A Short Note on Solving Partial Differential Equations Using Convolutional Neural Networks

Viktor Grimm, Alexander Heinlein, and Axel Klawonn

1 Introduction

Solving partial differential equations (PDEs) is a common task in numerical mathematics and scientific computing. Typical discretization schemes, for example, finite element (FE), finite volume (FV), or finite difference (FD) methods, have the disadvantage that the computations have to be repeated once the boundary conditions (BCs) or the geometry change slightly; typical examples requiring the solution of many similar problems are time-dependent and inverse problems or uncertainty quantification. Every single computation, however, can be very time consuming, motivating the development of surrogate models that can be evaluated quickly. There exist some possible surrogate models, including linear reduced order models [9, 21, 26, 29] and neural network-based models [6, 7, 8, 14, 19, 22, 24, 25].

In this work, we will discuss an approach for predicting the solution of boundary value problems using convolutional neural networks (CNNs). This approach is particularly interesting in the context of surrogate models which predict the solution based on a parametrization of the model problem, for instance, with respect to variations in the geometry or BCs; cf. Fig. 1 for a sketch of the CNN-based surrogate modeling approach. If the parametrization is high-dimensional, that is, if it consists of a large number of parameters, neural network-based approaches are particularly well-suited since they are know to be able to overcome the curse of dimensional-

Viktor Grimm
Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931 Köln, Germany, e-mail: viktor.grimm@uni-koeln.de

Alexander Heinlein
Delft University of Technology, Delft Institute of Applied Mathematics, Mekelweg 4, 2628 CD Delft, Netherlands e-mail: a.heinlein@tudelft.nl

Axel Klawonn
Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931 Köln, Germany e-mail: axel.klawonn@uni-koeln.de
Center for Data and Simulation Science, University of Cologne, Germany

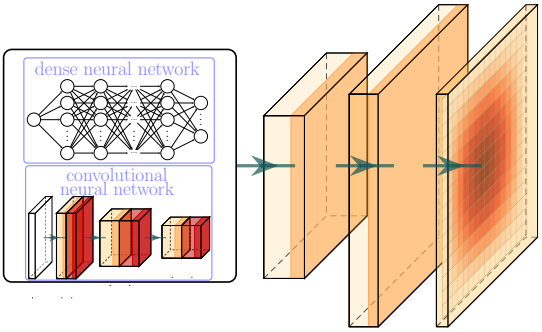


Fig. 1 Exemplary CNN-based surrogate model. The first block transforms the problem parametrization into a low-dimensional representation (latent representation) of the solution, and the right part of the model decodes the corresponding image of the solution field.

ity [4, 17]. In [5, 6, 15], a CNN model has been trained to predict stationary flow inside a channel with an obstacle of varying geometry; the model is trained in a purely data-based way using high-fidelity simulation data.

Here, we use a physics-based loss function in the CNN approach, that is, we optimize the network with respect to the residual of the partial differential equation (PDE) as well as the BCs of the BVP; this is also denoted as physics-informed or physics-aware machine learning (ML). Therefore, our approach is related to physics-informed neural networks (PINNs), which have been introduced in [28] and are an extension of the pioneering work [20]. However, different from [20, 28], we employ a finite difference-based discretization inside the loss function and predict the coefficients using a CNN. In the classical PINN approach, however a dense neural network (DNN) is employed as the discretization, and the derivatives are computed exactly via the backpropagation algorithm.

Physics-informed CNN approaches have already been considered. In particular, in [31], a model for predicting the solutions of the stationary diffusion equation for a single fixed geometry but varying BCs, encoded as an input image, is proposed. In [10], the authors employ a physics-based CNN model for predicting incompressible Navier–Stokes flow in parameterized geometries that is, the exact placement of the boundaries of the geometries depend on a parameter. More recently, the authors of this work have extended the previous approaches to a physics-aware CNN for predicting incompressible Navier–Stokes flow in more general geometries and also varying boundary conditions; cf. [13]. For further works on CNN-based surrogate models for the approximating the solutions of PDE, see, for instance, [7, 8, 11, 22, 25]. Furthermore, for scientific machine learning (SciML) overview papers with a broader scope and additional references on related approaches, we refer to [3, 35].

In this paper, we will compare the accuracy and convergence of a CNN model, optimized using a (stochastic) gradient descent-type method using a physics-based loss function, with a classical FD discretization, solving the resulting discrete linear

system of equations using an (unpreconditioned) conjugate gradient (CG) method, for a simple stationary diffusion problem. In order to focus on these aspects and remove any other complexities, we focus on a single problem configuration, that is, we neglect the encoder part in Fig. 1 and focus on training the decoder path. The paper is organized as follows: In Section 2, we introduce our stationary diffusion model problem and the simple difference discretization employed. Then, in Section 3, we briefly discuss how to solve the resulting discrete system of equations using the CG method as well as how to optimize a CNN model for predicting the same solution. Finally, we compare the performance of both solution frameworks with respect to accuracy and convergence in Section 5.

2 Model problem and discretization

Finite difference discretization

Let us consider a simple stationary diffusion problem on computational domain $\Omega := [0,1]^2$: find a function u, such that

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega, \end{aligned} \tag{1}$$

where f is some right hand side function. We discretize (1) using FDs. In particular, we consider a uniform grid $\Omega_h = \{(x_i, y_j)_{i,j}\}$ with $x_i := ih$ and $y_j := jh$, the step size $h = 1/n$, and $u_{i,j} := u(x_i, y_j)$. Using a central difference scheme, we obtain the following approximation of the Laplacian:

$$\Delta u(x_i) \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}. \tag{2}$$

Hence, the discrete form of (1) corresponds to the sparse system of linear equations

$$Au = f. \tag{3}$$

with a symmetric positive definite (SPD) matrix. Here, for simplicity, we use the same symbol for the solution and right hand side as in (1).

Reformulation of the finite difference problem via the cross-correlation

Before we explain our physics-based network model, let us discuss how (3) can be written equivalently using the cross-correlation operation

$$(I * K)_{ij} = \sum_m \sum_n I_{i-m,j-n} K_{m,n},$$

where I and K are two matrices. For simplicity, we omit the the range of the sums, and regard each matrix coefficient as zero which is outside the range of indices. Note that the discrete convolution and cross-correlation operations are related in the sense that one can be obtained from the other by transposition. Moreover, the cross corelation is actually implemented as the operation of convolutional layers in NN libraries; cf. [12, Section 9.1].

Now, let $U = \left(u_{i,j}\right)_{i,j}$ and $F = \left(f(x_i, y_j)\right)_{i,j}$ be $n \times n$ matrices resulting from re-arranging the solution and right hand side vectors in (3). Then, we obtain

$$Au = f \quad \Leftrightarrow \quad U * K = F, \tag{4}$$

where $*$ is the cross-correlation operation and K is given by

$$K = \frac{1}{h^2}\begin{pmatrix} K_{-1,-1} & K_{-1,0} & K_{-1,1} \\ K_{0,-1} & K_{0,0} & K_{0,1} \\ K_{1,-1} & K_{1,0} & K_{1,1} \end{pmatrix} = \frac{1}{h^2}\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \tag{5}$$

which is also denoted as the kernel matrix or filter. This can be easily seen by comparing the coefficients in (2), (5). Only for enforcing the boundary conditions for certain coefficients or pixels, respectively, the kernel K has to be modified, as is standard in the implementation of boundary conditions in finite difference discretizations.

3 Solving the finite difference problem using classical methods versus using convolutional neural networks

Efficient classical numerical solvers

Since our model problem, that is, stationary diffusion on the unit square, is arguably one of the most investigated problems for the development of solvers, there is a wide range of efficient solvers for (3). Hence, we keep this discussion rather short. A standard solver for systems with an SPD matrix is the conjugate gradient (CG) method. The convergence of the CG method is determined by the spectrum of the matrix, and in particular, it can be bounded in terms of the condition number of the system matrix A, which scales with $\frac{1}{h^2}$ for our model problem. The h dependence of the convergence of the CG method can be fixed by acceleration using preconditioners, such as domain decomposition [32] and multigrid [33] methods, to name just two popular classes of efficient and scalable preconditioners for (3).

For the purpose of comparing numerical solvers against a closely related ML approach for solving a stationary problem, we will use the CG method without preconditioning as the prototypical solver. It would be interesting to include state-of-the-art preconditioners in our study and discuss if and how preconditioning could be applied in the optimization of the CNNs. However, this is out of the scope of this short paper, and therefore, we will leave this to future research.

A finite difference solver based on convolutional neural networks

Solving (3) corresponds to finding the coefficients $u_{i,j}$, which are structured based on the uniform grid $\Omega_h = \{(x_i, y_j)_{i,j}\}$. We can simply interpret the discrete solution as a pixel image, with each pixel corresponding to one coefficient in the solution vector u. Hence, in several works, CNNs, which are very effective in image processing, have been trained to learn the discrete solution of a partial differential equation; cf. Fig.1 for a sketch of this approach and the discussion below. In practice, as we will also see in Section 5, this approach is not competitive for solving a single BVP. However, when used as a reduced order model for a parametrized model problem (e.g., with respect to the geometry), the higher computing costs for the training can be justified if the solutions of multiple BVPs can be predicted using a single model.

Here, we focus on training a neural network using a physics-informed, sometimes also referred to as physics-aware or physics-constraint, approach. Then, a neural network $\mathcal{NN}$ is trained to minimize the norm of the residual of the differential equation, i.e.,

$$\|\Delta\mathcal{NN} + f\|_\Omega^2 + \|\mathcal{NN}\|_{\partial\Omega}^2 \to \min,$$

where $\|\cdot\|_\Omega$ and $\|\cdot\|_{\partial\Omega}$ are some norms defined based on collocation points inside the domain Ω and on the boundary $\partial\Omega$; as mentioned in Section 1, this corresponds to the classical PINN approach if a dense NN is used as the discretization. If the output of the neural network corresponds to an image, that is, if the output data is a discrete vector on the uniform grid $\Omega_h = \{(x_i, y_j)_{i,j}\}$, we can employ an FD scheme to formulate the residual of the PDE, resulting in

$$\|b - A \cdot \mathcal{NN}\|_2^2 \to \min, \tag{6}$$

where the term corresponding to the boundary conditions vanishes since they are hard-coded within the matrix A. Note that this can be efficiently implemented in state-of-the-art ML libraries, such as Tensorflow: the matrix A does not have to be assembled, but it can be applied in a matrix-free fashion by using the FD stencil (2) as a fixed kernel in a convolutional layer and applying it to the output of the network; cf. the discussion in Section 2.

We note that solving (6) directly for u is equivalent to solving the least-squares problem corresponding to (3), which amounts to solving the normal equations

$$A^\top A u = A^\top b. \tag{7}$$

The system matrix $A^\top A$ is still SPD, so (7) can also be solved using the CG method. However, the convergence will be much slower, as the condition number

$$\kappa\left(A^\top A\right) = \kappa\left(A\right)^2.$$

The situation is changed further once u is replaced by a neural network $\mathcal{NN}$. Hence, minimizing the loss function with respect to the network parameters θ does not correspond to solving a linear system anymore. Moreover, the loss function is, in

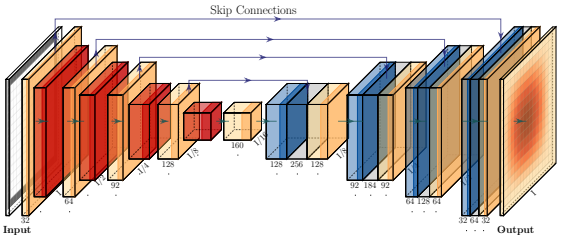


Fig. 2 Exemplary model architecture with a depth of four levels, resulting in 8×8 feature maps on the deepest level. Each level is composed of convolutions (orange), strided convolution (red), upsampling (blue) and/or concatenation (grey) layers. In total, this model has 834 627 parameters.

general, not even a convex function with respect to the network parameters anymore. Thus, in addition to solving a problem (6) that has a significantly worse conditioning than the original problem (3), we cannot use the CG method let alone another Krylov subspace method anymore.

Minimizing (6) with respect to the network parameters, which is also denoted as training the neural network, is usually performed using either a variant of stochastic gradient descent (SGD), such as the Adam (adaptive moments) optimizer [18], or a second order quasi-Newton method, such as L-BFGS [23]. Those optimizers and their parameters are typically chosen based on heuristics, which clearly shows that, at this point, we have lost most of the properties of the original problem (3) beneficial for a numerical solver.

Extension to more complex problems

Even though, in this paper, we focus on a linear problem on a simple square domain, our approach can be extended to nonlinear problems on more general geometries in a straight-forward way. In particular, the linear operator A in (6) can be easily replaced by a nonlinear operator F, which yields the minimization problem

$$\|b - F(\mathcal{NN})\|_2^2 \to \min. \tag{8}$$

In particular, in the CNN approach for a nonlinear PDE, the operator F corresponds to the finite difference discretization of the nonlinear differential operator of the PDE; cf. [13] for the application to the Navier–Stokes equations. Even though it cannot be directly implemented using a simple cross-correlation anymore, it can typically be written as a composition of cross-correlations and element-wise tensor-operations. Hence, it can still be easily and efficiently implemented using optimized functions from state-of-the-art deep learning libraries. To extend the approach to more complex geometries, boundary conditions have to be implemented for the corresponding output coefficients or pixels, respectively. A parametrization of the problem, for instance, with the respect to the geometry, can be incorporated via the input of the CNN; cf. Figs. 1 and 2. For more details, we refer to [13].

4 Network architecture and hyper parameters

As is usual in the context of NNs, the training performance and prediction accuracy of model strongly depend on the choice of the hyperparameters, which include the specific network architecture and parameters of the optimizer. In advance of our numerical study, we have carried out a detailed hyperparameter optimization to obtain a good performance of the CNN models. In particular, we used the optimized model for more complex computational fluid dynamics problems with varying geometries; cf. [13]. Similar to [5, 6, 15], in [13], the CNN model is employed as a reduced order surrogate model for varying geometries. As a result of the hyper parameter optimization, we ended up using an architecture which is inspired by the U-Net [30]; cf. Fig. 2. The model is composed of an encoder and a decoder part, each consisting of several levels. The corresponding levels of the encoder and decoder are connected with skip connections. Here, each level of the encoder part consists of a convolutional layer with an increasing number of 3×3 filters and a downsizing convolutional layer with 2×2 filters and stride of 2. In the decoder part, each level consists of a normal 3×3 convolutional layer, a concatenation layer for the skip connections and an up-sampling through nearest-neighbor interpolation layer.

In the hyper parameter optimization, we varied the activation function, the number of filters in the convolutional layers as well as the number of levels of the U-Net type architecture. Moreover, we performed numerical experiments for different learning rates, indicating the best performance for GD with a learning rate of 10^{-5} and for Adam with a learning rate of $5.0 \cdot 10^{-5}$. For more details on the hyper parameter optimization, we refer to [13].

In this paper, we focus on the effect of different solvers rather than the effect of different choices of the neural network architecture. In this sense, our major concern was to obtain a model architecture which is sufficient for approximating the solution of our the considered model problem. As we can observe based on the results in Section 5, this is the case for our model. In fact, the number of parameters and the model capacity could probably be reduced significantly for this model problem, at the cost of an additional hyper parameter optimization. Of course, a variation of the hyperparameters could have some impact on the convergence results in Section 5 but it is not obvious how to take the hyperparameter optimization into account in the comparison in a fair way. Moreover, we do not expect a major difference in the performance of the different approaches when varying the hyper parameters.

5 Numerical results

In this section, we compare different solution methods for an FD discretization of (1). In particular, we employ the gradient descent (GD) and conjugate gradient (CG) methods for the original equations (3) as well as the normal equations (7) arising from a least-squares formulation of the problem. We compare those results against training a CNN to predict the coefficient vector using the GD and Adam [18] meth-

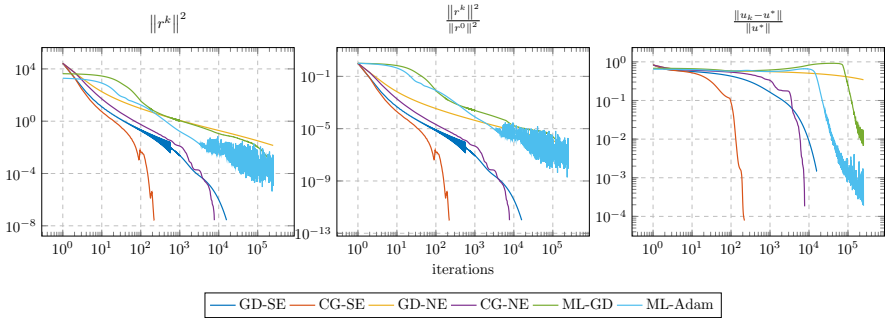


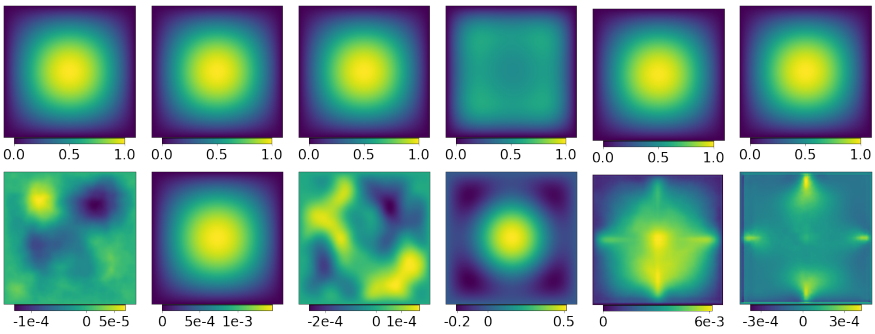
Fig. 3 Convergence of the GD, CG and Adam methods for the original linear equation system (3) and the least-squares problem eq. (6) for the FD discretization u and the CNN u_{NN}. Comparison of the absolute and relative residuum $\|r_k\|^2 \,/\, \|r_0\|^2$ where $r_k = b - Au_k$, and the relative error $\|u_k - u^*\| \,/\, \|u^*\|$.

ods for the physics-informed loss function, which corresponds to the least-squares formulation (6). All CNN computations were performed on NVIDIA V100-GPUs with CUDA 10.1 using python 3.6 and tensorflow-gpu 2.4 [1].

For our experiments, we choose $f = 2\pi^2 \sin(\pi x)\sin(\pi y)$ as the right hand side. The resulting BVP has the analytical solution $u^* = \sin(\pi x)\sin(\pi y)$, which we use as the reference. In this work, we exlusively consider an FD discrezation of the computational domain Ω with $N = 128$ grid nodes in each direction; this results in a total problem size of 16 384 nodes or degrees of freedom, respectively. For the classical methods, we use a fixed but random initial guess, the parameters of the CNNs are randomly initialized using the He normal initialization [16]. We compare the convergence of the methods via the squared relative residual $\|r_k\|^2 / \|r_0\|^2$, which corresponds to a relative mean squared error (MSE). For the classical numerical methods, we stop the iteration once a tolerance of 10^{-12} for the relative residual or an iteration count of 250 k iterations is reached. The CNNs are always trained for 250 k iterations or epochs.

We compare the relative residuals for the various methods applied to the standard and normal equations in Fig. 3. As expected, the CG method applied to the standard equation (CG-SE) converges the fastest after 221 iterations; note again that the convergence could be significantly improved using preconditioning techniques. The CG method applied to the normals equation (CG-NE) converges within 7 737 iterations, the GD method on the original equation (GD-SE) in 15 811 iterations. The GD method on the normal equations (GD-NE) does not converge within 250 k iterations and reaches a relative residual of $5.2 \cdot 10^{-7}$ at termination of the iteration.

As can be seen in Fig. 4d, the GD-NE solution, which has not converged within 250 k iterations, has a large relative L_2-error of 34 % compared with the analytical solution. For CG-SE, CG-NE, and GD-SE, we obtain errors of 0.008 %, 0.02 %, and 0.15 %, respectively, at convergence. In terms of convergence with respect to the relative squared residual norm, the ML approaches perform worse. Both ML-GD



(a) CG-SE (b) GD-SE (c) CG-NE (d) GD-NE (e) ML-GD (f) ML-Adam

Fig. 4 The solutions (top row) achieved with the various methods and the corresponding erorrs $(u^* - u)$ (bottom row) w.r.t the analytical solution at the grid nodes.

and ML-Adam do not achieve a relative tolerance of 10^{-12} and the training is stopped after 250 k iterations/epochs with a final relative residual of $1.5 \cdot 10^{-7}$ for ML-GD and $3.1 \cdot 10^{-8}$ for ML-Adam. Nonetheless, we achieve relative L_2-errors of 0.7 % for ML-GD and 0.02 % for ML-Adam. These are significantly lower than for GD-NE, even though the methods terminate at a similar relative residual. In fact, the accuracy is within one order of magnitude of the CG solutions and even better than the GD-SE solution; cf. also Fig, 4.

Spectral bias in the CNN training

Let us discuss why, in comparison, the error may be much lower for the CNN compared to the classical numerical solvers for a residual in the same order of magnitude. In particular, for the error e and the residual r, we have

$$Ae = A(u^* - u) = b - Au = r$$

Hence, of course, the relation of $\|e\|$ and $\|r\|$ depends on how the error decomposes into eigenfunctions of high/low eigenvalues. Since the CNNs were able to achieve comparatively low error while exhibiting higher absolute and relative residual, especially compared to the CG solutions, this suggests that the corresponding error is mainly composed of eigenfunctions corresponding to high eigenvalues. In particular, this implies that the CNNs exhibit some form of spectral bias, i.e., that they tend to learn eigenfunctions corresponding to low eigenvalues. Note that the spectral bias has been previously studied for DNNs [2, 27] and for PINNs [34]. However, to the best of the authors' knowledge, it has not been studied for the physics-informed CNN approach considered here. A more detailed study is out of the scope of this paper but will be discussed in future research.

6 Conclusion

In this work, we have compared physics-informed CNNs with classical methods for solvinge PDEs on the example of the stationary diffusion problem. We have shown that solution methods that take advantage of properties of the problem, such as the CG method, outperform the ML approach both in the accuracy achieved and in the speed of convergence. Yet, the ML solutions learned were within an order of magnitude of the CG solutions, i.e., they were not infeasible. But the much slower convergence coupled with the need for hyperparameter optimization as well as the heuristic nature of the choice of method parameters argue for the use of classical methods. Nonetheless, with an ML approach it is possible to include parameters, such as boundary conditions, geometry, etc., as input. In such cases, ML approaches are superior to classical methods and thus there is a sound reason again to use them. The extension of this study to more complex problems, the incorporation of preconditioning, as well as a more detailed discussed of the spectral bias will be the subject of future research.

Acknowledgements This work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centers. We gratefully acknowledge the use of the computational facilities of the Center for Data and Simulation Science (CDS) at the University of Cologne.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2016).
2. Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X., and Gu, Q. Towards Understanding the Spectral Bias of Deep Learning (2020).
3. Cuomo, S., di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *arXiv* (2022).
4. De Ryck, T. and Mishra, S. Error analysis for physics-informed neural networks (PINNs) approximating Kolmogorov PDEs. *Advances in Computational Mathematics* **48**(6), 79 (2022).
5. Eichinger, M., Heinlein, A., and Klawonn, A. Stationary flow predictions using convolutional neural networks. In: *Numerical Mathematics and Advanced Applications ENUMATH 2019*, 541–549. Springer (2021).
6. Eichinger, M., Heinlein, A., and Klawonn, A. Surrogate convolutional neural network models for steady computational fluid dynamics simulations. *Electronic Transactions on Numerical Analysis* **56**, 235–255 (2022).
7. Franco, N. R., Fresca, S., Manzoni, A., and Zunino, P. Approximation bounds for convolutional neural networks in operator learning (2023). ArXiv:2207.01546 [cs, math].
8. Fresca, S., Dede', L., and Manzoni, A. A Comprehensive Deep Learning-Based Approach to Reduced Order Modeling of Nonlinear Time-Dependent Parametrized PDEs. *Journal of Scientific Computing* **87**(2), 61 (2021).

9. F.R.S, K. P. 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**(11) (1901). Publisher: Taylor & Francis pages = 559–572,.
10. Gao, H., Sun, L., and Wang, J. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics* **428**, 110079 (2021).
11. Gonzalez, F. J. and Balajewicz, M. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems (2018). ArXiv:1808.01346 [physics].
12. Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press (2016).
13. Grimm, V., Heinlein, A., and Klawonn, A. Physics-aware convolutional neural networks for two-dimensional flow predictions. In preparation.
14. Guo, X., Li, W., and Iorio, F. Convolutional Neural Networks for Steady Flow Approximation. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 481–490. Association for Computing Machinery, New York, NY, USA (2016).
15. Guo, X., Li, W., and Iorio, F. Convolutional neural networks for steady flow approximation. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 481–490. Association for Computing Machinery, New York, NY, USA (2016).
16. He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification (2015). ArXiv:1502.01852 [cs].
17. Jentzen, A., Salimova, D., and Welti, T. A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *Communications in Mathematical Sciences* **19**(5), 1167–1205 (2021). Publisher: International Press of Boston.
18. Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
19. Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural Operator: Learning Maps Between Function Spaces (2022). ArXiv:2108.08481 [cs, math].
20. Lagaris, I., Likas, A., and Fotiadis, D. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* **9**(5), 987–1000 (1998). Conference Name: IEEE Transactions on Neural Networks.
21. Lassila, T., Manzoni, A., Quarteroni, A., and Rozza, G. Model order reduction in fluid dynamics: challenges and perspectives. *Reduced Order Methods for modeling and computational reduction* 235–273 (2014).
22. Lee, K. and Carlberg, K. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders (2019). ArXiv:1812.08373 [cs].
23. Liu, D. and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Mathematical Programming* **45**, 503–528 (1989).
24. Lu, L., Jin, P., and Karniadakis, G. E. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *Nature Machine Intelligence* **3**(3), 218–229 (2021). ArXiv:1910.03193 [cs, stat].
25. Maulik, R., Lusch, B., and Balaprakash, P. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids* **33**(3), 037106 (2021). Publisher: American Institute of Physics.
26. Quarteroni, A., Manzoni, A., and Negri, F. *Reduced basis methods for partial differential equations*, *Unitext*, vol. 92. Springer, Cham (2016).
27. Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., Bengio, Y., and Courville, A. On the Spectral Bias of Neural Networks (2019). ArXiv:1806.08734 [cs, stat].
28. Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* **378**, 686–707 (2019).
29. Rathinam, M. and Petzold, L. R. A New Look at Proper Orthogonal Decomposition. *SIAM Journal on Numerical Analysis* **41**(5), 1893–1925 (2003).

30. Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F. (eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Lecture Notes in Computer Science, 234–241. Springer International Publishing, Cham (2015).
31. Sharma, R., Farimani, A. B., Gomes, J., Eastman, P., and Pande, V. Weakly-supervised learning of heat transport via physics informed loss. *arXiv* (2018). URL arXiv:1807.11374.
32. Toselli, A. and Widlund, O. *Domain decomposition methods-algorithms and theory*, vol. 34. Springer Science & Business Media (2004).
33. Trottenberg, U., Oosterlee, C. W., and Schuller, A. *Multigrid*. Elsevier (2000).
34. Wang, S., Yu, X., and Perdikaris, P. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics* **449**, 110768 (2022).
35. Willard, J., Jia, X., Xu, S., Steinbach, M., and Kumar, V. Integrating scientific knowledge with machine learning for engineering and environmental systems. *arXiv* (2021).