MODELLISTICA E CALCOLO SCIENTIFICO

MOX

MODELING AND SCIENTIFIC COMPUTING

MOX-Report No. 45/2022

Approximation bounds for convolutional neural networks in operator learning

Franco, N.; Fresca, S.; Manzoni, A.; Zunino, P.

Approximation bounds for convolutional neural networks in operator learning

Nicola R. Franco[1], Stefania Fresca[1], Andrea Manzoni[1], Paolo Zunino[1]

[1] *MOX, Department of Mathematics, Politecnico di Milano,*
Piazza Leonardo da Vinci 32, Milan, 20133, Italy

Abstract

Recently, deep Convolutional Neural Networks (CNNs) have proven to be successful when employed in areas such as reduced order modeling of parametrized PDEs. Despite their accuracy and efficiency, the approaches available in the literature still lack a rigorous justification on their mathematical foundations. Motivated by this fact, in this paper we derive rigorous error bounds for the approximation of nonlinear operators by means of CNN models. More precisely, we address the case in which an operator maps a finite dimensional input $\boldsymbol{\mu} \in \mathbb{R}^p$ onto a functional output $u_{\boldsymbol{\mu}} : [0,1]^d \to \mathbb{R}$, and a neural network model is used to approximate a discretized version of the input-to-output map. The resulting error estimates provide a clear interpretation of the hyperparameters defining the neural network architecture. All the proofs are constructive, and they ultimately reveal a deep connection between CNNs and the discrete Fourier transform. Finally, we complement the derived error bounds by numerical experiments that illustrate their application.

Keywords.
Operator learning · Convolutional neural networks · Approximation theory

1 Introduction

Convolutional Neural Networks (CNNs) have become very popular after their tremendous success in computer vision, with applications ranging from image processing to generative models for images generation (Dosovitskiy, Tobias Springenberg, and Brox, 2015; Sultana, Sufian, and Dutta, 2020). From a mathematical point of view, image-like data are equivalent to discrete functional signals defined over rectangular domains and vice versa.

Indeed, each continuous function $u : [0,1]^2 \to \mathbb{R}$ can be discretized in matrix form as

$$\mathbf{U} := \begin{bmatrix} u(x_{1,1}) & \dots & u(x_{1,N}) \\ \dots & \dots & \dots \\ u(x_{N,1}) & \dots & u(x_{N,N}) \end{bmatrix} \in \mathbb{R}^{N \times N},$$

where $\{x_{i,j}\}_{i,j=1,\dots,N}$ are the vertices of some uniform grid defined over the unit square. In light of this, CNNs have been recently employed for tasks that go beyond computer vision, such as operator learning and reduced order modelling of parameter-dependent PDEs (Franco, Manzoni, and Zunino, 2021; Fresca, Dede, and Manzoni, 2021; Fresca and Manzoni, 2022; Lee and Carlberg, 2020; Mücke, Bohté, and Oosterlee, 2021).

As an example, let $\Omega = (0,1)^d$ and assume we are given an operator $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$ that maps a finite dimensional input $\boldsymbol{\mu} \in \mathbb{R}^p$ onto some functional signal $u_{\boldsymbol{\mu}} :

$\Omega \to \mathbb{R}$. This is a classical set-up in parameter-dependent PDE models, where each parameter instance is associated with the corresponding PDE solution. Once a suitable, discrete grid of points $\{x_{i_1,\ldots,i_d}\}_{i_1,\ldots,i_d=1,\ldots,N} \subset \Omega$ has been introduced, the operator of interest can be expressed as

$$\mathbb{R}^p \ni \boldsymbol{\mu} \to \mathbf{U}_{\boldsymbol{\mu}} \in \mathbb{R}^{N \times \cdots \times N},$$

denoting by $\mathbf{U}_{\boldsymbol{\mu}}^{i_1,\ldots,i_d} \approx u_{\boldsymbol{\mu}}(x_{i_1,\ldots,i_d})$. The final goal is then to construct a neural network model

$$\Phi : \mathbb{R}^p \to \mathbb{R}^{N \times \cdots \times N}, \ \text{ such that } \ \Phi(\boldsymbol{\mu}) \approx \mathbf{U}_{\boldsymbol{\mu}},$$

with the idea of replacing an operator that is otherwise computationally expensive to evaluate. As previously mentioned, this task can be successfully achieved by CNNs, as they allow to intrinsically account for underlying spatial correlations. However, the literature still lacks a comprehensive mathematical analysis and foundation motivating the remarkable performance shown by CNNs, and the role played by each hyperparameter in a CNN model remains unclear. In this work, we aim at addressing these critical points, showing rigorous estimates on the error $\mathcal{E} := \sup_{\boldsymbol{\mu}} \sup_{j \in \{1,\ldots,N\}^d} |u_{\boldsymbol{\mu}}(x_j) - \Phi_j(\boldsymbol{\mu})|$ generated when approximating the operator of interest by means of CNNs.

Literature review

Neural Networks (NNs) were known to be universal approximators since Cybenko (1989), however the design of effective NN architectures able to preserve desired accuracy properties had not been in-depth investigated until recent years. A substantial improvement was achieved in (Yarotsky, 2017), where a rigorous mathematical meaning to structural NN properties such as width and depth of a NN model has been first provided. In particular, Yarotsky proved that any s-differentiable scalar-valued map $f : [0,1]^p \to \mathbb{R}$ can be approximated uniformly with error $\varepsilon > 0$ by some ReLU Deep Neural Network (DNN) with $c \log(1/\varepsilon)$ layers and $c\varepsilon^{-s/p} \log(1/\varepsilon)$ active weights, where $c = c(p,s,f)$ is some constant that depends on the derivatives of f. This result was later extended to more general activation functions and different norms, see e.g. Gühring, Kutyniok, and Petersen (2020); Gühring and Raslan (2021); Siegel and Xu (2022), and adapted to the case of CNNs exploiting some algebraic arguments that link dense and convolutional layers, see e.g. Zhou (2020); He, Li, and Xu (2021).

However, all these results are limited to the approximation of scalar-valued maps and they are not suited for operator learning. To this end, it is worth to note the following aspect. Assume that we are interested in approximating a vector-valued map $f : [0,1]^p \to \mathbb{R}^n$, $f(\boldsymbol{\mu}) = [f_1(\boldsymbol{\mu}), \ldots, f_n(\boldsymbol{\mu})]$, with a DNN model Φ. Clearly, we could exploit the aforementioned results to approximate each f_i with some DNN ϕ_i, and then stack together the models to get $\Phi := [\phi_1, \ldots, \phi_n]$. However, with this construction the number of active weights in Φ would grow linearly with n as $n \to +\infty$. In our context, where we deal with functional outputs and $n = N^d$ comes from having discretized $\Omega = (0,1)^d$ with a computational grid with N nodes per side, this would be rather undesirable.

Nevertheless, new approaches are now appearing in the literature, in a first attempt to employ NNs for operator learning. Some of these, such as Neural Operators (Kovachki, Li, Liu, Azizzadenesheli, Bhattacharya, Stuart, and Anandkumar, 2021) and DeepONets (Lu, Jin, Pang, Zhang, and Karniadakis, 2021), work with a continuous functional output space, while a second class of approaches relies on a discretization of the output space, see e.g. Kutyniok, Petersen, Raslan, and Schneider (2021). In this work, we focus on the latter family of approaches.

Neural Operators provide a novel framework for building models between infinite-dimensional spaces, and are essentially based on integral operators. Among them, those that have been mostly investigated are Fourier Neural Operators, for which several error estimates have been derived, see, e.g., Kovachki, Lanthaler, and Mishra (2021). Conversely, DeepONets are a class of models based on a separation of variables approach, which decouples the input parameter and the space variable at output. Error estimates for DeepONets are also available, see Lanthaler, Mishra, and Karniadakis (2022), and they are mostly settled on the aforementioned results in the scalar case and on those by Schwab and Zech (2019) for high-dimensional inputs.

Besides these methods, deep learning approaches that discretize the functional output space are also available. This need usually arises, for instance, when dealing with parameter-dependent PDEs, whose solutions are usually computed through numerical discretization schemes like, e.g., the finite element method. In this case, the functional output space – usually given by a Sobolev space – is replaced by a finite-dimensional trial space (e.g., the space of finite elements of degree r built over a triangulation of the spatial domain Ω). Deep learning approaches of this type were proposed in Bhattacharya, Hosseini, Kovachki, and Stuart (2020) and Kutyniok et al. (2021). The former relies on linear reduction methods to deal with the functional component at output, and it is able to recover mesh independence. The second one, instead, is purely based on DNNs. Both works provide error estimates, most of which are derived by exploiting the results in the scalar-case and projection arguments.

This flourishing literature indicates a growing interest aimd at understanding the properties of DNN models and their potential in operator learning. However, at the best of our knowledge, no comprehensive study has been proposed for CNN models yet, despite these latter are extremely popular in practical applications. One reason might be that CNN architectures can be traced back to sparse versions of dense models, which led researchers to focus on deriving error bounds for DNNs, see e.g. Petersen and Voigtlaender (2020). Moreover, CNN models have been mostly studied for handling high-dimensional data at input and not as output, as in He et al. (2021). As a consequence, the available literature is left with a missing piece, which is to understand the approximation properties of convolutional layers when reconstructing functional signals. In the present work, we aim at addressing this issue.

Our contribution

Let $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$ be some nonlinear operator whose output are functions $u_{\boldsymbol{\mu}} : [0,1]^d \to \mathbb{R}$ defined over the unit hypercube. We provide error bounds for the approximation of such an operator via a CNN model $\Phi : \mathbb{R}^p \to \left(\mathbb{R}^N\right)^d$. In particular, we characterize the model architecture in terms of the approximation error

$$\mathcal{E} := \sup_{\boldsymbol{\mu} \in \Theta} \sup_{j \in \{1,\dots,N\}^d} |u_{\boldsymbol{\mu}}(x_j) - \Phi_j(\boldsymbol{\mu})|,$$

where $\Theta \subset \mathbb{R}^p$ is some parameter space and $\{x_j\}_j \subset [0,1]^d$ is a suitable $N \times \cdots \times N$ grid. By doing so, we also provide a clear interpretation to the model hyperparameters, including the number of dense and convolutional layers, the amount of active weights and the convolutional channels. In the present work, we limit ourselves to the 1-dimensional case, $d = 1$, even if the ideas at the core of our proofs can be extended to higher-dimensions with little effort.

We report below our main result, Theorem 2, in which we characterize the approximation error in terms of the complexity of a DNN model comprised of a

3

dense and a convolutional block. In what follows, we denote by $H^s(\Omega)$ the Sobolev space of s-times weakly-differentiable maps with square-integrable derivatives.

Theorem 2 *Let $\Omega := (0,1)$ and let $\{x_j\}_{j=1}^{N_h} \subset \Omega$ be a uniform grid with stepsize $h = 2^{-k}$. Let $\Theta \subset \mathbb{R}^p \ni \boldsymbol{\mu} \to u_{\boldsymbol{\mu}} \in H^s(\Omega)$ be a (nonlinear) operator, where Θ is a compact domain and $s \geq 1$. For some $r \geq 0$, assume that the operator is r-times Fréchet differentiable and that its r-derivative is Lipschitz continuous. Then, for any $0 < \varepsilon < 1$, there exists a Deep Neural Network $\Phi : \mathbb{R}^p \to \mathbb{R}^{N_h}$ such that*

$$|u_{\boldsymbol{\mu}}(x_j) - \Phi_j(\boldsymbol{\mu})| < \varepsilon$$

uniformly for all $\boldsymbol{\mu} \in \Theta$ and all $j = 1, \dots, N_h$. Additionally, Φ can be defined to have at most

  i) $C \log(1/\varepsilon)$ *dense layers, with ReLU activation, and $C \log(1/h)$ convolutional layers,*

  ii) $C\varepsilon^{-2/(2s-1)} \left[\varepsilon^{-p/(r+1)} \log(1/\varepsilon) + \log(1/h) \right]$ *active weights,*

  iii) $C\varepsilon^{-2/(2s-1)}$ *channels in input and output,*

where $C > 0$ is some constant dependent on Θ and on the operator $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$, thus also on s, r, p.

In particular, the above result shows that:

  (i) The number of dense layers depends logarithmically on the desired accuracy, while that of the convolutional layers depends logarithmically on the mesh resolution, i.e. on the number of discretization points.

  (ii) The width of the dense block is related to the regularity of the operator itself, with smooth operators requiring less neurons.

  (iii) The number of convolutional features depends on the regularity of the signals $u_{\boldsymbol{\mu}}$ at output.

We mention that, while being the ultimate focus of our work, Theorem 2 is only proved at the end of the paper, as we first need to derive some preliminary results. More precisely, the paper is organized as follows. First, in Section 2 we establish a link between convolutional layers and the Fourier transform. Then, in Section 3, we exploit these results to build a CNN model capable of reconstructing any functional output. Finally, in Section 4, we resort to the parametrized setting and we prove rigorously Theorem 2. In addition, we report in Section 5 some numerical experiments, where we assess the predicted error bounds. A discussion on possible extensions of our results to higher-dimensions can be found in Section 6; on the other hand, preliminary notions, such as the formal definition of CNN models, are reported in the Appendix to make the paper self-contained.

2 Interpolation of Fourier modes

Convolution operations are intimately connected to the Fourier transform via the so-called Convolution Theorem, see e.g. Katznelson (1976). Here, we further investigate this connection by deriving some preliminary results that will serve as building blocks for Theorems 1 and 2. The idea can be stated as follows. Given any dyadic partition of the unit interval,

$$\{x_j\}_{j=1}^{N_h} := \{(j-1)2^{-k}\}_{j=1}^{N_h},$$

where $N_h := 2^k + 1$, and any positive integer m, we construct a CNN model $\mathcal{S}_m$ that interpolates the (discrete) map

$$[\mathbf{z}_{-m}, \cdots, \mathbf{z}_m] \to \left[\sum_{k=-m}^{m} \mathbf{z}_k e^{2\pi \mathbf{i} k x_0}, \ldots, \sum_{k=-m}^{m} \mathbf{z}_k e^{2\pi \mathbf{i} k x_{N_h}} \right]$$

associating the coefficient $\mathbf{z}_k \in \mathbb{C}$, $k = -m, \ldots, m$, to the truncated Fourier transform at the points x_j, $j = 1, \ldots, N_h$, with $\mathbf{i}$ the imaginary unit. The construction of $\mathcal{S}_m$ is detailed step-by-step, starting at Lemma 1 and concluding with Lemma 3. The proofs are constructive, as they explicitly describe how to implement $\mathcal{S}_m$. In particular, we are able to characterize the complexity of $\mathcal{S}_m$ in terms of those specific features that are typical of CNNs, such as depth, kernel size, stride, dilation, padding, and number of input-output channels. For instance, we show in Lemma 3 that the depth of $\mathcal{S}_m$ grows logarithmically with the grid resolution, while the active weights grow linearly with m. These observations will play a key role for deriving the upper bounds in Theorems 1 and 2.

In what follows, we make use of the embedding $\mathbb{C} \hookrightarrow \mathbb{R}^4$,

$$\mathbf{z} \to [\mathrm{Re}(\mathbf{z}), \mathrm{Im}(\mathbf{z}), \mathrm{Re}(\mathbf{z}), \mathrm{Im}(\mathbf{z})],$$

to represent complex numbers. This will come in handy when trying to mimic the algebra of elements of $\mathbb{C}$ by using neural networks. With this convention, we also let $\mathbb{C}^n \hookrightarrow \mathbb{R}^{4 \times n}$ in the obvious way.

Lemma 1 *For any $k \in \mathbb{N}$ and any $\mathbf{z} \in \mathbb{C}$ there exists a convolutional neural network $\phi_{\mathbf{z}}^k : \mathbb{C}^{2^{k-1}} \to \mathbb{C}^{2^k}$ such that*

   i) it is linear (no activation at any level),

  ii) it only employs 1D convolutional and reshaping operations,

 iii) it has an architecture of at most six layers,

 iv) the input and the output of its convolutional layers have at most 8 channels,

  v) the kernels of the convolutional layers have size at most equal to 2,

and such that

$$\phi_{\mathbf{z}}^k([\mathbf{w}_1, \ldots, \mathbf{w}_{2^{k-1}}]) = [\mathbf{w}_1, \mathbf{z}\mathbf{w}_1, \ldots, \mathbf{w}_{2^{k-1}}, \mathbf{z}\mathbf{w}_{2^{k-1}}]$$

for all $\mathbf{w}_1, \ldots, \mathbf{w}_{2^{k-1}} \in \mathbb{C}$.

Proof. Let $n = 2^{k-1}$ be the (complex) input dimension. Let f_1 be a 1D transposed convolutional layer with the following specifics. The layer has four channels at input and four channels at output. It uses a 2-sized window that acts with a stride of 2. The layer has no bias and its weight matrix $\mathbf{W}_1 \in \mathbb{R}^{4 \times 4 \times 2}$, which is obtained by stacking together the convolutional kernels, is zero at all but six entries. These are given by the relations below

$$\begin{bmatrix} \mathbf{W}_1^{1,1,1} & \mathbf{W}_1^{1,1,2} \\ \mathbf{W}_1^{2,2,1} & \mathbf{W}_1^{2,2,2} \\ \mathbf{W}_1^{3,3,1} & \mathbf{W}_1^{3,3,2} \\ \mathbf{W}_1^{4,4,1} & \mathbf{W}_1^{4,4,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \mathrm{Re}(\mathbf{z}) & \mathrm{Im}(\mathbf{z}) \\ -\mathrm{Im}(\mathbf{z}) & \mathrm{Re}(\mathbf{z}) \end{bmatrix}.$$

Note that above we are also listing some of the zero entries in $\mathbf{W}_1$. In this way, it is easier to see what the purpose of f_1 is. The first block in $\mathbf{W}_1$ is used to

mimic the action of the identity matrix. Conversely, the second block encodes a 2×2 matrix representation of the complex number $\mathbf{z}$. The idea is that these two blocks should provide a way of computing the map $\mathbf{w} \to [\mathbf{w}, \mathbf{zw}]$. However, for these computations to be actually carried out, we also need a further layer that performs a suitable summation of the outputs given by f_1. To this end, we define the second layer, f_2, as a 1D convolution that maps 4-channeled inputs onto 2-channeled outputs. The latter uses a 1-sized window that acts with a stride of 1. The layer has no bias and its weight matrix $\mathbf{W}_1 \in \mathbb{R}^{2 \times 4 \times 1}$ contains either zeros or ones. The positive entries are

$$\mathbf{W}_2^{1,1,1}, \ \mathbf{W}_2^{1,2,1}, \ \mathbf{W}_2^{2,3,1}, \ \mathbf{W}_2^{2,4,1} = 1.$$

Then, $f_2 \circ f_1 : \mathbb{R}^{4 \times n} \to \mathbb{R}^{2 \times 2n}$, and, upto some basic calculations, we have

$$(f_2 \circ f_1)\left([\mathbf{w}_1, \dots, \mathbf{w}_n]\right) =$$

$$\left[\begin{array}{cccccc} \mathrm{Re}(\mathbf{w}_1), & \mathrm{Im}(\mathbf{w}_1), & \dots, & \mathrm{Re}(\mathbf{w}_n), & \mathrm{Im}(\mathbf{w}_n) \\ \mathrm{Re}(\mathbf{zw}_1), & \mathrm{Im}(\mathbf{zw}_1), & \dots, & \mathrm{Re}(\mathbf{zw}_n), & \mathrm{Im}(\mathbf{zw}_n) \end{array} \right].$$

In practice, the desired output of $\phi_{\mathbf{z}}^k$ is already there, but we need to adjust the the output dimension in order to match our convention for complex numbers.

To this end, we start by introducing a reshape operation $R_1 : \mathbb{R}^{2 \times 2n} \to \mathbb{R}^{1 \times 4n}$ that flattens the whole output. Then, we add a third convolutional layer, f_3, whose purpose is to double the entries in input. More precisely, we define f_3 has a 1D convolution that has 1 channel at input and 4 at output. The layer uses a 2-sized kernel that acts with a stride of 2. Once again, the layer introduces no bias and has a weight matrix $\mathbf{W}_3 \in \mathbb{R}^{4 \times 1 \times 2}$ given by

$$\mathbf{W}_3 = [[[1, 0]], \ [[0, 1]], \ [[1, 0]], \ [[0, 1]]].$$

With the notation adopted to represent complex numbers, the current action of $f_3 \circ R_1 \circ f_2 \circ f_1$ becomes

$$[\mathbf{w}_1, \dots, \mathbf{w}_n] \to [\mathbf{w}_1, \dots, \mathbf{w}_n, \mathbf{zw}_1, \dots, \mathbf{zw}_n].$$

Let us now act further on the output to sort the entries in the desired order. To do so, we introduce a 1D convolutional layer, f_4, that has a dilation factor of 2^k (this is because we want to group $\mathbf{w}_1$ with $\mathbf{zw}_1$, which is 2^k entries faraway, and so on). We let f_4 go from 4 to 8 channels, and employ a kernel of size 2 with unit stride. Once again, f_4 does not have a bias term, while its weight matrix satisfies

$$\mathbf{W}_4^{i,j,k} = \begin{cases} 1 & \text{if } i = j + 4(k-1) \\ 0 & \text{otherwise.} \end{cases}$$

At this point we have,

$$f_4 \circ f_3 \circ R_1 \circ f_2 \circ f_1 : [\mathbf{w}_1, \dots, \mathbf{w}_n] \to \left[\begin{array}{ccc} \mathbf{w}_1, & \dots, & \mathbf{w}_n \\ \mathbf{zw}_1, & \dots, & \mathbf{zw}_n \end{array} \right],$$

and we only need to add a final reshaping operation R_2. We let R_2 to act as follows. First, it transposes the input by mapping $\mathbb{R}^{8 \times n} \to \mathbb{R}^{n \times 8}$. Then, it performs the reshaping $\mathbb{R}^{n \times 8} \to \mathbb{R}^{2n \times 4}$, where entries are read by rows, and finally it transposes back the input so that it ends up in $\mathbb{R}^{4 \times 2n} \cong \mathbb{C}^{2^k}$. Finally, letting $\phi_{\mathbf{z}}^k := R_2 \circ f_4 \circ f_3 \circ R_1 \circ f_2 \circ f_1$ concludes the proof. $\qquad \square$

Lemma 2 *Let $k \in \mathbb{N}$ and $h = 2^{-k}$. Let $\{x_1, \ldots, x_{N_h}\}$ be an uniform partition of $[0, 1]$ with stepsize h. For any $\omega \in \mathbb{R}$, there exists a convolutional neural network $\mathcal{F}_\omega : \mathbb{C} \to \mathbb{C}^{N_h - 1}$ such that*

  i) $\mathcal{F}_\omega$ *is linear (no activation at any level),*

  ii) $\mathcal{F}_\omega$ *has at most depth $C \log(1/h)$,*

  iii) $\mathcal{F}_\omega$ *has at most $C \log(1/h)$ active weights,*

  iv) $\mathcal{F}_\omega(\mathbf{w}) = [\mathbf{w} e^{\mathbf{i}\omega x_1}, \ldots, \mathbf{w} e^{\mathbf{i}\omega x_{N_h - 1}}]$.

where $C > 0$ is a constant independent on h and ω. Furthermore, up to reshape operations, $\mathcal{F}_\omega$ only uses 1D convolutional layers that have at most 8 channels at both input and output. Moreover, the kernel size of all convolutional layers in $\mathcal{F}_\omega$ is at most 2.

Proof. Let $\mathbf{z} \in \mathbb{C}$. For all $j = 1, \ldots, k$, define the CNNs $\phi^j_{\mathbf{z}^{2^{j-1}}}$ as in Lemma 1. For the sake of simplicity assume that $k \geq 2$. Then it is straightforward to check that $\phi^1_{\mathbf{z}} : \mathbf{w} \to [\mathbf{w}, \mathbf{wz}]$, while $\phi^2_{\mathbf{z}^2} \circ \phi^1_{\mathbf{z}} : \mathbf{w} \to [\mathbf{w}, \mathbf{wz}^2, \mathbf{wz}, \mathbf{wz}^3]$ and so on. In particular,

$$\phi^k_{\mathbf{z}^{2^{k-1}}} \circ \cdots \circ \phi^1_{\mathbf{z}} : \mathbf{w} \to \pi\left([\mathbf{w}, \mathbf{wz}, \mathbf{wz}^2, \ldots, \mathbf{wz}^{2^k}]\right)$$

where $\pi : \mathbb{C}^{N_h - 1} \to \mathbb{C}^{N_h - 1}$ is some map that acts as a permutation over the entries. Let now $\mathbf{z} := e^{\mathbf{i}\omega h}$ and define $\mathcal{F}_\omega := \pi^{-1} \circ \phi^k_{\mathbf{z}^{2^{k-1}}} \circ \cdots \circ \phi^1_{\mathbf{z}}$. Then,

$$\mathcal{F}_\omega : \mathbf{w} \to [\mathbf{w} e^{\mathbf{i}\omega 0 h}, \mathbf{w} e^{\mathbf{i}\omega 1 h}, \ldots, \mathbf{w} e^{\mathbf{i}\omega (N_h - 2) h}],$$

and the conclusion follows. $\square$

Lemma 3 *Let $k \in \mathbb{N}$ and $h = 2^{-k}$. Let $\{x_1, \ldots, x_{N_h}\}$ be a uniform partition of $[0, 1]$ with stepsize h. For any positive integer m, there exists a convolutional neural network $\mathcal{S}_m : \mathbb{C}^{2m+1} \to \mathbb{C}^{N_h}$ such that*

  i) $\mathcal{S}_m$ *is linear (no activation at any level),*

  ii) $\mathcal{S}_m$ *has at most depth $C \log(1/h)$,*

  iii) $\mathcal{S}_m$ *has at most $Cm \log(1/h)$ active weights,*

  iv) $\mathcal{S}_m$ *uses convolutional layers with at most Cm channels,*

  v) for any complex vector $\mathbf{Z} = [\mathbf{z}_{-m}, \ldots, \mathbf{z}_m] \in \mathbb{C}^{2m+1}$ one has

$$\mathcal{S}_m(\mathbf{Z})_i = \sum_{k=-m}^{m} \mathbf{z}_k e^{2\pi \mathbf{i} k x_i},$$

  for all $i = 1, \ldots, N_h$, where $\mathcal{S}_m(\mathbf{Z})_i$ is the ith component of the output vector $\mathcal{S}_m(\mathbf{Z})$.

Here, $C > 0$ is a universal constant independent on h and m. Furthermore, up to reshape operations, $\mathcal{S}_m$ only uses 1D convolutional layers whose kernel size does not exceed 8.

Proof. Fix any $k \in \{-m, \dots, m\}$. Let $\mathcal{F}_{(k)}$ be the CNN in Lemma 2 when $\omega = 2\pi k$. We note that, as k varies, the structure of $\mathcal{F}_{(-m)}, \dots \mathcal{F}_{(m)}$ does not change: these architectures have the same depth and they employ convolutional layers with the same specifics. Also, the reshaping operations entailed by the networks occur at the same locations.

Therefore, we can stack all these models on top of each other to obtain a global CNN $\tilde{\mathcal{S}}_m$ such that

$$\tilde{\mathcal{S}}_m(\mathbf{Z}) = \left[\mathcal{F}_{(-m)}(\mathbf{z}_{-m}), \dots, \mathcal{F}_{(m)}(\mathbf{z}_m) \right]$$

where $\mathbf{Z} = [\mathbf{z}_{-m}, \dots, \mathbf{z}_m] \in \mathbb{C}^{2m+1}$ is a generic input vector. This can be done as follows.

To stack $2m+1$ convolutional layers with c_{in} channels at input and c_{out} channels at output each, we define a single CNN layer with $(2m+1)c_{in}$ channels at input and $(2m+1)c_{out}$ at output. Then, to avoid the introduction of redundant kernels, we constrain the new layer to group its kernels in subsets of $(2m+1)$. This ensures the wished behavior, i.e. that we actually stack the outputs of the $2m+1$ original layers as if they work in parallel (thus each seeing only the part of interest of the input). Similarly, reshaping and transpositions can be easily stacked together. For instance, stacking $(2m+1)$ transpositions of the form $\phi : \mathbb{R}^{a \times b} \to \mathbb{R}^{b \times a}$ results in a map from $\mathbb{R}^{(2m+1) \times a \times b}$ to $\mathbb{R}^{(2m+1) \times b \times a}$.

Since $\tilde{\mathcal{S}}_m$ takes values in $\mathbb{C}^{(2m+1) \times (N_h-1)}$, our next purpose is to append a further layer L such that

$$(L \circ \tilde{\mathcal{S}}_m)(\mathbf{Z}) = \sum_{k=-m}^{m} \mathcal{F}_{(k)}(\mathbf{z}_k) \in \mathbb{C}^{N_h-1}.$$

It is easy to see that L can be obtained with a convolutional layer having $(2m+1)$ channels at input and 1 at output, no stride or dilation, and a kernel of size 1 whose weight is constantly equal to 1.

Finally, we note that for all $k \in \{-m, \dots, m\}$ we have

$$e^{2\pi \mathbf{i} k x_1} = e^{2\pi \mathbf{i} k 0} = 1 = e^{2\pi \mathbf{i} k} = e^{2\pi \mathbf{i} k x_{N_h}}$$

due to periodicity. Therefore, we may simply define $\mathcal{S}_m := A \circ L \circ \tilde{\mathcal{S}}_m$, where A has the only purpose of appending a copy of its first output at the end, that is

$$A(\mathbf{w}_1, \dots, \mathbf{w}_{N_h-1}) = [\mathbf{w}_1, \dots, \mathbf{w}_{N_h-1}, \mathbf{w}_1].$$

This can be seen as a form of reshaping or padding.

By construction, $\mathcal{S}_m$ satisfies (v). Similarly, (i) and (iv) hold. In fact, each of the $\mathcal{F}_{(k)}$ has length $C \log(1/h)$, where C is a common constant. Since we stacked them in parallel to get $\tilde{\mathcal{S}}_m$, our final model has depth $C \log(1/h) + 2 = \tilde{C} \log(1/h)$. Also, the CNNs $\mathcal{F}_{(k)}$ featured at most 8 channels, thus $\mathcal{S}_m$ uses no more than $(2m+1)8 = \tilde{C}m$ channels at input-output. Property (iii) follows similarly by recalling that we grouped the CNNs kernels in order to properly stack the architectures. □

3 Approximation of a single function

We shall now exploit the theory developed in Section 2 in order to derive suitable error bounds for CNNs. To this end, our primal goal is to show that, for any desired accuracy, there exists a single CNN architecture that is able to provide as output an approximation for any function belonging to a given class in terms of

smoothness. More precisely, let $s \geq 1$ be a smoothness index, and fix some $m \in \mathbb{N}$. Let also $\{x_j\}_{j=1}^{N_h}$ be some dyadic grid defined over $\Omega = (0,1)$. We build a CNN model $\Psi : \mathbb{C}^{2m+1} \to \mathbb{R}^{N_h}$ such that

$$\forall f \in H^s(\Omega) \ \exists \mathbf{Z}_f \in \mathbb{C}^{2m+1} \ \text{ such that}$$

$$\sup_{j=1,\dots,N_h} |f(x_j) - \Psi_j(\mathbf{Z}_f)| < C m^{1/2-s} ||f||_H^s(\Omega),$$

where $C = C(s) > 0$ is some constant and Ψ_j is the jth component of the CNN output. The above states that any smooth function f can be well approximated by Ψ, provided that the model is fed with a suitable input vector. As for Lemma 3, we characterize the network complexity in terms of depth, channels and active weights. Furthermore, we show that the map $f \to \mathbf{Z}_f$ can be realized by some continuous linear operator that depends, at most, on s. Before stating this rigorously in Theorem 1, it is worth to remark that this result concerns the approximation of any functional output in $H^s(\Omega)$. In particular, although the proof is based on classical estimates coming from the literature of Fourier series, no periodicity is required.

Theorem 1 *Let $k \in \mathbb{N}$ and $h = 2^{-k}$. Let $\{x_j\}_{j=1}^{N_h}$ be a uniform partition of $\Omega := (0,1)$ with stepsize h, so that $N_h = 2^k + 1$. For any positive integer m and a universal constant C independent on m and h, there exists a linear convolutional neural network $\Psi : \mathbb{C}^{2m+1} \to \mathbb{R}^{N_h}$ with*

i) at most $C \log(1/h)$ layers,

ii) at most $C m \log(1/h)$ active weights,

iii) at most $8m$ channels in input and output, with kernels grouped by m,

such that for any $s \geq 1$ and all $f \in H^s(\Omega)$ one has

$$\sup_{j=1,\dots,N_h} |f(x_j) - \Psi_j(Tf)| \leq c m^{1/2-s} ||f||_{H^s(\Omega)}.$$

Here, $c = c(s) > 0$ and $T : H^s(\Omega) \to \mathbb{C}^{2m+1}$ are a positive constant and a continuous linear operator that depend on s, respectively.

Proof. Let us denote by $\mathbb{T}$ the 1-dimensional torus, $\mathbb{T} := \mathbb{R}/\mathbb{Z}$, so that the spaces $\mathcal{C}^k(\mathbb{T})$ refer to those functions that are k-times differentiable on the torus, namely

$$\mathcal{C}^k(\mathbb{T}) = \left\{ f \in \mathcal{C}^k[0,1], \ f^{(j)}(0) = f^{(j)}(1) \ \forall j = 1,\dots,k \right\}.$$

We start by defining an operator $T_0 : H^s(\Omega) \to H^s(\Omega) \cap \mathcal{C}^{s-1}(\mathbb{T})$ that turns any input function into a periodic signal. To this end, we recall that there exist polynomials $p_1,\dots,p_{s-1}$ of degree $2s - 1$ such that

$$p_j^{(k)}(0) = \delta_{j,k}, \quad p_j^{(k)}(1) = 0$$

for $j,k \in \{1,\dots,s-1\}$, see e.g. Spitzbart (1960). Then, the linear operator

$$\mathbf{v} \to P\mathbf{v} := \sum_{j=1}^{s-1} v_j p_j$$

maps any input vector $\mathbf{v} \in \mathbb{R}^{s-1}$ into a periodic signal in $H^s(\Omega) \cap \mathcal{C}^{s-1}(\mathbb{T})$. In fact, while the s-derivative of $P\mathbf{v}$ may not be periodic, it is still continuous over the unwrapped domain $\Omega = (0,1)$.

By recalling that $H^s(\Omega) \hookrightarrow \mathcal{C}^{s-1}(\Omega)$ thanks to classical Sobolev inequalities, we are then allowed to define

$$T_0 : f \to f + P[f^{(0)}(1) - f^{(0)}(0), \dots, f^{(s-1)}(1) - f^{(s-1)}(0)]$$

so that $T_0 : H^s(\Omega) \to H^s(\Omega) \cap \mathcal{C}^{s-1}(\mathbb{T})$ as desired. We now introduce the following notation. For any $f \in H^s(\Omega)$ we let $\tilde{f}$ be defined as

$$\tilde{f}(x) = \begin{cases} (T_0 f)(2x) & 0 \le x \le 1/2 \\ f(2x-1) & 1/2 < x \le 1 \end{cases}. \tag{1}$$

It is straightforward to see that $\tilde{f} \in H^s(\Omega) \cap \mathcal{C}^{s-1}(\mathbb{T})$. Also, the mapping $f \to \tilde{f}$ is linear and continuous, in the sense that for some constant $C > 0$ depending only on s we have

$$||\tilde{f}||_{H^s(\Omega)} \le C||f||_{H^s(\Omega)} \tag{*}$$

for all $f \in H^s(\Omega)$. For any positive integer m, let $S_m \tilde{f}$ be the m truncated Fourier series of the function $\tilde{f}$,

$$\left(S_m \tilde{f} \right)(x) = \sum_{j=-m}^{m} c_{\tilde{f}}^k e^{2\pi \mathbf{i} k x}$$

where

$$c_{\tilde{f}}^k := \int_\Omega \tilde{f}(x) e^{-2\pi \mathbf{i} k x} dx.$$

Since $\tilde{f} \in \mathcal{C}^{s-1}(\mathbb{T})$ and its s-derivative is in $L^2(\mathbb{T})$, we have

$$||\tilde{f} - S_m \tilde{f}||_{L^\infty(\Omega)} \le \sqrt{\frac{2}{2s-1}} m^{1/2-s} ||\tilde{f}||_{H^s(\Omega)} \tag{**}$$

by exploiting classical estimates of Fourier analysis. Let now $T : H^s(\Omega) \to \mathbb{C}^{2m+1}$ be defined as

$$T : f \to \left[c_{\tilde{f}}^{-m}, \dots, c_{\tilde{f}}^m \right],$$

so that T maps each signal into the Fourier coefficients of its periodic alias. Let $\{y_0, \dots, y_{2N_h - 1}\}$ be a uniform partition of $(0,1)$ that is twice as fine as the original one $\{x_0, \dots, x_{N_h}\}$, that is $y_{j+1} - y_j = h/2$. With this partition as a reference, let then $\mathcal{S}_m$ be the CNN in Lemma 3. By definition, we have

$$\left(S_m \tilde{f} \right)(y_i) = \mathcal{S}_m(Tf)_i.$$

Finally, let $\Psi := E \circ R \circ \mathcal{S}_m$, where

- R is a reshape truncation layer,

$$R(\mathbf{w}_1, \dots, \mathbf{w}_{2N_h - 1}) = [\mathbf{w}_{N_h}, \dots, \mathbf{w}_{2N_h - 1}],$$

  that we use to remove the undesired output. Note in fact that, the signal $\tilde{f}$ over $(1/2, 1)$ is practically f over $(0, 1)$. Thus, in light of (**), we are only interested in the second half of the output.

- $E : \mathbb{C}^{N_h} \to \mathbb{R}^{N_h}$ is the embedding that only keeps the real part of the input. This can also be seen as a reshape layer with a truncation at the end. Since f, and thus $\tilde{f}$, are real valued, so are $S_m \tilde{f}$ and $\mathcal{S}_m(Tf)$. Therefore, we are not losing any information.
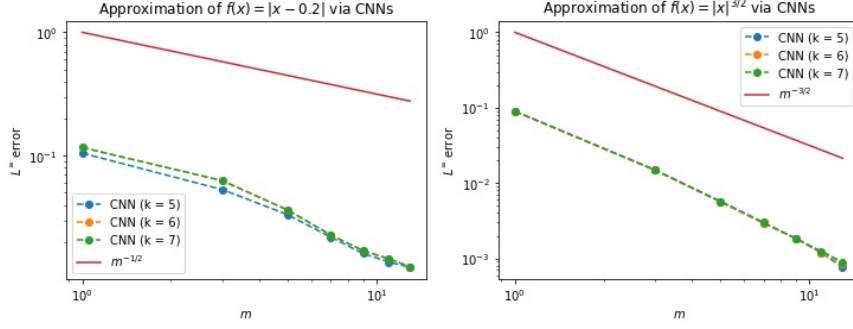
Figure 1: Numerical validation of the upper bounds in Theorem 1. The two panels show the results obtained for signals of different smoothness, respectively $H^1(0,1)$ on the left and $H^2(0,1)$ on the right. The results are reported for different grid resolutions, the mesh stepsize being $h = 2^{-k}$.

Finally, let $j \in \{1, \ldots, N_h\}$. We have

$$|f(x_j) - \Psi_j(Tf)|$$
$$= |f(x_j) - \mathcal{S}_m(Tf)_{2j-1}|$$
$$= |\tilde{f}(y_{2j-1}) - \mathcal{S}_m(Tf)_{2j-1}|$$
$$= \left| \tilde{f}(y_{2j-1}) - \left(S_m \tilde{f} \right)(y_{2j-1}) \right|.$$

Thus, by putting together (**) and (*) we get

$$|f(x_j) - \Psi_j(Tf)| \leq \cdots \leq Cm^{1/2-s} ||\tilde{f}||_{H^s(\Omega)}$$
$$\leq Cm^{1/2-s} ||f||_{H^s(\Omega)}.$$

$\square$

We remark that, as for the results in the previous Section, the proof is constructive. The pictures in Figure 1 show the approximation rates obtained by the actual implementation of Ψ (and T) along the lines detailed in the proof. The code used to obtain these results – as well as all the other results reported in the paper – is written in Python3 by exploiting the Pytorch library for CNNs, and it is available upon request. Note that we do not train the network, as we directly initialize Ψ with the wished weights and biases. The left panel in Figure 1 shows the results obtained for a mildly smooth signal, $f(x) = |x - 1/5|$. In this case we have $s = 1$, and the L^∞ error between the desired output, f, and the CNN approximation, is shown to decay at the expected rate, that is $1/\sqrt{m}$. This is also true regardless of the grid resolution, coherently with Theorem 1. Indeed, we obtained nearly the same results for $N_h = 33, 65, 129$. Finally, the right panel in Figure 1 refers to a smoother case, $f(x) = |x|^{3/2}$, where $s = 2$. Here we can remark, once again, the expected behavior.

4 Learning operators in a parametrized setting

We now extend the results in Section 3 to the parameter dependent setting. Let $\Theta \subset \mathbb{R}^p$ be some compact parameter space. We are interested in the approximation of an operator

$$\Theta \ni \boldsymbol{\mu} \to u_{\boldsymbol{\mu}} \in H^s(\Omega).$$

This framework is typically encountered in the case of parameter dependent PDEs, where each value of the input parameter vector $\boldsymbol{\mu} \in \Theta$ usually defines a different PDE solution $u_{\boldsymbol{\mu}}$. In this case, the approximation of the so-called parameter-to-solution map $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$ is of remarkable importance, especially when it comes to expensive many-query routines. For the interested reader, we refer to the general literature on Reduced Order Modeling for PDEs Hesthaven, Rozza, and Stamm (2016); Quarteroni, Manzoni, and Negri (2016) and to more recent contributions on the use of DNNs for the nonintrusive construction of efficient reduced order models in this context Franco et al. (2021); Fresca et al. (2021); Lee and Carlberg (2020).

Here, we aim at characterizing the approximation of such an operator in terms of convolutional neural networks. In particular, we seek for some DNN $\Phi : \mathbb{R}^p \to \mathbb{R}^{N_h}$ such that

$$\Phi_i(\boldsymbol{\mu}) \approx u_{\boldsymbol{\mu}}(x_i),$$

where the nodes $\{x_i\}_i^{N_h} \subset \overline{\Omega}$ come from a given discretization of the domain. We build Φ by considering an architecture that is made of two blocks, ϕ and Ψ. The former consists of dense layers, and it has the purpose of pre-processing the input. The latter is instead of convolutional type, and it is used to provide the desired output. We design Ψ along the lines of Theorem 1, thanks to which we are able to characterize the approximation error in terms of the network architecture as a whole.

In particular, we show that: (i) the depth of the dense block depends on the desired accuracy, while the number of convolutional layers only depends on the chosen discretization, (ii) fewer channels in the convolutional layers are required to approximate operators that have highly regular outputs, (iii) the width of the dense layers depends on the smoothness of the operator. We formalize these statements in the Theorem below.

Theorem 2 *Let $\Omega := (0,1)$ and let $\{x_j\}_{j=1}^{N_h} \subset \Omega$ be a uniform grid with step size $h = 2^{-k}$. We are given a (nonlinear) operator $\Theta \subset \mathbb{R}^p \ni \boldsymbol{\mu} \to u_{\boldsymbol{\mu}} \in H^s(\Omega)$, where Θ is a compact domain and $s \geq 1$. For some $r \geq 0$, assume that the operator is r-times Fréchet differentiable and that its r-derivative is Lipschitz continuous. Then, for any $0 < \varepsilon < 1$, there exists a Deep Neural Network $\Phi : \mathbb{R}^p \to \mathbb{R}^{N_h}$ such that*

$$|u_{\boldsymbol{\mu}}(x_j) - \Phi_j(\boldsymbol{\mu})| < \varepsilon$$

uniformly for all $\boldsymbol{\mu} \in \Theta$ and all $j = 1, \ldots, N_h$. Additionally, Φ can be defined to have at most

- *i) $C \log(1/\varepsilon)$ dense layers, with ReLU activation, and $C \log(1/h)$ convolutional layers,*

- *ii) $C\varepsilon^{-2/(2s-1)} \left[\varepsilon^{-p/(r+1)} \log(1/\varepsilon) + \log(1/h) \right]$ active weights,*

- *iii) $C\varepsilon^{-2/(2s-1)}$ channels in input and output,*

where $C > 0$ is some constant dependent on Θ and on the operator $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$, thus also on s, r, p.

Proof. Let $\varepsilon > 0$ and let $c = c(s) > 0$ be the constant in Theorem 1. We take advantage of the compactness of Θ and the continuity of the operator to define

$$M := \max_{\boldsymbol{\mu} \in \Theta} ||u_{\boldsymbol{\mu}}||_{H^s(\Omega)} < +\infty.$$

Let now Ψ be the CNN in Theorem 1, where we fix $m = \lceil (\varepsilon/2)^{-2/(2s-1)} Mc \rceil$. Then,

$$|u_{\boldsymbol{\mu}}(x_j) - \Psi_j(Tu_{\boldsymbol{\mu}})| < \varepsilon/2 \qquad (*)$$

for all $j = 1, \ldots, N_h$ and $\boldsymbol{\mu} \in \Theta$, where $T : H^s(\Omega) \to \mathbb{C}^{2m+1} \cong \mathbb{R}^{4m+2}$ is some continuous linear operator. We now note that, by composition, the map

$$\boldsymbol{\mu} \to Tu_{\boldsymbol{\mu}}$$

is an element of the Sobolev space $W^{r+1,\infty}(\Theta; \mathbb{R}^{4m+1})$.

In particular, by Theorem 1 in Yarotsky (2017), there exists a ReLU DNN $\phi : \mathbb{R}^p \to \mathbb{R}^{4m+2}$ with $C \log(1/\varepsilon)$ hidden layers and $Cm\varepsilon^{-p/(r+1)} \log(1/\varepsilon)$ active weights, such that

$$\sup_{\boldsymbol{\mu} \in \Theta} ||Tu_{\boldsymbol{\mu}} - \phi(\boldsymbol{\mu})||_1 < \varepsilon/2,$$

where $|| \cdot ||_1$ is the ℓ_1 norm over $\mathbb{R}^{4m+2}$, while $C > 0$ is a constant that depends on r, p, Θ, s and the operator $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$. The dependence on s comes from the Lipschitz constant of T, which may inflate the magnitude of the partial derivatives of $\boldsymbol{\mu} \to Tu_{\boldsymbol{\mu}}$.

Let now consider the composition $\Phi := \Psi \circ \phi$. It is easy to see that this DNN architecture satisfies the requirements claimed in the Theorem as soon as we replace the constant C with $\tilde{C} := Cc$. Also, for any $\boldsymbol{\mu} \in \Theta$ and $j = 1, \ldots, N_h$ we have the desired bound. In fact, by (*),

$$|u_{\boldsymbol{\mu}}(x_j) - \Phi_j(\boldsymbol{\mu})|$$
$$\leq |u_{\boldsymbol{\mu}}(x_j) - \Psi_j(Tu_{\boldsymbol{\mu}})| + |\Phi_j(\boldsymbol{\mu}) - \Psi_j(Tu_{\boldsymbol{\mu}})|$$
$$< \frac{\varepsilon}{2} + |\Psi_j(\phi(\boldsymbol{\mu})) - \Psi_j(Tu_{\boldsymbol{\mu}})|. \quad (**)$$

Now, we note that $|\Psi_j(\mathbf{a}) - \Psi_j(\mathbf{b})| \leq ||\mathbf{a} - \mathbf{b}||_1$. In fact, in Theorem 1, Ψ was defined as $E \circ R \circ \mathcal{S}_m$, where E and R were reshape layers, while $\mathcal{S}_m$ was as in Lemma 3. In particular,

$$|\Psi_j(\mathbf{a}) - \Psi_j(\mathbf{b})| \leq \sup_{x \in [0,1]} \left| \sum_{k=-m}^{m} a_k \mathrm{e}^{2\pi \mathrm{i}kx} - \sum_{k=-m}^{m} b_k \mathrm{e}^{2\pi \mathrm{i}kx} \right|$$
$$\leq \sum_{k=-m}^{m} |a_k - b_k|.$$

Therefore, relationship (**) finally yields

$$|u_{\boldsymbol{\mu}}(x_j) - \Phi_j(\boldsymbol{\mu})| \leq \cdots < \frac{\varepsilon}{2} + ||\phi(\boldsymbol{\mu}) - Tu_{\boldsymbol{\mu}}||_1 < \varepsilon.$$

$\square$

5 Numerical validation

We finally present some numerical experiments that confirm the decay rates predicted in Theorem 2. We proceed as follows. Once introduced the operator to be learned, we identify the smoothness indices s and r that appear in Theorem 2. Then, we fix a guess architecture $\Phi^{(1)}$ that serves as a starting point. Following the ideas of Theorem 2, we prescribe $\Phi^{(1)}$ as a DNN that is made by two blocks,

$$\Phi^{(1)} = \Psi^{(1)} \circ \phi^{(1)},$$

where $\phi^{(1)}$ is dense, while $\Psi^{(1)}$ is of convolutional type. More precisely:

- we fix $\Psi^{(1)}$ to be exactly the CNN in Theorem 1, where $m = m_1 \in \mathbb{N}$ is some initial guess for the dimensionality of the channels;

- we let $\phi^{(1)}$ have L_1 hidden layers of constant width w_1. We employ the leaky-ReLU activation at the hidden layers and no activation at the output. The weights and biases of $\phi^{(1)}$ are then initialized randomly accordingly to the approach introduced by He et al. in He, Zhang, Ren, and Sun (2015).

We train $\Phi^{(1)}$ over a training set $\{\boldsymbol{\mu}_i, u_{\boldsymbol{\mu}_i}\}_{i=1}^{N_{\text{train}}}$ in such a way that the loss function below is minimized

$$\mathcal{L}(\Phi^{(1)}) := \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left(h \sum_{j=1}^{N_h} |u_{\boldsymbol{\mu}_i}(x_j) - \Phi_j^{(1)}(\boldsymbol{\mu}_i)|^2 \right), \tag{2}$$

where $x_1, \ldots, x_{N_h}$ is some dyadic partition of $(0, 1)$ associated to a given grid resolution $h = 2^{-k}$. Note that, during this step, only the weights and biases of the dense block $\phi^{(1)}$ are actually optimized, as we keep $\Psi^{(1)}$ frozen. Although suboptimal, this will help us in removing possible drawbacks introduced by the difficulty of the optimization itself. Finally, we evaluate $\Phi^{(1)}$ over a test set of unseen instances $\{\boldsymbol{\mu}_i^{\text{test}}, u_{\boldsymbol{\mu}_i^{\text{test}}}\}_{i=1}^{N_{\text{test}}}$ in order to compute the empirical error given by

$$E(\Phi^{(1)}) = \max_{i,j} |u_{\boldsymbol{\mu}_i^{\text{test}}}(x_j) - \Phi_j^{(1)}(\boldsymbol{\mu}_i^{\text{test}})|. \tag{3}$$

Then, we exploit Theorem 2 in an attempt to define a second architecture, $\Phi^{(2)}$, that can be twice as accurate by halving the error over the testing set, that is by requiring that $E(\Phi^{(2)}) \approx E(\Phi^{(1)})/2$. We do this as follows:

- we update the number of channels according to (iii) in Theorem 2. In particular, up to rounding operations, we let

$$m_2 := 2^{2/(2s-1)} m_1;$$

- we increase the number of weights per dense layer coherently with (ii) in Theorem 2, that is

$$w_2 := 2^{p/(r+1)+2/(2s-1)} w_1;$$

- as suggested by (i) in Theorem 2, we also increase the number of dense hidden layers. In principle, the depth of the dense block should be increased by a constant factor $C \log(2)$. In practice, we let

$$L_2 = L_1 + l,$$

where l is either 1 or 2. This is to ensure that the obtained architectures are still feasible to train, as very deep models may become hard and expensive to optimize.

We then train $\Phi^{(2)}$ and iterate the above steps to generate $\Phi^{(3)}$, so that

$$E(\Phi^{(j)}) \propto 2^{-j}.$$

We highlight that, according to Theorem 2, this procedure should be robust with respect to the space discretization. In other words, we expect to obtain similar results regardless of the number of grid points employed in the discretization. To assess whether this behavior is actually observed in practice, we repeat our analysis for different mesh step sizes $h = 2^{-k}$.
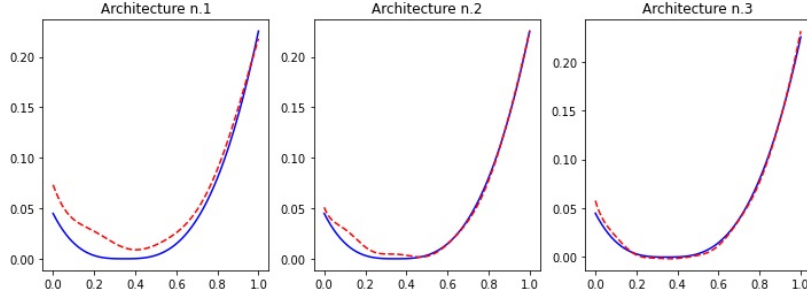
14

Figure 2: Benchmark example, Section 5.1. In blue, an instance $u_{\boldsymbol{\mu}}$ coming from the test set, here for $\boldsymbol{\mu} = [0.3462, 0.2935, 0.0818]$. The red dashed lines report the approximations proposed by the three DNNs, respectively $\Phi^{(1)}, \Phi^{(2)}, \Phi^{(3)}$. Grid resolution is $h = 2^{-7}$.

5.1 Benchmark example

To start, we consider the approximation of an operator that is defined analytically. More precisely, let $\Theta = [0,1] \times [0,1] \times [1,2] \subset \mathbb{R}^3$. For any fixed $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3) \in \Theta$ let

$$u_{\boldsymbol{\mu}}(x) = \mu_3 |x - \mu_1|^3 \mathrm{e}^{-\mu_2 x}.$$

We are interested in learning the map $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$. To this end, we note that $\{u_{\boldsymbol{\mu}}\}_{\boldsymbol{\mu} \in \Theta} \subset H^3(\Omega) \setminus H^4(\Omega)$. Also, the operator is at most twice differentiable with respect to $\boldsymbol{\mu}$, as its third derivative becomes discontinuous. According to the notation in Theorem 2, this results in $s = 3$ and $r = 2$.

For the space discretization, we consider three different mesh resolutions, $h = 2^{-5}, 2^{-6}, 2^{-7}$, corresponding respectively to $N_h = 33, 65, 129$ grid points. We train the networks by minimizing (2) via the so-called L-BFGS optimizer, where the training set consists of 500 randomly sampled parameters instances. We do not use batching strategies and we set the learning rate to its default value of 1. To avoid possible biases introduced by the optimization, we initialize and train each architecture multiple times (here, five), only to keep the best out of all the training sessions. This is a common practice known as *ensemble training*. For our starting architecture, $\Phi^{(1)}$, we set

$$m_1 = 5, \ w_1 = 4, \ L_1 = 3.$$

In this case, we have $2^{2/(2s-1)} \approx 1.32$ and $2^{p/(r+1)+2/(2s-1)} \approx 2.64$, since $p = 3$. In particular, our strategy for enriching the architectures can be stated as follows: to obtain a model that is twice as accurate, we increase the number of channels in the convolutional layers by nearly 30%, while we double (actually almost triple) the number of neurons in the dense layers. The theory also suggests to increase the depth of the dense block by some constant factor l. Here, we let $l = 2$.

Results are in Table 1, Figures 2 and 3. The first picture compares the output of the three architectures with that of the operator, for an unseen value of the input parameter $\boldsymbol{\mu}$. The quality of the approximation clearly increases as we consider richer and richer models. We also note that the regions with lower regularity are the most difficult to capture, coherently with what we expected. Figure 3, instead, reports the errors $E(\Phi^{(j)})$ in comparison with the expected decay rate 2^{-j}. We see that the numerical results perfectly match the theory, regardless of the grid resolution that is considered.
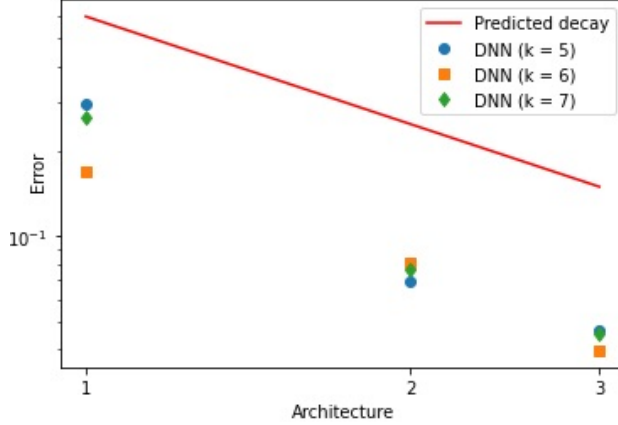
Figure 3: Numerical validation of Theorem 2 for the Benchmark example, Section 5.1. Axis are in loglog scale. The red line corresponds to the predicted decay rate, 2^{-j}, while the markers refer to the DNN models. Different markers correspond to different grid resolutions. The errors $E(\Phi^{(j)})$ obtained for the architectures $j = 1, 2, 3$, are computed as in Equation (3).

Model	m_j	w_j	L_j	Active weights	$E(\Phi^{(j)})$
$\Phi^{(1)}$	5	3	4	9'719	0.171
$\Phi^{(2)}$	6	5	10	12'147	0.081
$\Phi^{(3)}$	8	7	27	20'301	0.040

Table 1: Architectures and corresponding errors for the Benchmark example, Section 5.1. Results are reported limitedly to the case of grid resolution $h = 2^{-6}$. The hyperparameters read as in Section 5, that is: m_j = maximum number of convolutional features in the CNN block, upto a multiplicative constant; w_j = number of neurons per dense layer; L_j = depth of the dense block. The errors $E(\Phi^{(j)})$ are computed as in Equation (3).

5.2 Application to a parametrized time-dependent nonlinear PDE

We now consider a benchmark consisting of a one-dimensional coupled PDE-ODE nonlinear system

$$
\begin{cases}
\mu \dfrac{\partial u_{\boldsymbol{\mu}}}{\partial t} - \mu^2 \dfrac{\partial^2 u_{\boldsymbol{\mu}}}{\partial x^2} + R(u_{\boldsymbol{\mu}}) + w_{\boldsymbol{\mu}} = 0, & (x, t) \in \Omega \times (0, T) \\[2mm]
\dfrac{dw_{\boldsymbol{\mu}}}{dt} + (2w_{\boldsymbol{\mu}} - 0.5u_{\boldsymbol{\mu}}) = 0, & (x, t) \in \Omega \times (0, T) \\[2mm]
\dfrac{\partial u_{\boldsymbol{\mu}}}{\partial x}(0, t) = 50000t^3 e^{-15t}, & t \in (0, T) \\[2mm]
\dfrac{\partial u_{\boldsymbol{\mu}}}{\partial x}(1, t) = 0, & t \in (0, T) \\[2mm]
u_{\boldsymbol{\mu}}(x, 0) = 0, \ w_{\boldsymbol{\mu}}(x, 0) = 0, & x \in \Omega,
\end{cases}
\tag{4}
$$

where $R(u_{\boldsymbol{\mu}}) := u_{\boldsymbol{\mu}}(u_{\boldsymbol{\mu}} - 0.1)(u_{\boldsymbol{\mu}} - 1)$, while $\Omega = (0, 1)$ and $T = 2$. The above consists in a parametrized version of the monodomain equation coupled with the FitzHugh-Nagumo cellular model, describing the excitation-relaxation of the cell

membrane in the cardiac tissue FitzHugh (1961); Nagumo, Arimoto, and Yoshizawa (1962). System (4) has been discretized in space through linear finite elements, by considering $N_h = 2^k$, with $k \in \mathbb{N}$, grid points, and using a one-step, semi-implicit, first order scheme for time discretization with time-step $\Delta t = 5 \times 10^{-3}$; see, e.g., Pagani, Manzoni, and Quarteroni (2018) for further details. The solution of the former problem consists in a parameter-dependent traveling wave, which exhibits sharper and sharper fronts as the parameter μ gets smaller. The numerical transmembrane potential solution $u_{\boldsymbol{\mu}}$ represent the ground truth data in the experiments reported in the following.

Here, we consider the map $(\mu, t) \to u_{\boldsymbol{\mu}}(\cdot, t)$ as our operator of interest. In particular, the two dimensional vector parameter $\boldsymbol{\mu} := (\mu, t)$ consists of the scalar parameter μ and the time variable t. We let $\boldsymbol{\mu}$ vary in the (time-extended) parameter space $\Theta := \Theta_0 \times [0, T]$, where $\Theta_0 := 5 \cdot [10^{-3}, 10^{-2}]$.

In this case, it is not straightforward to identify the smoothness indices s and r. The numerical simulations show that the solutions $u_{\boldsymbol{\mu}}$ to (4) tend to have sharp gradients for certain values of the scalar parameter μ. In light of this, we let $s = 1$; if the solutions are actually smoother, then we expect the errors to decay faster than the predicted rate. Conversely, we make the assumption that $r = +\infty$, i.e. that the parameter-to-solution map is infinitely differentiable. We remark that the constant C appearing in Theorem 2 actually depends on r. To this end, we make the further assumption that $C = C(r)$ is bounded with respect to r.

As a starting point, we consider the following structural hyperparameters

$$m_1 = 1, \ w_1 = 3, \ L_1 = 3,$$

to build our reference architecture $\Phi^{(1)}$. In this case, we have $2^{p/(r+1)+2/(2s-1)} = 2^{2/(2s-1)} = 4$. In particular, Theorem 2 suggests to quadruplicate the number channels and neurons per dense layer in order to half the test error. We also increase the depth of the dense block by a constant factor of $l = 1$ when moving from an architecture to a more complex one. In this case, we do not assess the model performance for varying resolution levels as we stick to the same grid employed by the Finite Element solver. Instead, in order to collect more data, we repeat the same analysis for a different guess architecture, namely

$$m_1 = 1, \ w_1 = 2, \ L_1 = 4.$$

To collect the training and test sets, we proceed as follows. We sample $N_{train} = 20$ equally spaced values for the scalar parameter $\mu \in \Theta_0$, and we consider their midpoints to obtain $N_{test} = 19$ test instances. For each $\mu \in \Theta_0$ fixed, we then extract uniformly $N_t = 25$ time snapshots from the global trajectory defined over the interval $[0, T]$. Once again, we train the DNN models using the L-BFGS optimizer (no batching, learning rate = 1).

Results are reported in Figures 4 and 5. As for the benchmark example, we see that the DNN models become more and more expressive as we move from $\Phi^{(1)}$ to $\Phi^{(3)}$. The architectures mostly struggle in capturing flat regions, which is understandable as these entail discontinuities in the higher-derivatives (cf. Figure 5). Finally, the error trend, reported in Figure 4, is in agreement with the estimates presented in Theorem 2 regardless of the initial guess for the architecture. Note, once again, that here we only consider one resolution level, as we employ the same step size h adopted by the Finite Element solver.

Since we included time as an additional parameter, the plots in Figure 5 fix both the scalar parameter μ and the time instant t. However, we recall that according to Equation (3) the model was evaluated in terms of worst-case errors. In particular,
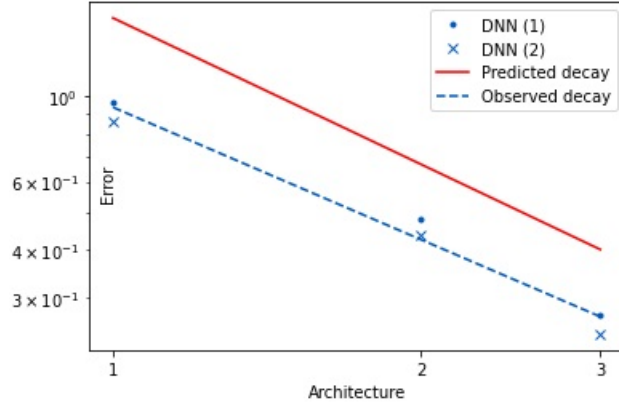
Figure 4: Numerical validation of Theorem 2 for the coupled FitzHugh-Nagumo problem, Section 5.2. Axis read in loglog scale. The red line corresponds to the predicted decay rate, 2^{-J}, while the blue markers report the errors $E(\Phi^{(j)})$ obtained for the architectures $j = 1, 2, 3$. Different markers correspond to different choices of the initial guess architecture $\Phi^{(1)}$, respectively. Errors are computed accordingly to Equation 3.

Model	m_j	w_j	L_j	Active weights	$E(\Phi^{(j)})$
$\Phi^{(1)}$	1	2	4	3'131	0.859
$\Phi^{(2)}$	4	3	16	10'305	0.435
$\Phi^{(3)}$	16	4	64	54'553	0.241

Table 2: Architectures and corresponding errors for the coupled FitzHugh-Nagumo problem, Section 5.2. Results are reported limitedly to one of the initial guess architectures. Hyperparameters read as in Section 5 and Table 1. Errors are computed as in Equation (3).
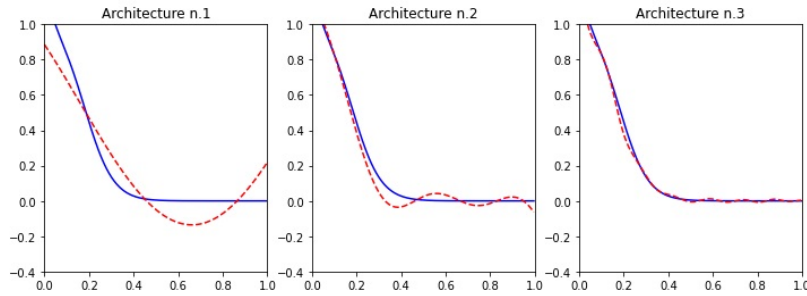

Figure 5: Learning the parameter-to-solution operator of a parametrized time-dependent nonlinear PDE, Section 5.2. In blue, a snapshot $u_{\boldsymbol{\mu}}(\cdot, t)$ coming from the test set, here for $\mu = 0.0488$ and $t = 1.905$. The red dashed lines correspond to the approximations proposed by the three DNN models, respectively $\Phi^{(1)}, \Phi^{(2)}, \Phi^{(3)}$.
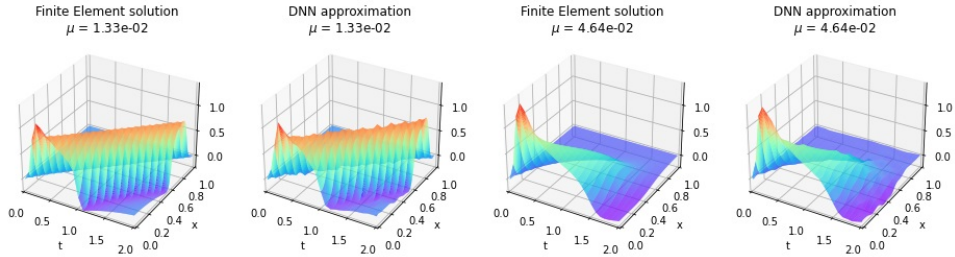
18

Figure 6: Learning the parameter-to-solution operator of a parametrized time-dependent nonlinear PDE, Section 5.2. Comparison between Finite Element solutions and DNN approximations for different μ. The first and the third plot report the spacetime surface $[0, T] \times \Omega \to \mathbb{R}$ representing the Finite Element simulation, thus $(t, x) \to u_\mu(x, t)$. Conversely, the second and the fourth picture show the corresponding DNN approximation over the same spatial grid, $(t, x_j) \to \Phi_j(t, \mu)$. Here, Φ is constructed considering the third architecture generated during the iterative augmentation process in Section 5, starting from the second guess architecture in Section 5.2.

the quality of the approximation is guaranteed over the whole time interval $[0, T]$ and for any choice of the scalar parameter $\mu \in \Theta_0$. Figure 6, shows the overall dynamics of the solution for two different choices of μ, with a comparison between Finite Element solutions and DNN approximations. Despite containing a few numerical artifacts, we see that the DNN model fully captures the general behavior of the solutions, both in the hyperbolic and diffusive case ($\mu = 1.33 \cdot 10^{-2}$ and $\mu = 4.64 \cdot 10^{-2}$ respectively). Of note, the spurious oscillations in the DNN approximation are in perfect agreement with the errors reported in Table 2. Accordingly to Theorem 2, these can be removed by considering larger architectures and, possibly, more training data.

6 Conclusions

In this paper, we have established and verified theoretical error bounds for the approximation of nonlinear operators by means of CNNs. Our results shed a light on the role played by convolutional layers and their hyperparameters, such as input-output channels, depth and others. In particular, they show how operator learning problems can be decoupled in two parts: on the one hand, the difficulty in characterizing the dependence with respect to the input parameters; on the other hand, the issue in having to reconstruct complex space-dependent outputs. The presented research is original and timely. Indeed, at the best of our knowledge, all the available results on DNNs and operator learning do not address the peculiar properties of CNNs, instead they consider classic fully connected architectures. Conversely, those works that focus on CNN models are typically not framed in the context of operator learning.

Our analysis is limited to the 1-dimensional case, $d = 1$, that is when the output of the operator are functions defined over an interval. However, we note that the main ideas underlying our proofs can be extended to higher dimensions with little effort. The critical points are Lemmas 1, 3 and Theorem 1. For the first two results, one needs to define suitable convolutional layers that are able to advance along different dimensions separately, which can be carried out via 2D and 3D convolutions whenever $d = 2, 3$. Conversely, Theorem 1 has to be adapted in a proper way, since it becomes trickier to turn generic maps $f : [0, 1]^d \to \mathbb{R}$ onto periodic functions. Furthermore, as the spatial dimension d plays an important role

in Sobolev inequalities, it may be convenient to replace the original output space $H^s(\Omega)$ with other functional spaces, such as $W^{s,\infty}(\Omega)$ or $\mathcal{C}^s(\overline{\Omega})$, when addressing the case $d > 1$.

Nevertheless, we believe that our results motivate the recent success of CNNs, especially in areas such as Reduced Order Modeling of PDEs. This is because, as shown in Theorem 2, smooth outputs are those that are better approximated by CNNs. Solutions to partial differential equations often enjoy regularity properties that make them an appealing area of application for the proposed analysis. This further promotes the practical use of CNNs as well as their theoretical study from a purely mathematical point of view.

Fundings

References

A. Dosovitskiy, J. Tobias Springenberg, T. Brox, Learning to generate chairs with convolutional neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1538–1546.

F. Sultana, A. Sufian, P. Dutta, Evolution of image segmentation using deep convolutional neural network: a survey, Knowledge-Based Systems 201 (2020) 106062.

N. R. Franco, A. Manzoni, P. Zunino, A deep learning approach to reduced order modelling of parameter dependent partial differential equations, arXiv preprint arXiv:2103.06183 (2021).

S. Fresca, L. Dede, A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes, Journal of Scientific Computing 87 (2021) 1–36.

S. Fresca, A. Manzoni, POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition, Computer Methods in Applied Mechanics and Engineering 388 (2022).

K. Lee, K. T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, Journal of Computational Physics 404 (2020) 108973.

N. T. Mücke, S. M. Bohté, C. W. Oosterlee, Reduced order modeling for parameterized time-dependent pdes using spatially and memory aware deep learning, Journal of Computational Science 53 (2021) 101408.

G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (1989) 303–314.

D. Yarotsky, Error bounds for approximations with deep relu networks, Neural Networks 94 (2017) 103–114.

I. Gühring, G. Kutyniok, P. Petersen, Error bounds for approximations with deep relu neural networks in w s, p norms, Analysis and Applications 18 (2020) 803–859.

I. Gühring, M. Raslan, Approximation rates for neural networks with encodable weights in smoothness spaces, Neural Networks 134 (2021) 107–130.

J. W. Siegel, J. Xu, High-order approximation rates for shallow neural networks with cosine and reluk activation functions, Applied and Computational Harmonic Analysis 58 (2022) 1–26.

D.-X. Zhou, Universality of deep convolutional neural networks, Applied and computational harmonic analysis 48 (2020) 787–794.

J. He, L. Li, J. Xu, Approximation properties of deep relu cnns, arXiv preprint arXiv:2109.00190 (2021).

N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces, arXiv preprint arXiv:2108.08481 (2021).

L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (2021) 218–229.

G. Kutyniok, P. Petersen, M. Raslan, R. Schneider, A theoretical analysis of deep neural networks and parametric pdes, Constructive Approximation (2021) 1–53.

N. Kovachki, S. Lanthaler, S. Mishra, On universal approximation and error bounds for fourier neural operators, Journal of Machine Learning Research 22 (2021) Art–No.

S. Lanthaler, S. Mishra, G. E. Karniadakis, Error estimates for deeponets: A deep learning framework in infinite dimensions, Transactions of Mathematics and Its Applications 6 (2022) tnac001.

C. Schwab, J. Zech, Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in uq, Analysis and Applications 17 (2019) 19–55.

K. Bhattacharya, B. Hosseini, N. B. Kovachki, A. M. Stuart, Model reduction and neural networks for parametric pdes, arXiv preprint arXiv:2005.03180 (2020).

P. Petersen, F. Voigtlaender, Equivalence of approximation by convolutional neural networks and fully-connected networks, Proceedings of the American Mathematical Society 148 (2020) 1567–1581.

Y. Katznelson, An Introduction To Harmonic Analysis, Dover, 1976.

A. Spitzbart, A generalization of hermite's interpolation formula, The American Mathematical Monthly 67 (1960) 42–46.

J. Hesthaven, G. Rozza, B. Stamm, Certified Reduced Basis Methods for Parametrized Partial Differential Equations, Springer Briefs in Mathematics, 1 ed., Springer International Publishing, 2016.

A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations: An Introduction, Springer International Publishing, 2016.

K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, In Proceedings of the IEEE international conference on computer vision (2015) 1026–1034.

R. FitzHugh, Impulses and physiological states in theoretical models of nerve membrane, Biophysical Journal 1 (1961) 455–466.

J. Nagumo, S. Arimoto, S. Yoshizawa, An active pulse transmission line simulating nerve axon, Proceedings of the IRE 50 (1962) 2061–2070.

S. Pagani, A. Manzoni, A. Quarteroni, Numerical approximation of parametrized problems in cardiac electrophysiology by a local reduced basis method, Computer Methods in Applied Mechanics and Engineering 340 (2018) 530–558.

Appendix

We report, in mathematical terms, the formal definition of convolutional layers and CNNs. These correspond to the ones adopted in the literature and reported within the Pytorch documentation. For tensor objects we use the following notation.

Given $\mathbf{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, we write $\mathbf{A}_{i_1, \ldots, i_p}$ for the subtensor in $\mathbb{R}^{n_{p+1} \times \cdots \times n_d}$ obtained by fixing the first p dimensions along the specified axis, where $1 \le i_j \le n_j$. We also adopt the usual abuse of notation for which scalar-valued activation functions operate componentwise on vectors, that is

$$\rho([x_1, \ldots, x_l]) := [\rho(x_1), \ldots, \rho(x_l)]$$

whenever $\rho : \mathbb{R} \to \mathbb{R}$.

Definition 1 *Let $m, m's, t, d$ be positive integers and let g be a common divisor of m and m'. A 1D Convolutional layer with m input channels, m' output channels, grouping number g, kernel size s, stride t, dilation factor d and activation function $\rho : \mathbb{R} \to \mathbb{R}$, is a map of the form*

$$\Phi : \mathbb{R}^{m \times n} \to \mathbb{R}^{m' \times \left\lfloor \frac{n - d(s-1) - 1}{t} + 1 \right\rfloor}$$

whose action on a given input $\mathbf{X} \in \mathbb{R}^{m \times n}$ is defined as

$$\Phi(\mathbf{X})_{k'} = \rho \left(\sum_k \mathbf{W}_{k', k} \otimes_{t,d} \mathbf{X}_k + \mathbf{B}_{k'} \right),$$

where $1 \le k' \le m'$, while the sum index k runs as below,

$$k = \lfloor g(k'-1)/m \rfloor m/g + 1, \ldots, (\lfloor g(k'-1)/m \rfloor + 1) \, m/g.$$

Here,

- $\mathbf{W} \in \mathbb{R}^{m' \times (m/g) \times s}$ *is the weight tensor*

- $\otimes_{t,d}$ *is the cross-correlation operator with stride t and dilation d. That is, for any $\mathbf{w} \in \mathbb{R}^s$ and $\mathbf{x} \in \mathbb{R}^n$ one has*

  $$\mathbf{w} \otimes_{t,d} \mathbf{x} \in \mathbb{R}^{\left\lfloor \frac{n - d(s-1) - 1}{t} + 1 \right\rfloor},$$

  where

  $$\left(\mathbf{w} \otimes_{t,d} \mathbf{x} \right)_j := \sum_{i=1}^{s} w_i x_{(j-1)t + (i-1)d + 1}.$$

- $\mathbf{B} \in \mathbb{R}^{m' \times \left\lfloor \frac{n - d(s-1) - 1}{t} + 1 \right\rfloor}$ *is the bias term.*

The default values for stride and dilation are $t = 1$, $d = 1$. For this reason, with little abuse of notation, one says that Φ has no stride and no dilation to intend that $t = 1$, $d = 1$. Similarly, we assume $g = 1$ whenever the grouping number is not declared explicitly.

Definition 2 *Let $m, m's, t, d$ be positive integers and let g be a common divisor of m and m'. A 1D Transposed Convolutional layer with m input channels, m' output channels, grouping number g, kernel size s, stride t, dilation factor d and activation function $\rho : \mathbb{R} \to \mathbb{R}$, is a map of the form*

$$\Phi : \mathbb{R}^{m \times n} \to \mathbb{R}^{m' \times (n-1)t + d(s-1) + 1}$$

whose action on a given input $\mathbf{X} \in \mathbb{R}^{m \times n}$ is defined as

$$\Phi(\mathbf{X})_{k'} = \rho \left(\sum_k \mathbf{W}_{k,k'} \otimes_{t,d}^{\top} \mathbf{X}_k + \mathbf{B}_{k'} \right),$$

where $1 \leq k' \leq m'$, while the sum index k runs as below,

$$k = \lfloor g(k'-1)/m \rfloor m/g + 1, \ldots, (\lfloor g(k'-1)/m \rfloor + 1) \, m/g.$$

Here,

- $\mathbf{W} \in \mathbb{R}^{(m/g) \times m' \times s}$ *is the weight tensor*

- $\otimes_{t,d}^{\top}$ *is the transposed cross-correlation operator with stride t and dilation d. That is, for any $\mathbf{w} \in \mathbb{R}^s$ and $\mathbf{x} \in \mathbb{R}^n$ one has*

$$\mathbf{w} \otimes_{t,d}^{\top} \mathbf{x} \in \mathbb{R}^{(n-1)t - d(s-1) + 1},$$

  where

$$\left(\mathbf{w} \otimes_{t,d}^{\top} \mathbf{x} \right)_j := \sum_i w_{\lfloor \frac{(i-1)t+1-j}{d} \rfloor + 1} x_i,$$

  the sum index i running as below,

$$i = \left\lfloor \frac{j-1}{t} + 1 \right\rfloor, \ldots, \left\lfloor \frac{(s-1)d + j - 1}{t} + 1 \right\rfloor.$$

- $\mathbf{B} \in \mathbb{R}^{m' \times (n-1)t - d(s-1) + 1}$ *is the bias term.*

Definition 3 *Let $\rho : \mathbb{R} \to \mathbb{R}$. A dense layer with activation function ρ is a map $\Phi : \mathbb{R}^n \to \mathbb{R}^{n'}$ of the form*

$$\Phi(\mathbf{x}) = \rho \left(\mathbf{W} \mathbf{x} + \mathbf{b} \right)$$

where $\mathbf{W} \in \mathbb{R}^{n' \times n}$ and $\mathbf{b} \in \mathbb{R}^{n'}$ are respectively the weight matrix and the bias vector.

Definition 4 *A Convolutional Neural Network (CNN) is any map that, up to re-shaping operations, can be written as the composition of (transposed) convolutional layers. Conversely, a Deep Neural Network (DNN) is any map that, up to reshaping operations, can be written as the composition of dense layers.*

Since (transposed) convolutional layers can be seen as a particular class of dense layers, every CNN is a DNN. Similarly, CNNs and DNNs can be easily composed to build more complex DNN models.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

44/2022   Peli, R.; Dovera, L.; Fighera, G.; Menafoglio, A.; Secchi, P.
Forecasting Oil Production Rates in Primary Depletion using the Physics-based Residual Kriging functional approach

43/2022   Zappon E.; Manzoni A.; Gervasio P.; Quarteroni A.
A reduced order model for domain decompositions with non-conforming interfaces

41/2022   Arnone, A.; Ferraccioli, F.; Pigolotti, C.; Sangalli, L.M.
A roughness penalty approach to estimate densities over two-dimensional manifolds

40/2022   Fumagalli, A.; Patacchini, F. S.
Well-posedness and variational numerical scheme for an adaptive model in highly heterogeneous porous media

42/2022   Gatti, F.; Fois, M.; de Falco, C.; Perotto, S.; Formaggia, L.
Parallel simulations for fast-moving landslides: space-time mesh adaptation and sharp tracking of the wetting front

39/2022   Ferro, N.; Perotto, S.; Gavazzoni, M.
A new fluid-based strategy for the connection of non-matching lattice materials

37/2022   Boon, W. M.; Fumagalli, A.
A Reduced Basis Method for Darcy flow systems that ensures local mass conservation by using exact discrete complexes

38/2022   Burzacchi, A.; Landrò, M.; Vantini, S.
Object-oriented Classification of Road Pavement Type in Greater Maputo from Satellite Images

35/2022   Perotto, S.; Bellini, G.; Ballarin, F.; Calò, K.; Mazzi, V.; Morbiducci, U.
Isogeometric hierarchical model reduction for advection-diffusion process simulation in microchannels

36/2022   Vaccaro, F.; Brivio, S.; Perotto, S.; Mauri, A.G.; Spiga, S.
Physics-based Compact Modelling of the Analog Dynamics of HfOx Resistive Memories