Technical Report Series
Center for Data and Simulation Science

Matthias Eichinger, Alexander Heinlein, Axel Klawonn

Surrogate Convolutional Neural Network Models for Steady Computational Fluid Dynamics Simulations

Technical Report ID: CDS-2020-12
Available at https://kups.ub.uni-koeln.de/id/eprint/29760
Submitted on December 14, 2020

SURROGATE CONVOLUTIONAL NEURAL NETWORK MODELS FOR STEADY COMPUTATIONAL FLUID DYNAMICS SIMULATIONS *

MATTHIAS EICHINGER†, ALEXANDER HEINLEIN†§‡, AND AXEL KLAWONN†§

Abstract. A convolution neural network (CNN)-based approach for the construction of reduced order surrogate models for computational fluid dynamics (CFD) simulations is introduced; it is inspired by the approach of Guo, Li, and Iori [X. Guo, W. Li, and F. Iorio, Convolutional neural networks for steady flow approximation, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, New York, USA, 2016, ACM, pp. 481–490]. In particular, the neural networks are trained in order to predict images of the flow field in a channel with varying obstacle based on an image of the geometry of the channel. A classical CNN with bottleneck structure and a U-Net are compared while varying the input format, the number of decoder paths, as well as the loss function used to train the networks. This approach yields very low prediction errors, in particular, when using the U-Net architecture. Furthermore, the models are also able to generalize to unseen geometries of the same type. A transfer learning approach enables the model to be trained to a new type of geometries with very low training cost. Finally, based on this transfer learning approach, a sequential learning strategy is introduced, which significantly reduces the amount of necessary training data.

Key words. Convolutional neural networks, computational fluid dynamics, reduced order surrogate models, U-Net, transfer learning, sequential learning

1. Introduction. The development of machine learning techniques for the solution of differential equations is an important topic in the new, rapidly evolving field of scientific machine learning (SciML) [3]. For instance, machine learning techniques may be used in order to discretize the differential equations, in order to enhance classical numerical models and methods, or as reduced order surrogate models.

Examples of methods, where neural networks are used as discretizations for partial differential equations are the approaches by Lagaris et al. [24], physics-informed neural networks (PINNs) by Raissi et al. [31, 32], and the Deep Ritz approach by E and Yu [9]. In [15], PINNs have recently been applied to a parameter identification problem for systems of ordinary differential equations.

There is also a large number of recent publications on hybrid algorithms which use machine learning techniques to enhance classical numerical methods for solving differential equations; see, e.g., [17, 34, 18, 4, 19].

Here, we are interested in constructing surrogate models for computational fluid dynamics (CFD) simulations. This is particularly important since CFD simulations arise in many application areas, such as civil and mechanical engineering, meteorology, geosciences, or medical science. Accordingly, there is a wide range of settings and fluids with varying complexity in their modeling. In most cases, however, the computational work in order to obtain accurate results is considerably high. Therefore, the development of reduced order models, which may reduce the computational cost, is of great interest. There is a wide range of classical model order reduction (MOR) techniques, such as principal component analysis (PCA) or proper orthogonal decom-

*Submitted to the editors DATE.

†Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931 Köln, Germany. eichingm@smail.uni-koeln.de, alexander.heinlein@uni-koeln.de, axel.klawonn@uni-koeln.de, url: http://www.numerik.uni-koeln.de.

‡Institute for Applied Analysis and Numerical Simulation, University of Stuttgart, Germany.

§Center for Data and Simulation Science, University of Cologne, Germany, url: http://www.cds.uni-koeln.de.

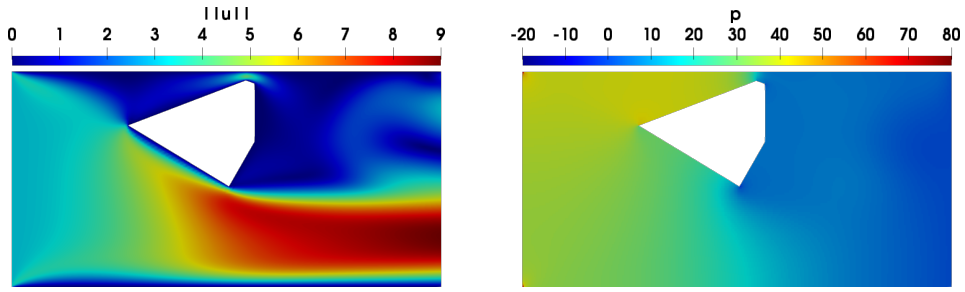


FIG. 1. *Solution of the steady Navier–Stokes equations (2.1) with boundary conditions (2.2) corresponding to the configuration in Figure 2.*

position (POD), Reduced Basis (RB), or simplified physics methods. In this regards, we refer to the extensive literature, e.g., [29, 33, 30, 37].

Here, we are interested in the use of artificial neural networks as reduced order surrogate models for CFD simulations. One approach of this category is the hybrid algorithm introduced in [8], which uses classical reduced basis solvers as the last layer in a neural network, to approximate the solution of parametrized PDEs. Also in [11], the authors build a reduced order model for cardiac electrophysiology by combining a convolutional auto-encoder and a deep feedforward neural network. In contrast to most other approaches, we are interested in geometry-dependent flow predictions, where the geometry is used as the input for the neural network and the whole flow field is obtained as the output. Our approach is inspired by the work of Guo, Li, and Iorio [16] and has already been partly presented in [10]. In particular, we employ convolutional neural networks (CNNs) in order to map from an image of the geometry to images of the resulting flow field, making use of the strengths of CNNs in processing image data. Therefore, we employ the bottleneck CNN from [16] as well as the U-Net, which has been introduced in Ronneberger et al. [35] for biomedical image segmentation.

In order to generate synthetic flow data for the training and validation phase, we use a software pipeline including the mesh generation tool snappyHexMesh and the simpleFoam CFD solver; both are part of OpenFOAM 5.0 [14]. In order to implement and train the neural networks, we use Keras 2.2.4 [6] with Tensorflow 1.12 [2] backend.

This paper is structured as follows. First, in section 2, we introduce the considered stationary CFD boundary value problem. Then, we describe our approach of constructing a surrogate CNN model and the employed network architectures in section 3 and the data generation process in section 4. In sections 5 and 6, we describe the training procedure and implementation and efficiency of the neural networks, respectively. Next, we provide results for two types of obstacle geometries in section 7 and discuss a transfer learning approach for extending the model to another type of geometries in section 8. Finally, before we conclude the paper in section 10, we introduce an iterative sequential learning approach, which enables us to significantly reduce the amount of necessary training data in section 9.

2. Stationary flow problem. As the model problem, we consider the stationary flow of an incompressible Newtonian fluid with kinematic viscosity $\nu > 0$ within a computational domain Ω_P. This can be described by the steady Navier–Stokes

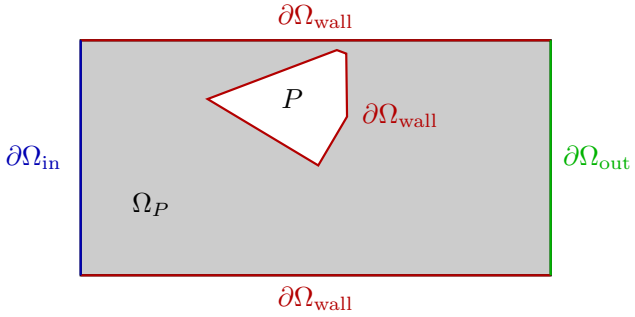


FIG. 2. *The computational domain Ω_P is a channel Ω of length 6 and width 3 with a polygonal obstacle P. We prescribe an inflow velocity at $\partial\Omega_{\text{in}}$, no-slip boundary conditions on $\partial\Omega_{\text{wall}}$, and do-nothing boundary conditions on $\partial\Omega_{\text{out}}$.*

equations

$$\begin{aligned} -\nu\Delta u + (u\cdot\nabla)\,u + \nabla p &= f \text{ in } \Omega_P, \\ \nabla\cdot u &= 0 \text{ in } \Omega_P, \end{aligned} \tag{2.1}$$

where u and p are the velocity and pressure variables. We will only consider the case where the volume force f is zero.

As a proof of concept, we will restrict ourselves to geometries of a specific type. In particular, we consider two-dimensional channels $\Omega_P := [0,6]\times[0,3]\setminus P$, where $P \subset [0,6]\times[0,3]$ is a polygonal star-shaped domain; see Figure 1 for an example with corresponding solution. Moreover, we apply the boundary conditions

$$\begin{aligned} u &= \begin{pmatrix} 3 \\ 0 \end{pmatrix} && \text{on } \partial\Omega_{\text{in}}, \\ \frac{\partial u}{\partial n} - pn &= 0 && \text{on } \partial\Omega_{\text{out}}, \text{ and} \\ u &= 0 && \text{on } \partial\Omega_{\text{wall}}; \end{aligned} \tag{2.2}$$

cf. Figure 2. In particular, we prescribe an inflow velocity on the left boundary of the channel and a do-nothing natural boundary condition at the right boundary of the channel. At the upper and lower boundaries as well as the boundary of the obstacle P we impose no-slip boundary conditions.

3. Surrogate convolutional neural network. Our main idea is to train a convolutional neural network (CNN)

$$\begin{aligned} \mathcal{CNN}: \quad \mathbb{R}^{w\times h} &\to \mathbb{R}^{2\times w\times h} \\ g &\mapsto u = \begin{pmatrix} u_x \\ u_y \end{pmatrix} \end{aligned}$$

which maps from an image representation of the geometry g to the corresponding velocity field u consisting of an image representations of its x component u_x and its y component u_y. The velocity field is obtained by solving (2.1) with boundary conditions (2.2). Here, w is the width of the input image and the output images and

h is the corresponding height. Our approach is inspired by the work of Guo, Li, and Iorio [16] and has already been presented partly in [10].

Convolutional neural networks [25] are specialized neural networks for data with a tensor product grid-like topology; see also [13, Chapter 9]. Examples for this type of data are, e.g., one-dimensional time series or 2D and 3D images. Whereas the application of one layer in a dense neural networks can be written as

$$y = \alpha\left(Wx + b\right)$$

with input vector x, output vector y, dense weight matrix W, bias vector b, and activation function $\alpha(\cdot)$, in a CNN, the matrix multiplication with the dense matrix W is replaced by a convolutional operation. Moreover, convolutional neural networks typically use so-called pooling operations. Both convolution and pooling operations are specialized linear operations which only allow for interaction of neighbored neurons, in the sense of the underlying grid-structure of the data. This is implemented by local multiplications with a smaller filter F matrix, which is moved point by point over the whole data grid. In particular, during the training, the coefficients of the filter matrix F are trained. Then, the application of a convolution or pooling operation can also be written as a matrix vector multiplications with a sparse matrix S, which can be obtained by assembly of the local filter matrix. Hence, the application of a convolutional layer corresponds to

$$y = \alpha\left(Sx + b\right).$$

Convolution and pooling operations may reduce the size of data vector. However, one typically uses multiple filters corresponding to the multiplication with multiple sparse matrices S_i. Hence, we also obtain multiple output vectors (channels)

$$y_i = \alpha\left(S_i x + b\right) \quad i = 1, ..., N.$$

This helps to prevent the loss of important features from a radical reduction of the dimension; cf., e.g., [13, Chapter 9] and [5, Chapter 5] for more details on convolutional neural networks.

In Figure 3, the different types of layers of the employed convolutional neural network architectures are visualized as boxes with different width and height to account for the dimension of the output image of the corresponding layer as well as the length accounting for the number of channels of the output. We consider three different types of convolutional layers:

Convolution: We apply padding before the convolution, such that the dimension of the data is kept the same.

Down-convolution: We apply striding, i.e., we increase the shift of the local filter within the convolution operation, such that the data dimension is reduced. In this case, we increase the number of channels.

Up-convolution: We use a transposed convolution with striding to increase the dimension of the data. At the same time, we reduce the number of channels.

As activation functions, we use the identity (*linear*) or the Rectified Linear Unit (ReLU) [21, 27, 12] function (*ReLU*). For more details on padding and striding, see also [13, Chapter 9] and [5, Chapter 5].

In contrast to other works, which use dense neural networks to approximate the solutions of fixed boundary value problems itself, such as [31, 9], we are here interested in constructing a functional relation between images of the geometry and the solution.

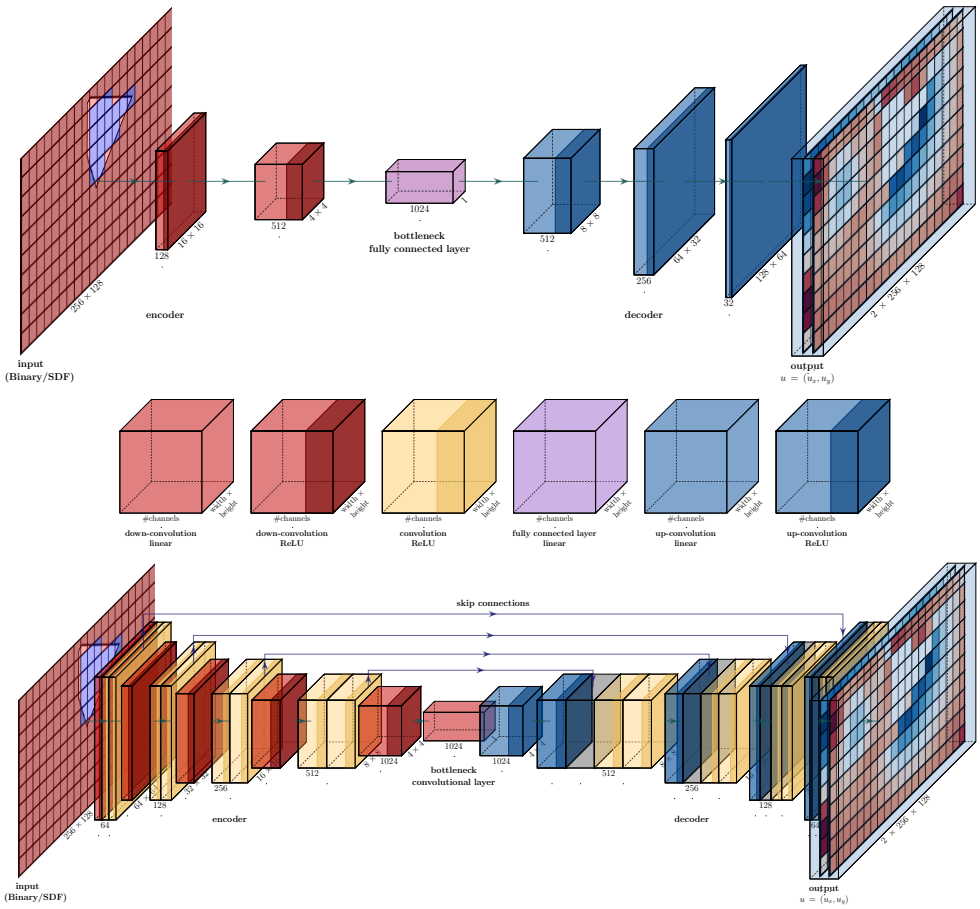


FIG. 3. *Architectures of the two considered convolutional neural networks: classical bottleneck CNN (top) and U-Net (bottom); cf. section 3 for more details on the graphical representation. Only one decoder path is depicted; cf. subsection 3.2 and Figure 4. Visualization using [1].*

This means that we are dealing with given discretization of the input and output, whereas other approaches use the neural network itself as the discretization of the PDE. Therefore, in contrast to those approaches, we are able to incorporate a variation of the geometry.

3.1. Network architectures. As surrogate models, we use two different convolutional neural network architectures.

Bottleneck CNN [16]. As the first network architecture, we consider the CNN from the work [16]; see Figure 3 (top). This CNN has a bottleneck structure such that a reduced dimension latent space of dimension 1 024 is learned from the data. The first part of the network is an encoder, the second part a corresponding decoder.

U-Net [35]. The second network architecture is inspired by the U-Net introduced by Ronneberger, Fischer, and Brox in [35]; see Figure 3 (bottom). Our network follows the basic structure of the U-Net but the convolutional layers are adapted to our problem. Similar to the previously described CNN, it is also a convolutional neural

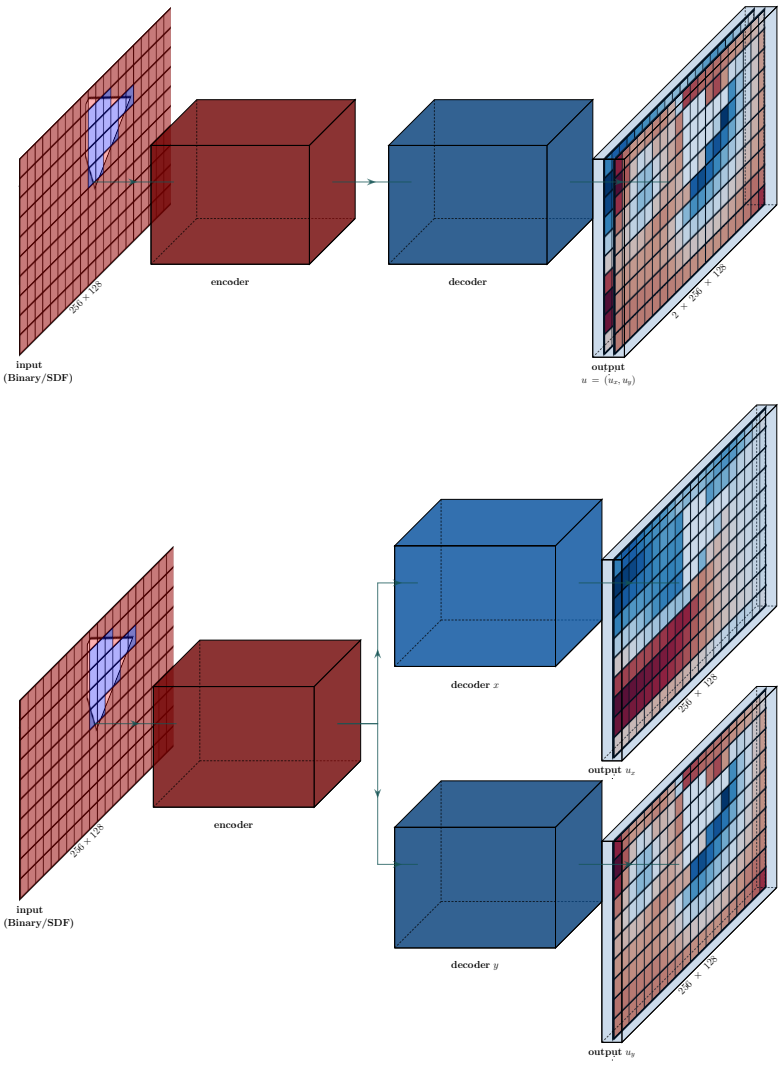


FIG. 4. *Abstract graphical representation of the network architectures with one (top) or two (bottom) decoder paths. Visualization using [1].*

network with bottleneck structure. However, additional skip connections are introduced as a regularization which is necessary due to the high number of convolutional layers. At the same time, we loose the property that all geometrical information has to be compressed into a vector of length 1 024. Originally, the U-Net has been introduced in the context of biomedical image segmentation, however, as we will observe in section 7, it also works very well for the prediction of flow fields.

3.2. One and two decoder paths. As in the work [16], we will also compare using one and two decoder paths in our neural networks. As pointed out in subsection 3.1, both networks have a bottleneck type structure, where the first part corresponds to the encoder, which learns the geometrical features of the input image, and the second part corresponds to the decoder, which predicts the flow images in x

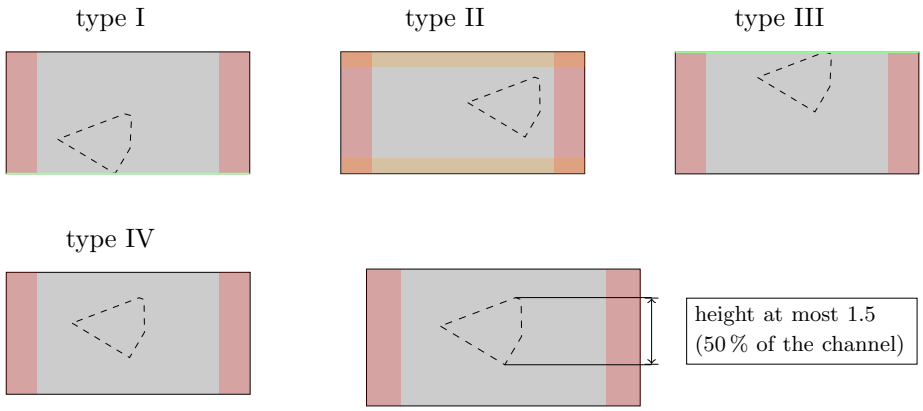


FIG. 5. *Graphical representation of all geometry configurations considered in this paper. In all cases, a minimum distance of* 1.5 *to the inlet (left) and outlet (right) is assumed and the maximum height of the obstacle is* 1.5*, i.e.,* 50 *% of the channel. Type I geometries touch the lower part of the boundary, type II geometries have a minimum distance of* 0.75 *to the lower and upper parts of the boundary, type III geometries touch the upper part of the boundary, and type IV have no restriction with respect to the vertical position of the obstacle.*

and y direction based on the geometrical features from the encoder. Based on this structure of the network architecture, we investigate in section 7 the case of training two separate decoder paths for the velocities in x and y direction. This clearly increases the number of parameters of the neural network because the decoder part is doubled in size; see Figure 4. On the other hand, this may help to better predict the two velocity components.

The total number of parameters of the bottleneck CNN and the U-Net with one or two decoder paths are listed in Table 1.

4. Generation of training data. In order to build a surrogate machine learning model for the high-fidelity CFD simulations, we first have to generate a large set of flow data. To ensure validity of the model without imposing additional (physical) knowledge, this training data has to sufficiently cover the possible input space, i.e., the space of possible geometries; cf., e.g., [7]. In order to create such an extensive database, we have set up an automatized software pipeline, which enables the simulation of a large amount of different geometrical configurations, without the need for additional user interaction. This pipeline consists of the following steps, which will be discussed in the further subsections:

- Definition of an obstacle geometry; see subsection 4.1
- Generation of a corresponding hexahedral simulation mesh; see subsection 4.2
- CFD simulation using the finite volume solver OpenFOAM; see subsection 4.3
- Data interpolation to a fixed pixel grid; see subsection 4.4

We will further describe the specific data sets used in our experiments in the respective sections 7 to 9.

4.1. Definition of the geometry. As previously described in section 2, we will consider a two-dimensional channel $\Omega := [0, 6] \times [0, 3]$ and then remove a polygonal star-shaped obstacle P, such that the computational domain is $\Omega_P = \Omega \setminus P$. Since the channel Ω is fixed, the only freedom in the geometry is the definition of the polygonal obstacle. This is performed by a random placement of the corners of the polygon

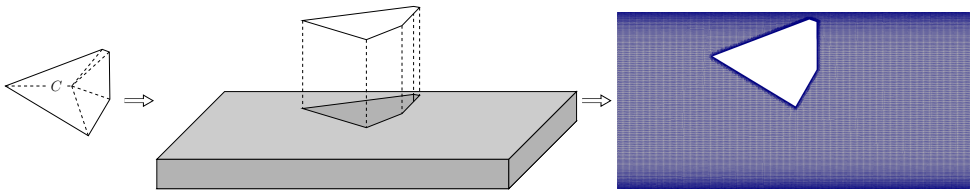


FIG. 6. *Mesh generation using snappyHexMesh: The obstacle geometry is defined in STL format and placed within the channel. Then, the mesh is generated, such that it is additionally refined near* $\partial\Omega_{\text{wall}}$.

under the following restrictions.

In general, we assume that the obstacle has a minimum distance of 0.75 from the inlet and outlet, i.e., from the left and the right boundary of the channel. Furthermore, the vertical extension of the obstacle should not be larger than 1.5, i.e., 50 % of the height of the channel; cf. Figure 5 (bottom right). We first only consider obstacles which touch the lower boundary (type I), obstacles which do not touch any boundary and have a minimum distance of 0.75 to the lower and upper boundary (type II), and obstacles which touch the upper boundary (type III). Later, in section 9, we will also consider type IV geometries, which have no restriction regarding the vertical location; see Figure 5.

In addition to the types I, II, and III, we also categorize the polygonal obstacles with respect to the number of edges. In general, we will only consider obstacles with less or equal than 20 edges; only in Figure 11, we neglect this constraint and consider a circular obstacle.

Note that we generally do not constrain the angles of the polygons, such that we may obtain thin obstacles which can only be poorly resolved by our input image; cf. subsection 4.4.

4.2. Mesh generation. The mesh generation procedure is sketched in Figure 6. In particular, we first describe the obstacle geometry, which has been previously generated along the description in subsection 4.1, using the STL (Standard Triangle Language) format. Then, we use the mesh generator blockMesh to create an underlying hexagonal mesh of the channel Ω. Then, we cut out the obstacle and create a mesh, which is refined near $\partial\Omega_{\text{wall}}$, using the tool snappyHexMesh. Both tools, blockMesh and snappyHexMesh, are part of OpenFOAM 5.0 [22, 14] and will not be discussed here in detail.

4.3. OpenFOAM simulations. The CFD simulations of the steady Navier–Stokes equations with corresponding boundary conditions as described in section 2 are performed using the simpleFoam solver in OpenFOAM 5.0 [22, 14]. In particular, the Navier–Stokes equations are first discretized using finite volumes [38, 26], and then, the resulting discrete nonlinear problem is solved using the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) method [28]. In particular, the pressure equations are solved using an geometric algebraic multi grid (GAMG) solver and the remaining velocity equations are then solved using an symmetric Gauss–Seidel iteration. An exemplary solution is depicted in Figure 1. For more details, we refer to the OpenFOAM user guide [14].

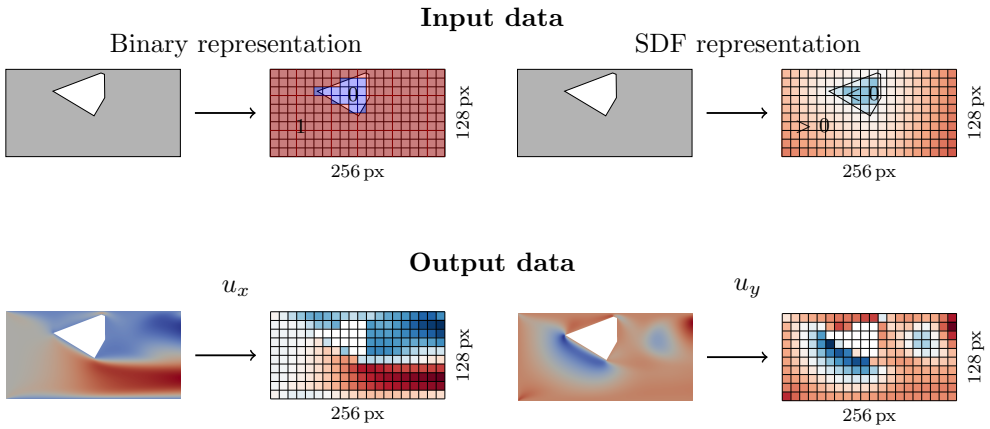


FIG. 7. *The structured input and output data for the CNNs is obtained by evaluation in the centers of the pixels of a* 256×128 *pixel grid.*

4.4. Data interpolation. As already pointed out in section 3, neural networks and, in particular, convolutional neural networks rely on input and output data with a fixed structure. Therefore, in order to train a neural network as a surrogate model for the previously described CFD simulations (subsection 4.3), the input and output data has to be transformed to a fixed structure. In particular, we are using a CNN that requires an image structure for the data; cf. section 3. Therefore, we will use two different image representations of the geometry, i.e., a signed distance function (SDF) representation and a binary representation; cf. [16, 10]. Both representations are 256×128 px images. Furthermore, we will interpolate the flow field onto a pixel grid of fixed 256×128 px size; see also Figure 7.

Signed distance function (SDF) input. The SDF input includes information about the obstacle as well as information about the minimum distance of any given pixel to the boundary of the obstacle. In particular, the signed distance function for a given pixel p is given as the minimum positive distance of the center of the pixel c_p to the boundary of the obstacle. However, in case that the center of the pixel lies in the interior of the obstacle, it is the negative distance.

Binary input. The binary input contains a reduced amount of information compared to the SDF input. In particular, the binary input value in a given pixel p is defined as zero if its center c_p is located in the interior of the obstacle and one otherwise. Hence, it can be obtained from the SDF input by mapping any positive value to one and any negative value to zero.

Output. The output data for a given configuration is obtained from the velocity field resulting from the corresponding OpenFOAM simulation; cf. subsection 4.3. In particular, each component of the discrete solution $u = \begin{pmatrix} u_x \\ u_y \end{pmatrix}$ is interpolated onto a 256×128 pixel image by evaluation in the center c_p of each pixel p. Hence, we obtain two pixel images representing the velocity field for each configuration.

Note that, since we do not constrain the angles of the corners of the polygon, it is possible that the geometry cannot be resolved accurately by a 256×128 pixel grid.

5. Training. In the training phase, we optimize the parameters of the neural networks to minimize different loss functions. In particular, we consider combinations

of a mean squared error (MSE) and a mean absolute error (MAE) with the mean relative error

$$\frac{1}{|D|}\sum_{P\in D}\frac{1}{|I_P|}\sum_{p\in I_P}\frac{\|u_p-\hat{u}_p\|_2}{\|u_p\|_2+10^{-4}}, \tag{5.1}$$

where u_p is the velocity prediction vector in the pixel p and $\hat{u}_p$ is the reference velocity in p, i.e., the evaluation of the CFD solution in the center c_p of the pixel. Furthermore, D is the set of all considered obstacles P and I_P is the set of all pixels p which are not covered by the respective obstacle; hence, we neglect the error of the velocity prediction within the obstacle. We add 10^{-4} as a regularization term in the denominator in case of very small values in the reference velocity, i.e., $\|u_p\|_2 \approx 0$. In particular, we consider combinations with (5.1) because we will later use this as the error measure on the validation and test data in order to investigate the performance of the surrogate model; cf. (7.1). Note that this error measure alone was not sufficient as the loss function to train the neural networks, and a reasonable reduction of the loss during the optimization was only possible when adding either the MSE or the MAE.

In total, we consider the four loss functions

$$\begin{aligned}
\mathrm{MSE} &= \frac{1}{|D|}\sum_{P\in D}\frac{1}{|I_P|}\sum_{p\in I_P}\left(\|u_p-\hat{u}_p\|_2^2\right),\\
\mathrm{MSE}^+ &= \frac{1}{|D|}\sum_{P\in D}\frac{1}{|I_P|}\sum_{p\in I_P}\left(\|u_p-\hat{u}_p\|_2^2+\frac{\|u_p-\hat{u}_p\|_2}{\|u_p\|_2+10^{-4}}\right),\\
\mathrm{MAE} &= \frac{1}{|D|}\sum_{P\in D}\frac{1}{|I_P|}\sum_{p\in I_P}\left(\|u_p-\hat{u}_p\|_1\right), \text{ and}\\
\mathrm{MAE}^+ &= \frac{1}{|D|}\sum_{P\in D}\frac{1}{|I_P|}\sum_{p\in I_P}\left(\|u_p-\hat{u}_p\|_1+\frac{\|u_p-\hat{u}_p\|_2}{\|u_p\|_2+10^{-4}}\right).
\end{aligned} \tag{5.2}$$

Furthermore, in subsection 7.2, we will discuss results for weighted variants of the MSE and MAE loss functions in order to improve the errors in the vicinity of the obstacles.

In order to optimize the parameters of the CNNs, we apply a stochastic gradient descent (SGD) up to a maximum number of 300 epochs; in case of stagnation in the reduction of loss for at least 50 epochs, we reduce the learning rate by 20 %. Moreover, we use a batch size of 64 and an adaptive scaling of the learning rate using the Adam (Adaptive moments) [23] algorithm with initial learning rate $\lambda = 0.001$. Furthermore, we observed that the training is improved if, in case of SDF input data, Z-normalization and, in case of binary input data, batch normalization is used; cf. [20].

6. Implementation and efficiency of the neural networks. Our implementation of the neural networks and the training algorithms uses the Keras 2.2.4 [6] with Tensorflow 1.12 [2] backend.

On an AMD Threadripper 2950X (8 × 3.8 Ghz) CPU with 32GB RAM, the average time for a serial computation of one configuration, including the mesh generation and the CFD simulation, took in the order of $O(10)$ s. In comparison, the evaluation of our neural networks took in the order of $O(0.1)$ s on the same CPU. Using a Nvidia GeForce RTX 2080Ti GPU, the evaluation of the neural networks was again accelerated by a factor of approximately 20. Therefore, we can confirm that our surrogate convolutional neural networks are significantly more efficient in the online phase.

	Bottleneck CNN		U-Net	
# decoders	1	2	1	2
parameters	≈ 47 m	≈ 85 m	≈ 34 m	≈ 53.5 m
time/epoch	180 s	245 s	195 s	270 s

TABLE 1

Training cost for the bottleneck CNN and the U-Net using one or two decoder paths. The times have been obtained from computations on a Nvidia GeForce RTX 2080Ti GPU; section 6.

However, the training phase of the neural networks is quite expensive, due to the very large number of parameters in the neural network. As listed in Table 1, one single epoch (for 90 000 configurations of training data and 10 000 configurations of validation data) during the training process took approximately 180 s - 270 s on the Nvidia GPU depending on the architecture of the network; the training cost on a CPU would be significantly higher. This highlights the high cost of the offline training phase.

7. Results on type I and II geometries. As a first step, we only consider geometries of type I and II; cf. subsection 4.1 and Figure 5. In particular, using our software pipeline described in section 4, we first generate a data set consisting of 100 000 geometry configurations, where 50 000 configurations correspond to type I obstacles and the remaining 50 000 to type II obstacles. Moreover, we restrict ourselves to polygonal obstacles with 3, 4, 5, 6, and 12 edges (10 000 each for type I and type II). Each obstacle is randomly generated under the conditions described in subsection 4.1. In order to train our neural networks, we perform a random split into 90 000 training data and 10 000 validation data.

As an error measure for our surrogate models, we use the mean relative error (5.1) evaluated on the validation data,

$$\frac{1}{|V|}\sum_{P\in V}\frac{1}{|I_P|}\sum_{p\in I_P}\frac{\|u_p-\hat{u}_p\|_2}{\|u_p\|_2+10^{-4}}, \tag{7.1}$$

where V is the set of all validation data. This error function is not equal to one of the loss functions, which are minimized for the 90 000 training data, but is part of two of the four considered loss functions (5.2); also compare for the discussion in section 5.

7.1. Results on the validation data. First, in Table 2, we compare the bottleneck CNN and the U-Net described in subsection 3.1 using SDF and binary input data, one or two decoder paths, and the four different loss functions (5.2). We list the relative errors for type I and type II validation data separately and additionally specify the relative error over all validation data. We observe that the U-Net generally yields better results compared to the bottleneck CNN. However, the best total relative errors obtained for the bottleneck CNN and the U-Net architectures are both very good with 3.85 % and 2.43 %. We observe that the bottleneck CNN benefits significantly from the use of the SDF input, which contains additional information compared to the binary input, whereas the U-Net is able to also extract this information from the binary input data; hence, the U-Net produces good results for both input data types.

It is not clear whether MSE or MAE is the better choice as the basis for the loss function. However, we clearly observe that adding the mean relative error (5.1) to the loss always improves the results, for both network architectures. However, as already

			Bottleneck CNN			U-Net		
input	# dec.	loss	total	type I	type II	total	type I	type II
SDF	1	MSE	61.16 %	110.46 %	11.86 %	17.04 %	29.42 %	4.66 %
		MSE$^+$	3.97 %	3.31 %	**4.63 %**	2.67 %	2.11 %	3.23 %
		MAE	25.19 %	41.52 %	8.86 %	9.10 %	13.89 %	4.32 %
		MAE$^+$	4.45 %	3.84 %	5.05 %	2.48 %	1.87 %	**3.10 %**
	2	MSE	49.82 %	89.12 %	10.51 %	13.01 %	21.59 %	4.42 %
		MSE$^+$	**3.85 %**	**3.05 %**	4.64 %	**2.43 %**	**1.78 %**	3.23 %
		MAE	45.23 %	81.38 %	9.08 %	5.47 %	7.06 %	3.89 %
		MAE$^+$	4.33 %	3.74 %	4.91 %	2.57 %	1.98 %	3.17 %
Binary	1	MSE	49.78 %	88.28 %	11.28 %	27.15 %	49.15 %	5.15 %
		MSE$^+$	10.12 %	11.44 %	8.80 %	5.49 %	6.25 %	4.74 %
		MAE	39.16 %	64.77 %	13.54 %	15.69 %	26.36 %	5.02 %
		MAE$^+$	10.61 %	12.34 %	8.87 %	**4.48 %**	**5.05 %**	**3.90 %**
	2	MSE	51.34 %	91.20 %	11.48 %	24.00 %	43.14 %	4.85 %
		MSE$^+$	10.03 %	11.37 %	8.69 %	5.56 %	6.79 %	4.33 %
		MAE	37.16 %	62.01 %	12.32 %	21.54 %	38.12 %	4.96 %
		MAE$^+$	**9.53 %**	**10.91 %**	**8.15 %**	6.04 %	7.88 %	4.20 %

TABLE 2

Comparison of the performance of the bottleneck CNN and the U-Net based on the error (7.1): variation of the input type, the number of decoder paths, and the loss function; cf. (5.2). The best errors for a given CNN architecture and input type are marked in **bold face**. *Taken fom [10].*

noted in section 5, the networks could not be trained using just the mean relative error (5.1) as the loss function.

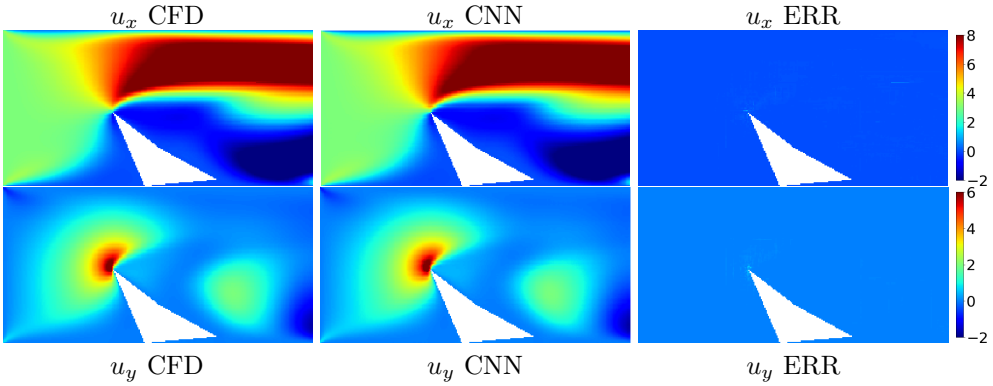
Furthermore, the results are not conclusive about whether a second decoder path should be used, or not. In particular, since the number of parameters and hence the trainings cost is significantly increased, it seems that the use of a single decoder is the more efficient choice.

Overall, many different configurations for the surrogate CNN yield very good results. However, we restrict ourselves to one specific network architecture and loss function for further experiments. In particular, following our findings, we will always consider the U-Net with one decoder path and MAE$^+$ loss.

Additionally, for this configuration and using SDF input, in Figure 8, we present exemplary results comparing the reference CFD solution and the CNN prediction. Whereas, in Figure 8, the results are in very good agreement qualitatively as well as quantitatively (errors of approximately 2 %), larger qualitative und quantitative differences can be observed in Figure 9 (errors of approximately 17 % and 15 %); see also [10] for additional results. To avoid these outliers and further reduce the errors will be subject of future research.

7.2. Spatial weighting of the loss function. In many applications, the velocities near the obstacle walls are of particular interest, e.g., for the computation of the wall shear stresses. In order to improve the accuracy of the prediction in the vicinity of the obstacles, we investigate weighted variants of the MSE and MAE loss

Type I:



Type II:

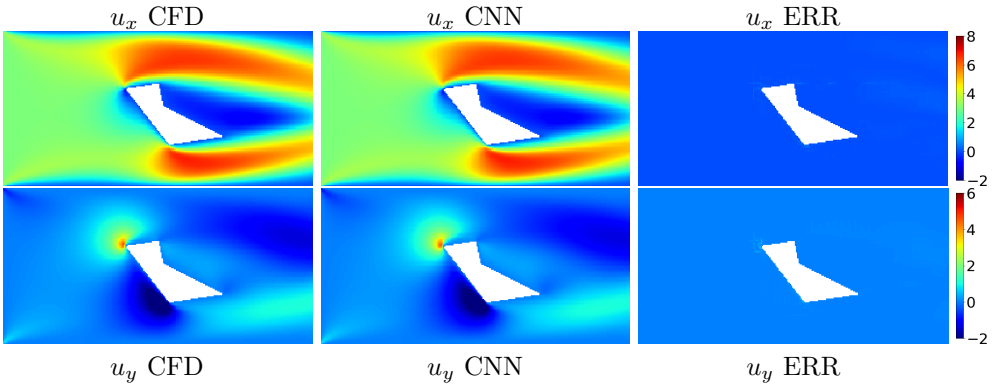


FIG. 8. *Type I & II geometry: comparison of the CFD flow field (left) computed using OpenFOAM, the CNN prediction (middle), and the pointwise error (right) for x (top) and y (bottom) component. Both examples yield a low mean relative error (7.1) of approximately 2 %.*

functions,

$$\text{MSE}_\omega = \frac{1}{|D|}\sum_{P\in D}\frac{1}{|I_P|}\sum_{p\in I_P} W_\omega(p)\left(\|u_p-\hat{u}_p\|_2^2\right) \quad \text{and} \tag{7.2}$$

$$\text{MAE}_\omega = \frac{1}{|D|}\sum_{P\in D}\frac{1}{|I_P|}\sum_{p\in I_P} W_\omega(p)\left(\|u_p-\hat{u}_p\|_1\right) \tag{7.3}$$

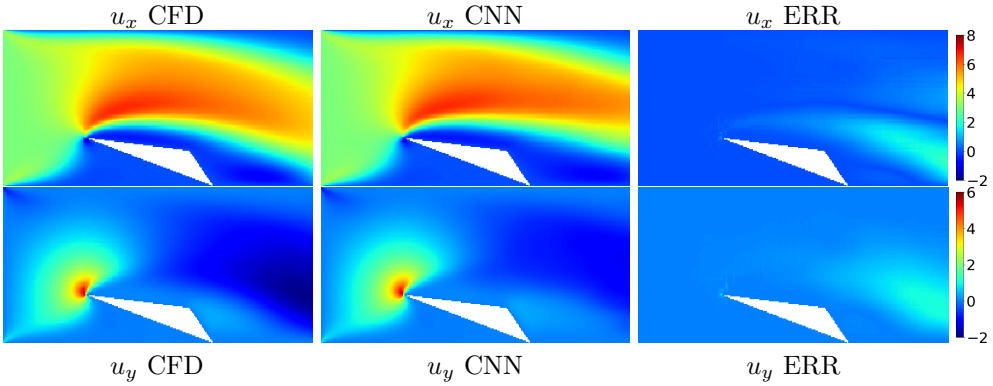
with the weight function

$$W_\omega(p) = \max\left\{1, \omega e^{-2\ln(\omega)\|c_p-P\|}\right\} \tag{7.4}$$

and the weight parameter $\omega > 1$. This weight function is chosen to be high at the boundary of the obstacle, i.e., $W_\omega(p) = \omega$ for $\|c_p - P\| = 0$. Then, the function decays exponentially and reaches a value of 1 in a distance of 0.5 to the obstacle, i.e., $W_\omega(p) = 1$ for $\|c_p - P\| \geq 0.5$. Moreover, if $\omega = 1$, $W(p) \equiv 1$ and the weighted loss functions are equal to the unweighted loss functions.

In order to investigate the effect of the weight function $W(p)$ on the error in the

Type I:



Type II:

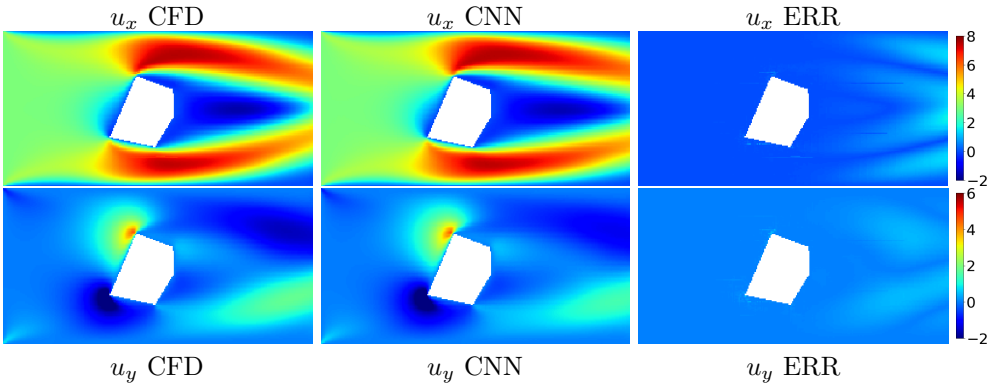


FIG. 9. *Type I & II geometry: comparison of the CFD flow field (left) computed using OpenFOAM, the CNN prediction (middle), and the pointwise error (right) for x (top) and y (bottom) component. Both examples yield higher mean relative errors (7.1) of approximately 17 % (type I) and 15 % (type II).*

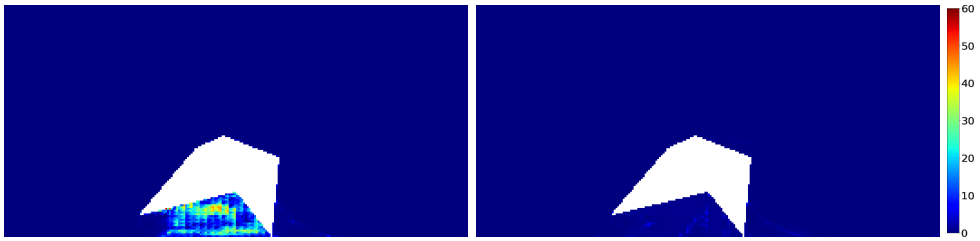


FIG. 10. *Visualization of the pixel-wise relative error (7.7) for a type I geometry. Comparison for the unweighted MAE loss function (5.2) (left) and the weighted MAE loss function (7.3) with $\omega = 250$ (right).*

vicinity of the obstacle, we investigate a split of the error function (7.1) into

$$\frac{1}{|V|}\sum_{P\in V}\frac{1}{|I_P|}\sum_{\substack{p\in I_P\\ \|c_p - I_P\|\leq 0.5}}\frac{\|u_p-\hat{u}_p\|_2}{\|u_p\|_2+10^{-4}}, \tag{7.5}$$

		error (7.1)			error (7.5)		error (7.6)	
loss	ω	total	type I	type II	type I	type II	type I	type II
MSE_ω (7.2)	1	17.04 %	29.42 %	**4.66 %**	28.24 %	2.39 %	**1.19 %**	**2.29 %**
	50	**9.41 %**	13.25 %	5.56 %	9.95 %	1.78 %	3.32 %	3.79 %
	100	12.11 %	18.32 %	5.90 %	14.74 %	1.76 %	3.59 %	4.16 %
	250	10.95 %	**14.21 %**	7.68 %	**8.19 %**	**1.69 %**	6.04 %	6.01 %
	500	13.94 %	17.72 %	10.16 %	9.89 %	2.10 %	7.85 %	8.07 %
MAE_ω (7.3)	1	9.11 %	13.89 %	4.32 %	12.96 %	2.19 %	**0.94 %**	**2.14 %**
	50	4.04 %	4.55 %	**3.53 %**	3.51 %	1.28 %	1.05 %	2.27 %
	100	3.68 %	3.82 %	3.54 %	2.61 %	1.19 %	1.22 %	2.37 %
	250	**3.63 %**	**3.64 %**	3.62 %	**2.33 %**	**1.18 %**	1.32 %	2.45 %
	500	3.83 %	3.88 %	3.78 %	2.46 %	1.19 %	1.43 %	2.60 %

TABLE 3

Results on validation data for a higher weight of the loss corresponding to errors in the vicinity of the obstacle using a U-Net with one decoder path and SDF input. The weight parameter ω in the function (7.4) is varied between 1 and 500, and the global error (7.1) as well as split of the error into pixels with a maximum distance of 0.5 to the obstacle (error (7.5)) and remaining pixels (error (7.6)) are listed. The best errors among the different weight factors are marked in **bold face**.

which corresponds to the error in the pixels near the obstacle, and

$$\frac{1}{|V|}\sum_{P\in V}\frac{1}{|I_P|}\sum_{\substack{p\in I_P\\ \|c_p-I_P\|>0.5}}\frac{\|u_p-\hat{u}_p\|_2}{\|u_p\|_2+10^{-4}}, \tag{7.6}$$

which corresponds to the remainder of the channel.

We present our results for varying values of ω in Table 3. As can be observed, the errors (7.5) and (7.6) are comparable for type II geometries, whereas the error near the obstacle is significantly higher for type I geometries; see Figure 10 (left) for an example of a type I obstacle with high relative errors

$$\frac{\|u_p-\hat{u}_p\|_2}{\|u_p\|_2+10^{-4}}. \tag{7.7}$$

in pixels near the obstacle for the unweighted MAE loss function (5.2). Hence, it is particularly important to improve the error near the obstacle for type I geometries.

The results in Table 3 show that, by increasing the weight parameter ω, we can reduce the error near the obstacle (7.5) while only slightly increasing the error in the remainder of the channel (7.6). The error reduction can also be seen in the example in Figure 10, where the pixel-wise relative error (7.7) near the obstacle is significantly reduced when using the weighted MAE loss function (7.3) with $\omega = 250$. Moreover, by choosing an appropriate value for ω, we can even improve the global error (7.1) averaged over all geometries compared to the unweighted loss functions. This shows that the distribution of the error can be controlled in a reasonable way by using a weighted loss function.

In the remainder of this paper, we will focus on using the MAE^+ loss function, however, it may be helpful in certain applications to also consider a weighted variant of this loss function.

7.3. Generalization to other type I and II geometries. In order to study the generalization properties of our U-Net-based surrogate models, we consider ad-

# polygon edges	SDF input			Binary input		
	total	type I	type II	total	type I	type II
7	2.71 %	1.89 %	3.53 %	4.39 %	4.61 %	4.16 %
8	2.82 %	1.98 %	3.65 %	4.67 %	4.89 %	4.44 %
10	3.21 %	2.32 %	4.10 %	5.23 %	5.51 %	4.94 %
15	4.01 %	3.16 %	4.86 %	7.76 %	7.85 %	6.66 %
20	5.08 %	4.22 %	5.93 %	9.70 %	10.43 %	8.97 %

TABLE 4

Results for the generalization properties of the U-Net with one decoder path, and MAE$^+$ loss function. Error (7.1) for polygonal obstacles with higher numbers of edges compared to the training and validation data: 1 000 *polygons (*500 *type I and* 500 *type II) for each number of edges; cf. [10].*

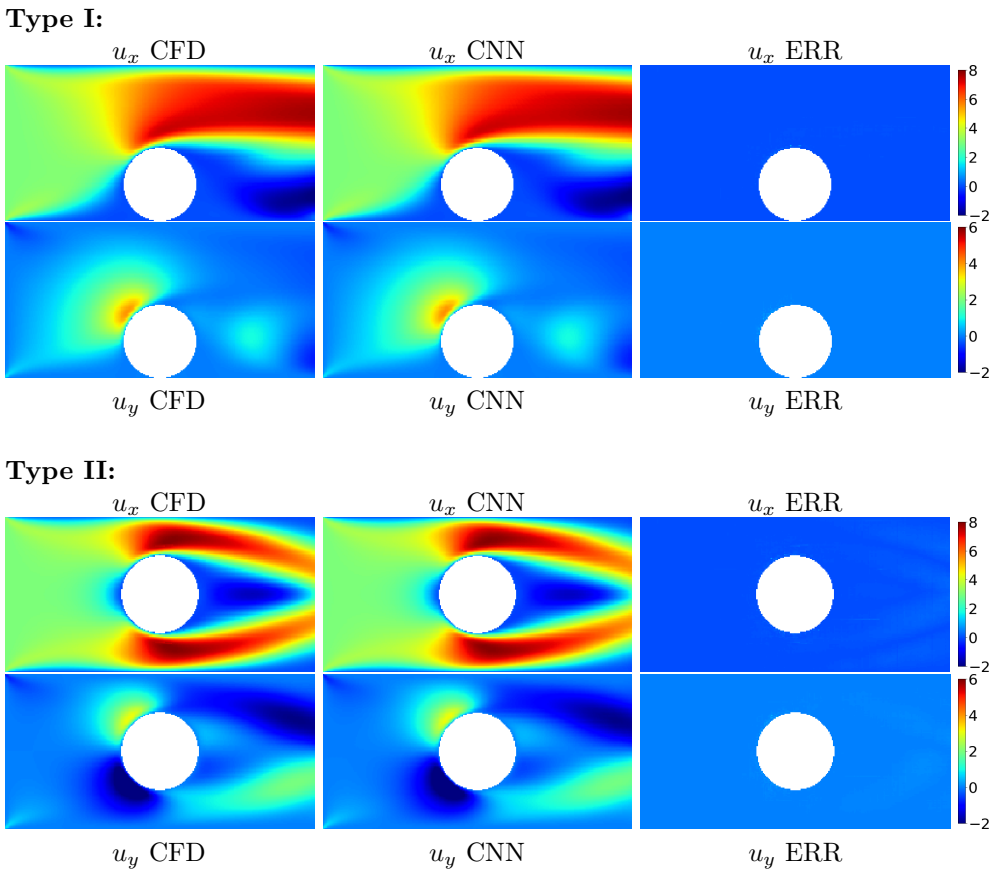


FIG. 11. *Comparison of the CFD flow field (top) computed using OpenFOAM, the CNN prediction (middle), and the pointwise error (bottom) for x (left) and y (right) component. Results for a geometry with circular obstacles of type I & II, which have not been part of the training or validation data. Nonetheless, the mean relative errors (7.1) are only approximately* 1 % *(type I) and* 3 % *(type II).*

ditional geometries, which have not been part of the initial data set. In particular, we consider polygonal obstacles with 7, 8, 10, 15, and 20 edges using the U-Net with one decoder path and MAE$^+$ loss function. Furthermore, we consider both SDF and binary input.

	SDF input	Binary input
type III	22 985.89 %	4 134.69 %

TABLE 5

Application to type III data of the U-Net with one decoder path, and MAE$^+$ loss function trained on type I and II data. The mean relative error (7.1) is given.

As can be observed in Table 4, the test results on geometries which were not part of the initial data set are very good with a maximum total error of less than 10 %. Again, the results are better using SDF input. Of course, the flow fields may become more complex for higher numbers of edges, and the corresponding network predictions become slightly worse.

In addition to that, we depict the results for two configurations with circular obstacles. The first example is of type I and the second example is a type II circular obstacle which is inspired by the benchmark problem introduced in [36]; see Figure 11. For these examples, we obtain good generalization results with mean relative errors of approximately 1 % and 3 %.

Finally, in Figure 12, we provide three examples of geometries which are clearly outside the validity range of our CNN model: a type III obstacle, two type II obstacles, and one large type II obstacle. For all corresponding model predictions, a degenerated flow field is visible, and the prediction error clearly deteriorates. In particular, we can observe that model prediction would rather fit to the case of a type II obstacle.

In order to further extend the validity range of our model and to overcome these issues, we will now discuss the application of transfer learning techniques. In particular, we focus on transfer learning for type III geometries.

8. Transfer learning for type III geometries. In order to investigate the generalization of our model to type III geometries, we generate a total of 5 000 additional polygonal obstacles of type III with 3, 4, 5, 6, and 12 edges; cf. subsection 4.1 and Figure 5 for the description of type III obstacles.

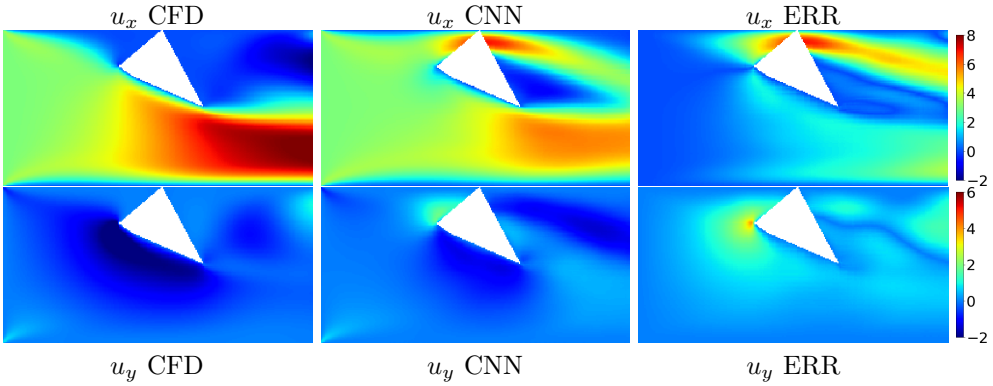
As can be observed in Table 5, the prediction performance on 2 500 randomly chosen validation data deteriorates independent of the input type; the errors 22 985.89 % and 4 134.69 % indicate that the neural network was essentially not able to predict the flow at all; see also the example in Figure 12. As previously described in section 4, this is due to the fact that the training data does not sufficiently cover the possible input geometries of type III; see also [7].

Now, we consider three different strategies to train type III geometries:

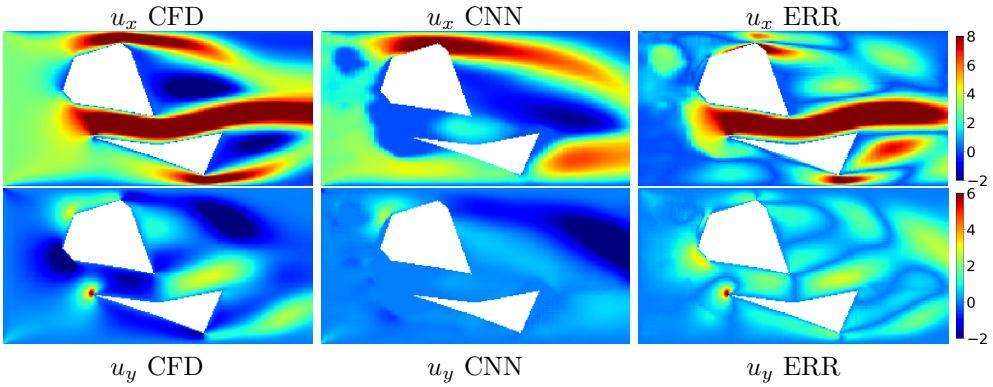
1. We train our U-Net with random initial parameters only using 2 500 training and 2 500 validation data of type III.
2. We use the trained U-Net from section 7 as the initial guess for training with only the 2 500 training and 2 500 validation data of type III.
3. We use the trained U-Net from section 7 as the initial guess for training on a combined data set. In particular, we use the 90 000 training and 10 000 validation data for type I and II geometries from section 7 and add the 2 500 training and 2 500 validation data of the type III geometries.

The results for the three different learning approaches are listed in Table 6. Using the first approach, we are not able to obtain any good results after 100 epochs of training on type III geometries; the error is in the order of 100 % for both SDF and binary input data. Presumably, this is due to the too small amount of training data. The second approach yields good results for type III geometries, which can

Type III:



2 × Type II:



Large Type II:

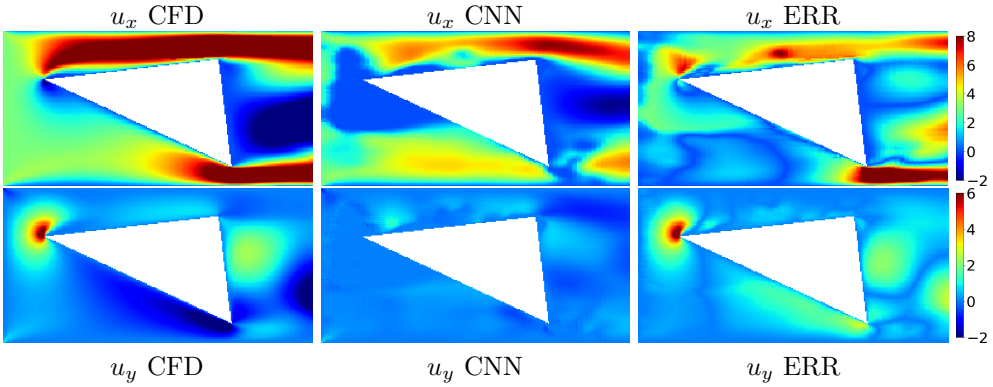


FIG. 12. *Comparison of the CFD flow field (top) computed using OpenFOAM, the CNN prediction (middle), and the pointwise error (bottom) for x (left) and y (right) component. Results for three types of geometries which have not been part of the training or validation data: a type III obstacle, two type II obstacles, and one large type II obstacle. The network which was trained on type I and type II geometries is not able to generalize to these geometries, which are clearly outside the validity range of the model. Mean relative errors (7.1) of approximately* 15777 % *(type III obstacle),* 140 % *(two type II obstacles), and* 67 % *(large type II obstacle) are obtained.*

		type I & II		type III	
learning approach	# training epochs	SDF input	Binary input	SDF input	Binary input
1	100	-	-	98.02 %	111.75 %
2	100	208.02 %	105.43 %	7.18 %	11.81 %
3	3	3.33 %	7.06 %	4.94 %	11.28 %

TABLE 6

Results of the error (7.1) for different learning approaches for type III data: training with random initial guess for type III data (approach 1), using a pre-trained neural network (on type I and II geometries) as the initial guess in training for type III data (approach 2), and using a pre-trained neural network (on type I and II geometries) as the initial guess in training for a combined data set with type I, II, and III data.

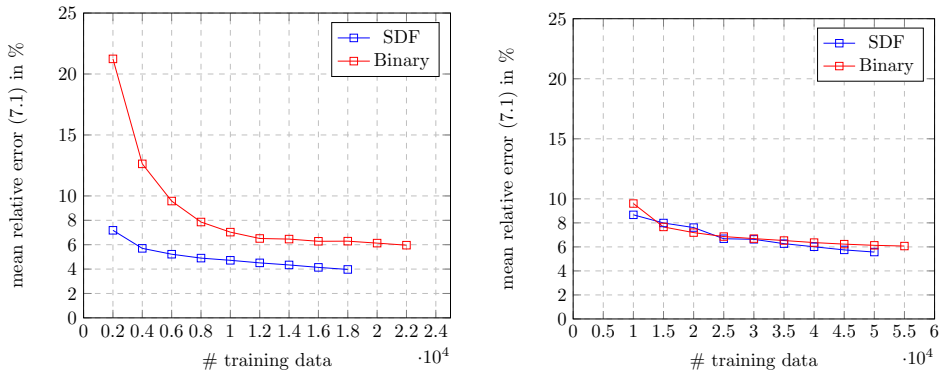


FIG. 13. *Mean relative error (7.1) for our sequential learning strategy as described in section 9: results for type I–III data (left) and for type IV data (right).*

be attributed to the good initial guess for the parameters of the neural network. However, since no type I or II geometries were part of the training data for the last 100 epochs, the corresponding prediction accuracy deteriorated. Our third approach is able to maintain the good prediction properties with respect to type I and II data while providing even better results for type III geometries compared to the second approach after only 3 epochs of additional training. This is remarkable since the same amount of type III as in the first two approaches has been used.

9. Sequential learning strategy. Based on the very promising results of the third transfer learning approach for type III geometries in section 8, we now propose a sequential learning strategy. The main idea is to start with a rather small training data set and enlarge it iteratively until a certain threshold for the validation error is reached. Therefore, we are able to reduce the amount of necessary data significantly.

First, we apply a sequential learning strategy for type I-III geometries. In particular, we first generate a fixed set of 10 000 validation data of type I, II, and III with polygonal obstacles with 3, 4, 5, 6, and 12 edges. We start with 2 000 random configurations as training data and random parameters of the neural network. Then, we iteratively add 2 000 additional random configurations and train the U-net using the third transfer learning strategy from section 8; in particular, we always use the neural network from the previous iteration as the initial guess. As can be observed

from Figure 13 (left), 18 000 configurations are sufficient to obtain a validation error of approximately 4 % for SDF input and 22 000 configurations with binary input to obtain an error of approximately 6 %. Hence, we are able to obtain better results compared to sections 7 and 8 using approximately 20 % of the data.

Finally, we apply the same sequential learning strategy to type IV data, i.e., data with no restriction of the vertical position. This data set is even more general than type I–III. In particular, it includes geometries which are close to the lower and upper wall, resulting in relatively high velocities close to these walls. Therefore, we start with a larger initial training data set of 10 000 and enlarge it by 5 000 in each iteration. We observe that a significantly higher amount of data is necessary to obtain an error of approximately 6 %, i.e., 45 000 and 55 000 configurations for SDF and binary input, respectively; cf. Figure 13 (right). However, this is still only approximately 50 % of the data used in sections 7 and 8. One further observation is that the errors for SDF and binary input do not differ as much as for the previous data sets.

Overall, using our sequential learning strategy, we are able to obtain very good prediction results with a relatively low amount of data.

10. Conclusion. In this paper, we have extensively investigated our approach of constructing a surrogate model for high-fidelity CFD simulations using convolutional neural networks; see also [16, 10]. In particular, in order to facilitate the prediction of flow fields for varying geometries, we employed convolutional neural networks mapping from a pixel image of the geometry to pixel images of velocity components in the computational domain. In this regard, our approach differs from the majority of the existing approaches, where a single neural network is used to discretize the solution of one single boundary value problem with a fixed geometry, whereas our approach is able to predict flow fields for varying geometries.

In order to investigate our approach, we have set up a software pipeline, which enables the generation of a large set of geometry and flow data. In a comparison of two convolutional neural network architectures using one or two decoders, different loss functions, and different input data, we have found that, in particular, the U-Net architecture, which has originally been introduced for biomedical image segmentation, is very robust and yields good predictions. Furthermore, we have obtained good generalization properties when applying the trained neural network to unseen geometries of the same type. Furthermore, we have presented an approach to reduce the error near the obstacles by introducing higher weights for the corresponding loss terms.

In order to transfer the model to new types of geometries, we have investigated an efficient transfer learning approach. Based on this, we have also described a sequential learning approach, where we iteratively enlarge the data set until an acceptable validation error is obtained. Using this approach, we were able to significantly reduce the amount of necessary training data.

On one exemplary machine, the evaluation of the convolutional neural networks is in the order of 100 times faster compared to generating a computational mesh and performing a CFD simulation using OpenFOAM.

REFERENCES

[1] *PlotNeuralNet GitHub repository*. https://github.com/HarisIqbal88/PlotNeuralNet. Accessed: 2020-08-14.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané,

R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015, https://www.tensorflow.org/. Software available from tensorflow.org.

[3] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, and K. Willcox, *Brochure on basic research needs for scientific machine learning: Core technologies for artificial intelligence*, https://doi.org/10.2172/1484362.

[4] A. Beck, D. Flad, and C.-D. Munz, *Deep neural networks for data-driven les closure models*, Journal of Computational Physics, 398 (2019), p. 108910, https://doi.org/10.1016/j.jcp.2019.108910, http://dx.doi.org/10.1016/j.jcp.2019.108910.

[5] F. Chollet, *Deep Learning with Python*, Manning Publications Co., USA, 1st ed., 2017.

[6] F. Chollet et al., *Keras*. https://keras.io, 2015.

[7] P. Courrieu, *Three algorithms for estimating the domain of validity of feedforward neural networks*, Neural Networks, 7 (1994), pp. 169 – 174, https://doi.org/https://doi.org/10.1016/0893-6080(94)90065-5, http://www.sciencedirect.com/science/article/pii/0893608094900655.

[8] N. Dal Santo, S. Deparis, and L. Pegolotti, *Data driven approximation of parametrized pdes by reduced basis and neural networks*, Journal of Computational Physics, 416 (2020), p. 109550, https://doi.org/10.1016/j.jcp.2020.109550, http://dx.doi.org/10.1016/j.jcp.2020.109550.

[9] W. E and B. Yu, *The Deep Ritz method: a Deep Learning-Based Numerical Algorithm for Solving Variational Problems*, Communications in Mathematics and Statistics, 6 (2018), pp. 1–12.

[10] M. Eichinger, A. Heinlein, and A. Klawonn, *Stationary flow predictions using convolutional neural networks*, technical report, Universität zu Köln, December 2019, https://kups.ub.uni-koeln.de/10440/.

[11] S. Fresca, A. Manzoni, L. Dedè, and A. Quarteroni, *Deep learning-based reduced order models in cardiac electrophysiology*, 2020, https://arxiv.org/abs/2006.03040.

[12] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, pp. 315–323.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[14] C. J. Greenshields, *Openfoam user guide, v5. 0*, OpenFOAM foundation Ltd, (2017).

[15] V. Grimm, A. Heinlein, A. Klawonn, M. Lanser, and J. Weber, *Estimating the time-dependent contact rate of sir and seir models in mathematical epidemiology using physics-informed neural networks*, technical report, Universität zu Köln, September 2020, https://kups.ub.uni-koeln.de/12159/.

[16] X. Guo, W. Li, and F. Iorio, *Convolutional neural networks for steady flow approximation*, in Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, New York, NY, USA, 2016, ACM, pp. 481–490, https://doi.org/10.1145/2939672.2939738, http://doi.acm.org/10.1145/2939672.2939738.

[17] D. Hartmann, C. Lessig, N. Margenberg, and T. Richter, *A neural network multigrid solver for the navier-stokes equations*, 2020, https://arxiv.org/abs/2008.11520.

[18] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber, *Machine Learning in Adaptive Domain Decomposition Methods - Predicting the Geometric Location of Constraints*, SIAM J. Sci. Comput., 41 (2019), pp. A3887–A3912.

[19] M. W. Hess, A. Quaini, and G. Rozza, *A comparison of reduced-order modeling approaches for pdes with bifurcating solutions*, 2020, https://arxiv.org/abs/2010.07370.

[20] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv:1502.03167, (2015).

[21] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, *What is the best multi-stage architecture for object recognition?*, 2009, pp. 2146–2153.

[22] H. Jasak, *Openfoam: Open source cfd in research and industry*, International Journal of Naval Architecture and Ocean Engineering, 1 (2009), pp. 89 – 94, https://doi.org/https://doi.org/10.2478/IJNAOE-2013-0011, http://www.sciencedirect.com/science/article/pii/S2092678216303879.

[23] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 1412.6980, (2014).

[24] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE transactions on neural networks, 9 (1998), pp. 987–

1000.
[25] Y. LeCun et al., *Generalization and network design strategies*, Connectionism in perspective, 19 (1989), pp. 143–155.
[26] F. Moukalled, L. Mangani, M. Darwish, et al., *The finite volume method in computational fluid dynamics*, vol. 6, Springer, 2016.
[27] V. Nair and G. E. Hinton, *Rectified linear units improve restricted boltzmann machines*, in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.
[28] S. Patankar and D. Spalding, *A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows*, International J. on Heat and Mass Transfer, 15 (1972), pp. 1787–1806.
[29] K. Pearson F.R.S., *LIII. On lines and planes of closest fit to systems of points in space*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2 (1901), pp. 559–572, https://doi.org/10.1080/14786440109462720, https://doi.org/10.1080/14786440109462720, https://arxiv.org/abs/https://doi.org/10.1080/14786440109462720.
[30] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced basis methods for partial differential equations: an introduction*, vol. 92, Springer, 2015.
[31] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations*, arXiv:1711.10561, (2017).
[32] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations*, arXiv:1711.10566, (2017).
[33] M. Rathinam and L. R. Petzold, *A new look at proper orthogonal decomposition*, SIAM Journal on Numerical Analysis, 41 (2003), pp. 1893–1925, https://doi.org/10.1137/S0036142901389049, https://doi.org/10.1137/S0036142901389049, https://arxiv.org/abs/https://doi.org/10.1137/S0036142901389049.
[34] D. Ray and J. S. Hesthaven, *An artificial neural network as a troubled-cell indicator*, Journal of Computational Physics, 367 (2018), pp. 166 – 191, https://doi.org/https://doi.org/10.1016/j.jcp.2018.04.029, http://www.sciencedirect.com/science/article/pii/S0021999118302547.
[35] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, (2015), pp. 234–241, https://doi.org/10.1007/978-3-319-24574-4_28, http://dx.doi.org/10.1007/978-3-319-24574-4_28.
[36] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher, *Benchmark computations of laminar flow around a cylinder*, Springer, 1996.
[37] W. H. Schilders, H. A. Van der Vorst, and J. Rommes, *Model order reduction: theory, research aspects and applications*, vol. 13, Springer, 2008.
[38] P. Wesseling, *Principles of computational fluid dynamics*, vol. 29 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2001, https://doi.org/10.1007/978-3-642-05146-3, https://doi.org/10.1007/978-3-642-05146-3.