SUPPLEMENTARY MATERIALS: Deep Neural Networks and PIDE discretizations

Bastian Bohn[*][†], Michael Griebel[†][‡], and Dinesh Kannan[‡]

SM1. Pseudo-differential operators.

Definition SM1.1. *Let $\Omega \subset \mathbb{R}^n$ be open, $0 \le \rho \le 1$, $0 \le \delta \le 1$, $m \in \mathbb{R}$, $n \in \mathbb{N}$, $n \ge 1$. Then the space of symbols of order m and of type (ρ, δ), denoted by $S^m_{\rho,\delta}$, is the space of all $p \in C^\infty(\Omega \times \mathbb{R}^n)$ such that, for all compact sets $K \subset \Omega$ and all multi-indices $\alpha, \beta \in \mathbb{N}^n$, there is a constant $C_{K,\alpha,\beta}$ such that*

$$|\, \partial^\alpha_{\mathbf{x}}\, \partial^\beta_\xi\, p(\mathbf{x}, \xi)| \le C_{K,\alpha,\beta}(1 + |\xi|)^{m-\rho|\beta|+\delta|\alpha|},$$

where $\mathbf{x} \in K$, $\xi \in \mathbb{R}^n$.

Definition SM1.2. *Let $v \in \mathcal{S}(\mathbb{R}^n)$, $p \in S^m_{\rho,\delta}$. A pseudo-differential operator $P(\mathbf{x}, D)$ on $\mathbb{R}^n$ with symbol $p(\mathbf{x}, \xi)$ is defined as*

$$P(\mathbf{x}, D)v(\mathbf{x}) := \int\limits_{\mathbb{R}^n} p(\mathbf{x}, \xi)\, \hat{v}(\xi)\, e^{2\pi i \mathbf{x}\cdot\xi}\, d\xi. \tag{SM1.1}$$

SM2. Addressing the singularity.

The kernel under the integral in equation (3.5) has a singularity when $\mathbf{x} = \mathbf{y}$, but for $v \in \mathcal{S}(\mathbb{R}^n)$, we can write

$$\begin{aligned}\int\limits_{\mathbb{R}^n} \frac{[v(\mathbf{x}) - v(\mathbf{y})]}{|\mathbf{x} - \mathbf{y}|^{n+2s}}\, d\mathbf{y} &\le C \int\limits_{B_R(\mathbf{x})} \frac{|\mathbf{x} - \mathbf{y}|}{|\mathbf{x} - \mathbf{y}|^{n+2s}}\, d\mathbf{y} + \|v\|_{L^\infty(\mathbb{R}^n)} \int\limits_{\mathbb{R}^n\setminus B_R(\mathbf{x})} \frac{1}{|\mathbf{x} - \mathbf{y}|^{n+2s}}\, d\mathbf{y}\\ &= C\Bigg(\int\limits_{B_R(\mathbf{x})} \frac{1}{|\mathbf{x} - \mathbf{y}|^{n+2s-1}}\, d\mathbf{y} + \int\limits_{\mathbb{R}^n\setminus B_R(\mathbf{x})} \frac{1}{|\mathbf{x} - \mathbf{y}|^{n+2s}}\, d\mathbf{y}\Bigg)\\ &= C\Bigg(\int\limits_0^R \frac{1}{|\mathbf{r}|^{2s}}\, d\mathbf{r} + \int\limits_R^\infty \frac{1}{|\mathbf{r}|^{2s+1}}\, d\mathbf{r}\Bigg) < +\infty.\end{aligned}$$

C is a positive constant depending on n and $\|v\|_{L^\infty}$. The integral shown above is finite only for $0 < s < 1/2$, but we can make the integral in equation (3.5) well-defined for $0 < s < 1$.

It can be shown [SM4] that the fractional Laplacian operator given by (3.5) can also be written as $(-\Delta)^s v(\mathbf{x}) := \frac{c_{n,s}}{2} \int\limits_{\mathbb{R}^n} \frac{[2v(\mathbf{x}) - v(\mathbf{x}+\mathbf{y}) - v(\mathbf{x}-\mathbf{y})]}{|\mathbf{y}|^{n+2s}}\, d\mathbf{y}$. Using this reformulation of the

[*]Fraunhofer Center for Machine Learning, Schloss Birlinghoven, 53754 Sankt Augustin, Germany (bastian.bohn@scai.fraunhofer.de).

[†]Fraunhofer Institute for Algorithms and Scientific Computing SCAI, Schloss Birlinghoven, 53754 Sankt Augustin, Germany.

[‡]Institute for Numerical Simulation, University of Bonn, Friedrich-Hirzebruch-Allee 7, 53115 Bonn, Germany (griebel@ins.uni-bonn.de, kannan@ins.uni-bonn.de).

fractional Laplacian and the second-order Taylor expansion of u and assuming v to be smooth, the integral term

$$\int_{\mathbb{R}^n} \frac{[2v(\mathbf{x}) - v(\mathbf{x}+\mathbf{y}) - v(\mathbf{x}-\mathbf{y})]}{|\mathbf{y}|^{n+2s}} \leq \frac{\|D^2 v\|_{L^\infty}}{|\mathbf{y}|^{n+2s-2}}$$

is integrable at the origin for any $0 < s < 1$. Thus, we can remove $P.V.$ from the integral as long as $v \in \mathcal{S}(\mathbb{R}^n)$. The idea is that, near $\mathbf{x}$, $[v(\mathbf{x})-v(\mathbf{y})]$ has the approximation $\nabla v(\mathbf{x})\cdot(\mathbf{x}-\mathbf{y})$, and the term under the integral is of the form $\frac{\nabla v(\mathbf{x})\cdot(\mathbf{x}-\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^{n+2s}}$. This term is odd with respect to $\mathbf{x}$, and consequently, it averages out for any $\mathbf{y}$ in the neighborhood of $\mathbf{x}$ by symmetry, and the immediate neighborhood does not contribute to the integral.

SM3. Speeding up tensor computations.

Rearranging the two-step computation. The affinity kernel ω is stored in a matrix form for each batch of training data. We can reorder the terms in equation (4.1) to enable faster computations of tensors. For example, a part of the term in the first equation of (4.1) can be written as

$$\sum_j \omega(\mathbf{X}_i, \mathbf{X}_j)(\mathbf{X}_j - \mathbf{X}_i) = \sum_j \omega(\mathbf{X}_i, \mathbf{X}_j)\mathbf{X}_j - \sum_j \omega(\mathbf{X}_i, \mathbf{X}_j)\mathbf{X}_i \tag{SM3.1}$$

$$= \sum_j \omega(\mathbf{X}_i, \mathbf{X}_j)\mathbf{X}_j - \mathbf{X}_i \sum_j \omega(\mathbf{X}_i, \mathbf{X}_j). \tag{SM3.2}$$

The second part of this term is just the sum of the i-th row of the matrix representing ω, times the pixel strip $\mathbf{X}_i$. Similarly, for the second stage, we have

$$[\mathbf{B}_2]_i = \cdots \left[\cdots \sum_j \omega(\mathbf{X}_i, \mathbf{X}_j)[\mathbf{B}_1]_j - [\mathbf{B}_1]_i \sum_j \omega(\mathbf{X}_i, \mathbf{X}_j) \right]. \tag{SM3.3}$$

Nonlocal diffusion operator. Figure SM1 shows how the tensors are dealt with and propagated forward in the nonlocal block, where the shape of the input to the nonlocal block is assumed to be $H \times W \times 1024$, 1024 being the number of channels of the input. H and W are the spatial height and width of the feature maps, respectively. The diagram only shows the computations for one stage of the nonlocal block since the other stages are just repetitions but with a pre-computed kernel ω. In Figure SM1, $\otimes$ represents matrix multiplication, $\oplus$ and $\ominus$ denote element-wise addition and subtraction, respectively, and $\odot$ denotes the dot product. For the dot product, each row of the tensor with shape $HW \times 1024$ is multiplied by a corresponding element from the row vector of shape $HW \times 1$. The first multiplication is used to compute the kernel ω. The second multiplication represents the first term in equations (SM3.2) and (SM3.3), and the dot product is used to compute the second part of equations (SM3.2) and (SM3.3).

Fractional Laplacian operator $(-\Delta)^s$. The tensors in the nonlocal block are computed similarly, as shown in Figure SM1. In the case of pseudo-differential operators, the $\otimes$ symbol for the kernel computation represents pair-wise distance computations. The other $\otimes$ symbol

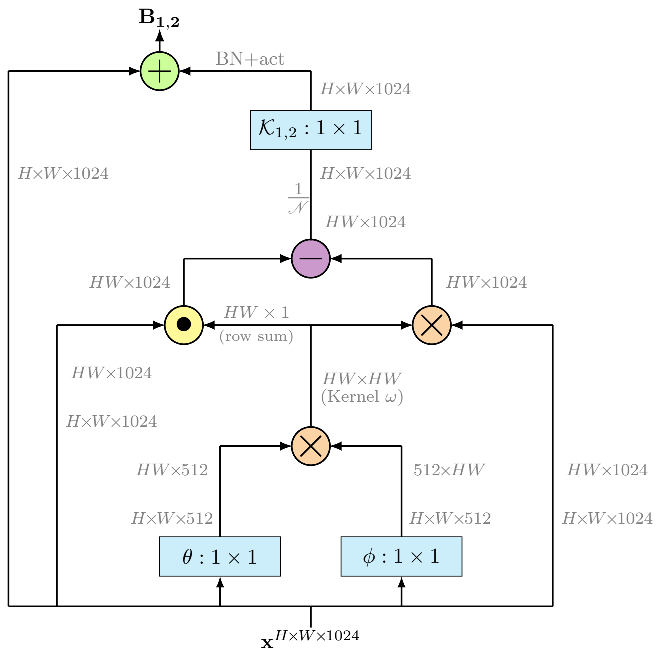

Figure SM1. *Forward propagation of tensors in a nonlocal block with the nonlocal diffusion operator $\mathcal{L}$.*

in the figures still stands for matrix multiplication as before. The only difference here is that we compute $(\mathbf{X}_i - \mathbf{X}_j)$ (and $([\mathbf{B}_1]_i - [\mathbf{B}_1]_j)$), instead of $(\mathbf{X}_j - \mathbf{X}_i)$ (and $([\mathbf{B}_1]_j - [\mathbf{B}_1]_i)$), to be in line with the definitions of the two operators. However, this does not make any difference to the learning problem.

Subsampled nonlocal blocks. As we can see in Section 6.1, the global computation can be quite expensive. In order to minimize the floating-point operations, several remedies exist. Firstly, all the convolutions in the nonlocal block $(\theta, \phi, \mathcal{K}_1, \mathcal{K}_2)$ are 1×1 convolutions. This itself reduces the computational effort drastically. Secondly, the embeddings θ and ϕ can be used to reduce the number of channels by half, as shown in Figure SM1. This reduces the computation of a nonlocal block by half when the kernel ω is computed.

There is one more way of reducing the computational effort of the nonlocal blocks, namely subsampling the image before the affinity of the pixel strips is computed. This is shown in Figure 2 (right). Figure 2 shows how the input $\mathbf{X}$ (of shape $H \times W \times C$) is spatially subsampled to $\hat{\mathbf{X}}$ (of shape $H' \times W' \times C$) before the affinity between the regions of the image is computed. Experiments have shown that this has very little impact on the quality of the results of the network. This tells us that we do not need to compare each pair of pixel strips. Instead, it is good enough if we compare patches of the image with each other and learn the correlations between them while we compute the kernel ω. The computational savings due to this subsampling trick [SM6] are discussed in Section 6.1.

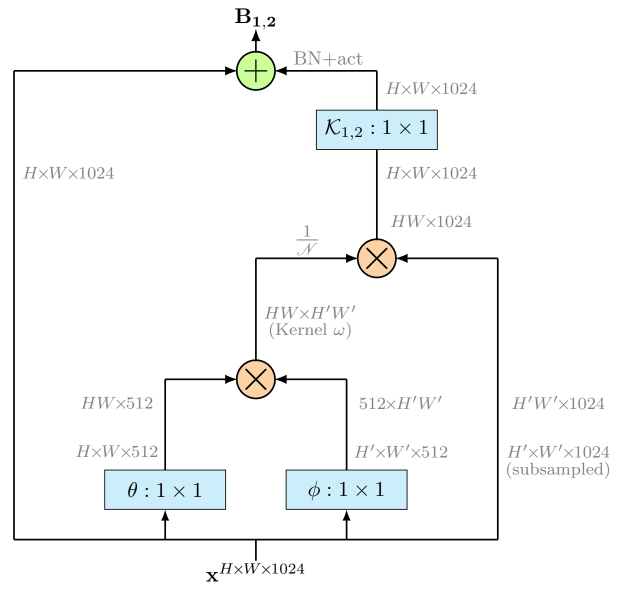

Figure SM2. *Forward propagation of tensors in a* subsampled *nonlocal block with the inverse Laplacian operator* $(-\Delta)^{-s}$.

SM4. Implementation details and choice of hyperparameters. All the networks are implemented and trained using the Tensorflow library [SM1] with a single Nvidia Tesla P100 GPU. The networks for the semantic segmentation task (Section 5.3) are trained using an Nvidia Tesla V100 GPU. All the trainable weights in the neural network are initialized based on the suggestions in [SM5]. The discretization step size h is kept between 0.04 and 0.08. Very small values of h lead to slow propagation of the information down the network, and as a result, the network performs worse. On the other hand, we see exploding gradients and sudden feature transformations in the network for bigger values of h, which cause instability and adversely affect training and convergence.

The loss function S in equation (2.2) is chosen to be the cross-entropy loss function. Stochastic Gradient Descent (SGD) is used as the optimizer with momentum 0.9 and with a learning rate of 0.01 for the first epoch to warm up the training. Then we go back to a learning rate of 0.1 and reduce it by a factor of 10 after 80, 120, 160 and 180 training epochs. The weight decay constant α_1 for weights in the normal blocks is 2×10^{-4} for CIFAR-10/CIFAR-100/BDD100K and 5×10^{-4} for STL-10. The weight smoothness decay constant α_2 is 1×10^{-8}. The reason for such a small value for α_2 is that, in our case, we have a nonlocal block after the second block in each Unit. While we do want the weights to vary smoothly, the weights and the features of the second and third Hamiltonian block will be invariably quite different because of the presence of the nonlocal operation between the two blocks. Therefore, α_2 is kept smaller than the value suggested in [SM2]. As discussed before, the convolutional weights in the nonlocal block are only regularized by weight decay (2×10^{-4} for all the datasets, i.e. CIFAR-10/100, STL-10, BDD100K) and not by the weight smoothness decay. The scaling

Table SM1

Number of floating-point operations depending on the subsampling pool size in the nonlocal blocks while training the nonlocal diffusion network on the STL-10 dataset.

Network	**Subsample pool size**	**STL-10**
		FLOPs (M)
Nonlocal diffusion $\mathcal{L}$	0	9341.2
	2	3471.4
	4	2003.7
	6	1731.9
	8	1636.8
	12	1568.9
Hamiltonian-74	NA[1]	1432.5

factor λ is 0.1 for all the nonlocal operators with the dimensional constant $n = 2$. The value of s for the pseudo-differential operators is $1/2$. From the experiments, it has been seen that the nature of the nonlocal operator plays a more important role than the power of the fractional Laplacian and the inverse fractional Laplacian. This is to say, any value of s between 0 and 1 works for the two pseudo-differential operators, and the performance is rather determined by the nature of the nonlocal interaction between the pixels strips and not by the value of s in the kernel of the integral operator.

SM5. Computational costs for the STL-10 dataset. When it comes to datasets like STL-10 [SM3] (or ImageNet, etc.), there is a trade-off that needs to be taken into consideration. Often, the training of networks is bound by memory constraints and not by computational constraints. In such cases, this increase is within acceptable limits. But when the aim is to bring down the number of floating-point operations, one could use the subsampling trick in the nonlocal block (Figure 2 and Section SM3 in this supplementary material). This drastically reduces the number of floating-point operations for the computations of the kernel ω while measuring the affinity between each pair of pixel strips/sections of the image, without changing the nonlocal nature of the block. Table SM1 shows the effect of the subsampling pool size on the number of floating-point operations for the nonlocal diffusion network while training on the STL-10 dataset. The zero in the table stands for non-subsampled nonlocal blocks. Clearly, the computational cost is quite high if no subsampling is performed. When the pool size is increased, the number of floating-point operations gradually approaches the number of FLOPs for the original Hamiltonian network (on STL-10). When the pool size is too large, the nonlocal character of the block is degraded a bit. Thus, it is a balancing act between computational costs versus the increase in performance of a network in the presence of nonlocal blocks. One has to choose the fitting pool size for the subsampling in the nonlocal block, based on the computational constraints that one has and the resolution of each image.

[1]Not applicable because the original Hamiltonian network has no nonlocal blocks.

SM6. Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$.

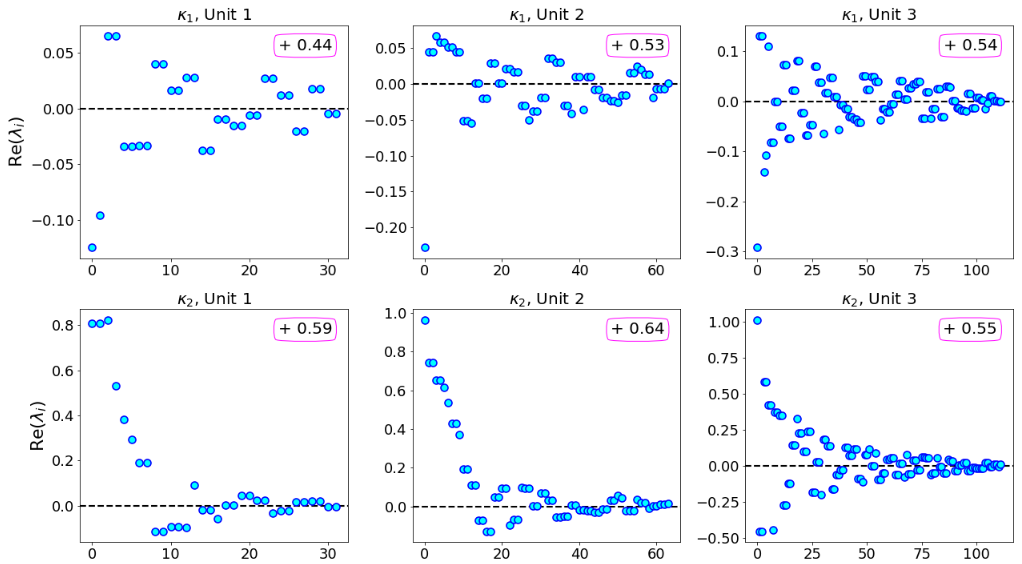

Figure SM3. *Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the nonlocal block placed in each Unit of the pseudo-differential $(-\Delta)^{1/2}$ network while training on STL-10.*

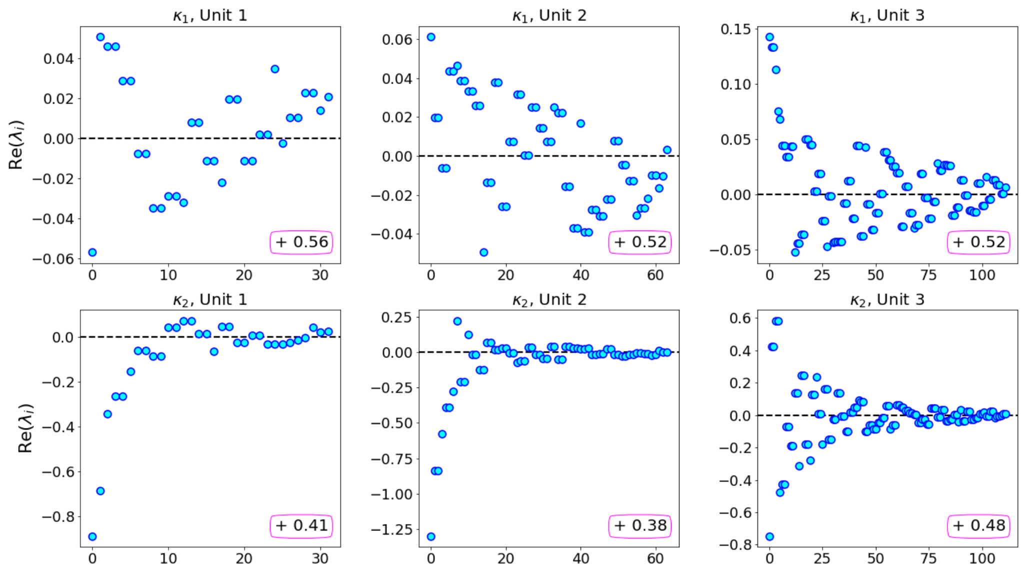

Figure SM4. *Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the nonlocal block placed in each Unit of the pseudo-differential $(-\Delta)^{-1/2}$ network while training on STL-10.*

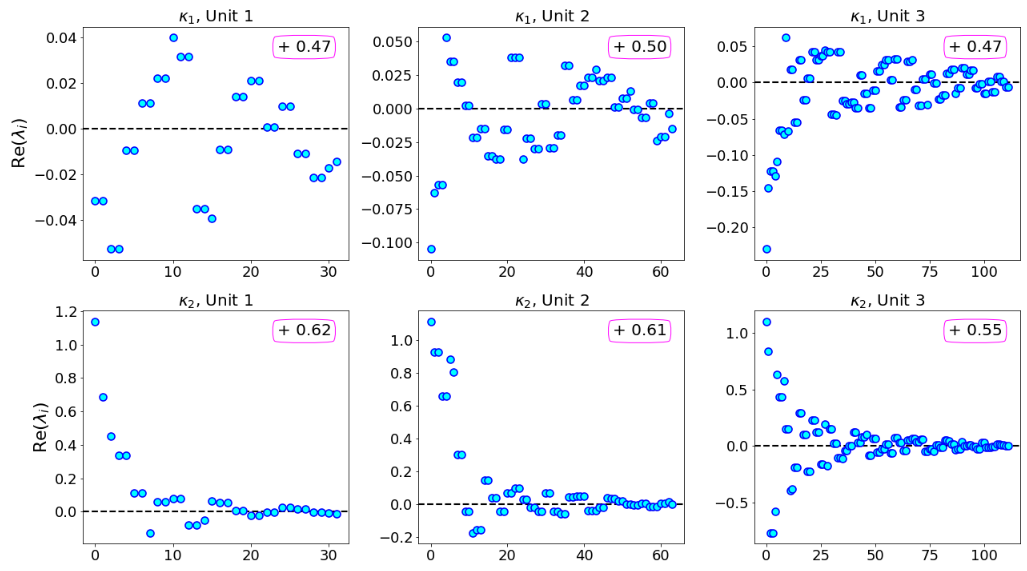

Figure SM5. *Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the nonlocal block placed in each Unit of the pseudo-differential $(-\Delta)^{-1}$ network while training on STL-10.*

SM7. Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$.

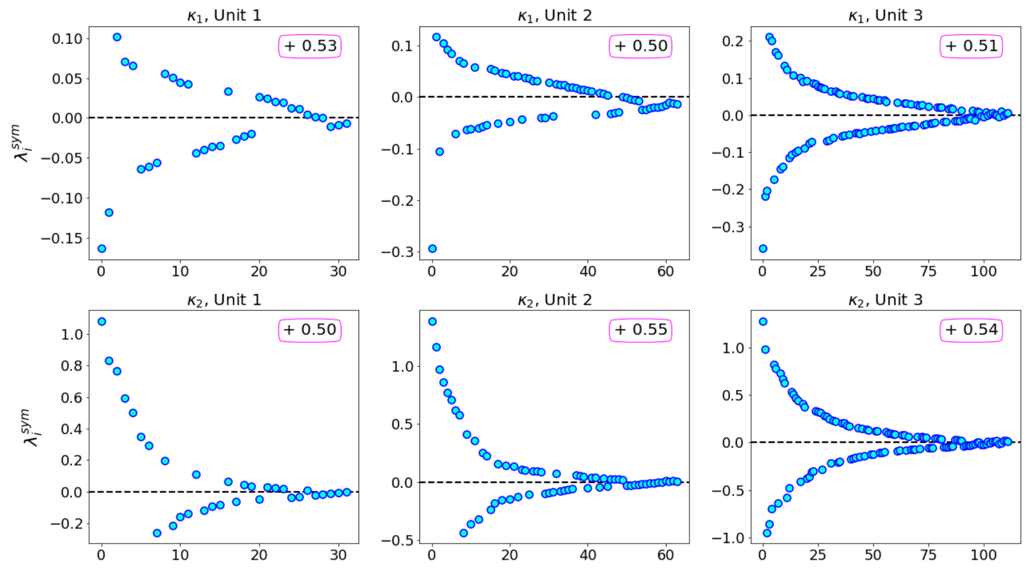

Figure SM6. *Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the nonlocal block placed in each Unit of the pseudo-differential $(-\Delta)^{1/2}$ network while training on STL-10.*

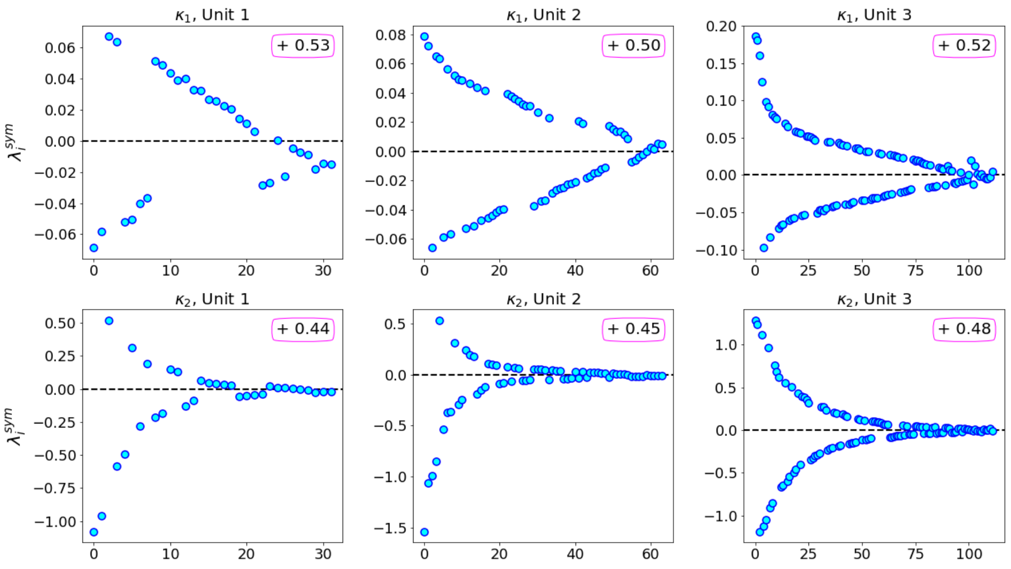

Figure SM7. *Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the nonlocal block placed in each Unit of the pseudo-differential $(-\Delta)^{-1/2}$ network while training on STL-10.*

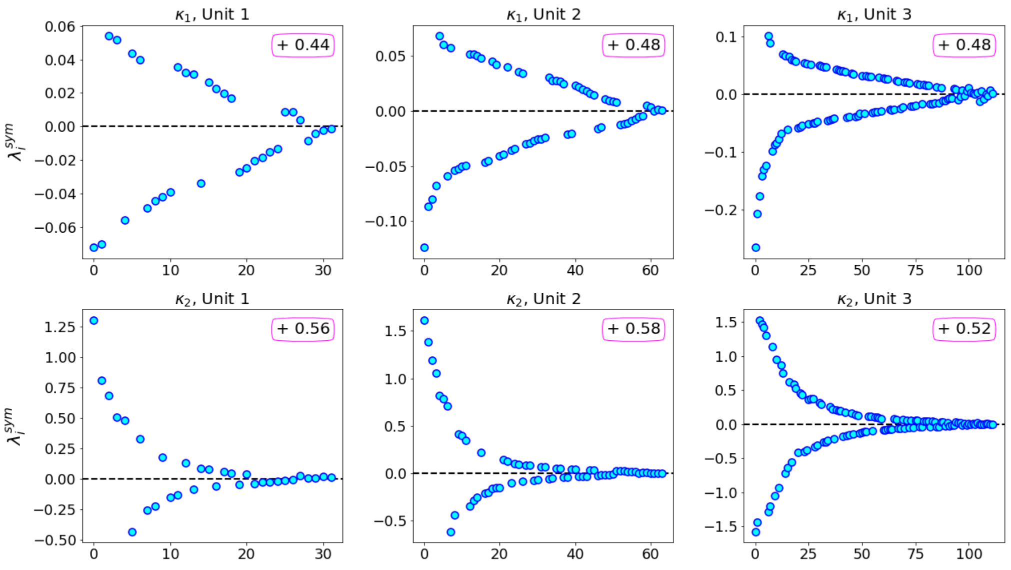

Figure SM8. *Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the nonlocal block placed in each Unit of the pseudo-differential $(-\Delta)^{-1}$ network while training on STL-10.*

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015, https://www.tensorflow.org/. Software available from tensorflow.org.

[2] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, *Reversible architectures for arbitrarily deep residual neural networks*, in 32nd AAAI Conference on Artificial Intelligence, 2018.

[3] A. Coates, A. Ng, and H. Lee, *An analysis of single-layer networks in unsupervised feature learning*, in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, vol. 15, 2011, pp. 215–223.

[4] E. Di Nezza, G. Palatucci, and E. Valdinoci, *Hitchhiker's guide to the fractional Sobolev spaces*, Bulletin des Sciences Mathématiques, 136 (2012), pp. 521–573.

[5] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*, in 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026–1034.

[6] X. Wang, R. Girshick, A. Gupta, and K. He, *Non-local neural networks*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7794–7803.