Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Seminar for Applied Mathematics

Convolutional Neural Operators

B. Raonic and R. Molinaro and T. Rohner and S. Mishra and E. de Bézenac

Research Report No. 2023-11
February 2023

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Convolutional Neural Operators

Bogdan Raonić
Seminar for Applied Mathematics,
ETH, Zurich, Switzerland

Roberto Molinaro
Seminar for Applied Mathematics,
ETH, Zurich, Switzerland

Tobias Rohner
Seminar for Applied Mathematics,
ETH, Zurich, Switzerland

Siddhartha Mishra
Seminar for Applied Mathematics,
ETH AI Center,
ETH, Zurich, Switzerland

Emmanuel de Bézenac
Seminar for Applied Mathematics,
ETH, Zurich, Switzerland

Abstract

Although very successfully used in machine learning, convolution based neural network architectures – believed to be inconsistent in function space – have been largely ignored in the context of learning solution operators of PDEs. Here, we adapt convolutional neural networks to demonstrate that they are indeed able to process functions as inputs and outputs. The resulting architecture, termed as convolutional neural operators (CNOs), is shown to significantly outperform competing models on benchmark experiments, paving the way for the design of an alternative robust and accurate framework for learning operators.

1 Introduction

Partial Differential Equations (PDEs) are mathematical models for an enormous variety of phenomena of interest in the sciences and engineering Evans (2010). *Solving* a PDE amounts to computing the underlying solution *operator* which maps given *input* functions such as initial and boundary conditions, source terms, coefficients etc to the solution. Currently, numerical methods such as finite difference, finite element and spectral methods are used to compute this PDE solution operator Quarteroni & Valli (1994). However, these methods can be prohibitively expensive, particularly in several dimensions. Moreover these methods are *data agnostic*, Mishra (2018) and references therein, and not designed to learn from or adapt to the available large datasets, either generated through simulations or from observations. Consequently, there has been considerable amount of interest in recent years to use *data driven machine learning* methods for the fast, robust and accurate solution of PDEs.

Given that operators are the underlying objects of interest in the context of PDEs, *operator learning* ML architectures which map functions to functions are being increasingly viewed as the suitable paradigm for applying ML techniques to PDEs Kovachki et al. (2021). A widely used framework in this regard is that of DeepONets and its variants Lu et al. (2021); Mao et al. (2020); Cai et al. (2021); Lanthaler et al. (2022) whereas an alternate paradigm is that of neural operators Kovachki et al. (2021); Li et al. (2020a;b), which includes the popular Fourier Neural Operator (FNO) Li et al. (2021) architecture. Although these architectures have been successfully applied in various examples, many pressing issues, such as the limited expressivity of DeepONets Lanthaler et al. (2023) and *aliasing* errors for FNOs Fanaskov & Oseledets (2022), still hinder the widespread adoption of operator learning frameworks in the simulation of PDEs.

In this context, it is worth noting that convolutional neural networks (CNNs) are often the state of the art models for a variety of tasks in image processing such as classification and generation LeCun et al. (2015). However, CNNs entail finite dimensional inputs and outputs and are not directly applicable for operator learning. Naive use of CNNs in solving PDEs often leads to results that depend heavily on the underlying grid resolution Zhu & Zabaras (2018) and references therein. Hence, CNNs have been largely ignored as ML models in this important area. Despite this background, CNNs are appealing in many respects, given their locality, computational efficiency and widespread use in other ML contexts and bringing them back into the reckoning for learning PDE operators could be advantageous. This is precisely the goal of the current paper where we will show that by making simple modifications, for instance reinterpreting those suggested in Karras et al. (2022) for image generation, CNNs can be adapted to learn operators. The resulting architecture, termed as *Convolutional Neural Operators* (CNOs) maps input functions to output functions. Moreover, we demonstrate through numerical experiments that CNOs significantly outperform existing operator learning architectures on benchmark problems, highlighting the utility of convolution based architectures and paving the way for an alternative efficient operator learning framework.

2 Convolutional Neural Operators

Setting. For simplicity of notation and definiteness, we focus on two spatial dimensions by letting the domain $D = \mathbb{T}^2$ be a 2-d torus. Let $\mathcal{G}^\dagger : H^r(D) \mapsto H^s(D)$ be the solution operator of some underlying PDE, with $H^{r,s}$ being Sobolev spaces. Without loss of generality, we set $s = r$ hereafter. The goal is to learn $\mathcal{G}^\dagger$ from finite data of measurements of input and output function pairs $\{u_i, \bar{u}_i\}_{i=1}^N$. As in practice, the underlying data is either generated by numerical simulations or observations, we assume that we can only access the realizations of any function $f \in H^r(D)$ in the form of point-wise evaluations $\{f(x_j)\}_{j=1}^{s\times s}$ on a $s \times s$ uniform grid on D.

Bandlimited Approximations. Next, we approximate the solution operator $\mathcal{G}^\dagger$ with an operator $\mathcal{G} : \mathcal{B}_w(D) \mapsto \mathcal{B}_w(D)$, where $B_w(D)$ is a space of bandlimited functions Vetterli et al. (2014) i.e., functions whose non-zero Fourier coefficients can be atmost of modulus $w \in \mathbb{R}_+$. The motivation behind the use of bandlimited functions is twofold: (1) the fourier coefficients of Sobolev functions $f \in H^r(D)$ decay (rapidly). Therefore, they can be well approximated by bandlimited functions $\tilde{f} \in \mathcal{B}_w(D)$ with a large enough band w: for any $\epsilon > 0$, there exists $w > 0$ such that $||f - \tilde{f}||_{L^2(D)} < \epsilon$. (2) for a sufficiently resolved grid i.e. $s > 2w$, there exists a direct equivalence between the function and its grid (point) values, given by the Shannon-Whittaker-Kotel'nikov sampling theorem Vetterli et al. (2014). Hence, any (*discrete*) operations on gridvalues are guaranteed to yield a *unique* continuous analogue in the space of bandlimited functions. This exact correspondence between the continuous and discrete representations of the functions, defined in **SM** A.2, is a *necessary* condition for working with continuous objects such as functions. If it is not satisfied, multiple functions could have the same discrete representation, leading to the well-known *aliasing* phenomenon Vetterli et al. (2014), resulting in subsequent errors. Throughout the following, we implicitly assume that all the functions have a bandlimit at most $s/2$, so we may suppose that $w = s/2$ in the rest of the paper.

CNO Block. We will now introduce CNO, our convolution-based neural operator, which we define as a compositional mapping between functions as,

$$\mathcal{N}^{CNO} : u = v_1 \mapsto v_2 \mapsto \dots v_L = \bar{u}, \quad v_{l+1} = \mathcal{P}_l \circ \Sigma_l \circ \mathcal{K}_l(v_l), \quad 1 \leq \ell \leq L. \tag{1}$$

Here, the input function u is processed through the composition of a series of mappings between functions (layers), with each layer consisting of three elementary mappings, i.e., $\mathcal{P}_l$ is either upsampling or downsamping operator, $\mathcal{K}_l$ is the convolution operator and Σ_l is the activation operator. These elementary operators are defined below. See also Figure 1 for a schematic representation of CNO.

Our goal in defining the elementary operations below is to maintain an equivalence between the continuous operations and the discrete computations such that the Shannon-Whittaker-Kotel'nikov theorem applies at every step. This requires that *each* layer of the operator is a mapping between bandlimited functions (the bands need not be the same), and the size of the sampling grid is chosen accordingly. This constitutes the main difference from classic realizations of CNNs as their operations do not respect this requirement, leading to aliasing errors Karras et al. (2022).

Convolution. The convolutional operator in our case has a discrete kernel $K_w = \sum_{i,j=1}^k k_{ij} \cdot \delta_{z_{ij}}$, defined on the $s \times s$ grid with z_{ij} being the grid points, $k \in \mathbb{N}$ being the discrete kernel size and δ

the Dirac measure. The convolution operator $\mathcal{K}_w : \mathcal{B}_w(D) \mapsto \mathcal{B}_w(D)$ is defined by

$$\mathcal{K}_w f(x) = (K_w \star f)(x) = \int_D K_w(x-y) f(y) dy = \sum_{i,j=1}^{k} k_{ij} f(x - z_{ij}), \quad \forall x \in D,$$

where the last identity arises from the fact that f is a bandlimited function. Thus, our convolution operator is directly parametrized in physical space, providing locality in this operator. This is in contrast to FNO Li et al. (2021), where the convolution operator is defined in Fourier space.

Up and Downsampling. Let $h_w(x) = \text{sinc}(2wx_0) \cdot \text{sinc}(2wx_1)$ for $x = (x_0, x_1) \in \mathbb{R}^2$ be an *ideal low-pass interpolation filter*. We define the *downsampling* operator from the bandlimit w to the bandlimit $\underline{w} < w$ as $\mathcal{D}_{w,\underline{w}} : \mathcal{B}_w(D) \mapsto \mathcal{B}_{\underline{w}}(D)$, defined by

$$\mathcal{D}_{w,\underline{w}} f(x) = \left(\frac{\underline{w}}{w}\right)^2 (h_{\underline{w}} \star f)(x) = \left(\frac{\underline{w}}{w}\right)^2 \int_D h_{\underline{w}}(x-y) f(y) dy, \quad \forall x \in D,$$

where $\star$ is the convolution operation on functions. The *upsampling* operator from the bandlimit w to $\overline{w} > w$ is simply defined by $\mathcal{U}_{w,\overline{w}} : \mathcal{B}_w(D) \mapsto \mathcal{B}_{\overline{w}}(D)$ as

$$\mathcal{U}_{w,\overline{w}} f(x) = f(x), \quad \forall x \in D.$$

Activation. Assume that we want to apply a (pointwise) activation function σ to a function $f \in \mathcal{B}_w(D)$. Following Karras et al. (2022), we modify this operation by firstly upsampling the signal to a bandlimit $\overline{w}$, then apply the activation and finally downsample the signal back to the bandlimit w. As the functions of our interest (solutions of PDEs) have a fast decaying spectrum, it is reasonable to assume that newly introduced frequencies above $\overline{w} >> w$ are negligible. Consequently, the activation function can be approximated by an operator between the bandlimited spaces, namely $\sigma : \mathcal{B}_{\overline{w}}(D) \mapsto \mathcal{B}_{\overline{w}}(D)$. Therefore, the modified activation function $\Sigma_{w,\tilde{w}} : \mathcal{B}_w(D) \mapsto \mathcal{B}_w(D)$ is defined by

$$\Sigma_{w,\overline{w}} f(x) = \mathcal{D}_{\overline{w},w}(\sigma \circ \mathcal{U}_{w,\tilde{w}} f)(x), \quad \forall x \in D.$$

Instantiation. We propose to use a deep convolutional encoder-decoder architecture for Convolutional Neural Operators equation 1. The encoder network gradually downsamples the input in the spatial domain, while increasing the number of channels at the same time. The decoder network does the opposite. As the network goes deeper in the encoder, more global features are extracted. The extracted information from the multiple scales is gradually gathered in the decoder to produce the relevant output. In the convolutional encoder-decoder neural operator, the first M iterations are devoted to the **encoder**, namely

$$v_{l+1} = \mathcal{D}_{s_l, s_{l+1}} \circ \Sigma_{s_l, s_{l+1}} \circ \mathcal{K}_{s_l} v_l, \quad v_l \in \mathcal{B}_{s_l}(D),$$

where $s_l = s/2^l$ is the current bandlimit. The next $M-1$ iterations are devoted to the **decoder**. Let $\tilde{s}_l = s_{2M-l-1}$. The decoder is defined as

$$v_{l+1} = \mathcal{U}_{\tilde{s}_l, \tilde{s}_{l+1}} \circ \Sigma_{\tilde{s}_l, \tilde{s}_{l+1}} \circ \mathcal{K}_{\tilde{s}_l} v_l, \quad v_l \in \mathcal{B}_{\tilde{s}_l}(D).$$

Optional residual blocks could be added between the encoder and decoder, while optional resolution invariant blocks could be added after each up/downsampling block. A schematic representation of the encoder-decoder blocks is given in **SM** A.3. All the continuous operators introduced above can be equivalently defined in a discrete setting, see **SM** A.2 for details.

3 Experiments

We empirically test CNO on two different benchmark problems. As baselines in the following experiments, we choose three models. First, we consider an end-to-end fully convolutional neural-network (CNN) architecture, but *ablate* the interpolation filtering operation and perform the downsampling operation with a standard *average (mean) pooling*. Next, we benchmark the proposed architecture with Fourier Neural Operator FNO and DeepONet, with the CNN encoder as a branch-net.

For both experiments, we will consider the widely used fluid dynamics model, the incompressible Navier-Stokes equations (see **SM B.1**) on the two-dimensional domain $D = [0,1]^2$ with periodic

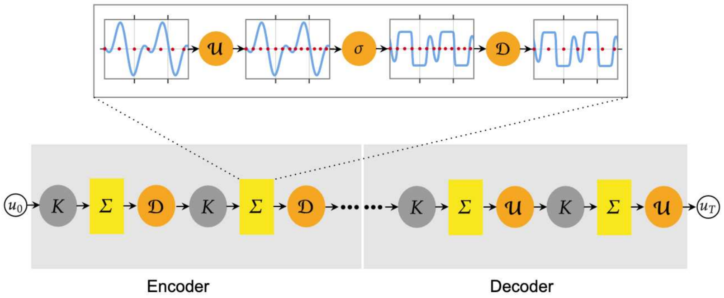

Figure 1: Schematic representation of the Convolutional Neural Operator architecture, see equation 1 for notation.

boundary conditions. In the first experiment, which we abbreviate as **NS1**, the Navier-Stokes equation is considered with a fluid viscosity of $\nu = 10^{-3}$. Following, Li et al. (2021); Prasthofer et al. (2022), the initial conditions are drawn from the measure $\mathcal{N}(0, \mathbf{C})$, where the covariance matrix is $\mathbf{C} = 7^{3/2}(-\Delta + 49\mathbf{I})^{-5/2}$ and task is to learn the operator that maps the initial vorticity to the vorticity of the resulting solution at time $T = 5$. To this end, the underlying data is generated with a spectral method as described in Prasthofer et al. (2022) and all the models are trained with 500 samples on a 33^2-grid. The test error is also computed on 500 samples. We compare the performance of CNO to CNN, FNO and DeepONet baselines and present the test errors in Table 3. We observe from this table that not only is CNO the best performing architectures among all the models compared here, it outperforms CNN by a factor of almost 4. This demonstrates that simple modifications to CNNs such as adding interpolation filters at every layer can significantly improve model performance.

In the second experiment, abbreviated as **NS2**, the Navier-Stokes equations are again considered, but with a viscosity of $\nu = 4 \cdot 10^{-4}$, applied to only Fourier modes with modulus greater than 12 and the initial data corresponds to a thin shear layer (or vortex sheet), see **SM B.2** for illustrations. The aim is to consider a problem with significantly larger range of scales and more complicated temporal dynamics than **NS1**. To generate the training and test data, we simulate the Navier-Stokes equations with a spectral viscosity method on a 128^2-grid and downsample the data to a 64^2-grid to learn the operator mapping the initial velocity to velocity at $T = 1$. We train all models on 890 training samples and present the test errors (on 128 samples) in Table 3. We observe from the table that CNO is the best performing model, not only outperforming CNN but also FNO, which is considered state of the art among operator learning models, by reducing the test error by a factor of 1.6.

	DONet	**CNN**	**FNO**	**CNO**
NS1	2.23%	3.50%	1.15%	$\mathbf{0.96}\%$
NS2	11.28%	4.69%	5.14%	$\mathbf{3.27}\%$

Table 1: Relative median L^1-error computed over testing samples for different benchmarks and models with the best performing model highlighted in bold.

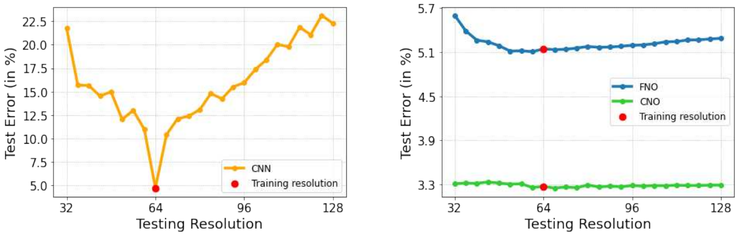

Figure 2: Shear Layer experiment. The errors are computed on different testing resolution.

Varying Resolutions. It is widely believed that errors with Neural operators should (approximately) be independent when they are tested on different grid resolutions representing the same continuous function Kovachki et al. (2021). We test this issue for CNO by considering the same setup as the **NS2** experiment, with training data downsampled on the 64^2 grid. These trained models are then tested by downsampling the original 128^2 data on a wide range of different resolutions. Results are illustrated in Figure 2. On the left plot, just as expected (as the scale of the convolutional filters do not vary with the resolution) and reported previously Zhu & Zabaras (2018); Li et al. (2021), the CNN model performs very poorly. We believe that for this reason, classical convolutional architectures have largely been overlooked in operator learning. On the right of the same plot, but on a different scale, we compare CNO and FNO at different test resolutions and observe that CNO –despite being a convolutional based architecture– not only is very stable to changes in the resolution, but seems to be even more stable than FNO, for both higher and lower resolution data. We hypothesize that this observed lack of stability of FNO could be due to aliasing, which implicitly ties the model to the resolution of the training data.

REFERENCES

Shengze Cai, Zhicheng Wang, Lu Lu, Tamer A Zaki, and George Em Karniadakis. DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, 436:110296, 2021.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

V. Fanaskov and I. Oseledets. Spectral neural operators. *arXiv preprint arXiv:2205.10573v1*, 2022.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

T. Karras, M. Aittala, S. Laine, E. Härkönen, J Hellsten, J. Lehtinen, and T. Aila. Alias-free generative adversarial networks. *arXiv preprint arXiv:2106.12423v4*, 2022.

N. Kovachki, Z. Li, B. Liu, K. Azizzadensheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481v3*, 2021.

S. Lanthaler, R. Molinaro, P. Hadorn, and S. Mishra. Nonlinear reconstruction for operator learning of pdes with discontinuities. In *International Conference on Learning Representations*, 2023.

Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. Error estimates for DeepONets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 2022.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Zongyi Li, Nikola B Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *CoRR*, abs/2003.03485, 2020a.

Zongyi Li, Nikola B Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew M Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 6755–6766. Curran Associates, Inc., 2020b.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Z. Mao, L. Lu, O. Marxen, T. Zaki, and G. E. Karniadakis. DeepMandMnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. Preprint, available from arXiv:2011.03349v1, 2020.

S. Mishra. A machine learning framework for data driven acceleration of computations of differential equations,. *Math. in Engg.*, 1(1):118–146, 2018.

M. Prasthofer, T. De Ryck, and S. Mishra. Variable input deep operator networks. *arXiv preprint arXiv:2205.11404*, 2022.

A. Quarteroni and A. Valli. *Numerical approximation of Partial differential equations*, volume 23. Springer, 1994.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

M. Vetterli, J. Kovacevic, and V.K. Goyal. *Foundations of Signal Processing*. Cambridge University Press, 2014.

Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 336:415–447, 2018.

Supplementary Material for: Convolutional Neural Operators.

A Details of the models

In the main text, we defined continuous operators that we apply to the bandlimited functions. Below, we describe these operators, including implementation of the filter h_w, data sampling, etc., in the discrete settings.

A.1 Filter design

Since perfect filters h_w have infinite impulse response and cause ringing artifacts around high-gradient points (e.g. discontinuities) due the Gibbs phenomenon, one usually uses *windowed-sinc* filters. The *windowed-sinc* filters are constructed by multiplying the ideal filter h_w by a corresponding window function. That is equivalent to convolving the filter with the window function in the frequency domain. In practice, we use standard Python libraries and their functions such as *scipy.signal.firwin* to design the filters. They enable us to manually control the cutoff frequency w_c and the half-width of the transition band w_h of the designed filters. We design discrete filters with a prescribed compact support. We usually choose our filters to have the kernel size of at least 16 (or to have at least 16 ”taps”).

In all the experiments, we use $w_c = s/(2+\epsilon)$, where $\epsilon \ll 1$. As we mentioned, we control the half-width of the filter $w_h = c_h \cdot s$. (Almost) Perfect *sinc* filter can be designed by setting $c_h = 0.5$. However, it is usually beneficial to allow *some amount* of aliasing and, hence, we set $c_h = 1$. In rest of the text, we will denote the designed filters by $H_{s,M}$, where s is the sampling rate of the signals, while M is the number of taps. One can implement a 2D filter by first convolving a 1D filter with each row and then with each column. Different filter designs (in 1D) are shown in the Figure 3.

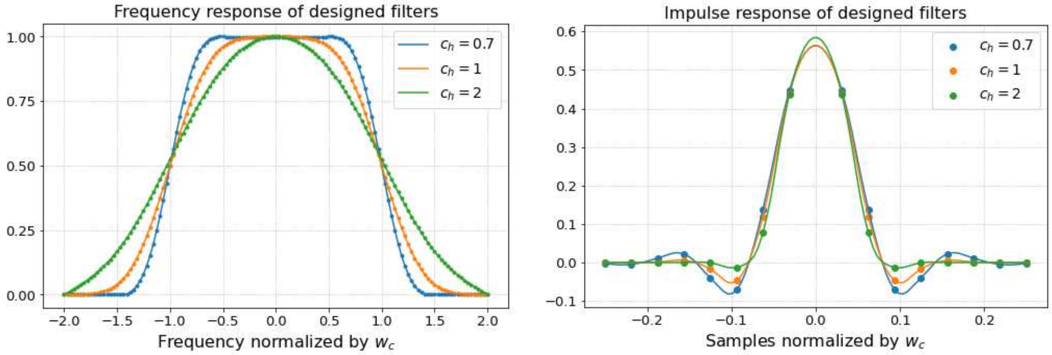

Figure 3: On the left: Frequency responses of different designed filters. On the right: Impulse responses of different designed filters. The sampling rate is $s = 128$, the cutoff frequency is $w_c = s/2.001$, while the halfwidth each filter is $w_h = c_h \cdot s$. Each filter has $M = 16$ taps.

A.2 Discrete data structures and operations

Discrete Data Structures. Given a function $f \in \mathcal{B}_w(D)$, one can always pass to its discrete representation $f_s \in \mathbb{R}^{s\times s}$ by sampling it with a sampling rate $s = 2w$. Formally, the discrete representation of the signal can be written as

$$f_s[i,j] = \text{III}_s\big(i/s, j/s\big) \cdot f\big(i/s, j/s\big) \quad i,j = 1\dots s,$$

where $\text{III}_s = \sum_{n\in\mathbb{Z}^2} \delta_{n/s}$ is the Dirac comb. Given a vector $f_s \in \mathbb{R}^{s\times s}$, one recovers the continuous representation $f \in \mathcal{B}_{s/2}(D)$ by convolving it with the interpolation filter $h_{s/2}$:

$$f(x_0,x_1) = \sum_{n=-\infty}^{+\infty} \sum_{i,j=1}^{s} f_s[i,j] \cdot h_{s/2}\left(x_0 - \frac{ns+i}{s}, x_1 - \frac{ns+j}{s}\right), \quad \forall (x_0,x_1) \in D.$$

Discrete Convolution. Let $f \in \mathcal{B}_{s/2}(D)$ and $f_s \in \mathbb{R}^{s\times s}$ be its sampled version. Let $H_{s,M} \in \mathbb{R}^{M\times M}$ be a discrete interpolation filter with M taps, designed as above. For $i,j = 1\dots s$ and $n \in \mathbb{N}$, we define $f_s[i+ns, j+ms] = f_s[i,j]$. Discrete convolution between the filter $H_{s,M}$ and f_s is defined as

$$(f_s \star H_{s,M})[n,m] = \sum_{k_1,k_2=1}^{M} H_{s,M}[k_1,k_2] \cdot f_s[n-k_1, m-k_2] \quad n,m \in \mathbb{N}.$$

The convolution of f_s with a discrete kernel $K \in \mathbb{R}^{k\times k}$ is defined in the same way.

Discrete Upsampling. Let $\mathcal{F}(f_s) \in \mathbb{C}^{s\times s}$ be the DFT of the signal f_s. The signal upsampling in the *frequency domain* (by a factor 2) is defined by the function $\mathcal{U}_{\mathcal{F},s} : \mathbb{C}^{s\times s} \mapsto \mathbb{C}^{2s\times 2s}$, with

$$\mathcal{U}_{\mathcal{F},s}(g)[i,j] = \begin{cases} \mathcal{F}(g)[i,j], & |i| \le s/2, |j| \le s/2 \\ 0, & \text{otherwise} \end{cases}$$

for all $i,j = -s,\dots,0,\dots s-1$ Then, the discrete upsampling of the signal f_s is

$$\mathcal{U}_s : \mathbb{R}^{s\times s} \mapsto \mathbb{R}^{2s\times 2s}, \quad \mathcal{U}_s(f_s) = \mathcal{F}^{-1}\big(\mathcal{U}_{\mathcal{F},s}(f_s)\big)$$

Alternatively, one can upsample the signal by adding the zeros between every 2 grid points to get the signal $f_{s,\uparrow 2s}$ and then convolve the new signal with the discrete windowed filter $H_{s,M} \in \mathbb{R}^{M\times M}$. This is the approach used in the implementation of CNO.

Discrete Downsampling. The discrete downsampling of the signal f_s (by a factor 2) is done by convolving it discretely with $H_{s/2,M} \in \mathbb{R}^{M\times M}$ and then keeping *every other point* of the resulting output. We implicitly assume that $s/2 \in \mathbb{N}$. More rigorously, the discrete downsampling is defined as the function $\mathcal{D}_s : \mathbb{R}^{s\times s} \mapsto \mathbb{R}^{s/2\times s/2}$ such that

$$\mathcal{D}_s(f_s) = (H_{s/2,M} \star f_s)_{\downarrow s/2}$$

Discrete Modified Activation Function. Finally, given the definitions above, the discrete activation function is defined as

$$\Sigma_s : \mathbb{R}^{s\times s} \mapsto \mathbb{R}^{s\times s}, \quad \Sigma_s(f_s) = \mathcal{D}_s \circ \sigma \circ \mathcal{U}_s(f_s),$$

where $\sigma : \mathbb{R} \mapsto \mathbb{R}$ is a point-wise activation functions (such as leaky ReLu).

A.3 Architecture Details

Below, details concerning the model architectures are discussed.

Fourier Features. Features features have been first introduce in Tancik et al. (2020) to improve the learning of high frequencies. Let $v \in [0,1]^2$ be input coordinates and $m \in \mathbb{N}$. Featurized version of v with $2m$ Fourier features is given by,

$$\Big(\cos(2\pi\, b_1^T \cdot v),\ \sin(2\pi\, b_1^T \cdot v),\ \dots,\ \cos(2\pi\, b_m^T \cdot v),\ \sin(2\pi\, b_m^T \cdot v)\Big),$$

where $b_i \in \mathbb{R}^2$ are i.i.d. drawn from the standard normal distributions.

Convolutional Neural Operator. The CNO architecture is designed based on 4 different "blocks", i.e. the downsampling block (D), the upsampling block (U), the invariant block (I) and the ResNet block (R).

The downsampling block (D) consists of the following operations:

- *convolution*, which leaves the input size unchanged and, and doubles the number of channels (usually, in the very first (D) block, the number of channels is increased to 64);
- *modified activation function* Σ;
- downsampling by a factor 2.

The upsampling (U) block is similar to the first one and consists of:

- *convolution*, defined as before;
- upsampling by a factor 2;
- *modified activation function* Σ.

The invariant block (I) includes:

- *convolution* operation. Differently from the (D) and (U) blocks, neither the input size nor the number of channels are changed;
- *modified activation function* Σ;

The invariant block usually follows a dowsampling or precedes an upsampling bock and allows for better exploration of the signal features at a certain sampling rate. The ResNet block (R) is similar to the (I) block, but with the additional skip connections added between the consecutive invariant blocks. An example of CNO architecture can be seen in the Figure 4.

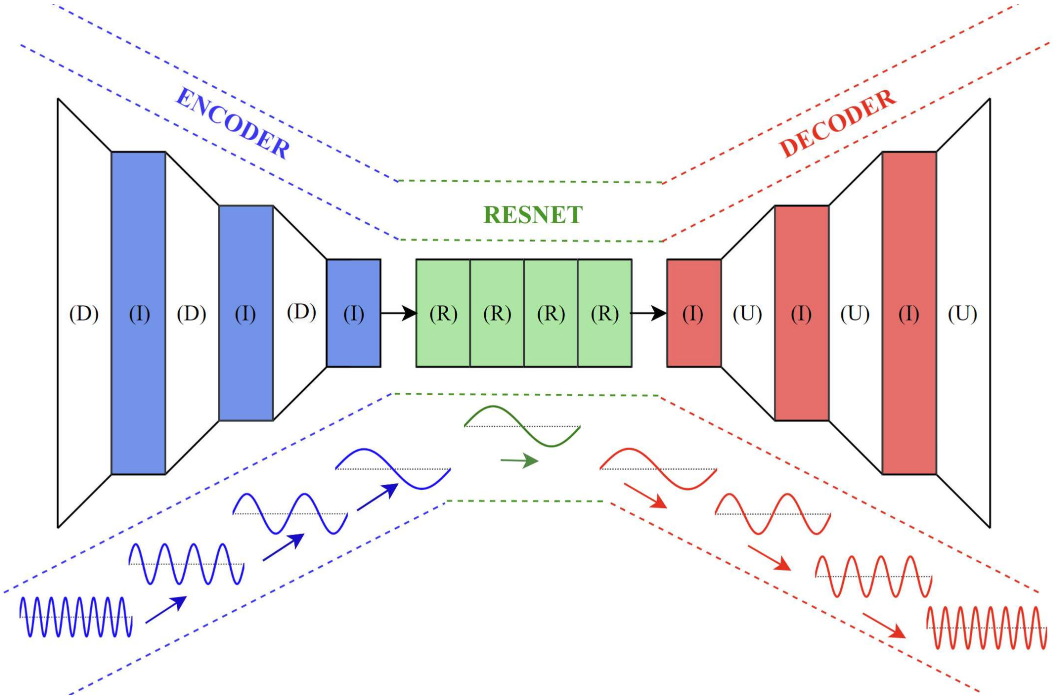

Figure 4: This is an example of a CNO architecture with 6 (I) blocks, 3 (D) blocks, 3 (U) blocks and 4 (R) blocks. Larger the height, larger is the resolution. The number of channels first gradually grows in the encoder, stays constant in the ResNet, and drops in the decoder. The information from the multiple scales is sequentially gathered. In this specific case, assuming that the initial sampling rate is 64, the sampling rate changes in accordance with the sequence $64 \to 32 \to 16 \to 8$ in the (D) encoder and $8 \to 16 \to 32 \to 64$ in the decoder. The number of channels changes as per $17 \to 64 \to 128 \to 256$ in the encoder and $256 \to 128 \to 64 \to 1$ in the decoder. Note that the sequence of channels starts with 17, as we include 16 Fourier features in the input.

Convolutional Neural Networks. The architecture of Convolutional Neural Network is the same as the one of CNO, but the downsampling operation is performed through *average pooling*, while the upsampling operation is done by a standard interpolation.

Feed Forward Dense Neural Networks. Given an input $y \in \mathbb{R}^m$, a feed-forward neural network (also termed as a multi-layer perceptron) transforms it to an output, through a layer of units (neurons) which compose of either affine-linear maps between units (in successive layers) or scalar non-linear activation functions within units Goodfellow et al. (2016), resulting in the representation,

$$u_\theta(y) = C_{L_t} \circ \sigma \circ C_{L_t-1} \ldots \circ \sigma \circ C_2 \circ \sigma \circ C_1(y). \tag{2}$$

Here, $\circ$ refers to the composition of functions, and σ is a scalar (non-linear) activation function. For any $1 \le \ell \le L_t$, we define

$$C_\ell z_\ell = W_\ell z_\ell + b_\ell, \text{ for } W_\ell \in \mathbb{R}^{d_{\ell+1}\times d_\ell}, z_\ell \in \mathbb{R}^{d_\ell}, b_\ell \in \mathbb{R}^{d_{\ell+1}}., \tag{3}$$

and denote,

$$\theta = \{W_\ell, b_\ell\}_{\ell=1}^{L_t}, \tag{4}$$

to be the concatenated set of (tunable) weights for the network. Thus, in the machine learning terminology, a feed-forward neural network equation 2 consists of an input layer, an output layer, and L_t hidden layers with d_ℓ neurons, $1 < \ell < L_t$. In all numerical experiments, the trunk net of DeepONet is a feed-forward neural network. Moreover, we consider a uniform number of neurons across all the layers of the network $d_\ell = d_{\ell-1} = d$, $1 < \ell < L_t$.

Fourier Neural Operator. Fourier neural operator (FNO) $\mathcal{N}^{FNO} : H^r(D) \mapsto H^s(D)$ is a composition

$$\mathcal{N}^{FNO} = Q \circ \mathcal{L}_T \circ \cdots \circ \mathcal{L}_1 \circ R. \tag{5}$$

with $\bar{u}(x) \mapsto R(\bar{u}(x), x)$ being a "lifting operator", represented by a linear transformation $R : \mathbb{R}^{d_u} \times \mathbb{R}^d \to \mathbb{R}^{d_v}$ where d_u is the number of components of the input function, d is the dimension of the domain and d_v is the "lifting dimension" (a hyperparameter). The operator Q is a non-linear projection, instantiated by a shallow neural network with a single hidden layer with 128 neurons and $GeLU$ activation function. Each *hidden layer* $\mathcal{L}_\ell : v^\ell(x) \mapsto v^{\ell+1}(x)$ is of the form

$$v^{\ell+1}(x) = \sigma\left(W_\ell \cdot v^\ell(x) + \left(K_\ell v^\ell\right)(x)\right),$$

with $W_\ell \in \mathbb{R}^{d_v \times d_v}$ a weight matrix (residual connection), σ an activation function, and the *non-local Fourier layer*,

$$K_\ell v^\ell = \mathcal{F}_N^{-1}\left(P_\ell(k) \cdot \mathcal{F}_N v^\ell(k)\right),$$

where $\mathcal{F}_N v^\ell(k)$ denotes the (truncated)-Fourier coefficients of the discrete Fourier transform (DFT) of $v^\ell(x)$, computed based on the given n grid values in each direction. Here, $P_\ell(k) \in \mathbb{C}^{d_v \times d_v}$ is a complex Fourier multiplication matrix indexed by $k \in \mathbb{Z}^d$, d the total number of retained Fourier coefficients, and $\mathcal{F}_N^{-1}$ denotes the inverse DFT. The residual connection derives from a convolutional layer with kernel size 1.

DeepONet. DeepONet Lu et al. (2021) is the operator, $\mathcal{N}^{DONet} : H^r(D) \mapsto H^s(D)$, given by

$$\mathcal{N}^{DONet}(\bar{u})(y) = \sum_{k=1}^{p} \beta_k(\bar{u})\tau_k(y) \tag{6}$$

where the *branch-net* β is a neural network that maps $\mathcal{E}(\bar{u}) = (\bar{u}(x_1), \ldots, \bar{u}(x_m)) \in \mathbb{R}^m$, evaluations of the input $\bar{u}$ at sensor points $x := (x_1, \ldots, x_m) \in D$, to $\mathbb{R}^p$:

$$\beta : \mathbb{R}^m \to \mathbb{R}^p,\ \mathcal{E}(\bar{u}) \mapsto (\beta_1(\mathcal{E}(\bar{u})), \ldots, \beta_p(\mathcal{E}(\bar{u})), \tag{7}$$

and the *trunk-net* $\tau(y) = (\tau_1(y), \ldots, \tau_p(y))$ is another neural network mapping,

$$\tau : U \to \mathbb{R}^p, \quad y \mapsto (\tau_1(y), \ldots, \tau_p(y)). \tag{8}$$

Thus, a DeepONet combines the branch net (as coefficient functions) and trunk net (as basis functions) to create a mapping between functions.

In particular, in all numerical experiments, we employ standard feed-forward neural networks as trunk-net. In contrast, the branch is obtained as a composition of the encoder and Resnet of CNO architecure (without interpolation filter), and a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^p$, where n denotes the number of channels in the last layer of the ReseNet and p the number of basis functions.

B Details of the experiments

B.1 2D Navier-Stokes equations

Navier-Stokes equations describe the flow of an incompressible fluid with viscosity ν. The equations are given by

$$\frac{\partial u}{\partial u} + u \cdot \nabla u + \nabla p = \nu \Delta u, \quad \nabla \cdot u = 0, \quad u(t=0) = u_0,$$

where $u \in \mathbb{R}^2$ is the fluid velocity, $p \in \mathbb{R}$ is the fluid pressure and $u_0 \in \mathbb{R}^2$ is the initial velocity of the fluid. The fluid vorticity is defined as $\omega = \nabla \times u$.

B.2 Initial Conditions: Experiment 2

In the second experiment, we first generate the (pre-)initial velocity, defined by

$$u_0(x,y) = \begin{cases} \tanh\left(2\pi\frac{y-0.25}{\rho}\right), & y+\sigma(x) \leq \frac{1}{2} \\ \tanh\left(2\pi\frac{0.75-y}{\rho}\right), & \text{otherwise} \end{cases}, \quad v_0(x,y) = 0$$

Here, $\rho = 0.1$ and $\sigma : [0,1] \mapsto \mathbb{R}$ is the perturbation function given by$\sigma(x) = \sum_{k=1}^{10} \alpha_k \sin(2\pi kx - \beta_k)$, where α_k and β_k are i.i.d. uniformly distributed in $[0,1]$ and $[0,2\pi]$. The real initial velocity is obtained by Leray projection onto the divergence free manifold (to assure incompressibility).

B.3 Hyperparameters

Experiment 1. The CNO model has 3 (D) (and (U) blocks), 6 (I) blocks and 6 (R) blocks. The convolution size is $k = 3$. The filter properties are $f_c = f_s/2.0001$, $c_h = 1$ and number of taps $M = 16$. The number of Fourier features that we include is $m_{CNO} = 16$. The sequence of channels in the encoder is $17 \to 64 \to 128 \to 256$. There are approximately 5.3M parameters. The FNO architecture has 4 Fourier layers, $d = 16$ Fourier modes and the lifting dimension (width) $d_v = 64$. The model accounts for approximately 8.4M parameters. The trunk-net of DeepONet accounts for 4 hidden layers, with 256 neurons and $LeakyReLU$ activation function. On the other hand the branch consists of 4 (D), (I) and (R) blocks. Moreover, we reconstruct the output function as linear combination of 100 basis and we consider $m_{DON} = 4$ Fourier features. The model has roughly 3.4M parameters.

Experiment 2. The CNO model that we trained has 3 (D) (and (U) blocks), 6 (I) blocks and 10 (R) blocks. The convolution size is $k = 3$. The filter properties, the sequence of channels in the encoder and the number of Fourier features m_{CNO} is the same as in the previous experiment. There are approximately 1.9M parameters. As far as DeepONet is concerned, we employ 4 hidden layers and 128 neurons in the trunk, 6 (D) (and (I)) blocks, and 4 (R) blocks in the branch. Moreover, $m = 16$ Fourier features are used and 50 basis functions. The total number of parameters is 4.7M.

B.4 Illustration of Results reported in Section 3.

We start by elaborating on the results obtained on the benchmarks and presented in Table 3. In Figure 5, we show 4 randomly drawn test samples for the experiment 1. For all the test samples, we observe that CNO, FNO and DONet can accurately approximate the ground truth without any visible artifacts, whereas CNN assigns the same value to batches of neighboring cells, resulting in a visually coarser prediction.

Next, in Figure 6, we focus on the experiment 2 and plot 4 testing samples. CNO clearly yields to significantly more accurate predictions compared to the other models. In particular, FNO shows aliasing artifacts that explain the high error. Instead, DeepONet is not able to learn the operator, predicting for all the samples the same output.

We also show an example of input and output samples at different resolutions which are used in the **NS2** experiment in Figure 7

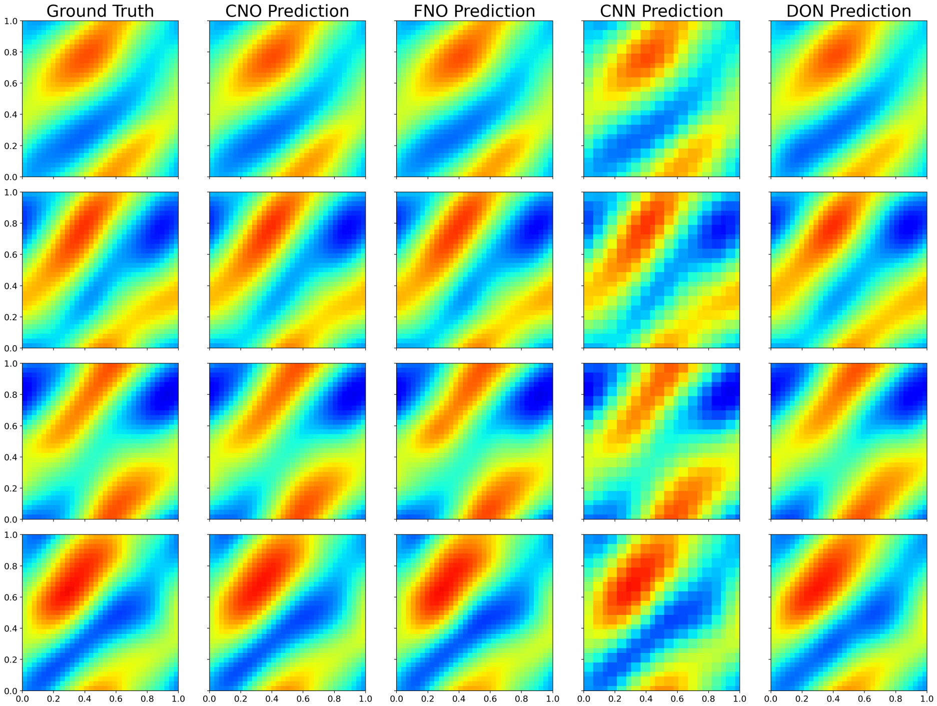

Figure 5: Exact and predicted coefficients for 4 different test samples (rows) and for different models (columns) for the NS Experiment 1. From left to right: ground truth, CNO, FNO, CNN, and DeepONet. The input and output resolutions are 33^2.

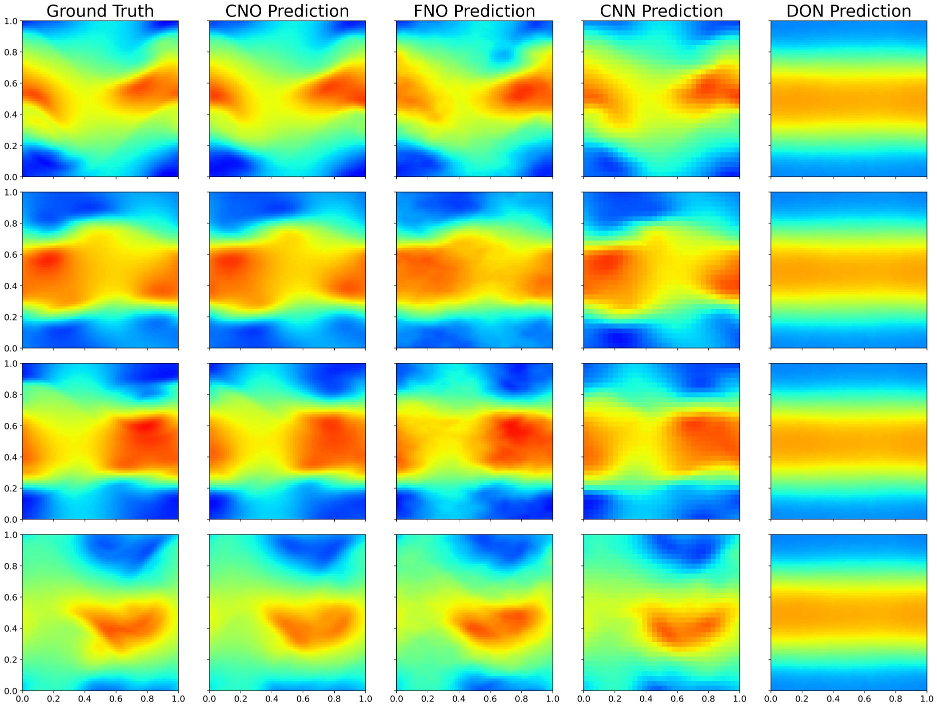

Figure 6: Exact and predicted coefficients for 4 different test samples (rows) and for different models (columns) for the NS Experiment 2. From left to right: ground truth, CNO, FNO, CNN, and DeepONet. The input and output resolutions are 64^2.

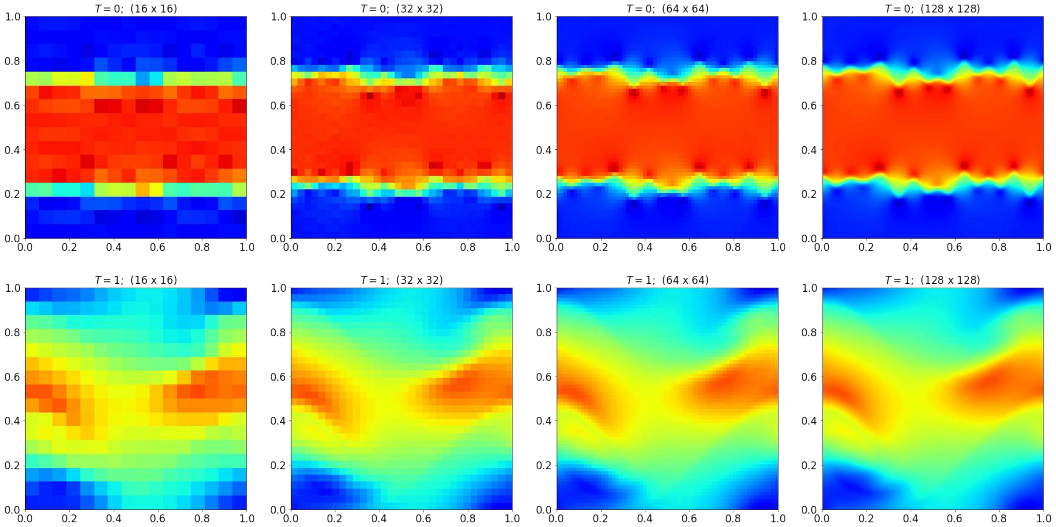

Figure 7: An example of input and output (ground truth) samples at 4 different resolutions for the NS Experiment 2. The input samples are on the top, while the corresponding output samples are on the bottom.