Robust Emulator for Compressible Navier-Stokes using Equivariant Geometric Convolutions

Wilson G. Gregory
Dept. of Applied Mathematics and Statistics
Johns Hopkins University
Baltimore, MD, USA

David W. Hogg
Center for Cosmology and Particle Physics
Dept. of Physics
New York University, New York, NY, USA
Max-Planck-Institut für Astronomie
Heidelberg, Germany
Center for Computational Astrophysics
Flatiron Institute
New York, NY, USA

Kaze W. K. Wong
Dept. of Applied Mathematics and Statistics
Johns Hopkins University
Baltimore, MD

Soledad Villar
Dept. of Applied Mathematics and Statistics
Mathematical Institute for Data Science
Johns Hopkins University
Baltimore, MD, USA
Center for Computational Mathematics
Flatiron Institute
New York, NY, USA

Abstract

Recent methods to simulate complex fluid dynamics problems have replaced computationally expensive and slow numerical integrators with surrogate models learned from data. However, while the laws of physics are relationships between scalars, vectors, and tensors that hold regardless of the frame of reference or chosen coordinate system, surrogate machine learning models are not coordinate-free by default. We enforce coordinate freedom by using geometric convolutions in three model architectures: a ResNet, a Dilated ResNet, and a UNet. In numerical experiments emulating 2D compressible Navier-Stokes, we see better accuracy and improved stability compared to baseline surrogate models in almost all cases. The ease of enforcing coordinate freedom without making major changes to the model architecture provides an exciting recipe for any CNN-based method applied on an appropriate class of problems.

1 Introduction

Understanding and modeling the Navier-Stokes equations for fluid dynamics is a critical problem in astronomy, climate science, and many other fields. Traditional numerical solvers are accurate and are considered a robust standard for solving the Navier-Stokes equations, but they can be computationally expensive for systems that require a high resolution. Creating surrogate models with machine learning (ML) methods has shown promise as an alternative. Once trained on the desired spatial and temporal scales, these surrogate models can generate a solution from an initial condition much faster than a traditional solver. However, long term stability in surrogate models remains a concern.

One potential culprit for unstable rollouts is that ML models are not coordinate-free by default; they operate on the *components* of the vectors rather than the vectors themselves. Since the laws of physics are independent of our choice of coordinates [26], not respecting this coordinate independence may undermine the accuracy and stability of the model. With the tools of equivariant machine learning, we can make better and more efficient models by incorporating the rules of coordinate freedom.

The concept of equivariance is simple. Given a function $f : X \to Y$ and a group G with an action on both X and Y, we say f is *equivariant* with respect to G if for all $x \in X$ and $g \in G$ we have $f(g \cdot x) = g \cdot f(x)$. For equivariant machine learning, we learn a function f over a class of equivariant functions with respect to a relevant group, such as translations [17], gauge symmetries [4], permutations [20], or rotations/reflections [3, 5, 28, 29]. We use equivariant ML for dynamical systems by making our model equivariant to a group that expresses coordinate transformations.

There are many approaches to building equivariant models, such as using group convolutions [3] or irreducible representations [5, 15, 29]. Closest to our paper in aims are [28] and [2], but they implement the symmetries with irreducible representations and Clifford algebras respectively. Instead, this paper will use *geometric convolutions* that operate on images and filters of vectors and tensors. Our follow-on work [10] explores the topic more thoroughly, including more background and all proofs.

In section 2 we provide the background for geometric convolutions and how to use them to create an equivariant model, and in section 3 we discuss the problem setup and numerical results.

2 Methods

2.1 The Geometric Image

We start by describing tensors, their operations, and geometric images following the roadmap of [10]. Our geometric objects are scalars, vectors, and tensors and their negative parity counterparts (i.e. pseudoscalars, pseudovectors, and pseudotensors) in fixed dimension d, usually assumed to be 2 or 3. In this setting, the coordinate transformations will be given by the action of the orthogonal group $O(d)$.

A scalar is a value $s \in \mathbb{R}$ that is unchanged under coordinate transforms, i.e. for group element $g \in O(d)$, we have $g \cdot s = s$. A vector is an element $v \in \mathbb{R}^d$ that is transformed by $g \cdot v = M(g)\, v$ for all $g \in O(d)$. Here, $M(g)$ is the standard matrix representation of g. A rank-1 order k tensor is the tensor (outer) product of k vectors, i.e. $t = v_1 \otimes \ldots \otimes v_k$, and the action of $O(d)$ on a tensor is defined as the product of the action in each order of the tensor:

$$g \cdot (v_1 \otimes \ldots \otimes v_k) = (g \cdot v_1) \otimes \ldots \otimes (g \cdot v_k)\ . \tag{1}$$

To get higher rank tensors, we can add multiple rank-1 tensors and the action of $O(d)$ extends linearly.

When a group element g includes a reflection and acts on a negative parity object, that object will also undergo a sign flip. For example, a familiar pseudovector is angular velocity. We will refer to all these geometric objects as $k_{(p)}$-tensors, where $p = \pm 1$ is the parity.

We will use Einstein summation notation to define the allowed operations on tensors. For example, if a and b are $k_{(p)}$-tensors then $a + b$ is a $k_{(p)}$-tensor defined as $[a + b]_{i_1,\ldots,i_k} = [a]_{i_1,\ldots,i_k} + [b]_{i_1,\ldots,i_k}$. If a is a $k_{(p)}$-tensor and b is a $k'_{(p')}$-tensor, then the tensor product is a $(k + k')_{(p\,p')}$-tensor defined as

$$[a \otimes b]_{i_1,\ldots,i_k,i_{k+1},\ldots,i_{k+k'}} = [a]_{i_1,\ldots i_k} [b]_{i_{k+1},\ldots,i_{k+k'}} \tag{2}$$

If a is a $(k' + 2k)_{(p)}$-tensor, then the k-contraction $\iota_k(a)$ is defined as

$$[\iota_k(a)]_{i_1,\ldots,i_{k'}} = [a]_{j_1,\ldots,j_k,j_1,\ldots,j_k,i_1,\ldots,i_{k'}} \tag{3}$$

where, by Einstein summation notation, $j_1, \ldots, j_k$ are implicitly summed over. Using the k-contraction, we define the ℓ_2-norm extended to $k_{(p)}$-tensor a as $\|a\|_2 = \sqrt{\iota_k(a \otimes a)}$.

A $k_{(p)}$-tensor geometric image is a N^d grid where each pixel has a geometric object of the same type. We also consider $k_{(p)}$-tensor images on the d-torus, where the grid is given the algebraic structure of $(\mathbb{Z}/N\mathbb{Z})^d$. All the tensor operations described above have analogous definitions for the geometric image where each operation is performed pixel-wise.

Finally, [10] introduces a new operation: geometric convolution. If A is a $k_{(p)}$-tensor image and C is a $k'_{(p')}$-tensor image then the geometric convolution, denoted $A * C$, is a $(k + k')_{(p\,p')}$-tensor image

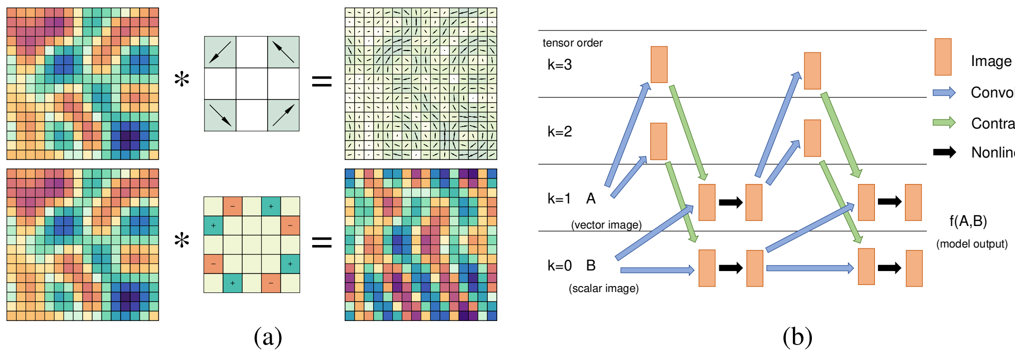


Figure 1: (a) Convolution of a scalar image with a pseudovector and pseudoscalar filter. (b) Example architecture taking a vector image and scalar image as input and output. Linear layers are shown by the blue convolution arrows followed by green contraction arrows. The black arrows represent nonlinearities. The orange blocks represent multiple channels of images at that tensor order.

where the normal multiplication of convolution has been replaced with the tensor product. See Figure 1 for an example of geometric convolution.

2.2 Equivariance and the Geometric Image

The coordinate transformations of geometric images will be given by a group of translations of pixels on the torus and rotations/reflections that rotate both the overall image and the geometric objects in each pixel. The tensor operations of the preceding section are equivariant to $O(d)$ [21]. However, since the geometric image is not a continuous tensor field but a discretization onto a grid, we want to use a discrete subgroup of $O(d)$ to avoid rotations that do not return the pixels to the same grid. Thus we work with B_d, the Euclidean symmetries of the d-dimensional hypercube.

The group B_d acts on geometric images by rotating the entire image as well as rotating the tensor in each pixel. Our full group, denoted $G_{N,d}$, is the semi-direct product of translations on the d-torus and B_d. A function f on geometric images is *equivariant* to $G_{N,d}$ if for all $k_{(p)}$-tensor images A and all $g \in G_{N,d}$ we have $f(g \cdot A) = g \cdot f(A)$. Additionally, f is *invariant* to $G_{N,d}$ if $f(g \cdot A) = f(A)$ and an image B is *isotropic* if $g \cdot B = B$. It is straightforward to show that convolutions are equivariant when we consider both the image and the filter as input, that is, $g \cdot (A * C) = (g \cdot A) * (g \cdot C)$. However, we would like our filter C to be fixed for all inputs A and maintain the equivariance. This can be achieved by using the *isotropic* filters, a result shown in [3]. We combine this idea with ideas from [16] and the rules of linear tensor functions [7] to get the result below. This theorem is described and proven in [10, Theorem 1] and is similar to [9, Theorem 3.1], which characterizes polynomials on point clouds that are equivariant to permutations and rotations.

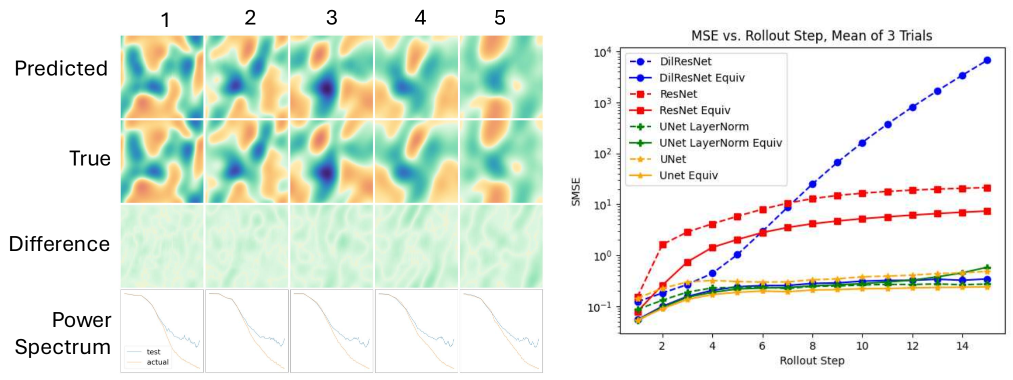
Theorem 2.1 (informal)**.** *A function from $k_{(p)}$-tensor images to $k'_{(p')}$-tensor images is linear and $G_{N,d}$-equivariant if and only if it can be written as $\iota_k(A * C)$ for some B_d-isotropic $(k+k')_{(p\,p')}$-tensor filter C.*

2.3 Building the Equivariant Model

We now have the necessary tools to build our model. Each layer in the network operates on a collection of geometric images of any combination of tensor order and parity with one or more channels. The layers used in this work are linear, pointwise nonlinear, LayerNorm [1], and max pool.

The linear layers take an input collection of geometric images $\{(k_i, p_i)\}_{i=1}^{W_\text{in}}$ with c_i channels and the desired output tensor orders and parities $\{(k_j, p_j)\}_{j=1}^{W_\text{out}}$ with c_j channels and computes all the convolutions and contractions to map between those two sets. Following Theorem A.1, there are $\ell = 1, \ldots, c_j$ functions

$$\sum_{i=1}^{W_\text{in}} \sum_{z=1}^{c_i} \iota_{k_i}(A_{i,z} * C_{\ell,i,z}) \tag{4}$$



(a) (b)

Figure 2: (a) Five steps of M0.1 rollout using the best performing model, the equivariant UNet without LayerNorm. The x-component of the velocity is plotted. (b) Comparison of test performance over a 15 step rollout. The SMSE is shown for *each* step, rather than a cumulative loss.

for each desired output tensor order and parity. Per the theorem, these convolution filters $C_{\ell,i,z}$ must be B_d-isotropic to guarantee that this layer is $G_{N,d}$ equivariant. We construct a basis of B_d-isotropic filters using group averaging as detailed in [10], and then take a parameterized linear combination to get our filters. However, using filters as large as the input image is impractical in most cases, so we use deeper networks of 3×3 or 5×5 filters, as is commonly done in CNNs [23].

The typical pointwise nonlinear functions such as ReLU or tanh break equivariance when applied to the individual components of a tensor. Properly building $O(d)$-equivariant nonlinear functions is a challenging and active area of research; for a larger exploration, see [30] and references therein. For this model, we extend the Vector Neuron nonlinearity [6] for any tensor order and parity. See Figure 1 for an example of a typical architecture interlacing linear and nonlinear layers.

The final layer types we will use in our model are LayerNorm and max pool. To make an equivariant version of LayerNorm, we follow the strategy of vector whitening used in [2], based on a similar strategy developed for neural networks with complex values [27]. Max pooling layers use the ℓ_2 tensor norm to determine the max tensor for each channel of each input image.

3 Numerical Experiments

We use 2D compressible Navier-Stokes simulation data from the excellent PDEBench [25]. This data consists of a velocity field, a density field, and a pressure field. The baseline methods will treat these as four input channels of a typical CNN, but of course our equivariant methods recognize these as a vector image and two scalar images. The images are 128×128 pixels on the torus–that is, the data is generated with periodic boundary conditions. We use data generated with two distinct set of parameters: Mach number $M = 0.1$, shear viscosity $\eta = 0.01$, and bulk viscosity $\zeta = 0.01$ and $M = 1.0, \eta = 0.1, \zeta = 0.1$. The simulations are saved at 21 time points which are a subset of the integrator timesteps.

We use 128 simulation trajectories with random initial conditions as training data and another 128 trajectories as test data. Each model takes as input four consecutive time steps of each field, and outputs each field of the next time step. Thus we can turn those 128 trajectories into $2,176$ training data points because each trajectory has 17 overlapping sections of four input steps and one output step. We train a Dilated ResNet [24], a ResNet [12], and a UNet [22] with and without LayerNorm [1] and equivariant counterparts of each of those models. We train with the MSE loss of a single step, but at test time we are also interested in the performance of autoregressively rolling out the model over 15 time steps. The baseline models and training setup generally follow those described in [11], and additional data, model, and training details are in Appendix B.

The numerical results are given in Table 1. With the exception of the 15 step rollout of the UNet LayerNorm, the equivariant version of each model outperforms the non-equivariant version in every case. In Figure 2, we can see with more granularity the test performance for each rollout step. In the

model	M0.1 1-step	M0.1 rollout	M1.0 1-step	M1.0 rollout
DilResNet	0.040	13318.773 ± 18824.855	0.005	9.574 ± 9.608
DilResNet Equiv	**0.018**	$\mathbf{3.770 \pm 0.090}$	**0.001**	$\mathbf{0.153 \pm 0.023}$
ResNet	0.039	175.736 ± 17.846	0.009	$\mathbf{0.835 \pm 0.097}$
ResNet Equiv	$\mathbf{0.024 \pm 0.001}$	$\mathbf{57.508 \pm 9.157}$	**0.003**	2.943 ± 0.992
UNet LayerNorm	0.027	3.414 ± 0.217	0.009 ± 0.001	1.067 ± 0.190
UNet LayerNorm Equiv	$\mathbf{0.018 \pm 0.002}$	$\mathbf{3.971 \pm 1.158}$	**0.001**	$\mathbf{0.136 \pm 0.047}$
UNet	0.047 ± 0.001	5.086 ± 0.105	0.012 ± 0.002	2.074 ± 0.066
Unet Equiv	**0.018**	$\mathbf{2.813 \pm 0.257}$	**0.001**	$\mathbf{0.124 \pm 0.018}$

Table 1: Loss values for each model, averaged over three trials. All losses are the sum of the mean squared error losses over the channels: density, pressure, and velocity. The rollout loss is the sum of the error over 15 steps. The std $\pm 0.xxx$ is provided if its at least 0.001.

most drastic example, the rollout error for the Dilated ResNet explodes, while the equivariant Dilated ResNet is stable and accurate over all 15 steps. In [24], the authors combat this issue by adding a small amount of Gaussian noise during training; we instead achieve stability in a physically-motivated way by enforcing $O(d)$-equivariance. The equivariance also helps with parameter efficiency; we chose the channel depth so that the equivariant model was comparable to the baseline model in parameter count (Table 1), however we could also have aimed for the same accuracy to get a large reduction in the number of parameters.

One interesting area for further research is how and why the equivariance helps. One interesting observation of Figure 2 (a) is that the power spectrum for the equivariant model output is still quite different from the ground truth at higher frequencies. It may be that equivariance is advantageous at certain scales and not at others.

Acknowlegements: It is a pleasure to thank Roger Blandford (Stanford), Drummond Fielding (Flatiron), Leslie Greengard (Flatiron), Ningyuan (Teresa) Huang (JHU), Kate Storey-Fisher (NYU), and the Astronomical Data Group at the Flatiron Institute for valuable discussions and input. All the code used for making the data and figures in this paper is available at `https://github.com/WilsonGregory/GeometricConvolutions`.

Funding: WG was supported by NSF CISE 2212457. SV was partly supported by the NSF–Simons Research Collaboration on the Mathematical and Scientific Foundations of Deep Learning (MoDL) (NSF DMS 2031985), NSF CISE 2212457, ONR N00014-22-1-2126, and NSF CAREER 2339682.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[2] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K. Gupta. Clifford neural layers for pde modeling, 2023.

[3] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[4] Taco S. Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral cnn, 2019.

[5] Taco S. Cohen and Max Welling. Steerable cnns, 2016.

[6] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas Guibas. Vector neurons: A general framework for so(3)-equivariant networks, 2021.

[7] Yu. I. Dimitrienko. *Tensor Analysis and Nonlinear Tensor Functions*. Springer Dordrecht, 1 edition, June 2013.

[8] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[9] Ben Finkelshtein, Chaim Baskin, Haggai Maron, and Nadav Dym. A simple and universal rotation equivariant point-cloud network, 2022.

[10] Wilson Gregory, David W. Hogg, Ben Blum-Smith, Maria Teresa Arias, Kaze W. K. Wong, and Soledad Villar. Geometricimagenet: Extending convolutional neural networks to vector and tensor images, 2023.

[11] Jayesh K. Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling, 2022.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016.

[14] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[15] Erik Jenner and Maurice Weiler. Steerable partial differential operators for equivariant neural networks. *arXiv preprint arXiv:2106.10163*, 2021.

[16] Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[17] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[18] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[20] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks, 2019.

[21] M. M. G. Ricci and Tullio Levi-Civita. Méthodes de calcul différentiel absolu et leurs applications. *Mathematische Annalen*, 54(1):125–201, 1900.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[24] Kim Stachenfeld, Drummond Buschman Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned simulators for turbulence. In *International Conference on Learning Representations*, 2022.

[25] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning, 2024.

[26] Kip S. Thorne and Roger D. Blandford. *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*. Princeton University Press, 2017.

[27] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks, 2018.

[28] Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization, 2021.

[29] Maurice Weiler and Gabriele Cesa. General $e(2)$-equivariant steerable cnns, 2021.

[30] Yinshuang Xu, Jiahui Lei, Edgar Dobriban, and Kostas Daniilidis. Unified fourier-based kernel and nonlinearity design for equivariant networks on homogeneous spaces, 2022.

A Claims and Extensions

The formal statement of Theorem 2.1 is given below. The proof is given in [10].

Theorem A.1. *A function $f : \mathcal{A}_{N,d,k,p} \to \mathcal{A}_{N,d,k',p'}$ is linear and $G_{N,d}$-equivariant if and only if it can be written as $\iota_k(A * C)$ for some B_d-isotropic $C \in \mathcal{A}_{M,d,k+k',p\,p'}$, where $M = N$ if N is even and $M = N + 1$ otherwise.*

We extend the vector neuron nonlinearity of [6] for any set of $k_{(p)}$-tensor images as follows.

Definition A.2. Let $A_i \in \mathcal{A}_{N,d,k,p}$ for $i = 1, \ldots, c$ be c channels of input geometric images. Let $\alpha_i, \beta_i \in \mathbb{R}$ for $i = 1, \ldots, c$ be learned scalar parameters, and $Q = \sum_{i=1}^{c} \alpha_i\, A_i, K = \sum_{i=1}^{c} \beta_i\, A_i$. Then the nonlinearity $\sigma : (\mathcal{A}_{N,d,k,p})^c \to \mathcal{A}_{N,d,k,p}$ is defined:

$$\sigma((A_i)_{i=1}^{c}) = \begin{cases} Q & \text{if } \iota_k(Q \otimes K) \geq 0 \\ Q - \iota_k\Big(Q \otimes \frac{K}{\|K\|_2}\Big)\frac{K}{\|K\|_2} & \text{otherwise} \end{cases} \tag{5}$$

where $\|\cdot\|_2$ is the tensor norm. To get c output channels, we can repeat this function c times with different learned parameters α_i, β_i.

The max pool function for a geometric image is defined as follows.

Definition A.3 (max pool$_b$)**.** Let b be a positive integer and let $A \in \mathcal{A}_{N,d,k,p}$, where b divides N. Then the function $\text{max pool}_b : \mathcal{A}_{N,d,k,p} \to \mathcal{A}_{N/b,d,k,p}$ is defined for each pixel index $\bar{\imath} \in [0, (N/b) - 1]^d$:

$$\text{max pool}_b(A)(\bar{\imath}) = A\left(b\,\bar{\imath} + \underset{\bar{a} \in [0,b-1]^d}{\arg\max}\ \|A(b\,\bar{\imath} + \bar{a})\|_2\right) \tag{6}$$

B Experimental Details

B.1 Data

The data is the PDEBench file

```
2D_CFD_Rand_M0.1_Eta0.01_Zeta0.01_periodic_128_Train.hdf5
```

which can be found at `https://darus.uni-stuttgart.de/file.xhtml?fileId=164687&version=8.0`. We used the first 128 trajectories as training data, the next 32 trajectories as a validation set, and the next 128 trajectories as a test data set.

B.2 Models

Model specifics are described below. For equivariant models, we always use ReLU for scalars and the Vector Neuron activation for non-scalars. For equivariant encoder and decoder blocks, we use 3×3 filters instead of 1×1 filters because for some order and parity pairs, there are no 1×1 B_d-isotropic filters. All convolutions use biases except for the UNet. For equivariant models, the bias is a scale of the mean tensor of that image. Additional details are in Table 2.

- **Dilated ResNet [24]:** The model starts with two "encoder" convolutions with 1×1 filters and ReLU activations. There are four blocks, each consisting of seven convolutions with dilations of $1, 2, 4, 8, 4, 2, 1$ with associated ReLU activations. There are residual connections connecting each block. The model concludes with two "decoder" convolutions with 1×1 filters and a ReLU activation between the two.

- **ResNet [12]:** This model consists of 8 blocks of 2 convolutions each with residual connections between each block. Each block also has LayerNorm and a GeLU activation [14]. We put the LayerNorm and activation prior to the convolution (preactivation order [13]) following [11]. This model also uses two "encoder" 1×1 convolutions and two "decoder" 1×1 convolutions.
- **UNet LayerNorm [11]:** This model is referred to as "UNetBase" in [11]. This starts with an embedding block with a convolution with a 3×3 filter following by LayerNorm and a GeLU activation [14]. Next comes a max pool$_2$ followed by two convolutions with LayerNorm and GeLU activation. This is process is repeated for 4 total downsamples, and notably the number of convolution channels is doubled for every down sample. Then the process happens in reverse, with max pooling replaced with transposed convolution to double the spatial size instead of halving it each time. See [8] for a description of transposed convolution. The number of convolution channels is also halved each time we upsample. The final kicker is that there are also residual connections from before each downsample to after each upsample for the appropriate spatial size. The model concludes with a final convolution. In the equivariant model we do not include the LayerNorm because it hurt the performance.
- **UNet [22]:** This model is the same as the one above, except is uses BatchNorm instead of LayerNorm and the convolutions are without biases.

model	params	CNN channels	norm	bias	learning rate
DilResNet	1,043,651	64	-	Yes	`2e-3`
DilResNet Equiv	979,347	48	-	Mean	`1e-3`
ResNet	2,401,155	128	LayerNorm	Yes	`1e-3`
ResNet Equiv	2,558,703	100	LayerNorm	Mean	`7e-4`
UNet LayerNorm	31,053,251	64	LayerNorm	Yes	`8e-4`
UNet LayerNorm Equiv	27,077,139	48	-	Mean	`4e-4`
UNet	31,046,400	64	BatchNorm	No	`8e-4`
UNet Equiv	27,066,864	48	-	No	`3e-4`

Table 2: Comparison of various models. The number of channels of each model was chosen so that the equivariant and non-equivariant models have roughly the same number of parameters.

B.3 Training

For a loss function, we use the sum of mean squared error loss, or SMSE. This loss sums over the tensor components and the channels and takes the mean over the spatial components. If $\{A_i\}_{i=1}^c$ are the true $k_{i(p_i)}$-tensor images and $\left\{\hat{A}_i\right\}_{i=1}^c$ are our predicted $k_{i(p_i)}$-tensor images, then the $\mathcal{L}_{\text{smse}}$ is defined as,

$$\mathcal{L}_{\text{smse}}\left(\{A_i\}_{i=1}^c, \left\{\hat{A}_i\right\}_{i=1}^c\right) = \sum_{i=1}^c \frac{1}{N^d} \sum_{\bar{\imath}} \left\| A_i(\bar{\imath}) - \hat{A}_i(\bar{\imath}) \right\|_2^2 , \quad (7)$$

where $\|\cdot\|_2$ is the tensor norm. When calculating a rollout loss, we simply sum the loss of each rollout step.

We follow a similar training regime as in [11]. We train for 50 epochs using the AdamW optimizer [19] with a weight decay of `1e-5` and a cosine decay schedule [18] with 5 epochs of warmup. Learning rates were tuned for each model, searching for values between `1e-4` and `2e-3`, and are included in Table 2.

We trained on 4 RTX A5000 graphics cards with a batch size of 8, for an effective batch size of 32. Experiments we averaged over 3 trials, using the same training data each time. It possible that different optimizers, learning rate schedules, batch sizes, or other hyperparameters may perform better on the task, but we held those fixed and only tuned the learning rate since our focus is on comparing the equivariant and non-equivariant models.