MKNO: Multi-Kernel Neural Operator

Kiernan McGuigan[1]
K. Andrea Scott[2], Sirisha Rambhatla[3], Coauthor Coauthor[1],
[1]Vision and Image Processing Group, Systems Design Engineering, University of Waterloo
[2]Mechanical and Mechatronics Engineering, University of Waterloo
[3]Management Science and Engineering, University of Waterloo
{kmcguiga, ka3scott, sirisha.rambhatla}@uwaterloo.ca

Abstract

Neural operators learn resolution independent mappings between functional spaces, and are a popular way to generate solutions for an entire class of partial differential equations (PDE) as opposed to just one instance, leading to significant computational gains. However, these methods rely on a continuous-discrete equivalence between the functional form and the samples, which may be violated if the samples are not captured *faithfully*. We propose the multi-kernel neural operator (MKNO) which can capture different frequency components at varying levels of resolutions. MKNO accomplishes this by using the Fourier kernels to capture lower frequency global information and graph kernels to capture more local and high frequency feature information. MKNO is descretization invariant, and learns a general solution operator that can be applied to varying resolutions without retraining. To validate our architecture we apply MKNO to a number of different two dimensional PDEs and observe strong results.

1 Introduction

Partial Differential Equations (PDEs) are fundamental to various applications across science and engineering, including fluid dynamics [1, 2], aerodynamics [2], and mechanical systems [3]. Traditionally, these equations have been solved using numerical methods [4], however many applications rely on high resolution and fine grained solutions in which these methods can be slow and inefficient [5]. More recently, machine learning techniques have been leveraged for their ability to learn to approximate functions, allowing for the solving of these equations in a more efficient manner [6].

Early machine learning techniques included directly parameterizing a solution function as a neural network. This method, dubbed neural finite element method (neural FEM), is mesh invariant and has proven to be highly effective for solving PDEs [7, 8]. While effective, this method is designed to solve a singular instance of a PDE, rather than generalizing more broadly [5]. Learning one specific solution function means each new instance of the PDE requires retraining. Moreover, because this approach closely resembles traditional numerical methods for solving PDEs, the same computational scaling issues exist, limiting the general applicability of these approaches [9, 5].

Another group of methods involve parameterizing finite-dimensional operators [10, 11] as a neural network to map between finite dimensional domains. While these methods can be accurate, by operating between finite-dimensional domains they are inherently tied to a specific data resolution. This requires retraining for any different discritization of the domain, meaning the model would be unable to accurately make predictions on any mesh other than the one it was trained on.

More recently operator learning has been gaining popularity for its ability to model solution operators in a mesh free manner [12]. These methods aim to leverage some continuous representation of our input data to parameterize a neural network as an infinite dimensional operator [5]. This allows a single set of network parameters to generalize to arbitrary data resolutions. Since many real world processes are continuous in nature, this flexibility with respect to data resolution can be extremely advantageous for many applications. Another advantage of this flexibility is the ability to train at a lower resolution while still retaining the same accuracy at inference time on potentially higher resolution samples. Since training a model is typically more memory and computationally expensive that running inference, reducing the resolution of data can greatly increase our efficiency both memory-wise, computationally, and in the time to train.

To accomplish this a continuous functional representation is obtained from the discrete observations and a kernel function is used to map between functional spaces. Therefore, operator methods rely on a continuous-discrete equivalence [13, 14] which makes this representation possible [13]. Kernel functions then learn to map between infinite dimensional spaces. Some examples of these kernel functions include the graph kernel operator [9, 15, 16, 17] and the Fourier kernel operator [5, 18, 17, 19].

Graph kernel operators parameterize a graph neural network with nodes connected based on real world distance and edge information encapsulating distances and spatial locations. This establishes a constant visual informational radius, irrespective of the input data resolution [9]. In theory the flexibility of graphs should allow for any continuous function to be effectively modeled with enough message passing layers, however in practice graphs struggle to achieve this in an efficient manner. To limit computational complexity, the input graph is typically not fully connected, with edges only being placed between nodes within some real world distance threshold. This means that the graph will require more and more layers to effectively communicate global feature information around the entire domain [15, 17].

Alternatively, Fourier kernel operators leverage the fast Fourier transform (FFT) and multiply learned kernel weights in the frequency domain, acting as a fixed frequency global convolution [5]. While this results in a highly efficient method and is extremely effective at capturing global feature information, the Fourier representation struggles to capture higher frequency and localized feature information [19].

In this paper, we propose a multi-kernel neural operator architecture that leverages both graph and Fourier kernel operators to efficiently and effectively capture both high frequency local feature information as well as lower frequency global feature information simultaneously.

2 Proposed Methodology

2.1 Neural Operator

The goal of a neural operator is to map between infinite dimensional spaces based on a finite set of input-output observations. To define this concept we can begin with some bounded domain D in which we have real-valued functions A and U. There exists some mapping $G : A \rightarrow U$, which we will call the solution operator. Suppose we are given observations $\{a_j, u_j\}_{j=1}^{N}$ where $a_j \subset A, u_j \subset U$ our goal is to approximate the solution operator G with $G_\theta,\ \theta \in \Theta$, where Θ is some parameter space, to map from inputs to outputs.

$$G_\theta : A \rightarrow U,\ \theta \in \Theta \tag{1}$$

To optimize these parameters θ we can define a cost function C which we minimize.

$$\min_{\theta \in \Theta} \mathbb{E}_a[\, C(G_\theta(a),\ G(a)\,] \tag{2}$$

The approximate solution operator G_θ, which maps between functional spaces, is comprised of a number of iterative point-wise and kernel operator layers [5, 9].

$$G_\theta = Q \circ K_l \circ \cdots \circ K_0 \circ P \tag{3}$$

Where P is a point-wise transform used to map the input function $a \in A$ to some higher dimensional latent representation $\nu(0)$, thus $P : a \rightarrow \nu(0)$. Q is another point-wise transformation that maps latent space to the output dimensions $Q : \nu(l) \rightarrow u$. Between these we apply l kernel layers which aim to approximate the solution operator $K_l : \nu(l-1) \rightarrow \nu(l)$ using the kernel function κ_l within the domain $x, y \in D$.

$$\nu_l(x) = \int_D \kappa_l(x, y)\ \nu_{l-1}(y) \tag{4}$$

2.2 Graph Kernel Operator

The graph kernel uses a kernel function to transform the messages in the message passing layers of the graph neural network (GNNs) [9, 15]. To efficiently perform this operation, only nodes within a radius r are connected together. Therefore multiple layers are used to approximate a kernel capable of sharing information across the entire domain.

$$\nu_{l+1}(x) = \frac{1}{|N(x)|} \sum_{y \in N(x)} K_l(e(x, y) \cdot \nu(y)) \tag{5}$$

Where $N(x)$ is the set of nodes connected to query point x and $|N(x)|$ is the count. $e(x, y)$ is the edge connecting observations x and y and is commonly composed of domain indexes x, y as well as the input function values $a(x), a(y)$.

2.3 Fourier Kernel Operator

Fourier kernels [5], were motivated by the use of Fourier decomposition to solve PDEs. These also take advantage of the efficiency of the fast Fourier transform (FFT) and the ability to parameterize global spatial convolutions as multiplications in the spectral domain.

$$\nu_{l+1}(x) = \mathcal{F}^{-1}(\mathcal{F}(\kappa_\phi) \cdot \mathcal{F}(\nu_{l-1}))(x), \tag{6}$$

where $\mathcal{F}$ is the FFT, $\mathcal{F}^-1$ is the inverse FFT, and κ_ϕ is the convolutional kernel weights. In practice we learn weights R_ϕ directly in the Fourier domain.

2.4 Multi-Kernel Neural Operator

We propose a novel multi-branch kernel to efficiently handle both local and global feature information. To accomplish this we define a kernel that combines both the Fourier kernel and the graph kernel in parallel branches. We can define this mathematically as;

$$\mathcal{K}_{Fourier} = \mathcal{F}^{-1}(\mathcal{F}(\kappa^{(1)}_{\phi_l}) \cdot \mathcal{F}(\nu_{l-1}))(x) \tag{7}$$

$$\mathcal{K}_{graph} = \frac{1}{|N(x)|} \sum_{y \in N(x)} \kappa^{(2)}_l (e(x,y) \cdot \nu(y))(x) \tag{8}$$

$$\nu_{l+1}(x) = \mathcal{K}_{Fourier}(x) + \mathcal{K}_{graph}(x) \tag{9}$$

Where ν is the latent high dimensional representation, $\mathcal{F}, \mathcal{F}^{-1}$ is the Fourier transform and its inverse, κ^1_ϕ is the Fourier kernel weights, $N(x), |N(x)|$ are the connected nodes and its count, $e(x,y)$ is the edge information between connecting nodes, and K^2_l is the kernel weights for the graph branch.

By using both kernels in combination we aim to maintain a stronger continuous-discrete equivalence irrespective as to weather we have low frequency global feature information, higher frequency local feature information, or a combination of the two. This allows us to faithfully capture the continous-discrete equivalency for more equations and under more conditions.

As with most neural operators we use a lifting layer $P : a \rightarrow \nu_0$, a point-wise fully connected layer taking as input the last T solutions to the PDE and producing the high-dimensional latent representation $\nu_0(x)$. Following this we apply a number of MKNO blocks mapping between latent representations $MKNO_l : \nu_{l-1} \rightarrow \nu_l$. Finally, we apply a projection layer $Q(x) : \nu_l(x) \rightarrow u(x)$ mapping back down to the output dimensionality from the latent representation. Within the MKNO block, two kernels are applied in parallel with their results being summed before applying a non-linearity

3 Experiments

3.1 Metrics

To evaluate the results the relative errors are reported [5]. Relative error divides the sum of squares of residuals for each training sample by the sum of squares of the targets. This attempts to remove the impact of absolute value differences between different realizations of the PDE in our loss function. We report the relative error at the full and half resolutions. Error metrics are reported for both the full 64 by 64 resolution and the half 32 by 32 resolution that models were trained on.

3.2 Baselines

We compare our multi-kernel neural operator with a number of other popular operator based methods and non-operator based methods. We compare against Fourier neural operator [5] and Adaptive Fourier Neural Operator [20, 21] which leverage the Fourier kernel, and GNO [9] which leverages the graph kernel. We also present a GNN model inspired by the FNO paper [5] but with spatially invariant edge connectivity. Lastly we compare with a Conv LSTM [22] for our time based PDEs and a simple convolution network for our non-time based PDEs.

3.3 Equations

All our PDEs were generated with a resolution of 64 by 64. When training the resolution was reduced to 32 by 32. All results presented will be generated at the full (64 by 64) resolution or the half (32 by 32) resolution on a set of test data that was withheld during the training process.

3.3.1 Darcy Flow

Steady-state Darcy flow is described by the second order linear elliptic PDE,

$$-\nabla \cdot (a \nabla u) = f \tag{10}$$

with forcing term $f = 1$ and diffusion coefficient $a\ \psi\#\mu$ with μ being defined as a Gaussian Process with zero mean and squared exponential kernel [13].

Table 1: Relative Error for the Darcy Flow equation. Full resolution is 64 by 64 while half resolution is the 32 by 32 grid that models were trained on. All metrics are calculated on a test set of data that was withheld at training.

	Relative Error	
Model	Full Resolution	Half Resolution
Conv	0.2268	0.2593
FNO	0.1326	0.1304
AFNO	0.1527	0.1548
GNO	0.1555	0.1609
GNN	0.1516	0.1492
MKNO	0.1097	0.1010
MKNOeff	0.1242	0.1190

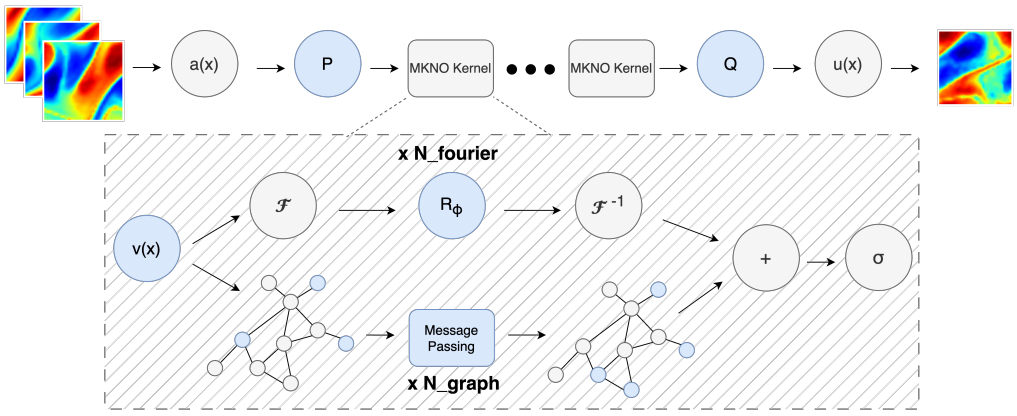

Figure 1: The Multi-Kernel Neural Operator diagram displays our methodology. We show both the Fourier branch and the graph branch used by our multi-kernel operator which are applied in parallel before their results are summed.

3.3.2 Poisson Equation

The linear elliptic PDE is described as,

$$-\Delta = f \tag{11}$$

with source term f being defined by a complex periodic function over the domain [13].

Table 2: Relative Error for the Poisson equation. Full resolution is 64 by 64 while half resolution is the 32 by 32 grid that models were trained on. All metrics are calculated on a test set of data that was withheld at training.

	Relative Error	
Model	Full Resolution	Half Resolution
Conv	0.1506	0.1554
FNO	0.0973	0.0974
AFNO	0.1509	0.1435
GNO	0.1684	0.1709
GNN	0.1609	0.1633
MKNO	0.1017	0.1031
MKNOeff	0.0861	0.0855

3.3.3 Wave Equation

A linear hyperbolic PDE described as,

$$u_{tt} - c^2\Delta u = 0, u_0(x, y) = f(x, y) \tag{12}$$

with constant propagation speed $c = 0.1$ initial conditions f [13].

3.3.4 Allen-Cahn Equation

The parabolic nonlinear PDE can be described as,

$$u_t = \Delta u - \epsilon^2 u(u^2 - 1) \tag{13}$$

with reaction rate $\epsilon = 220$ and initial conditions f [13].

Table 3: Relative Error for the Wave equation. Full resolution is 64 by 64 while half resolution is the 32 by 32 grid that models were trained on. All metrics are calculated on a test set of data that was withheld at training.

	Relative Error	
Model	Full Resolution	Half Resolution
Conv	0.1746	0.1561
FNO	0.0636	0.0606
AFNO	0.1603	0.1530
GNO	0.1752	0.1705
GNN	0.1808	0.1783
MKNO	0.0717	0.0688
MKNOeff	0.0659	0.0627

3.3.5 Navier-Stokes Equations

The incompressible fluid motion PDE is describe as,

$$u_t + (u \cdot \nabla)u + \nabla p = \nu\Delta u, divu = 0 \tag{14}$$

with viscosity $\nu = 1e^{-5}$ and time steps $t \in [1, 20]$ of which the first ten are input to the operators and the last ten are to be predicted.

4 Conclusion

In conclusion, in this paper we present a novel neural operator method MKNO which demonstrates a strong ability to faithfully capture both low frequency global feature information along with higher frequency more localized feature information simultaneously. This allows our method to be more robust for different PDEs with more complex dynamics, something that can be extremely applicable in different circumstances.

Table 4: Relative Error for the Allen-Cahn equation. Full resolution is 64 by 64 while half resolution is the 32 by 32 grid that models were trained on. All metrics are calculated on a test set of data that was withheld at training.

	Relative Error	
Model	Full Resolution	Half Resolution
Conv	0.4996	0.4849
FNO	0.2314	0.2270
AFNO	0.5116	0.4974
GNO	0.5185	0.5214
GNN	0.4900	0.4822
MKNO	0.2374	0.2301
MKNOeff	0.2281	0.2251

Table 5: Relative Error for the Navier-Stokes equation. Full resolution is 64 by 64 while half resolution is the 32 by 32 grid that models were trained on. All metrics are calculated on a test set of data that was withheld at training.

	Relative Error	
Model	Full Resolution	Half Resolution
ConvLSTM	0.5592	0.4600
FNO	0.1284	0.1277
AFNO	0.4207	0.4118
GNO	0.4908	0.4934
GNN	0.4766	0.4639
MKNO	0.1255	0.1254
MKNOeff	0.1361	0.1347

Acknowledgments

The authors want to thank the dedicated members of the VIP lab and all volunteers who have helped to organize the CVIS conference, computing resources provided by the Digital Research Alliance of Canada and funding provided by the National Research Council of Canada, AI4 Logistics Program.

References

[1] C. L. Fefferman, J. C. Robinson, J. L. R. Diez, and J. L. Rodrigo, *Partial differential equations in fluid mechanics*. Cambridge University Press, 2018, vol. 452.

[2] J.-J. Chattot, *Computational aerodynamics and fluid dynamics: an introduction*. Springer, 2002.

[3] A. P. Selvadurai, *Partial differential equations in mechanics 2: The Biharmonic equation, Poisson's equation*. Springer Science & Business Media, 2013.

[4] G. F. Pinder, *Numerical methods for solving partial differential equations: a comprehensive introduction for scientists and engineers*. John Wiley & Sons, 2018.

[5] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," no. arXiv:2010.08895, May 2021, arXiv:2010.08895 [cs, math]. [Online]. Available: http://arxiv.org/abs/2010.08895

[6] C. Beck, M. Hutzenthaler, A. Jentzen, and B. Kuckuck, "An overview on deep learning-based approximation methods for partial differential equations," *arXiv preprint arXiv:2012.12348*, 2020.

[7] W. E and B. Yu, "The deep ritz method: A deep learning-based numerical algorithm for solving variational problems," 2017.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, p. 686–707, 2019.

[9] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Graph kernel network for partial differential equations," no. arXiv:2003.03485, Mar. 2020, arXiv:2003.03485 [cs, math, stat]. [Online]. Available: http://arxiv.org/abs/2003.03485

[10] Y. Zhu and N. Zabaras, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification," *Journal of Computational Physics*, vol. 366, p. 415–447, Aug. 2018. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2018.04.018

[11] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik, "Prediction of aerodynamic flow fields using convolutional neural networks," *Computational Mechanics*, vol. 64, no. 2, p. 525–545, Jun. 2019. [Online]. Available: http://dx.doi.org/10.1007/s00466-019-01740-0

[12] N. B. Kovachki, S. Lanthaler, and A. M. Stuart, "Operator learning: Algorithms and analysis," 2024.

[13] B. Raonic, R. Molinaro, T. De Ryck, T. Rohner, F. Bartolucci, R. Alaifari, S. Mishra, and

E. de Bézenac, "Convolutional neural operators for robust and accurate learning of pdes," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, p. 77187–77200. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/f3c1951b34f7f55ffaecada7fde6bd5a-Paper-Conference.pdf

[14] A. S. Krishnapriyan, A. F. Queiruga, N. B. Erichson, and M. W. Mahoney, "Learning continuous models for continuous physics," *Communications Physics*, vol. 6, no. 1, p. 1–14, Nov. 2023.

[15] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Multipole graph neural operator for parametric partial differential equations," no. arXiv:2006.09535, Oct. 2020, arXiv:2006.09535 [cs, math, stat]. [Online]. Available: http://arxiv.org/abs/2006.09535

[16] H. Wang, J. Li, A. Dwivedi, K. Hara, and T. Wu, "Beno: Boundary-embedded neural operators for elliptic pdes," no. arXiv:2401.09323, Jan. 2024, arXiv:2401.09323 [cs]. [Online]. Available: http://arxiv.org/abs/2401.09323

[17] Z. Li, N. B. Kovachki, C. Choy, B. Li, J. Kossaifi, S. P. Otta, M. A. Nabian, M. Stadler, C. Hundt, and K. Azizzadenesheli, "Geometry-informed neural operator for large-scale 3d pdes."

[18] N. Liu, S. Jafarzadeh, and Y. Yu, "Domain agnostic fourier neural operators," in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, p. 47438–47450. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/940a7634dab556b67af15bacd337f7db-Paper-Conference.pdf

[19] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, "U-fno – an enhanced fourier neural operator-based deep-learning model for multiphase flow," no. arXiv:2109.03697, May 2022, arXiv:2109.03697 [physics]. [Online]. Available: http://arxiv.org/abs/2109.03697

[20] J. Guibas, M. Mardani, Z. Li, A. Tao, A. Anandkumar, and B. Catanzaro, "Adaptive fourier neural operators: Efficient token mixers for transformers," no. arXiv:2111.13587, Mar. 2022, arXiv:2111.13587 [cs]. [Online]. Available: http://arxiv.org/abs/2111.13587

[21] J. Pathak, S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, P. Hassanzadeh, K. Kashinath, and A. Anandkumar, "Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators," no. arXiv:2202.11214, Feb. 2022, arXiv:2202.11214 [physics]. [Online]. Available: http://arxiv.org/abs/2202.11214

[22] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," no. arXiv:1506.04214, Sep. 2015, arXiv:1506.04214 [cs]. [Online]. Available: http://arxiv.org/abs/1506.04214