Deep Neural Networks and PIDE discretizations

Dinesh Kannan

Born 8th October 1995 in Chennai, India

22nd September 2020

Master's Thesis Mathematics

Advisor: Prof. Dr. Michael Griebel

Second Advisor: Dr. Bastian Bohn

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Acknowledgements

I would like to thank my advisors Prof. Dr. Michael Griebel and Dr. Bastian Bohn for suggesting this topic to me and for the fruitful discussions and valuable insights during my work on this thesis. I thank *Institut für Numerische Simulation*, Bonn for providing me with the necessary hardware and software support that was necessary for this thesis. Lastly, I express my gratitude towards my loved ones for their constant support. I hope I can requite the kindness and love that I have received.

Contents

Notations and Abbreviations

Notations

$\langle \cdot, \cdot \rangle_{L^2}$	L^2-inner product
$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of real numbers
$\mathbf{1}$	Matrix with all ones
$\|\cdot\|_2$	L^2 norm
$\|\cdot\|_F$	Frobenius norm
$A_{i,j}$	Entry of the matrix A at the crossing of i-th row and j-th column

Abbreviations

BN	Batch Normalization
CNN	Convolutional Neural Network
DNN	Deep Neural Network
LSTM	Long Short Term Memory
mIoU	Mean Intersection-over-Union
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PIDE	Partial Integro-Differential Equation
ResNet	Residual Neural Network
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

Recent advancements in storage capacities and digitalization of data from all walks of life have made it possible to create and store huge amounts of raw data that is almost impossible to process manually. Deep learning is a class of machine learning methods that has garnered a lot of attention in the last couple of decades in this regard. Deep Neural Networks (DNNs) can be used to train machines to learn from examples, to recognize patterns in the data effectively and to make predictions on unseen examples. They have had a lot of success especially in supervised learning, where the relation between the data and the labels is highly nonlinear [GBC16]. The inputs are transformed through a series of layers with nonlinear transformations that contain trainable weights, and the network progressively extracts higher-level features from the input as the layers get deeper. These deep architectures learn very complex mappings and can represent a wide range of functions. DNN has found several applications in the last decade, such as natural language processing [CW08] and solving partial differential equations (PDEs) [SS18], just to name a few.

Convolutional Neural Networks (CNNs) are a special class of neural networks. They make use of spatial correlations between features by introducing a series of spatial convolution operators into the network with compactly supported stencils/kernels and point-wise nonlinearities, where the trainable kernel weights capture hierarchical patterns. CNNs increase the computational efficiency of the network due to the sparse connections between the features and due to the reduction in the number of weights via parameter sharing. Moreover, in [Zho20], it is argued that CNNs can approximate any continuous function to an arbitrary accuracy if the depth of the neural network is large enough. CNNs were introduced in the 1990s in order to solve character recognition tasks [LBB+98]. However, larger datasets, better algorithms and architectural designs, as well as improvements in computer hardware, have only recently made it possible to train truly deep CNNs. CNNs got widespread recognition in recent years when a deep CNN was used to beat the state-of-the-art architectures in the ImageNet image classification challenge [KSH12]. Over the last decade, CNNs have led to significant improvements in several machine learning tasks, such as image classification [KSH12; SZ15], semantic segmentation [LSD15] and object detection [CKZ+15; RHG+15]. In recent years, there have been several advancements with respect to initialization strategies [HZR+15; GB10], optimization algorithms [COO+18; MS12; BCN18] or changes in the number of channels or filter sizes [SVI+16].

There are several challenges when it comes to designing and training a neural network. Firstly, the training of deeper networks becomes more challenging because the parameter space is high-dimensional. The performance of the DNN is also very much dependent on the choice of the optimization algorithm [BCN18]. For example, gradient descent is a first-order optimization algorithm, which means that it does not take the second derivatives of the cost function into account. Due to the non-convexity of the optimization problem, it is possible that we do not have global optimal solutions because the optimizer is stuck in a local minimum. Secondly, in very deep networks, the features from the earlier layers are 'washed out' through repeated multiplication or convolution with weight matrices, making it hard for the deeper layers to learn the appropriate gradient directions [SGS15b]. Besides, there is the well-known issue of exploding and vanishing gradients [PMB13]. As the gradient information is backpropagated [LBD+89], repeated multiplication or convolution with small weights makes the gradient very small in the earlier layers. To circumvent this problem, several suggestions have been made, for example, the use of Batch Normalization [IS15] or careful initialization [HZR+15; GB10]. Furthermore, the long training time of very deep networks is another persistent issue. The forward and backward passes scale linearly with the depth of the network. Architectures like the ResNet-152 [HZR+16a] require several weeks to converge on the ImageNet dataset [DDS+09], even with several modern GPUs at its disposal.

The performance of a neural network is very much driven by its depth [SZ15]. It is a modern practice to develop deeper and deeper networks that can generalize better on test data. However, stacking together several convolutional layers in conventional feed-forward network architectures typically results in poor propagation. Very often, the accuracy stagnates and degrades subsequently as the network becomes deeper [GB10; SGS15b]. The quantum leap came when He, Zhang, Ren et al. introduced the so-called Residual Neural Networks (ResNets) in [HZR+16a] to alleviate this issue by using identity skip connections as bypassing paths, which made the training hundreds of layers possible and accelerated the convergence of deep networks. Since then, ResNets have been employed in several computer vision tasks, such as semantic segmentation [PHM+17] and object detection [HGD+17].

By introducing a small step size hyperparameter h, the forward propagation of ResNets can be interpreted as an explicit Euler discretization of an initial value problem that is governed by time-dependent nonlinear partial/ordinary differential equations (PDEs/ODEs), where each discretization step in the time domain can be seen as one layer of the ResNet [Wei17; HR17]. With this approach, we have a vast and rich theory of numerical methods and stability theory of PDEs/ODEs at our disposal, which can be used to understand neural networks better. With the help of this continuous interpretation, the learning problem can be recast as a parameter estimation of an inverse problem.

The stability of the forward propagation of a neural network is of prime importance. The output of a neural network or the solution to the underlying PDE/ODE is sensitive to perturbations of the input data that are not perceived by the human eye, which makes a network vulnerable to adversarial attacks [ZAG18; MDFF16]. A network with such instabilities is also harder to train in practice. The forward propagation of the network is stable if the PDE/ODE itself admits stable solutions. Based on this observation, several new architectures have been proposed, which are inspired by stable and well-posed solutions of the underlying PDE/ODE of the neural network [HR17; CMH+18b].

In CNNs, the convolution operation is local, and as a consequence, each layer draws inferences based on the information that is present in a small spatial neighborhood of the image (receptive field) and therefore lacks a holistic view of the feature maps. This is known as the *field-of-view* problem [LLU+16]. Recently, there have been attempts to model the spatial dependencies in an image [JSZ+15]. To make sure that the output layer has a large receptive field and no information of the image is left out while making the inference, modern architectures tend to go deep. By stacking more convolutional and pooling layers, the receptive field is increased, and large parts of the image end up having an indirect influence on the output layer. This is helpful for databases with long-range spatial dependencies.

However, the number of trainable weights increases rapidly with depth, and they impose challenges when using mediocre computational hardware with memory constraints. Thus, the dilemma is whether one should have a very deep network that has a better performance due to its widened receptive field but makes the training harder for any optimizer or to have shallower networks that are easier to train but perform relatively worse.

Objectives and own contributions

The aim of this thesis is to propose neural networks that tackle the problems of stability and field-of-view of a CNN. The Hamiltonian model from [CMH+18b] is used as the base architecture due to its stable forward propagation that preserves information as the features propagate through the layers. Instead of increasing the network's depth or width to improve a neural network's performance, the introduction of nonlocal operators is proposed. Spatial nonlocality is ubiquitous in computer vision, and its presence in the networks has been shown to work well for object detection and video classification tasks [WGG+18].

- Therefore, four integral-based spatially nonlocal operators are proposed that address the issue of field-of-view and reduce the need for deeper networks if one needs better performance. Three of them are pseudo-differential operators, namely fractional Laplacian and inverse fractional Laplacian operators that arise in several problems in physical sciences and applied mathematics [Cap67; Yam12]. The other operator is a global version of the well-known (weighted) Laplacian operator.

- The proposed operators are discretized in a Nonlocal Block and the associated forward propagation is inspired by partial integro-differential equations (PIDEs). These operators, although dense, are implemented efficiently and inserted into the Hamiltonian model such that we have a relatively low extra computational cost and very little increase in the number of trainable parameters. Moreover, they can be inserted into any neural network without any further effort.

- To examine their effectiveness, new network designs are proposed, and the resulting neural networks are then tested on several image classification benchmark datasets and on a synthetic dataset that depicts the need for spatial long-range interactions in a network. Additionally, this thesis demonstrates a real-world application, namely the networks are used for semantic segmentation tasks in autonomous driving.

- Their better efficacy in comparison to the state-of-the-art deeper architectures is studied, and their robustness to noise and perturbations to the input features, along with the networks' performance in the presence of less training data, is assessed.

- The optimal architectural design for the proposed neural networks is investigated. More precisely, the effect of the power of the fractional Laplacian and the inverse fractional Laplacian on the test accuracy and the consequences of over-usage of these integral-based operators are studied.
- Finally, the extra computational costs of the proposed neural networks along with their forward propagation stability is investigated.

Outline of this work

This thesis is organized as follows. In Chapter 2, a general introduction to Convolutional Neural Networks and its building blocks are discussed. The Residual Neural Network (ResNet) is introduced, and its link with stable dynamical systems is reviewed. Then, a few PDE/ODE-based stable CNN architectures are discussed, which have been proposed in the last couple of years. Chapter 3 introduces the PIDE-inspired nonlocal operators, and a few inherent properties of such operators are explored and reviewed. These nonlocal operators are then discretized in Chapter 4, and several strategies to save computational costs are proposed. This chapter also talks about the implementation details for the nonlocal operators and for the proposed CNNs in general. In Chapter 5, the numerical experiments related to image classification and semantic segmentation are conducted, where the proposed networks are put to test, and their effectiveness is examined. Also, we explore subtle changes in the network design and their effects on performance. Chapter 6 deals with the forward propagation stability of the proposed CNNs and the computational cost associated with them. The concluding Chapter 7 summarizes the results and provides a more general outlook on the topic. In the end, some related ideas for future research are mentioned.

Chapter 2

Basic tools in Deep Neural Networks and their stability

In this chapter, we have a brief overview of Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) and several basic notions related to it. We look at the well-known Residual Neural Networks (ResNets) and link it to ordinary and partial differential equations (ODEs/PDEs). Then, we review several stable architectures that have been proposed in this regard. Finally, we discuss a few recent works and results in this field of neural architectures with a dynamical system's viewpoint.

2.1 Deep feedforward networks for classification and segmentation tasks

A supervised learning problem can be seen as a data fitting problem in a high-dimensional space. One such machine learning task is image classification, which involves assigning each element of a dataset to one of the many classes or categories of images. It forms the basis for several other computer vision tasks such as detection, segmentation and localization. Let each d-channel image be of dimension $\mathbb{R}^{h\times w\times d}$, with each image belonging to one of the k classes. Then the objective is to find a classifier map $\mathcal{J} : \mathbb{R}^{h\times w\times d} \mapsto \mathbb{R}^k$ such that for a given $\mathbf{x} \in \mathbb{R}^{h\times w\times d}$, we have

$$\mathcal{J}(\mathbf{x}) \approx e_i \in \mathbb{R}^k,$$

if $\mathbf{x}$ belongs to class i, $1 \leq i \leq k$, e_i being the basis vector with all zeros except at position i. The task of semantic segmentation is similar and can be seen as a multi-dimensional extension of the image classification task. Here, each pixel of each image is supposed to be assigned to one of the many classes (car, bus, etc.), i.e. each pixel gets a meaning so that in the end, individual objects on an image can be distinguished from one another (see section 5.5).

The idea of a Deep Neural Network (DNN) is to emulate the visual cortex [LP16], where simple cells respond to specific edge-like patterns within their receptive field, and complex cells have larger receptive fields and are locally invariant to the exact position of the pattern on the image [HW62]. A DNN has layers of neurons that are connected to each other, and the signal is passed via these connections to adjacent neurons, depending on the state of each neuron. Recent research has shown that DNNs perform really well in modeling and inferring from complex datasets. The uniqueness of DNNs in comparison to other machine learning techniques lies in the fact that simple functions are used to approximate very complex ones.

We will be using *feedforward* networks for supervised learning, where connections between the nodes or neurons do not form a cycle, and the signal passes through in one direction only. Let $\mathbf{x} \in \mathbb{R}^d$ be the input layer to the neural network. Let d_l be the number of nodes in layer l, and let $\mathbf{x}_m^{(l)}$ be the m-th node in the l-th layer, $m = 1, \dots, d_l$. Then the feedforward network can be seen as a network of connections, with each arrow representing the weight with which each node value is multiplied in order to compute the nodes of the next layer (see Figure 2.1). If the values of the l-th layer are given, then the activation of the j-th node in the $l+1$-th layer is given as

$$\mathbf{x}_j^{(l+1)} = \sigma\left(\sum_{m=1}^{d_l} w_{m,j}^{(l)}\ \mathbf{x}_m^{(l)} + b_j^{(l+1)} \right), \tag{2.1}$$

where $w_{m,j}^{(l)}$, $b_j^{(l+1)}$ represent the trainable *weights* and *biases* respectively, and σ is the activation function (see page 10). Such connections can be easily represented using a matrix-vector product.

For classification tasks, the last layer is passed through a softmax function (see section 2.2) to obtain the probability of the input belonging to each of the k classes. Each node in the hidden layers is passed through an activation function (see section 2.2), which makes the model highly nonlinear. This concatenation of nonlinear transformations tries to approximate the classifier map $\mathcal{J}$. To improve this approximation and to make sure that it is as close as possible to $\mathcal{J}$, the network is trained, i.e. a certain loss function is optimized with respect to the trainable weights that connect the neurons, as we will see in section 2.3.2. Such networks extract features from raw data and can learn hidden features and relations between attributes. These relations are later used to predict results on unseen samples. For a detailed summary of deep learning, see [VBG+17]. For image processing, often, not every neuron is connected to another. The convolution operator is used instead to maintain sparsity of connections and nevertheless learn abstract details in an image and make predictions based on it.

2.2 Building blocks of Convolutional Neural Networks

Convolution is one of the most important and fundamental concepts in signal and image processing and analysis. Convolutional Neural Networks (CNNs) are neural networks that use convolutions on data with grid-like topology, like images or videos. CNNs have garnered a lot of interest in recent years. The first real breakthrough came in 2012, when the network AlexNet [KSH12] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by using convolution layers, which was not common at the time. The idea is to replace fully-connected layers in a neural network with convolution filters, whose weights are trainable, and after the neural network has been trained for a certain number of epochs, these filters start to look out for various features in the image. The convolution filters make use of both spatial and channel-wise information that is present within local receptive fields at each layer.

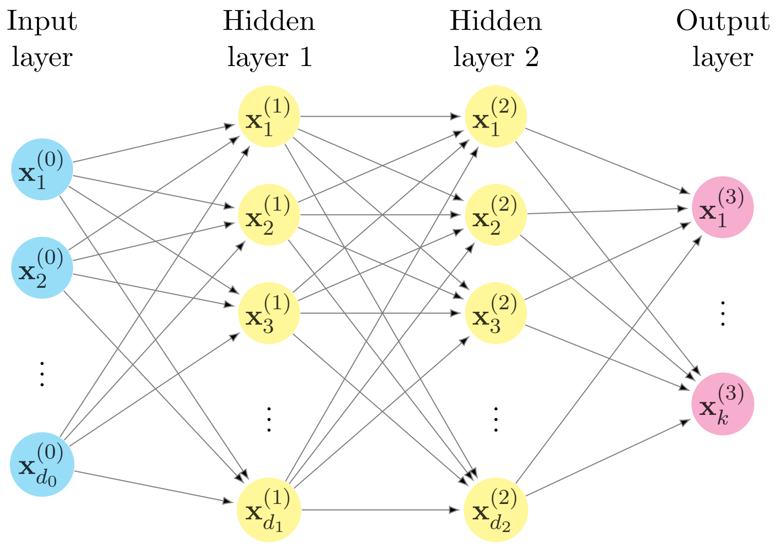

Figure 2.1: A fully-connected 4-layered feedforward neural network.

Convolution

Definition 2.1. Let $\mathbf{x} \in \mathbb{R}^{h\times w}$ be a 1-channel image and let $K \in \mathbb{R}^{f\times f}$ be a convolution filter. Then the per-channel convolution (with stride 1) of $\mathbf{x}$ and K is given by

$$(K * \mathbf{x})_{i,j} := \sum_{p,q=-f}^{f} K_{p+f+1,q+f+1}\, \mathbf{x}_{i+p,j+q} \tag{2.2}$$

for $i = 1, 2, \ldots, h$ and $j = 1, 2, \ldots, w$.

The filter size $f \times f$ is generally much smaller than the actual image size $h \times w$. Here, the negative indices of $\mathbf{x}$ can be treated in various ways. For instance, they can be considered as zeros. Strictly speaking, the operation should be called cross-correlation. Convolution involves an additional step of flipping the kernel. However, in machine learning, the flipping plays no significant role, and therefore both operations are mentioned interchangeably in the literature.

We will be dealing with mainly 3D images, and we use the so-called *2D convolutions* to this end, where we perform channel-wise convolution and add the results depth-wise (see Figure 2.2).

Definition 2.2. Let $\mathbf{x} \in \mathbb{R}^{h\times w\times d}$ be a d-channel image and let $K_l \in \mathbb{R}^{f\times f\times d}$ be c convolution filters, $l = 1, 2, \ldots, c$. Then the 2D convolution of $\mathbf{x}$, expressed by the map $\kappa : \mathbb{R}^{h\times w\times d} \mapsto \mathbb{R}^{m\times n\times c}$, is given by

$$[\kappa(\mathbf{x})]_l := \sum_{i=1}^{d} K_{l,i} * \mathbf{x}_i + b_i\mathbf{1} \tag{2.3}$$

for $l = 1, 2, \ldots, c$, where $\mathbf{x}_i \in \mathbb{R}^{h\times w}$ and $[\kappa(x)]_l$ represent each channel of the input and output respectively, $K_{l,i} \in \mathbb{R}^{f\times f}$ is the i-th channel of filter K_l, as defined in equation

(2.2), b_i is a trainable, per-channel bias parameter, and $\mathbf{1} \in \mathbb{R}^{m\times n}$ is the matrix with all ones. The output of this operation is sometimes called *feature maps*, i.e. each feature map is the output of each of the c filters.

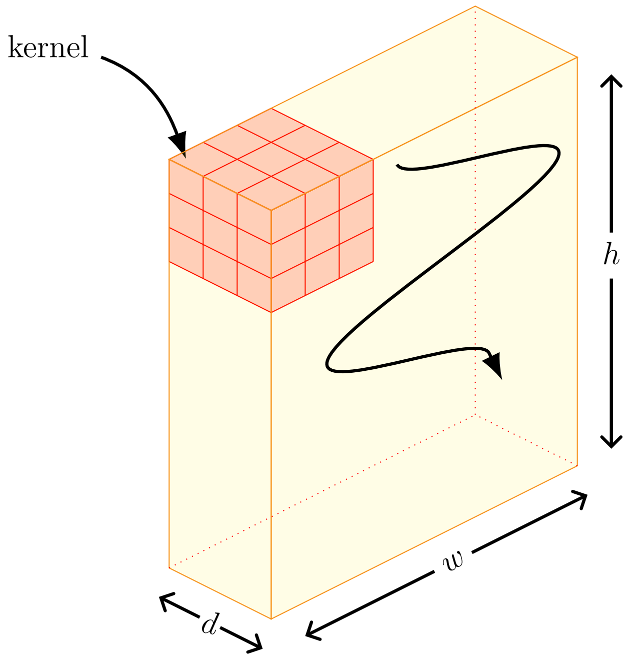

Figure 2.2: Kernel sliding over a 3-channel image.

There are a couple of advantages to this approach. Instead of flattening the image, as is the case in fully-connected layers, we can keep the 2D structure, thereby preserving the local information. Also, instead of each neuron being connected to another, we have the advantage of sparse connections (only a few input neurons contribute to a given output neuron), and kernel weights are instead shared throughout the image. The kernel is much smaller than the image, which makes sure that we have a certain number of trainable parameters that do not depend on the input size. Typically, one uses a stack of such layers, with deeper layers learning more abstract details, like faces, etc., while shallower layers learn edges or patterns in a certain direction.

Similarly, one can define a convolution operation with stride $s > 1$, where the filter f *jumps* or slides over by s pixels (horizontally or vertically) after the computation of each node of the output feature maps. This can be seen as a way of subsampling the image. Moving the kernel by jumps of two pixels, for instance, is equivalent to moving the kernel by jumps of one but retaining only odd output nodes.

Remark. Note that the height and width of the output image are reduced after the convolution. We would be using zero-padding, i.e. padding the input image with a certain number of bands of zeros before the convolution operation so that the output image is of the desired dimensions. Other common padding options include, among others, symmetric and periodic padding.

Such a discrete convolution can be seen as a matrix-vector product. For a 2D convolution, the linear transformation is a *doubly block circulant* matrix. It is a matrix made up of circulant matrix blocks, each of which is also a circulant matrix (see [GBC16]). This is what distinguishes CNNs from classical neural networks that have dense matrices.

Example 2.3. Let $\mathbf{x}$ be a 4×4 image and let $K = \begin{pmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{pmatrix}$ be a 3×3 filter. Then the convolution (with zero-padding) can be represented as

$$K * \widetilde{\mathbf{x}} = M \cdot \widetilde{\mathbf{x}} = \begin{pmatrix} A_2 & A_3 & O & O \\ A_1 & A_2 & A_3 & O \\ O & A_1 & A_2 & A_3 \\ O & O & A_1 & A_2 \end{pmatrix} \cdot \widetilde{\mathbf{x}} \, ,$$

where $O \in \mathbb{R}^{4\times 4}$ is the null matrix, $A_1, A_2, A_3 \in \mathbb{R}^{4\times 4}$ are tridiagonal matrices, i.e. $A_1 = \operatorname{tridiag}(k_1, k_2, k_3)$, $A_2 = \operatorname{tridiag}(k_4, k_5, k_6)$, $A_3 = \operatorname{tridiag}(k_7, k_8, k_9)$. Here, $\operatorname{tridiag}(a, b, c)$ stands for matrices with a, b and c on the subdiagonal, diagonal and superdiagonal respectively and zeros otherwise, and $\widetilde{\mathbf{x}} \in \mathbb{R}^{16}$ is the flattened column vector of the zero-padded version of $\mathbf{x}$ so that the output image is of the same spatial dimensions. For a multi-channel input image, one can similarly just stack up such doubly circulant matrices. Although this matrix representation is easy to implement using the Kronecker product, for example, it is memory inefficient due to the storage of repeated entries of the sparse doubly circulant matrix M. Instead, several more efficient implementations, such as the *im2col* method [VAG17], are used.

Pooling

To make sure that only the most important information propagates through the neural network, and to reduce computational effort at the same time, one performs *pooling*. Essentially, the pooling operation replaces the output of each pixel with a summary of its nearby outputs. For example, max pooling [ZC88] takes the maximum value within a small rectangular patch of the image. Other pooling options include L^1 or L^2 norms on the rectangular patch [SKM+13], average pooling and a stochastic combination of average pooling and max-pooling [YWC+14]. Average pooling, for example, can be seen as a convolution (with stride 2) of the image with a 2×2 convolution filter, i.e.

$$(P * \mathbf{x}) := \frac{1}{4}\left(\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} * \mathbf{x} \right) , \tag{2.4}$$

with P being the pooling kernel. The added benefit of such an operation is that it makes the network invariant under certain transformations, like spatial invariance to input translations. Such invariances are crucial in determining if some features are present in an image or not. It also shortens the training time and combats the problem of overfitting in a network during training, i.e. when the network performs significantly worse on the test examples in comparison to its performance on the training examples.

Batch Normalization

Batch Normalization (BN) [IS15] is a computation to normalize each batch of features by the batch mean and the batch standard deviation. This computation speeds up the training, makes the optimization landscape smoother and makes the neural network more robust to the choice of hyperparameters of the network.

Definition 2.4. Let $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(s)}\}$ be a batch of features. Then the Batch Normalization $\mathcal{B}$ is defined as

$$\mathcal{B}(\mathbf{x}^{(i)}) := \frac{\gamma(\mathbf{x}^{(i)} - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad \text{for} \ \ i = 1, 2, \cdots, s, \tag{2.5}$$

where $\mu = \frac{1}{s}\sum_{i=1}^{s} \mathbf{x}^{(i)}$ and $\sigma^2 = \frac{1}{s}\sum_{i=1}^{s}(\mathbf{x}^{(i)} - \mu)^2$, ϵ is some small value to avoid diving by zero, and γ and β are learnable scale and shift parameters.

Batch normalization adds some noise to each hidden layer's activations, which has a slight regularizing effect. Besides, it makes the training of deeper networks much easier. The network can use a higher learning rate without any major issues, such as the well-documented issue of vanishing or exploding gradients [PMB13]. Although the advantages of Batch normalization are well documented, the reasons for its effectiveness are still being discussed.

Remark. Batch normalization is generally used in training only. The moving average and variance from training are then used on the test data to track the test accuracy.

Activation functions

To introduce nonlinearity in the model and increase the model's expressive power, the output of a node in each layer is passed through a nonlinear *activation* function $\sigma : \mathbb{R} \mapsto \mathbb{R}$. Common activation functions include $\sigma(x) = \tanh(x)$ and the *rectified-linear unit* or ReLU function $\sigma(x) = \max(0, x)$ [NH10]. ReLU function is nearly linear, and hence preserves many of the properties that make optimizing linear models easy. Also, using ReLU leads to a sparsity of connections, with many neurons being zero.

There are two other activation functions that are usually used on the nodes of the last layer of a neural network, namely the sigmoid activation function

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{2.6}$$

which is used for binary classification tasks, and the softmax function $S : \mathbb{R}^k \mapsto \mathbb{R}^k$,

$$[S(x)]_i := \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}, \tag{2.7}$$

which is used for multi-class classification. It returns a probability distribution consisting of k probabilities that are proportional to the entries of the input vector x and are actually the different probabilities of the image/pixel belonging to each of the k classes.

Figure 2.3 shows a typical VGG-type CNN architecture [SZ15] for a k-class image classification task, where the input image is passed through several convolutional layers, followed by pooling. The feature size decreases gradually via pooling, and the number of channels is increased to assist the learning of more abstract features as the network gets deeper. Finally, the feature maps are flattened and connected to fully-connected layer(s) with softmax activation.

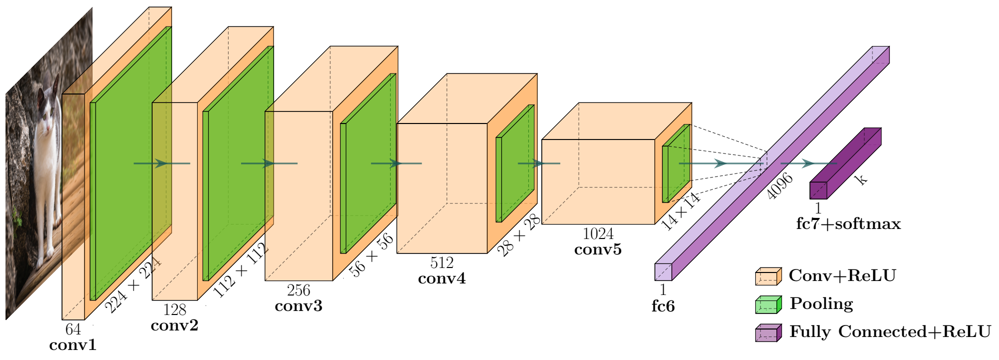

Figure 2.3: A typical VGG-type CNN with increasing number of channels and decreasing spatial feature size.

2.3 Residual Neural Networks and PDEs/ODEs

2.3.1 Residual Neural Networks

The depth of a neural network plays a key role in its performance [SZ15]. Deeper networks offer computational efficiency for more complex tasks [BCV13]. However, the optimization of such a network is considerably more difficult, and there is no guarantee that we have global optimal solutions due to the non-convexity of the optimization problem. When training such deeper and deeper neural networks, often the accuracy rather saturates and then degrades rapidly [SGS15b]. Intuitively, the performance of the deeper networks should ideally be at least as good as that of the shallower networks, if not better. To tackle this problem, [HZR+16a; HZR+16b] came up with the novel idea of identity *skip connections* or *shortcut connections*.

Let $\mathbf{Y}_0$ be the input to the neural network. The forward propagation of a Residual Neural Network (ResNet) with N layers is given by

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathcal{F}(\mathbf{Y}_j, \mathbf{K}_j) \quad \text{for } j = 0, \dots, N-1, \tag{2.8}$$

where $\mathbf{Y}_j$ are the feature values in the j-th layer, $\mathbf{K}_j$ are the learnable (convolutional and BN) weights in the j-th layer, and $\mathcal{F}$ is the residual module that can come in various forms, for example, $\mathcal{F}(\mathbf{Y}, \mathbf{K}) = \sigma(\mathbf{K}\mathbf{Y} + b)$ or $\mathcal{F}(\mathbf{Y}, \mathbf{K}) = -\mathbf{K}_2\sigma(\mathbf{K}_1\mathbf{Y} + b)$, where $\mathbf{K} = \{\mathbf{K}_1, \mathbf{K}_2\}$, i.e. the residual module has two convolutional layers. $\sigma(\cdot)$ is the activation function (assumed to be ReLU here) applied component-wise. So essentially, in each step of the iteration, the residual module tries to learn the update made to $\mathbf{Y}_j$.

Remark. Note that the residual module depends not just on layer weights $\mathbf{K}_j$, but also on layer biases b_j. For simplicity, however, we write $\mathcal{F}(\mathbf{Y}_j, \mathbf{K}_j)$ instead of $\mathcal{F}(\mathbf{Y}_j, \mathbf{K}_j, b_j)$. Also, to be precise, if we were to include all the components of the residual block, the forward propagation in a residual block of ResNet [HZR+16a] could be written as

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \sigma(\mathcal{B}(\mathbf{K}_{j,2}\, \sigma(\mathcal{B}(\mathbf{K}_{j,1}\mathbf{Y} + b_j)))),$$

with $\mathcal{B}$ representing the Batch Normalization (BN) layer. This equation follows the sequence of convolution-BN-ReLU. In a follow-up paper [HZR+16b], He, Zhang, Ren et al. suggest another sequence for computations, namely BN-ReLU-convolution.

Although ResNet has had a lot of success in several competitions such as ILSVRC [RDS+15], Microsoft COCO [LMB+14] and Pascal VOC challenge [EEVG+15], it was not the first to make use of these shortcut connections. In [SGS15a], *Highway Networks* with gated shortcut connections are proposed. It was inspired by Long Short Term Memory (LSTM) recurrent networks [HS97], where the parameterized gates control how much information is allowed to flow across the shortcut. Therefore, ResNets can be seen as a special case of Highway Networks, but tests have shown that Highway Networks perform no better than ResNets.

The success of ResNets has made it possible to train up to hundreds of layers and still achieve compelling performance. Whereas AlexNet [KSH12], GoogLeNet [SLJ+15] and VGG19 [SZ15] have 8, 22 and 19 layers respectively, the ResNet architecture featured 152 layers. Szegedy, Ioffe, Vanhoucke et al. showed in [SIV+17] that residual links in fact speed up the convergence of very deep networks. Consequently, over the past few years, ResNets and residual connections have been used in several fields, not just in classification tasks. These include, among other areas, machine translation [WSC+16], object detection [HZR+16a; HGD+17] and semantic segmentation [PHM+17].

Remark. One thing to remember is that, in equation (2.8), the dimensions of $\mathbf{Y}_j$ and $\mathcal{F}(\mathbf{Y}_j, \mathbf{K}_j)$ must be the same. But in a typical network for image classification and segmentation tasks, the width or the number of feature maps of the network increases as we go deeper into the network since deeper convolutional layers use more filters. To circumvent this issue, several strategies are suggested in [HZR+16a]. When the dimensions increase, one can pad zeros to match dimensions, or one can use 1×1 convolution to increase the number of channels of the skip connection $\mathbf{Y}_j$, which would introduce a few extra training parameters.

2.3.2 PDE/ODE formulation of ResNets

The forward propagation of a ResNet can be seen as a discretization of an underlying nonlinear ordinary differential equation (ODE), and in special cases, as a discretization of an underlying partial differential equation (PDE). Weinan, Haber and Ruthotto were among the first to come up with the idea of adding a step size h to the equation of the forward propagation of ResNets [Wei17; HR17], i.e.

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\mathcal{F}(\mathbf{Y}_j, \mathbf{K}_j) \quad \text{for } j = 0, \dots, N-1. \tag{2.9}$$

This equation could then be interpreted as a forward Euler discretization of the initial value problem

$$\dot{\mathbf{Y}}(t) = \mathcal{F}(\mathbf{Y}(t), \mathbf{K}(t)), \quad \mathbf{Y}(0) = \mathbf{Y}_0, \tag{2.10}$$

with the input layer being $\mathbf{Y}(0)$ and the output layer values $\mathbf{Y}(T)$ being the solution to the initial value problem at time T, with T being related to the depth of the network. Thus, each layer of the network can be seen as the discretization of the continuous solution at discrete time points in $[0, T]$, and the problem of learning the network weights $\mathbf{K}_j$ can be seen as a parameter estimation problem or an optimal control problem that is governed by equation (2.10).

With this interpretation, we have a vast mathematical framework of PDEs/ODEs, which could provide insights regarding the behavior of CNNs. In fact, in [TG18], it has been shown that the variational limit of ResNets results in an ODE system. In [ZHW+19], Zhang, Han, Wynter et al. claim that the presence of the step size h in ResNet-like networks stabilizes the gradients, allows larger learning rates (and hence faster convergence of the training), makes the network more robust to noise, and facilitates training of deeper networks with greater ease. From a dynamical systems viewpoint, the better performance of deeper networks can be attributed to the increase in the number of time steps while integrating an underlying PDE/ODE, which results in a more refined approximation of the continuous-time solution to the PDE/ODE.

After the computation of the final layer $\mathbf{Y}_N$, the results are generally connected to a dense layer, followed by the use of the softmax function while dealing with classification problems. Let the result of this computation be called $\mathbf{Y}_{N+1}$, with $\mathbf{W}$ representing the weights and biases of this extra layer. Let the collection of weights and biases of the other layers be represented as $\mathbf{K} = \{\mathbf{K}_j\}_{j=1}^{j=N}$, $b = \{b_j\}_{j=1}^{j=N}$ respectively. Then the learning problem can be cast as a high-dimensional, non-convex optimization problem

$$\begin{aligned} \min_{\mathbf{K},\mathbf{W},b} \quad & \frac{1}{2} S(\mathbf{Y}_{N+1}, \mathbf{C}) + R(\mathbf{K}, \mathbf{W}, b) \\ \text{such that} \quad & \mathbf{Y}_{j+1} = \mathbf{Y}_j + h\mathcal{F}(\mathbf{Y}_j, \mathbf{K}_j) \quad \text{for } j = 0, \ldots, N-1, \end{aligned} \tag{2.11}$$

where S is the loss function that is convex in the first argument and measures the closeness of the prediction to the ground truth. Usually, S is assumed to be the least-squares function and cross-entropy loss function for regression and classification tasks respectively [GBC16]. $\mathbf{C}$ is the label or the ground truth for the example $\mathbf{Y}_0$ that is propagated through the network, and R is a convex regularizer that penalizes large weights that are undesirable (see section 4.3).

Example 2.5. To demonstrate why deeper networks are necessary, we look at a simple working example. The task is to classify points in $\mathbf{Y}_0 \in \mathbb{R}^3$, with each coordinate chosen uniformly from $[0, 40]$ such that start vectors with $\|\mathbf{Y}_0\| \leq 20$ belong to one class, those with $20 < \|\mathbf{Y}_0\| \leq 40$ belong to another class, and the rest of the points belong to the third class. A given random start vector $\mathbf{Y}_0 \in \mathbb{R}^3$ is propagated through the network governed by the equation

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\mathbf{K}^T \sigma(\mathbf{K}\mathbf{Y}_j) \quad \text{for } j = 0, \ldots, N-1,$$

where the weight matrix $\mathbf{K} = \begin{pmatrix} \theta_1 + \theta_2 & -\theta_1 & -\theta_2 \\ -\theta_2 & \theta_1 + \theta_2 & -\theta_1 \\ -\theta_1 & -\theta_2 & \theta_1 + \theta_2 \end{pmatrix}$ is kept constant over the

N layers, θ_1, θ_2 are the two trainable parameters, and tanh is used as the activation function. The forward propagation is performed for two cases, namely $h = 1, N = 10$ and $h = 0.05, N = 200$. This means that we are integrating the underlying ODE to a final time $T = 10$ in both cases, with a different number of integration steps, i.e. once with a shallower network and once with a deeper one. After the final layer, the softmax function is applied to $\mathbf{Y}_N$ to obtain $\mathbf{Y}_{N+1}$. The objective is to minimize the loss

$$\min_{\theta_1,\theta_2} \frac{1}{2}\left\|\mathbf{Y}_{N+1}(\theta_1, \theta_2) - \mathbf{C}\right\|_2^2,$$

with $\mathbf{C}$ being the *one-hot* encoded vector, which states the class to which $\mathbf{Y}_0$ belongs. The loss landscape with respect to the parameters in this simple example is shown below (Figure 2.4). The plot shows that the deeper networks lead to a smoother loss landscape, which is often necessary for stability during the forward propagation and, of course, a smoother landscape is easier to optimize via backpropagation [LBD+89].

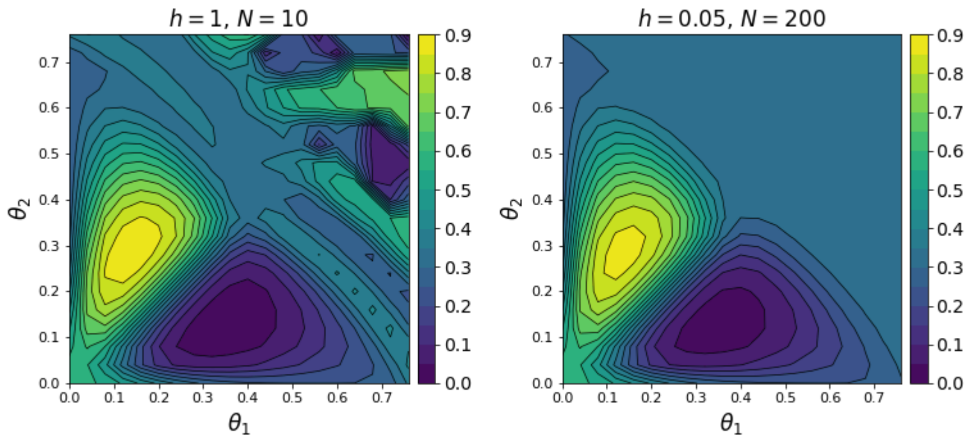

Figure 2.4: Loss landscape with respect to θ_1, θ_2 for different depths of the network.

Convolution as partial differential operators

With the introduction of the dummy time variable, we have a time derivative in the forward propagation. In special cases, such as image classification or segmentation, one mainly has spatial convolution weights $\mathbf{K}_j$ for each layer, instead of dense weight matrices. In such cases, the convolution weights can be expressed as a linear combination of spatial differential operators or as a coupled system of partial differential operators in case of multi-channel convolutions [RH20]. As a result, in the case of CNNs, equation (2.8) is a forward Euler discretization (in the time domain) of an underlying partial differential equation due to the inherent presence of spatial derivatives in the residual function $\mathcal{F}$. The CNN layer weights determine the order, type and stability of the underlying PDE. This connection motivates us to use the powerful tools of PDEs that have been used for the last couple of decades in image segmentation [MS89], registration, filtering [PM90; Wei98] and restoration [AK06]. Since we are dealing with image classification and segmentation, which involves convolutional layers, we refer to the ODE and PDE interpretation interchangeably, but we keep in mind that the convolution operation has an inherent spatial aspect to it.

2.4 Stable architectures for PDE/ODE-based ResNets

When one has to solve the inverse dynamic problem (2.11), the first question that comes to mind is if the forward propagation through the network is stable, i.e. if the output of the network varies continuously with respect to its input. This is because, in a network with unstable forward propagations, the network is sensitive to perturbations in the input image. Similar images might propagate through the network and yield vastly different outputs and predictions [GSS15], leaving the network vulnerable to adversarial attacks, and therefore the network generalizes poorly on test data. Also, such instabilities make the network harder to train via backpropagation. The well-known problem of exploding and vanishing gradients [PMB13] is also related to the stability of the forward propagation, with exploding gradients suggesting that the output of the network is highly sensitive to the inputs, and vanishing gradients indicating the insensitivity of the output with respect to the input. To remedy this, we look into the stability of underlying PDEs/ODEs and link it to the stability of the forward propagation of the network.

2.4.1 Stability of PDEs/ODEs

Definition 2.6 ([RH20]). Let $\mathbf{Y}(T, \mathbf{K}(T))$ be the solution to the initial value problem (2.10), with the initial value $\mathbf{Y}(0) = \mathbf{Y}_0$. Let $\widetilde{\mathbf{Y}}(T, \mathbf{K}(T))$ be another solution with the initial value $\widetilde{\mathbf{Y}}(0) = \widetilde{\mathbf{Y}}_0$. Then the given PDE/ODE is stable if there exists an $M > 0$ independent of the final time T such that

$$\|\mathbf{Y}(T, \mathbf{K}(T)) - \widetilde{\mathbf{Y}}(T, \mathbf{K}(T))\|_F \leq M \|\mathbf{Y}_0 - \widetilde{\mathbf{Y}}_0\|_F, \tag{2.12}$$

where $\|\cdot\|_F$ is the Frobenius norm.

Theorem 2.7 ([AP98]). *The solution to the PDE/ODE* (2.10) *is stable if* $\mathbf{K}(t)$ *changes sufficiently slowly and*

$$\max_i Re(\lambda_i(\mathbf{J}(t))) \leq 0, \quad \forall t \in [0, T], \tag{2.13}$$

where $Re(\cdot)$ *is the real part and* $\lambda_i(\mathbf{J}(t))$ *is the* i*-th eigenvalue of the Jacobian (with respect to* $\mathbf{Y}(t)$*) of the right-hand side of equation* (2.10).

Lemma 2.8 ([AP98; Asc08]). *The forward propagation, given by the forward Euler discretization* (2.9)*, is stable if*

$$\max_i |1 + h\lambda_i(\mathbf{J}_j)| \leq 1, \quad \forall j = 0, 1, \dots, N-1, \tag{2.14}$$

where $\mathbf{J}_j$ *stands for the Jacobian of the right-hand side of equation* (2.9) *at the* j*-th time step.*

2.4.2 ODE- and PDE-based neural architectures

There have been several approaches to enforce stability in a neural network, such as Lipschitz regularization [FCA+18] and weight normalization [MKK+18]. Using the interpretation of ResNets as a discretization of a PDE/ODE, Chang, Meng, Haber et al. suggest several stable neural architectures in [CMH+18b], which are based on the stability theory of PDEs/ODEs. They do this by making structural changes in the forward propagation and by restricting the space of CNNs in a way such that the underlying PDE/ODE obeys Theorem 2.7, which is satisfied if the real parts of the eigenvalues of the Jacobian of the right-hand side of the PDE are negative. To be more precise, we want the eigenvalues of the Jacobian of the right-hand side of any PDE to be imaginary because, as Haber and Ruthotto pointed out in [HR17], while Theorem 2.7 is necessary for the stability of the forward propagation, it is not sufficient for a well-posed learning problem that is given by equation (2.11). Positive eigenvalues amplify the signal in a network, whereas negative eigenvalues might lead to damping, and the energy in the system is dissipated over time. Both are related to the vanishing and exploding gradient issue in neural networks or correspondingly related to the over-sensitivity/insensitivity of the output features with respect to the input features. Hence, we would like to keep the real parts of the eigenvalues of the Jacobian as close to zero as possible. Using this idea, three new architectures have been proposed in [CMH+18b] that make this possible by design.

Hamiltonian network

By introducing an extra augmented variable and two sets of convolution filters $\mathbf{K}_1, \mathbf{K}_2$, the forward propagation can be interpreted as a Hamiltonian system, where we have a system of PDEs

$$\begin{aligned} \dot{\mathbf{Y}}(t) &= \mathbf{K}_1^T(t)\, \sigma(\mathbf{K}_1(t)\mathbf{Z}(t) + b_1(t)), \\ \dot{\mathbf{Z}}(t) &= -\mathbf{K}_2^T(t)\, \sigma(\mathbf{K}_2(t)\mathbf{Y}(t) + b_2(t)). \end{aligned} \tag{2.15}$$

Here, $\mathbf{Y}(t)$ and $\mathbf{Z}(t)$ are channel-wise partitions of the input features (see Figure 2.5), and for our purposes of image classification and segmentation, $\mathbf{K}_i$ is the convolution operator, and $\mathbf{K}_i^T$ is its transpose. One can rewrite equation (2.15) as

$$\begin{pmatrix} \dot{\mathbf{Y}}(t) \\ \dot{\mathbf{Z}}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{K}_1^T(t) & 0 \\ 0 & -\mathbf{K}_2^T(t) \end{pmatrix} \sigma\left(\begin{pmatrix} 0 & \mathbf{K}_1(t) \\ \mathbf{K}_2(t) & 0 \end{pmatrix} \begin{pmatrix} \mathbf{Y}(t) \\ \mathbf{Z}(t) \end{pmatrix} + \begin{pmatrix} b_1(t) \\ b_2(t) \end{pmatrix} \right) \tag{2.16}$$

The eigenvalues of the Jacobian matrix of the right-hand side of the equation above can be shown to be all imaginary [CMH+18b], thereby making the forward propagation stable and well-posed for all choices of the (convolutional) weights that the optimizer chooses while solving equation (2.11), where this new system of PDEs governs the forward propagation. Besides, the Hamiltonian systems are known to conserve energy. So the forward propagation, in this case, is expected to only moderately amplify or damp the features, as it is propagated down the network.

The discretization of equation (2.15) is done using the Verlet integration since such symplectic methods capture the long time features of Hamiltonian systems really well [Asc08]. With the Verlet scheme, we arrive at the following pair of coupled equations:

$$\begin{aligned} \mathbf{Y}_{j+1} &= \mathbf{Y}_j + h\, \mathbf{K}_{j1}^T\, \sigma(\mathbf{K}_{j1}\mathbf{Z}_j + b_{j1}), \\ \mathbf{Z}_{j+1} &= \mathbf{Z}_j - h\, \mathbf{K}_{j2}^T\, \sigma(\mathbf{K}_{j2}\mathbf{Y}_{j+1} + b_{j2}). \end{aligned} \tag{2.17}$$

This computation is placed in a Hamiltonian Block, as shown in Figure 2.5. $\mathbf{X}_j$ is assumed to be the input to the Hamiltonian Block, $\mathbf{Y}_j, \mathbf{Z}_j$ are channel-wise partitions of $\mathbf{X}_j$, and $\mathbf{X}_{j+1}$ is the output of the Block that is obtained by the channel-wise concatenation of $\mathbf{Y}_{j+1}$ and $\mathbf{Z}_{j+1}$.

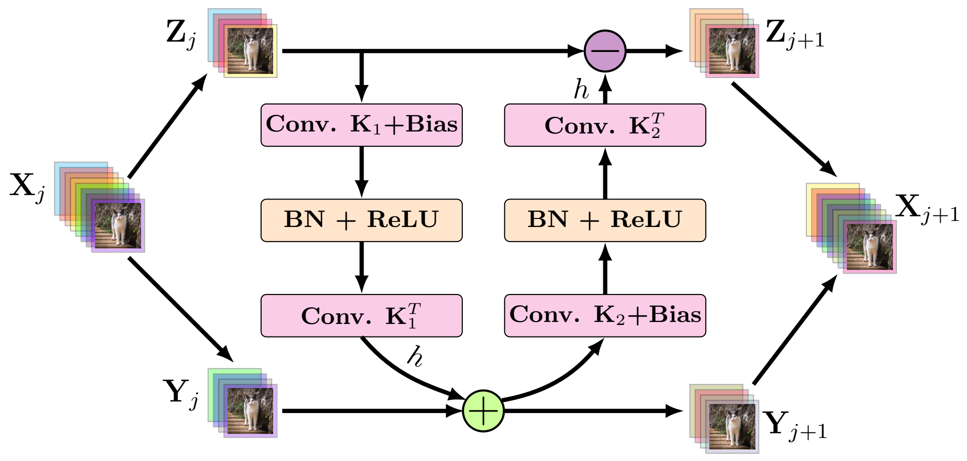

Figure 2.5: The Hamiltonian Block with channel-wise partition and concatenation.

Leapfrog network

In a special case of Hamiltonian networks, the PDE from equation (2.15) can also be discretized using the symplectic leapfrog scheme, where one of the kernels is an identity matrix and one of the activation functions is an identity function. In such a case, we get a second-order initial value problem

$$\ddot{\mathbf{Y}}(t) = -\mathbf{K}^T(t)\,\sigma(\mathbf{K}(t)\mathbf{Y}(t) + b(t)), \quad \mathbf{Y}(0) = \mathbf{Y}_0, \tag{2.18}$$

which is inspired by the system of coupled PDEs/ODEs for a Hamiltonian system, with one variable being eliminated. It can be shown that for this PDE, the eigenvalues of the Jacobian of the right-hand side are imaginary since this is just a special case of the Hamiltonian network, which makes the forward propagation stable. Using the leapfrog discretization, we have the discretized solution

$$\mathbf{Y}_{j+1} = \begin{cases} 2\mathbf{Y}_j - h^2\,\mathbf{K}_j^T\,\sigma(\mathbf{K}_j\mathbf{Y}_j + b_j), & j = 0, \\ 2\mathbf{Y}_j - \mathbf{Y}_{j-1} - h^2\,\mathbf{K}_j^T\,\sigma(\mathbf{K}_j\mathbf{Y}_j + b_j) & j > 0. \end{cases} \tag{2.19}$$

Midpoint network

The right-hand side of equation (2.10) can be altered to contain an anti-symmetric transformation

$$\mathcal{F}(\mathbf{Y}, \mathbf{K}) = \sigma((\mathbf{K} - \mathbf{K}^T)\mathbf{Y} + b).$$

Since the eigenvalues of an anti-symmetric matrix are always imaginary, the well-posedness and stability of the forward propagation are maintained (see [CCH+19]). The equation can be discretized using the central finite difference scheme

$$\frac{\mathbf{Y}_{j+1} - \mathbf{Y}_{j-1}}{2h} = \sigma((\mathbf{K} - \mathbf{K}^T)\mathbf{Y}_j + b), \tag{2.20}$$

which, when rearranged, can be written as

$$\mathbf{Y}_{j+1} = \begin{cases} 2h\sigma((\mathbf{K}_j - \mathbf{K}_j^T)\mathbf{Y}_j + b_j), & j = 0, \\ \mathbf{Y}_{j-1} + 2h\sigma((\mathbf{K}_j - \mathbf{K}_j^T)\mathbf{Y}_j + b_j), & j > 0. \end{cases} \tag{2.21}$$

Remark. This network is very similar to the *AntisymmetricRNN* suggested in [CCH+19]. There, Chang, Chen, Haber et al. use the same anti-symmetric matrices to maintain forward stability of the RNN but with forward Euler discretization instead of the central differencing scheme. At the same time, they introduce a tiny diffusion term to the right-hand side of equation (2.21) to make sure that the forward Euler scheme stays stable, thereby satisfying Lemma 2.8. Also, note that the Midpoint and the Leapfrog network have a *single* layer only, i.e. at each iteration, we perform a convolution and a transposed convolution operation, whereas, in the Hamiltonian Block (Figure 2.5), we have a *two-layer* block, i.e. convolution and transposed convolution are performed twice each.

Reversibility of the Networks

The three networks described above are reversible at least algebraically. This means that, given the activations of the last couple of layers, one could compute the activations of the shallower layers. Therefore, while performing backpropagation, one does not need to store the activations of most of the hidden layers, and hence, the network has a relatively small memory footprint. This, of course, increases the number of floating-point operations performed during training. The trade-off is not so huge, and a lot of the machine learning problems are anyway bound by memory constraints and not by computational constraints. For example, the Leapfrog method can be back-computed if we have the activations of the last two layers:

$$\mathbf{Y}_j = 2\mathbf{Y}_{j+1} - \mathbf{Y}_{j+2} - h^2\,\mathbf{K}_{j+1}^T\,\sigma(\mathbf{K}_{j+1}\mathbf{Y}_{j+1} + b_{j+1}), \tag{2.22}$$

for $j = N-2, N-3, \dots, 0$. Similarly, if one rewrites equation (2.17) as

$$\begin{aligned} \mathbf{Y}_{j+1} &= \mathbf{Y}_j + \mathcal{G}(\mathbf{Z}_j), \\ \mathbf{Z}_{j+1} &= \mathbf{Z}_j + \mathcal{H}(\mathbf{Y}_{j+1}), \end{aligned} \tag{2.23}$$

then for $j = N-1, N-2, \dots, 0$, the algebraically reversible equation reads

$$\begin{aligned} \mathbf{Z}_j &= \mathbf{Z}_{j+1} - \mathcal{G}(\mathbf{Y}_{j+1}), \\ \mathbf{Y}_j &= \mathbf{Y}_{j+1} - \mathcal{H}(\mathbf{Z}_j). \end{aligned} \tag{2.24}$$

As shown in [CMH+18b], although the network might be algebraically reversible, one needs to have restrictions on the residual functions $\mathcal{G}, \mathcal{H}$. To this end, the hyperparameter h plays a crucial role in making sure that the backward propagation is numerically stable. Besides, there have been a few other suggestions on reversibility in ResNets. A reversible variant of ResNet is suggested in [GRU+17], but the network lacks the stable forward propagation discussed above. Note that the reversibility can only be performed between pooling operations, i.e. on a per-Unit basis (see Figure 4.1). Recently, in [LHP19], an invertible pooling operator was suggested to make the CNN completely reversible.

2.5 Related network architectures and recent work

Over the last couple of years, the ResNet has been widely researched, and several improved versions have been proposed. Xie, Girshick, Dollár et al., in [XGD+17], proposed a modified architecture (*ResNeXt*), where they propose wider networks instead of deeper ones, with multi-branch convolutions and aggregated residuals. It is a bit similar to the well-known multi-branch Inception module [SLJ+15; SVI+16; SIV+17], where results of convolutional layers of different receptive field sizes are concatenated. In [HLVDM+17], the so-called *DenseNet* is proposed. Here, all layers with the same feature map size are connected with each other, i.e. more than one identity skip connection is used, and the input to each layer consists of feature maps (of similar size) of all the earlier layers. This supposedly enables feature reuse, making the network parameter-efficient. In [ZK16], Zagoruyko and Komodakis propose a new network (Wide ResNet), where they add more filters in each layer of the ResNet to increase the width of the network.

One drawback of ResNets is that very deep networks take a lot of time to train. To tackle this issue, in [HSL+16], the idea of randomly dropping layers of a ResNet during training is introduced, but the entire network is used during test time. This way, ResNets can be interpreted as an ensemble of networks with varying depths. This allows better information and gradient flow in deeper networks. It reduces training time substantially and improves the test error, which shows that there is a certain amount of redundancy in deeper networks. At the same time, this adds to the many benefits of skip connections because removing a layer from a traditional architecture such as VGG [SZ15] leads to a huge loss in performance (see [VWB16]). The dropping of ResNet layers can also be seen as skipping one time step in the discretization of the underlying PDE/ODE and has been studied in [CMH+18a]. Furthermore, in the same paper, a way of accelerating the training of ResNets was proposed, i.e. a *multi-level* strategy, where we first start with a shallow network, and step by step, more and more residual blocks are added, with the trained weights of the shallow network being prolongated to the deeper network. This way, one has a good starting guess for the newly added residual blocks in the deeper network, and it has shown to save training time, in comparison to the training time for networks that are quite deep from the beginning itself.

This idea of viewing DNNs as a discretization of a dynamical system or as a solution to an optimal control problem is currently being widely researched and has been used for a variety of learning tasks. For example, in [LCT+18], the training is cast as a control problem, and the necessary optimality conditions in continuous time are formulated using the Pontryagin's maximum principle. In [SM17; LS17a], a continuous flow model for ResNet is proposed, with the data flow being seen as the solution of the transport equation along the characteristic line. The relation between ResNets and the control problem of PDEs on manifolds is explored in [LS17b]. On the other hand, a more sophisticated differential equation solver is used in [CRB+18] to solve the forward propagation problem. Zhang and Schaeffer, in [ZS19], interpret ResNets as an optimal control problem with differential inclusions and provide several continuous-time stability results. In [Ces10], the neurons are modeled as discrete dynamical systems, and the effects of synaptic plasticity are studied.

The ODE/PDE interpretation of ResNets has also been used to train and test ResNets on different image resolutions [HRH+18]. Networks that were trained on low-resolution images, for instance, can be tested on high-resolution images and vice versa. The dynamical system view has also made its way into Recurrent Neural Networks (RNN). In [CCH+19], the same antisymmetric weight matrices are used to stabilize the network, which in turn helps in capturing long-term dependencies and outperforms the traditional LSTM models.

Even within the PDE/ODE framework, several discretization methods have been researched so far. Since a higher-order numerical scheme can have a lower truncation error, which leads to better accuracy, implicit and explicit Runge-Kutta methods [AP98] have been proposed as a discretization scheme for the forward propagation [ZCF18]. In [HLT+19], Haber, Lensink, Treister et al. introduce a semi-implicit discretization method [AP98] to accelerate the communication between the pixels, which is crucial in learning long-range correlations in an image. This has shown to work well for problems, where the output feature is high-dimensional, for example, in image segmentation or depth prediction in images. Reshniak and Webster go one step further in [RW19] and work with fully implicit residual blocks that are defined as fixed points of certain nonlinear transformations. This stabilizes the forward and backward propagation of the network. In [YWL+19], Yang, Wu, Li et al. propose an adaptive time-stepping scheme that depends on the parameters of the current step. Lastly, Lu, Zhong, Li et al., in [LZL+18], show that ResNets and other network architectures such as PolyNet [ZLCL+17], FractalNet [LMS16] and RevNet [GRU+17] can all be seen as different time-stepping schemes of the underlying differential equation that governs the forward propagation. Furthermore, they use the linear multistep (LM) numerical scheme for ODEs in their model. Moreover, one can view the Midpoint and Leapfrog discretizations as special cases of the LM-architecture. A summary of the different architectures and their discretization schemes is provided in Table 2.1 below.

Network	**Forward propagation**	**Numerical scheme**
ResNet [HZR+16a]	$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathcal{F}(\mathbf{Y}_j)$	Forward Euler
RevNet [GRU+17]	$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathcal{F}(\mathbf{Z}_j)$ $\mathbf{Z}_{j+1} = \mathbf{Z}_j + \mathcal{G}(\mathbf{Y}_{j+1})$	Forward Euler[1]
PolyNet [ZLCL+17]	$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathcal{F}(\mathbf{Y}_j) + \mathcal{F}(\mathcal{F}(\mathbf{Y}_j))$ $\approx (1 - h\mathcal{F})^{-1}\mathbf{Y}_j$	Backward Euler[2]
FractalNet [LMS16]	$f_{j+1}(\mathbf{Y}) = [(f_j \circ f_j)(\mathbf{Y})] \oplus [\mathrm{conv}(\mathbf{Y})]$	Runge-Kutta[3]
AntisymmetricRNN [CCH+19]	$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma((\mathbf{K}_j - \mathbf{K}_j^T)\mathbf{Y}_j + \mathbf{V}_j\mathbf{X}_j + b_j)$	Forward Euler[4]
LM-ResNet [LZL+18]	$\mathbf{Y}_{j+1} = (1 - k_j)\mathbf{Y}_j + k_j\mathbf{Y}_{j-1} + \mathcal{F}(\mathbf{Y}_j)$	Linear multistep[5]

Table 2.1: Discretization schemes for various neural architectures.

[1]Forward Euler for a coupled system of ODEs
[2]PolyNet can be seen as a one-step approximation of the Backward Euler scheme
[3]$\oplus$ represents a join operation that merges the result of two convolution layers
[4]$\mathbf{X}_j$ is the input to the RNN at time step j
[5]$k_j \in \mathbb{R}$ is a trainable parameter

Chapter 3

Nonlocal and pseudo-differential operators for Deep Learning

In a Convolutional Neural Network, the pooling operation has another important role to play. The pooling or subsampling of the image and the convolution operations allow far-away pixels in an image to communicate with each other, thereby exploiting long-range correlations in an image and improving the predictions made by the network. At the same time, convolution is a local operation, and therefore each convolutional layer's values depend only on the information present in certain local patches of the image. This is known as the *field-of-view* problem [LLU+16].

In fully-connected networks, the value of each neuron depends on the entire input to the network. On the other hand, the value of a neuron in a convolutional layer only depends on a certain subset of the input. This region in the input is the receptive field for that neuron. In learning tasks with images, each neuron in the output layer should ideally have a big receptive field, i.e. the values of the output neurons should indirectly depend on large parts of the image so that no important information of the image is left out when making the predictions. The receptive field size of a neuron can be increased, for example, by stacking more convolutional layers and thereby making the network deeper. This increases the receptive field size linearly in theory because each extra layer increases the receptive field size by the filter size. Meanwhile, pooling or subsampling allows each pixel to cover a larger area and facilitates information travel over larger distances, which increases the receptive field size multiplicatively. This also justifies the initial successes of AlexNet [KSH12], where convolutional and pooling layers were used in combination on a large scale for the first time. Most modern architectures have a combination of both convolutional and pooling layers to widen the field-of-view. For an empirical study on receptive fields, see [ZKL+15].

Just stacking up convolutional and pooling layers can have several disadvantages. In a deeper network, for example, the optimization problem (2.11) might be hard to solve and repeatedly performing local operations can be computationally inefficient because, as pointed out before on page 14, from a dynamical systems view, the convolution operations can be seen as a linear combination of spatial differential operators, and are hence local.

Long-range and nonlocal dependencies are ubiquitous in several fields, such as financial markets [Con05], video processing [BST+95] or physical sciences [DGL+12; CL06; WA05].

In image processing, nonlocal methods can be found, for example, in image denoising [BCM05a; DFK+07; BCM05b], image regularization [GO07] or image recovery [LZO+10]. Even in the area of deep learning, applications such as natural language processing use LSTM modules [HS97] to model long-range dependencies. In [WGG+18], a nonlocal operation is suggested that has been shown to work well for video classification. In section 6.2, we will see a few drawbacks of it, in comparison to the ones proposed in this thesis.

We now look at a few integral-based spatially nonlocal operators that could be used in CNNs to address this issue. The underlying and associated partial integro-differential equation (PIDE) is then discretized in a so-called Nonlocal Block and inserted in the neural network. By direct computation of interactions between each pair of pixels or pair of patches in an image, these operators make sure that the field-of-view increases drastically. Consequently, the information travels larger distances for a fixed number of layers in a neural network. This allows the network to learn global features from each example of the dataset. In other words, to achieve a certain given amount of accuracy on the test data, one needs fewer layers in the network since the far-apart pixels communicate and learn long-range dependencies in the image quicker. We will later see in section 4.2 that these operators, although dense, can be designed in a way so that we have relatively low computational costs.

3.1 The nonlocal diffusion operator

Definition 3.1. Let $\Omega = \mathbb{R}^n$ and let $u : \mathbb{R}^n \to \mathbb{R}^n$ be any function, then for $\mathbf{x}, \mathbf{y} \in \Omega$, the nonlocal diffusion operator can be defined as

$$\mathcal{L}u(\mathbf{x}) := \int\limits_{\Omega} \omega(\mathbf{x}, \mathbf{y})[u(\mathbf{y}) - u(\mathbf{x})]\, d\mathbf{y}, \tag{3.1}$$

where the kernel ω has the following two properties:

(i) ω is symmetric: $\omega(\mathbf{x}, \mathbf{y}) = \omega(\mathbf{y}, \mathbf{x}), \quad \forall \mathbf{x}, \mathbf{y} \in \Omega$,

(ii) ω is positivity-preserving: $\omega(\mathbf{x}, \mathbf{y}) \geq 0, \quad \forall \mathbf{x}, \mathbf{y} \in \Omega$.

The kernel can be viewed as a function that measures the affinity or similarity between points in Ω. The aim is to apply such an operator on a discrete image u, where $u : \Omega \to \mathbb{R}^c$ is the discrete image with c channels, and $u(\mathbf{x})$ is the set of pixel values of all the channels at a discrete spatial point $\mathbf{x} \in \Omega$ from the image domain Ω (the pixel strips in Figure 4.2). For details regarding the implementation of this operator, see section 4.2.

Note that this is just the global analogue of a local Laplacian operator. The Laplacian is a differential operator, and hence local. Each output is dependent on its neighboring values and not on all the values in the domain. Therefore, in a local setting, for each value $\mathbf{x} \in \Omega$, the integral in equation (3.1) is computed over a subset of the domain Ω and not over the entire domain.

Such operators have been an upshot of several image processing tasks. For example, for $\mathbf{x}, \mathbf{y} \in \Omega$, consider the functional

$$J(u) := \frac{1}{4} \int\limits_{\Omega \times \Omega} \omega(\mathbf{x}, \mathbf{y})[u(\mathbf{x}) - u(\mathbf{y})]^2\, d\mathbf{x}\, d\mathbf{y}, \tag{3.2}$$

which sums up weighted variations of the function u (similar total variational denoising in image processing). While minimizing the functional, the corresponding Euler-Lagrange descent flow is given by

$$u_t(\mathbf{x}) = -J'(u)(\mathbf{x}) = -\int\limits_{\Omega} \omega(\mathbf{x},\mathbf{y})[u(\mathbf{x}) - u(\mathbf{y})]\, d\mathbf{y} = \mathcal{L}u(\mathbf{x}). \tag{3.3}$$

Therefore, starting from an initial input image $u_0(\mathbf{x})$, the associated partial integro-differential equation (PIDE) can be written as

$$u_t(\mathbf{x},t) = \mathcal{L}u(\mathbf{x}), \quad u(\mathbf{x},0) = u_0(\mathbf{x}), \tag{3.4}$$

which is essentially a nonlocal weighted linear diffusion equation. The kernel function $\omega : \Omega \times \Omega \to \mathbb{R}$ is here assumed to be of Hilbert-Schmidt type [RR06], i.e.

$$\int\limits_{\Omega}\int\limits_{\Omega} |\omega(\mathbf{x},\mathbf{y})|^2\, d\mathbf{x}\, d\mathbf{y} < \infty. \tag{3.5}$$

3.1.1 Interpretations of the nonlocal diffusion operator

There are several interpretations of the nonlocal diffusion operator that arise from different mathematical applications. We provide two of them below.

Weighted graph Laplacians

The nonlocal diffusion operator can be seen as a continuous generalization of graph Laplacians. Let $G = (V, E)$ be a connected undirected weighted graph with finite vertex and edge sets V and E respectively. Each pair of vertices $l, k \in V$ is connected by an edge $e_{k,l} \in E$ with a weight $w_{k,l}$. The weights are symmetric and positive, i.e. $w_{k,l} = w_{l,k}$ and $w_{k,l} \geq 0$, $\forall l, k$. The discrete function $u : V \to \mathbb{R}$ represents the value of each vertex, i.e. $u(k) \in \mathbb{R}$ is the value of u at vertex k. Then the weighted graph Laplacian [CG97] is defined as

$$\Delta_G(u(k)) := \sum_{l \in \rho(k)} w_{k,l}\,(u(l) - u(k)), \quad \text{for } k, l \in V, \tag{3.6}$$

where $\rho(k)$ is the set of vertices that are connected by an edge to the vertex k. The corresponding weighted Laplacian matrix L, which is the difference of the degree matrix and the adjacency matrix of the graph, can be defined as

$$L_{l,k} = \begin{cases} \sum\limits_{l \in \rho(k)} w_{k,l} & \text{if } l = k, \\ -w_{l,k} & \text{if vertices } l \text{ and } k \text{ are connected by an edge,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

If for a given vertex, we allow only the nearest four vertices to be connected to the given vertex with edge weights $\frac{1}{h^2}$, then the weighted graph Laplacian reads

$$\Delta_G(u(k)) = \frac{1}{h^2} \sum_{l \in \rho(k)} (u(l) - 4u(k)), \quad \text{for } k, l \in V, \tag{3.8}$$

which is the discretized approximation of the Laplacian differential operator on a 2D grid with the discrete nodes being separated by a step size of h. Graph Laplacian finds its use in several image processing tasks such as image segmentation [Gra06] and graph-based semi-supervised learning [LC09].

Markov chains and jump processes

The nonlocal diffusion operator can also be seen as a random walk Markov chain on Ω. Let Ω be the finite set of discrete states $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M \in \Omega$. Let $\mathcal{Q}(\mathbf{x}_i) := \sum_{\mathbf{x}_j \in \Omega} \omega(\mathbf{x}_i, \mathbf{x}_j)$ for $\mathbf{x}_i, \mathbf{x}_j \in \Omega$, then the transition kernel of the discrete-time, time-homogeneous Markov chain on Ω can be defined as

$$\mathcal{P}(\mathbf{x}_i, \mathbf{x}_j) := \frac{\omega(\mathbf{x}_i, \mathbf{x}_j)}{\mathcal{Q}(\mathbf{x}_i)}, \tag{3.9}$$

which is positivity-preserving but no longer symmetric. At the same time, we have $\sum_{\mathbf{x}_j \in \Omega} \mathcal{P}(\mathbf{x}_i, \mathbf{x}_j) = 1$ for all $\mathbf{x}_i \in \Omega$. Hence, each entry of the matrix $\mathcal{P}(\mathbf{x}_i, \mathbf{x}_j)$ can be viewed as the one-step transition probability from state $\mathbf{x}_i$ to state $\mathbf{x}_j$. Then

$$\mathbf{x}_i^{N+1} = \sum_{\mathbf{x}_j \in \Omega} \mathcal{P}(\mathbf{x}_i, \mathbf{x}_j)\, \mathbf{x}_j^N \tag{3.10}$$

governs the discrete-time Markov jump process, with $\mathbf{x}_i^N$ being the probability mass of state $\mathbf{x}_i$ at a discrete time point N. Therefore the nonlocal diffusion operator from equation (3.1) can be written as a difference of probabilities for a state $\mathbf{x}_i$ between two consecutive time steps:

$$\mathbf{x}_i^{N+1} - \mathbf{x}_i^N = \sum_{\mathbf{x}_j \in \Omega} \mathcal{P}(\mathbf{x}_i, \mathbf{x}_j)\, [\mathbf{x}_j^N - \mathbf{x}_i^N]. \tag{3.11}$$

3.1.2 Properties of the nonlocal diffusion operator

The nonlocal diffusion operator $\mathcal{L}$ has many properties that are similar to a typical elliptic operator. We look at a few of them below.

Theorem 3.2. *Let the nonlocal diffusion operator $\mathcal{L}$ be defined as in equation* (3.1)*. Then $\mathcal{L}$ has the following properties:*

(a) If $u(\mathbf{x}) \equiv const$ for all $x \in \Omega$, then $\mathcal{L}u(\mathbf{x}) \equiv 0$ for all $x \in \Omega$.

(b) $-\mathcal{L}$ is a positive semi-definite operator, i.e.

$$\big\langle -\mathcal{L}u(\mathbf{x}), u(\mathbf{x})\big\rangle_{L^2} \geq 0$$

for all $\mathbf{x} \in \Omega$ and for any function u, where $\langle \cdot, \cdot \rangle_{L^2}$ is the L^2-inner product.

(c) The mean of the output of the nonlocal operation is zero, i.e.

$$\int_\Omega \mathcal{L}u(\mathbf{x})\, d\mathbf{x} = 0.$$

Proof.

(a) Since $u(\mathbf{x})$ is a constant for all $\mathbf{x} \in \Omega$, the term $[u(\mathbf{y}) - u(\mathbf{x})]$ inside the integral vanishes for all $x \in \Omega$, which proves the claim. Note that this property of $\mathcal{L}$ having only constant functions in its null space is equivalent to the graph Laplacian matrix L from equation (3.7) having a zero eigenvalue of multiplicity one, which holds for any connected graph.

(b)

$$\begin{aligned}
\big\langle -\mathcal{L}u(\mathbf{x}), u(\mathbf{x})\big\rangle_{L^2} &= -\int\limits_{\Omega\times\Omega} \omega(\mathbf{x},\mathbf{y})[u(\mathbf{y}) - u(\mathbf{x})]u(\mathbf{x})\, d\mathbf{y}\, d\mathbf{x} \\
&= \int\limits_{\Omega\times\Omega} \omega(\mathbf{x},\mathbf{y})[u(\mathbf{x}) - u(\mathbf{y})]u(\mathbf{x})\, d\mathbf{y}\, d\mathbf{x} \\
&= \frac{1}{2}\int\limits_{\Omega\times\Omega} \Big[\omega(\mathbf{x},\mathbf{y})\Big(u(\mathbf{x}) - u(\mathbf{y})\Big)u(\mathbf{x}) \\
&\qquad\qquad + \omega(\mathbf{y},\mathbf{x})\Big(u(\mathbf{y}) - u(\mathbf{x})\Big)u(\mathbf{y})\Big]\, d\mathbf{y}\, d\mathbf{x} \\
&= \frac{1}{2}\int\limits_{\Omega\times\Omega} \Big(u(\mathbf{y}) - u(\mathbf{x})\Big)^2 \omega(\mathbf{x},\mathbf{y})\, d\mathbf{y}\, d\mathbf{x} \geq 0,
\end{aligned}$$

where the third equality is due to the symmetry between $\mathbf{x}$ and $\mathbf{y}$.

(c)

$$\begin{aligned}
\int\limits_{\Omega} \mathcal{L}u(\mathbf{x})\, d\mathbf{x} &= \int\limits_{\Omega\times\Omega} \omega(\mathbf{x},\mathbf{y})[u(\mathbf{y}) - u(\mathbf{x})]\, d\mathbf{y}\, d\mathbf{x} \\
&= \frac{1}{2}\int\limits_{\Omega\times\Omega} \Big[\omega(\mathbf{x},\mathbf{y})\Big(u(\mathbf{x}) - u(\mathbf{y})\Big) + \omega(\mathbf{y},\mathbf{x})\Big(u(\mathbf{y}) - u(\mathbf{x})\Big)\Big]\, d\mathbf{y}\, d\mathbf{x} \\
&= 0.
\end{aligned}$$

□

Now we look at several defining properties of the underlying PIDE from equation (3.4) that characterize the feature transformations that the operator performs.

Theorem 3.3. *Let the nonlocal diffusion operator $\mathcal{L}$ be defined as in equation (3.1). Then the associated descent flow (3.4) has the following properties:*

(a) The mean value of u is preserved across the time horizon, i.e.

$$\frac{1}{|\Omega|}\int\limits_{\Omega} u(\mathbf{x},t)\, d\mathbf{x} = \frac{1}{|\Omega|}\int\limits_{\Omega} u_0(\mathbf{x})\, d\mathbf{x} \quad \textit{for all } t \geq 0,$$

where $u(\mathbf{x},0) = u_0(\mathbf{x})$ and $|\Omega|$ is the measure of the domain.

(b) *The system governed by equation* (3.4) *has energy decay i.e.*

$$\frac{d}{dt}\|u(\mathbf{x},t)\|_{L^2}^2 = \frac{d}{dt}\int\limits_\Omega u(\mathbf{x},t)^2\,d\mathbf{x} \leq 0 \qquad \textit{for all } x \in \Omega.$$

(c) *The solution to equation* (3.4) *converges to a constant function, namely to the mean value of* $u_0(\mathbf{x})$*:*

$$\lim_{t\to\infty} u(\mathbf{x},t) = \frac{1}{|\Omega|}\int\limits_\Omega u_0(\mathbf{x})\,d\mathbf{x}.$$

Proof.

(a) We compute the time derivative of the mean value to get

$$\frac{d}{dt}\int\limits_\Omega u(\mathbf{x},t)\,d\mathbf{x} = \int\limits_\Omega u_t(\mathbf{x},t)\,d\mathbf{x} = \int\limits_\Omega \mathcal{L}u(\mathbf{x})\,d\mathbf{x} = 0,$$

where the second last equality above is by equation (3.4) and the last equality is due to part (c) of the previous theorem.

(b) Using part (b) of the last theorem, we get

$$\frac{d}{dt}\int\limits_\Omega u(\mathbf{x},t)^2\,d\mathbf{x} = \frac{d}{dt}\big\langle u(\mathbf{x}),u(\mathbf{x})\big\rangle_{L^2} = \langle u(\mathbf{x}),u_t(\mathbf{x})\rangle_{L^2} = \big\langle u(\mathbf{x}),\mathcal{L}(\mathbf{x})\big\rangle_{L^2} \leq 0.$$

(c) The last inequality is strictly less than zero unless $\mathcal{L}(\mathbf{x}) \equiv 0$ and u is a constant function. Hence, we have a convergence to a constant function. Since the corresponding graph Laplacian matrix L has a zero eigenvalue of multiplicity one, it can be shown that there exists only one steady-state solution $u_t(\mathbf{x},t) = \mathcal{L}(\mathbf{x}) = 0$, which is the mean value of $u_0(\mathbf{x})$. □

Since the operator $\mathcal{L}$ performs diffusion, the features tend to smooth out over time, and hence reduce the variance of the image/features. We will see later that though the nonlocal interactions help in image classification and segmentation, over-using it can damp the features leading to loss of information (see section 5.3.3). We can quantify this in the next lemma.

Lemma 3.4. *Let* $u(\mathbf{x},t)$ *be the solution of equation* (3.4)*. Let the variance be defined as*

$$\mathrm{Var}[u(\mathbf{x})] := \frac{1}{|\Omega|}\int\limits_\Omega \left(u(\mathbf{x}) - \frac{1}{|\Omega|}\int\limits_\Omega u(\mathbf{y})\,d\mathbf{y}\right)^2 d\mathbf{x}.$$

Then we have

$$\frac{d}{dt}\mathrm{Var}[u(\mathbf{x})] \leq 0.$$

Proof.

$$\begin{aligned}\frac{d}{dt}\operatorname{Var}[u(\mathbf{x})] &= \frac{2}{|\Omega|}\int\limits_{\Omega}\Big(u(\mathbf{x},t) - \frac{1}{|\Omega|}\int\limits_{\Omega} u(\mathbf{y})\,d\mathbf{y}\Big)u_t(\mathbf{x},t)\,d\mathbf{x}\\ &= \frac{2}{|\Omega|}\int\limits_{\Omega}\Big(u(\mathbf{x},t) - \frac{1}{|\Omega|}\int\limits_{\Omega} u(\mathbf{y})\,d\mathbf{y}\Big)\mathcal{L}(\mathbf{x})\,d\mathbf{x}\\ &= \frac{2}{|\Omega|}\Big(\int\limits_{\Omega} u(\mathbf{x})\mathcal{L}u(\mathbf{x})\,d\mathbf{x} - \frac{1}{|\Omega|}\int\limits_{\Omega} u(\mathbf{y})\,d\mathbf{y}\cdot\int\limits_{\Omega}\mathcal{L}u(\mathbf{x})\,d\mathbf{x}\Big)\\ &= \frac{2}{|\Omega|}\int\limits_{\Omega} u(\mathbf{x})\mathcal{L}u(\mathbf{x})\,d\mathbf{x}\\ &\le 0,\end{aligned}$$

where the last two lines were obtained using results from Theorem 3.2. □

3.2 Pseudo-differential operators

Now we look at a few pseudo-differential operators that can also be represented as integral-based global operators whose implementations introduce nonlocality into the neural network. The aim is to define an operator $\mathcal{O}$ such that the associated forward propagation for a given input image $u_0(\mathbf{x})$ is governed by a PIDE that is similar to the form

$$u_t(\mathbf{x},t) = \mathcal{O}u(\mathbf{x}), \quad u(\mathbf{x},0) = u_0(\mathbf{x}). \tag{3.12}$$

3.2.1 The fractional Laplacian operator

Fractional differential operators appear in several interesting problems in applied mathematics [Yam12], physical sciences [Cap67; SV14], volume-constraint problems [DG13] and in financial mathematics as a pricing model for American options [Sil07]. Let us first define a few related concepts before we define the operator itself.

Definition 3.5. The Schwartz space of rapidly decreasing C^∞ functions on $\mathbb{R}^n$ is defined as

$$\mathcal{S}(\mathbb{R}^n) := \{f \in C^\infty(\mathbb{R}^n) : \forall\alpha,\beta \in \mathbb{N}^n, \|f\|_{\alpha,\beta} < \infty\},$$

where $\|f\|_{\alpha,\beta} := \sup\limits_{\mathbf{x}\in\mathbb{R}^n} |\mathbf{x}^\alpha(D^\beta f)(\mathbf{x})|$ for every multi-index α,β.

The functions in this space have derivatives that exist over the entire domain and tend to vanish faster than any inverse power of $\mathbf{x}$, as $\mathbf{x} \to \pm\infty$.

Definition 3.6. Let $\Omega = \mathbb{R}^n$, $0 < s < 1$, then the fractional-order Sobolev space is defined as

$$H^s(\Omega) := \Big\{u \in L^2(\Omega) : \int\limits_{\Omega}\int\limits_{\Omega} \frac{[u(\mathbf{y}) - u(\mathbf{x})]^2}{|\mathbf{x}-\mathbf{y}|^{n+2s}}\,d\mathbf{y}\,d\mathbf{x} < \infty\Big\},$$

where the Gagliardo seminorm and the norm are defined as

$$|u|^2_{H^s(\Omega)} := \int_\Omega \int_\Omega \frac{[u(\mathbf{y}) - u(\mathbf{x})]^2}{|\mathbf{x}-\mathbf{y}|^{n+2s}} \, d\mathbf{y} \, d\mathbf{x},$$

$$\|u\|_{H^s(\Omega)} := \left(\|u\|^2_{L^2(\Omega)} + |u|^2_{H^s(\Omega)} \right)^{1/2}$$

It can be shown that $H^1(\Omega) \subseteq H^s(\Omega)$ and that the space of compactly supported smooth functions $C^\infty(\mathbb{R}^n)$ is dense in $H^s(\mathbb{R}^n)$ even for $0 < s < 1$ [DNPV12]. Analogously, one can define the fractional trace theorem, the fractional Sobolev inequality, etc.

Definition 3.7. For $u \in \mathcal{S}(\mathbb{R}^n)$, $0 < s < 1$, the fractional Laplacian operator $(-\Delta)^s$ is defined as a singular integral operator as

$$(-\Delta)^s u(\mathbf{x}) := c_{n,s} \, P.V. \int_{\mathbb{R}^n} \frac{[u(\mathbf{x}) - u(\mathbf{y})]}{|\mathbf{x}-\mathbf{y}|^{n+2s}} \, d\mathbf{y} = c_{n,s} \lim_{\epsilon \to 0^+} \int_{\mathbb{R}^n \setminus B_\epsilon(\mathbf{x})} \frac{[u(\mathbf{x}) - u(\mathbf{y})]}{|\mathbf{x}-\mathbf{y}|^{n+2s}} \, d\mathbf{y}, \quad (3.13)$$

where P.V. stands for the Cauchy principal value (of the improper integral), $c_{n,s}$ is a constant depending on n, s and is defined as

$$c_{n,s} = \frac{4^s \Gamma(n/2 + s)}{\pi^{n/2} |\Gamma(-s)|},$$

where Γ is the gamma function with its improper integral representation for $a > 0$ given by

$$\Gamma(a) = \int_0^\infty \mathbf{x}^{a-1} e^{-\mathbf{x}} \, d\mathbf{x}$$

and the identity $\Gamma(a) = \frac{\Gamma(a+1)}{a}$ can be used to uniquely extend the integral definition to all negative real numbers except the integers that are less than or equal to 0.

Note that this operator can be seen as a special case of the nonlocal diffusion operator defined by equation (3.1), with $\Omega = \mathbb{R}^n$ and kernel $\omega(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}|^{-(n+2s)}$ and with the flipped term $u(\mathbf{x}) - u(\mathbf{y})$ instead of $u(\mathbf{y}) - u(\mathbf{x})$. The term is flipped because we want both operators to be elliptic, and the corresponding Poisson problem for the nonlocal diffusion operator is written as $-\mathcal{L}u(\mathbf{x}) = 0$, i.e. with an extra negative sign in front, whereas for the pseudo-differential operator, we have $(-\Delta)^s u(\mathbf{x}) = 0$.

Connection to pseudo-differential operators

This operator can in fact be defined as a *pseudo-differential* operator, which shows the origins of the operator and also characterizes the operator in the Fourier space. Let the Fourier transform and the inverse Fourier transform of $u \in \mathcal{S}(\mathbb{R}^n)$ be denoted by

$$\hat{u}(\xi) := \mathscr{F}u(\xi) := \int_{\mathbb{R}^n} u(\mathbf{x}) e^{-2\pi i \mathbf{x} \cdot \xi} \, d\mathbf{x}, \quad \xi \in \mathbb{R}^n,$$

$$u(\mathbf{x}) := \mathscr{F}^{-1}\hat{u}(\mathbf{x}) := \int_{\mathbb{R}^n} \hat{u}(\xi) e^{2\pi i \mathbf{x} \cdot \xi} \, d\xi, \quad \mathbf{x} \in \mathbb{R}^n.$$

Definition 3.8. Let $\Omega \subset \mathbb{R}^n$ be open, $0 \leq \rho \leq 1$, $0 \leq \delta \leq 1$, $m \in \mathbb{R}$, $n \in \mathbb{N}$, $n \geq 1$. 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

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

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

Definition 3.9. Let $u \in \mathcal{S}(\mathbb{R}^n)$, $p \in S^m_{\rho,\delta}$. A pseudo-differential operator $P(\mathbf{x}, D)$ on $\mathbb{R}^n$ with symbol $p(\mathbf{x}, \xi)$ is defined as

$$P(\mathbf{x}, D)u(\mathbf{x}) := \int_{\mathbb{R}^n} p(\mathbf{x}, \xi)\, \hat{u}(\xi)\, e^{2\pi i \mathbf{x}\cdot\xi}\, d\xi. \tag{3.14}$$

It can be then shown that the fractional Laplacian is the pseudo-differential operator [DNPV12] with the Fourier symbol $(2\pi|\xi|)^{2s}$ and satisfies

$$\mathscr{F}((-\Delta)^s u)(\xi) = (2\pi|\xi|)^{2s}\, \hat{u}(\xi) \quad \text{for } 0 < s \leq 1. \tag{3.15}$$

For $s = 1$, we get the classical Laplacian $(-\Delta)$ with the Fourier symbol $(2\pi|\xi|)^2$:

$$\begin{aligned} -\Delta u(\mathbf{x}) &= -\Delta(\mathscr{F}^{-1}\hat{u}(\mathbf{x})) = -\Delta\Big[\int_{\mathbb{R}^n} \hat{u}(\xi) e^{2\pi i \mathbf{x}\cdot\xi}\, d\xi\Big] \\ &= \int_{\mathbb{R}^n} (2\pi|\xi|)^2\, \hat{u}(\xi) e^{2\pi i \mathbf{x}\cdot\xi}\, d\xi = \mathscr{F}^{-1}\Big((2\pi|\xi|)^2\, \hat{u}(\xi)\Big) \end{aligned}$$

For classical PDE operators, the corresponding Fourier symbol is generally a polynomial in ξ, but by using symbols that are not only a function of ξ but also a function $\mathbf{x}$, we arrive at a large plethora of pseudo-differential operators.

Note that there are several other equivalent definitions of the fractional Laplacian operator that are defined via heat semi-groups [ST10], as an infinitesimal generator of Lévy processes [App09] (similar to the usual Laplacian that is the negative generator of Brownian motions), or as an inverse of the Riesz potential, etc. (see [Kwa17]). For our purposes of image classification and segmentation, we will be using the singular integral operator definition (Definition 3.7) that defines the output of the operator point-wise.

Remark. In [LLH+19], the result of the operator $(-\Delta)^{1/2}$ acting on several basic functions such as the $\sin, \cos$ and exponential functions are illustrated. Moreover, it can be shown that the classical Laplacian is a limit case of the fractional one [Sti10], i.e. for a certain class of bounded functions u, $\mathbf{x} \in \mathbb{R}^n$, we have the point-wise limits

$$\lim_{s\to 0^+} (-\Delta)^s u(\mathbf{x}) = u(\mathbf{x}), \qquad \lim_{s\to 1^-} (-\Delta)^s u(\mathbf{x}) = -\Delta u(\mathbf{x}).$$

Also, one can have the corresponding Poisson problem with Dirichlet boundary conditions for $0 < s < 1$

$$\begin{aligned} (-\Delta)^s u &= f \text{ in } \Omega, \\ u &= g \text{ in } \mathbb{R}^n \setminus \Omega, \end{aligned} \tag{3.16}$$

which has been an area of active research recently [AB17; ROS14; BDPM18]. In [AB17], Acosta and Borthagaray discuss the well-posedness of the problem (3.16) and suggest finite element algorithms for the homogeneous exterior boundary condition. Note that all the different definitions and interpretations of the fractional Laplacian are generally restricted to bounded domains, and the associated boundary conditions lead to very distinct operators that need to be treated differently. For example, the singular integral operator definition needs prescribed values on the entire region outside the domain (i.e. $\mathbb{R}^n \setminus \Omega$), not just of the boundary values, whereas some definitions only need prescribed boundary values.

Addressing the singularity

The kernel under the integral in equation (3.13) has a singularity when $\mathbf{x} = \mathbf{y}$, but for $u \in \mathcal{S}(\mathbb{R}^n)$, we can write

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

C is a positive constant depending on n and $\|u\|_{L^\infty}$. The integral shown above is finite only for $0 < s < 1/2$, but we can make the integral in equation (3.13) well-defined for $0 < s < 1$.

Lemma 3.10. *The fractional Laplacian operator given by* (3.13) *can also be written as*

$$(-\Delta)^s u(\mathbf{x}) := \frac{c_{n,s}}{2} \int\limits_{\mathbb{R}^n} \frac{[2u(\mathbf{x}) - u(\mathbf{x}+\mathbf{y}) - u(\mathbf{x}-\mathbf{y})]}{|\mathbf{y}|^{n+2s}}\, d\mathbf{y}. \tag{3.17}$$

Proof.

$$\begin{aligned}&\frac{c_{n,s}}{2} \int\limits_{\mathbb{R}^n} \frac{[2u(\mathbf{x}) - u(\mathbf{x}+\mathbf{y}) - u(\mathbf{x}-\mathbf{y})]}{|\mathbf{y}|^{n+2s}}\, d\mathbf{y} \\ &\quad= \frac{c_{n,s}}{2} \lim_{\epsilon\to 0^+} \int\limits_{\mathbb{R}^n\setminus B_\epsilon(0)} \frac{2u(\mathbf{x}) - u(\mathbf{x}+\mathbf{y}) - u(\mathbf{x}-\mathbf{y})}{|\mathbf{y}|^{n+2s}}\, d\mathbf{y} \\ &\quad= \frac{c_{n,s}}{2} \lim_{\epsilon\to 0^+} \Bigg[\int\limits_{\mathbb{R}^n\setminus B_\epsilon(0)} \frac{u(\mathbf{x}) - u(\mathbf{x}+\mathbf{y})}{|\mathbf{y}|^{n+2s}}\, d\mathbf{y} + \int\limits_{\mathbb{R}^n\setminus B_\epsilon(0)} \frac{u(\mathbf{x}) - u(\mathbf{x}-\mathbf{y})}{|\mathbf{y}|^{n+2s}}\, d\mathbf{y}\Bigg] \\ &\quad= \frac{c_{n,s}}{2} \lim_{\epsilon\to 0^+} \Bigg[\int\limits_{\mathbb{R}^n\setminus B_\epsilon(\mathbf{x})} \frac{u(\mathbf{x}) - u(\eta)}{|\mathbf{x}-\eta|^{n+2s}}\, d\eta + \int\limits_{\mathbb{R}^n\setminus B_\epsilon(\mathbf{x})} \frac{u(\mathbf{x}) - u(\zeta)}{|\mathbf{x}-\zeta|^{n+2s}}\, d\zeta\Bigg]\end{aligned}$$

$$
\begin{aligned}
&= c_{n,s} \lim_{\epsilon\to 0^+} \int\limits_{\mathbb{R}^n\setminus B_\epsilon(\mathbf{x})} \frac{u(\mathbf{x})-u(\eta)}{|\mathbf{x}-\eta|^{n+2s}}\, d\eta \\
&= c_{n,s} \lim_{\epsilon\to 0^+} \int\limits_{\mathbb{R}^n\setminus B_\epsilon(\mathbf{x})} \frac{u(\mathbf{x})-u(\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^{n+2s}}\, d\mathbf{y} \\
&= (-\Delta)^s u(\mathbf{x}),
\end{aligned}
$$

where the third equality is due to change of variables $\eta := \mathbf{x}+\mathbf{y}$, $\zeta := \mathbf{x}-\mathbf{y}$ and the second last equality was just a relabeling of η to $\mathbf{y}$. □

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

$$\int\limits_{\mathbb{R}^n} \frac{[2u(\mathbf{x})-u(\mathbf{x}+\mathbf{y})-u(\mathbf{x}-\mathbf{y})]}{|\mathbf{y}|^{n+2s}} \le \frac{\|D^2u\|_{L^\infty}}{|\mathbf{y}|^{n+2s-2}}$$

is integrable at the origin for any $0 < s < 1$. Thus, we can remove $P.V.$ from the integral as long as $u \in \mathcal{S}(\mathbb{R}^n)$. The idea is that, near $\mathbf{x}$, $[u(\mathbf{x})-u(\mathbf{y})]$ has the approximation $\nabla u(\mathbf{x})\cdot(\mathbf{x}-\mathbf{y})$, and the term under the integral is of the form

$$\frac{\nabla u(\mathbf{x})\cdot(\mathbf{x}-\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^{n+2s}}.$$

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.

3.2.2 The inverse fractional Laplacian operator

Now we try to define an analogous operator $(-\Delta)^{-s}$ for $s > 0$, i.e. an inverse fractional Laplacian, which we will see, is also a nonlocal operator. Note that, similar to equation (3.15), we have

$$\mathscr{F}((-\Delta)^{-s}u)(\xi) = (2\pi|\xi|)^{-2s}\,\hat{u}(\xi) \quad \text{for } 0 < s < n/2. \tag{3.18}$$

One needs the restriction $0 < s < n/2$ because if $s \ge n/2$, the Fourier symbol/multiplier $(2\pi|\xi|)^{-2s}$ fails to define a *tempered distribution* [Sil07], where the space of tempered distributions is the continuous dual of the Schwartz space $\mathcal{S}(\mathbb{R}^n)$. The symbol $(2\pi|\xi|)^{-2s}$ is decaying with respect to s, and in the sense of distributions (see [Ste70]), such symbols have the Fourier inverse

$$\mathscr{F}^{-1}(|\xi|^{-2s}) = c_{n,-s}|\mathbf{x}|^{-(n-2s)}.$$

Using the inverse property for convolutions

$$f * g = \mathscr{F}^{-1}\{\mathscr{F}\{f\}\cdot\mathscr{F}\{g\}\}$$

and equation (3.18), we get the following equality

$$(-\Delta)^{-s}u(\mathbf{x}) = \left(c_{n,-s}|\mathbf{x}|^{-(n-2s)}\right) * u(\mathbf{x}) = c_{n,-s}\int\limits_{\mathbb{R}^n} \frac{u(\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^{n-2s}}\, d\mathbf{y}. \tag{3.19}$$

Definition 3.11. For $u \in \mathcal{S}(\mathbb{R}^n)$, $0 < s < n/2$, the inverse fractional Laplacian operator $(-\Delta)^{-s}$ is defined as an integral operator as

$$(-\Delta)^{-s}u(\mathbf{x}) := c_{n,-s} \int\limits_{\mathbb{R}^n} \frac{u(\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^{n-2s}}\, d\mathbf{y}, \tag{3.20}$$

where $c_{n,-s}$ is the constant given by

$$c_{n,-s} = \frac{\Gamma(n/2-s)}{4^s \pi^{n/2}\Gamma(s)}.$$

The right-hand side of the definition is called the *Riesz potential.* Riesz potential is an important tool in harmonic analysis and linear PDEs [Hör15]. This singular integral is well-defined, provided u decays sufficiently rapidly at infinity (see [Ste70]), for instance, if $u \in L^p(\mathbb{R}^n)$ with $1 \leq p < \frac{n}{2s}$. For a more in-depth study of Riesz and Riesz-like potentials, see [Lan72; Rub96].

Recently, a fundamental solution for the case $s = n/2$ has been proposed [Sti19], which can also be used in the neural network as a nonlocal operation. As pointed out in [Ste70], the logarithmic kernel defined below can be seen as a vague limit of the Riesz potentials defined by (3.20).

Definition 3.12. Let $u \in \mathcal{S}(\mathbb{R}^n)$ with $\int\limits_{\mathbb{R}^n} u = 0$, $\mathbf{x} \in \mathbb{R}^n$, and $s = n/2$. Then we have

$$(-\Delta)^{-s}u(\mathbf{x}) := c_n \int\limits_{\mathbb{R}^n} \Big(-2\log|\mathbf{x}-\mathbf{y}| - \gamma \Big) u(\mathbf{y})\, d\mathbf{y}, \tag{3.21}$$

where the Euler-Mascheroni constant γ is given by

$$\gamma = -\int\limits_0^\infty e^{-r} \log r\, dr \approx 0.5772156649,$$

and the constant c_n is given by

$$c_n = \frac{1}{(4\pi)^{n/2}\Gamma(n/2)}.$$

The integral operator, defined by equation (3.20), can be seen as a special case of the Hilbert-Schmidt integral operator [RR06] that is continuous and hence bounded, if the kernel is well-defined.

Definition 3.13. Let $\Omega \subset \mathbb{R}^n$ be open and connected, and let $k : \Omega \times \Omega \to \mathbb{R}$ be a Hilbert-Schmidt kernel, i.e.

$$\int\limits_\Omega \int\limits_\Omega |k(\mathbf{x},\mathbf{y})|^2\, d\mathbf{x}\, d\mathbf{y} < \infty. \tag{3.22}$$

Then for $u \in L^2(\Omega)$, the Hilbert-Schmidt integral operator $\mathcal{T} : L^2(\Omega;\mathbb{R}) \to L^2(\Omega;\mathbb{R})$ corresponding to the kernel k is given by

$$\mathcal{T}u(\mathbf{x}) := \int\limits_\Omega k(\mathbf{x},\mathbf{y})u(\mathbf{y})\, d\mathbf{y}. \tag{3.23}$$

Lemma 3.14. *Let $\Omega \subset \mathbb{R}^n$ and let k be a Hilbert-Schmidt kernel. The corresponding Hilbert-Schmidt operator $\mathcal{T} : L^2(\Omega;\mathbb{R}) \to L^2(\Omega;\mathbb{R})$ is bounded.*

Proof. For $u \in L^2(\Omega)$, using the Cauchy-Schwarz inequality, we have

$$\begin{aligned}\|\mathcal{T}u\|^2_{L^2(\Omega)} &= \int\limits_{\Omega} \Big[\int\limits_{\Omega} k(\mathbf{x},\mathbf{y})u(\mathbf{y})\,d\mathbf{y} \Big]^2 d\mathbf{x} \\ &\leq \int\limits_{\Omega} \Big(\int\limits_{\Omega} |k(\mathbf{x},\mathbf{y})|^2\,d\mathbf{y} \Big) \Big(\int\limits_{\Omega} (u(\mathbf{y}))^2\,d\mathbf{y} \Big)\,d\mathbf{x} \\ &= \Big(\int\limits_{\Omega}\int\limits_{\Omega} |k(\mathbf{x},\mathbf{y})|^2\,d\mathbf{y}\,d\mathbf{x} \Big) \Big(\int\limits_{\Omega} u(\mathbf{y})^2\,d\mathbf{y} \Big) \\ &= C\|u\|^2_{L^2(\Omega)}.\end{aligned}$$

□

As such, the Riesz potential kernel $\dfrac{1}{|\mathbf{x}-\mathbf{y}|^{n-2s}}$ may not always satisfy the square-integrability condition (3.22). While implementing these nonlocal operators for images, the methods used to bypass this issue and to avoid the blowing up of values are discussed in sections 4.2.2 and 4.2.3. Besides, in the case of the log kernel from equation (3.21), the kernel is no longer positivity-preserving. Sections 4.2.2 and 4.2.3 also talk about the implementation of the log kernel that makes sure that it stays well-defined. It is interesting to note that the kernels of the pseudo-differential operators are translation invariant and isotropic. Also, from equation (3.19), it is clear that the operators from equations (3.20) and (3.21) can be seen as convolutions on a global scale (spatially), whereas the operators $\mathcal{L}$ and $(-\Delta)^s$ from equations (3.1) and (3.13) respectively define a diffusion-like operator on a global scale (spatially).

Chapter 4

PIDE-based discretization of nonlocal operators and implementation details

In this chapter, we first look at the implementation of the Hamiltonian network that will be used as the base network. Then, several discretization details about the Nonlocal Blocks, which contain the proposed integral operators, will be introduced. We explore a few strategies to reduce the computational effort of such global operators and discuss how the tensors are dealt with when such operators are applied on images or feature maps. Finally, the general implementation details for the numerical experiments, including the regularization techniques, are discussed.

4.1 Hamiltonian networks with Verlet scheme

The ODE and PDE-based networks discussed before have the advantages of stable forward propagation and reversibility during backpropagation. Among all the three ODE-based networks suggested by Chang, Meng, Haber et al. in [CMH+18b], the Hamiltonian network, using the Verlet scheme from equation (2.17), performs the best on standard benchmarks, possibly due to its two-layer network, where each Hamiltonian Block has two convolution and two transposed convolution operations. Hence, for the experiments, this network will be used as the base architecture, and changes are made to it by introducing nonlocality in the network. Similar to the ResNet architecture, several of the Hamiltonian Blocks (Figure 2.5) are stacked up to form a Unit and several of these reversible Units together form the entire Network. Figure 4.1 shows a 2-Unit Hamiltonian network, where we see that the input is passed through an initial convolution layer and then through each Unit of the network.

Similar to the original ResNet architecture [HZR+16a], after each Unit (consisting of m Hamiltonian Blocks), a pooling operation reduces the feature map size by half, and the number of channels is increased by using a 1×1 convolution layer (with ReLU). After the final Unit, the features are subsampled and then passed on to the fully-connected layer (for image classification tasks), where the predictions are made. The number of Units in the model is kept fixed. In our case, the networks will always contain three Units, but the

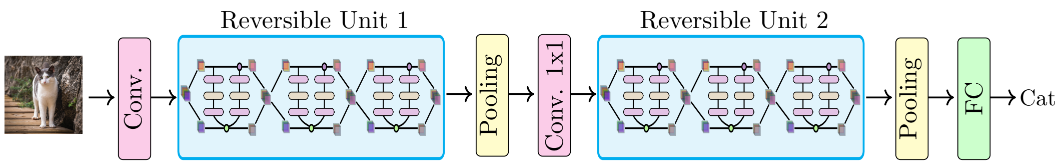

Figure 4.1: The Hamiltonian Network with two Units having three Hamiltonian reversible Blocks each when used for image classification.

number of Blocks (m) in each Unit (along with the feature map size and the number of channels in each Unit) is varied to alter the depth of the neural network. In terms of the ODE/PDE interpretation, each Block represents a time step of the discretization, with N and h deciding the depth of the network and the final output $\mathbf{Y}_N$ being the output of the last Block. The Leapfrog and the Midpoint network architectures are treated similarly, where the forward propagation in the Blocks are governed by equations (2.19) and (2.21), and m Blocks are stacked together to form a Unit.

4.2 Implementation of Nonlocal Blocks

We now look at how the global operators are included in the network. To this end, the nonlocality is added after the second Block in each Unit of the network, i.e. the feature maps that are obtained after the second Block are passed through the *Nonlocal Block*. The output of the Nonlocal Block is then fed into the next *normal* Block (Hamiltonian, Midpoint, etc.).

4.2.1 Nonlocal diffusion operator

Let $\mathbf{X}$ be the input to the *Nonlocal Block* that is supposed to contain the nonlocal interactions. Let $\mathbf{X}_i$ denote each pixel of the image. For instance, for an input $\mathbf{X}$ with 16 channels, $\mathbf{X}_i \in \mathbb{R}^{16}$ stands for the 16 channel values for each pixel position. These vectors will be called *pixel strips* from here on. To implement the nonlocal diffusion operator $\mathcal{L}$, defined by equation (3.1), in the Nonlocal Block, one can discretize the PIDE-inspired operator using a 2-step/2-stage computation as follows:

$$\begin{aligned}
[\mathbf{B}_1]_i &= \mathbf{X}_i + h\,\mathcal{K}_1\left[\frac{1}{\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)}\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)(\mathbf{X}_j-\mathbf{X}_i)\right],\\
[\mathbf{B}_2]_i &= \mathbf{X}_i + h\,\mathcal{K}_2\left[\frac{1}{\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)}\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)\Big([\mathbf{B}_1]_j-[\mathbf{B}_1]_i\Big)\right],
\end{aligned} \tag{4.1}$$

where $[\mathbf{B}_1]_i$ and $[\mathbf{B}_2]_i$ are the individual pixel strips of the intermediate feature maps $\mathbf{B}_1$ and the output feature maps $\mathbf{B}_2$ respectively, h is the discretization step size, and $\mathcal{K}_1$, $\mathcal{K}_2$ are 1×1 convolution operators, followed by ReLU activation and Batch Normalization. The output $\mathbf{B}_2$ is then fed into the next normal Block in the Unit. This 2-stage discretization strengthens the nonlocal interactions between the features further and can be roughly seen as a forward Euler scheme (in time) for the PIDE

$$u_t(\mathbf{x},t) = \mathcal{L}(u(\mathbf{x}) + h\mathcal{L}u(\mathbf{x})), \quad u(\mathbf{x},0) = u_0(\mathbf{x}), \tag{4.2}$$

where $u_0(\mathbf{x})$ is the initial value.

There are several things to note here. The kernel $\omega(\mathbf{x},\mathbf{y})$ measures the affinity or similarity between each pair of pixel strips of the image/feature maps. For choosing the right kernel, there are several suggestions made by Wang, Girshick, Gupta et al. in [WGG+18], such as the Gaussian function $e^{\mathbf{x}^T\mathbf{y}}$, the embedded Gaussian $e^{\theta(\mathbf{x})^T\phi(\mathbf{y})}$ and the embedded dot product $\theta(\mathbf{x})^T\phi(\mathbf{y})$. All these kernels seem to work equally well. We will be using the scaled embedded dot product $\lambda\theta(\mathbf{x})^T\phi(\mathbf{y})$, where θ and ϕ are 1×1 convolutional embeddings, and $\lambda > 0$ is a scaling factor. Essentially, it means that the input $\mathbf{X}$ is passed through two different 1×1 convolutions to obtain embeddings $\theta(\mathbf{X})$ and $\phi(\mathbf{X})$. Then the kernel ω measures the affinity between the pairs of pixel strips $\theta(\mathbf{X}_i)$ and $\phi(\mathbf{X}_j)$ (see Figure 4.2).

One can also use the scaled Gaussian kernel if one needs a positivity-preserving kernel $e^{\lambda\theta(\mathbf{x})^T\phi(\mathbf{y})}$. Note that the embedded kernels are not symmetric. If one needs a symmetric kernel with embeddings of the input, then one can pre-embed the features to get $\theta(\mathbf{X})$ and then feed it to the PIDE-inspired Nonlocal Block with the kernel $\omega(\mathbf{X}_i,\mathbf{X}_j)$ in (4.1) replaced by $\omega(\theta(\mathbf{X}_i),\theta(\mathbf{X}_j))$. Then one can use the scaled Gaussian kernel $\omega(\mathbf{x},\mathbf{y}) = e^{\lambda\mathbf{x}^T\mathbf{y}}$ or the scaled dot product $\omega(\mathbf{x},\mathbf{y}) = \lambda\mathbf{x}^T\mathbf{y}$, both of which are symmetric.

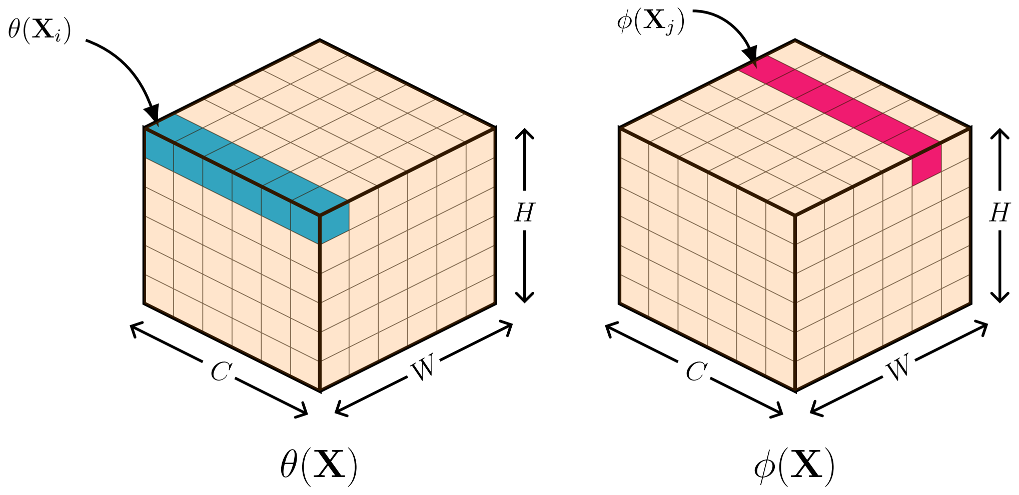

Figure 4.2: Pixel strips of embedded versions of $\mathbf{X}$ with C channels, which are used to compute the kernel entry $\omega(\mathbf{X}_i,\mathbf{X}_j)$.

The factor $\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)$ is a normalizing factor. For the sake of simplicity, and for easier gradient computations, as suggested in [WGG+18], we choose this factor to be $\mathcal{N}$, where $\mathcal{N}$ is the number of pixel strips in the input $\mathbf{X}$. For instance, in Figure 4.2, the number of pixel strips is 36 for a 6×6 image. Without this normalizing factor, the computations might blow up for inputs/images of larger sizes.

After the normalization, a 1×1 convolution $\mathcal{K}_1$ is applied, along with ReLU and Batch normalization, and then the result is added to the pixel strip of the original input, i.e. $\mathbf{X}_i$.

Instead of simply discretizing the PIDE (equations (3.4) and (3.12)) using the forward Euler scheme, this computation is iteratively repeated several times, as shown in Figure 4.3. This allows us to extract more nonlocal information from the dataset. Here, we restrict ourselves to a two-stage forward propagation.

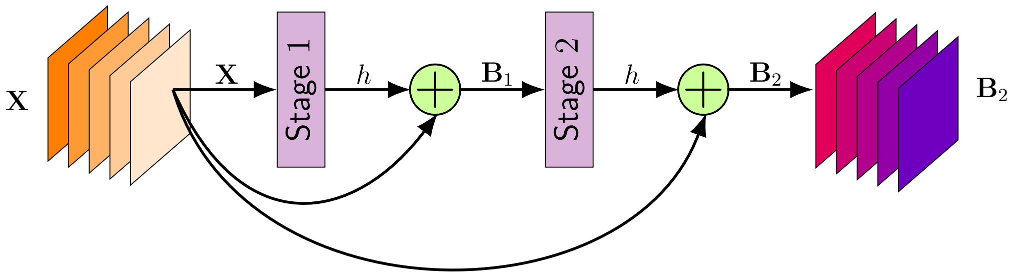

Figure 4.3: The two-stage computation of a Nonlocal Block with skip connections.

The advantage of this multi-stage operation is that one can perform nonlocal operations several times, which widens the field-of-view of the network further, but the kernel ω is computed only once at the start, which saves computational time and effort. At the same time, one cannot have too many stages in one Nonlocal Block because, for instance, after Stage 5, we have the output $\mathbf{B}_5$, but the pre-computed kernel would fail to characterize the affinity between pairs of pixel strips of the features properly since the features would change quite a bit after each stage (see section 5.3.3).

This Nonlocal Block differs from the one suggested in [TSD+18] because, as shown in Figure 4.3, it contains identity skip connections. Similar to skip connections in ResNets, which have performed well for deep networks, skip connections in the Nonlocal Block are introduced to make sure that the Nonlocal Block does not impede the flow of information, which would force the network to perform worse. The experiments in section 5.3.1 confirm this, showing that including skip connections in the nonlocal computations can be beneficial. This way, in the worst case, in the presence of Nonlocal Blocks, the network at least does not deliver a worse performance than the performance of a network without the nonlocality.

The affinity kernel 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

$$\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)(\mathbf{X}_j-\mathbf{X}_i) = \sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)\mathbf{X}_j \; - \; \sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)\mathbf{X}_i \tag{4.3}$$

$$= \sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)\mathbf{X}_j \; - \; \mathbf{X}_i \sum_j \omega(\mathbf{X}_i,\mathbf{X}_j). \tag{4.4}$$

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

$$[\mathbf{B}_2]_i = \cdots \left[\cdots \sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)[\mathbf{B}_1]_j \; - \; [\mathbf{B}_1]_i \sum_j \omega(\mathbf{X}_i,\mathbf{X}_j) \right]. \tag{4.5}$$

The exact details of how the tensors are dealt with and propagated forward in the Nonlocal Block are shown in Figure 4.4, 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.

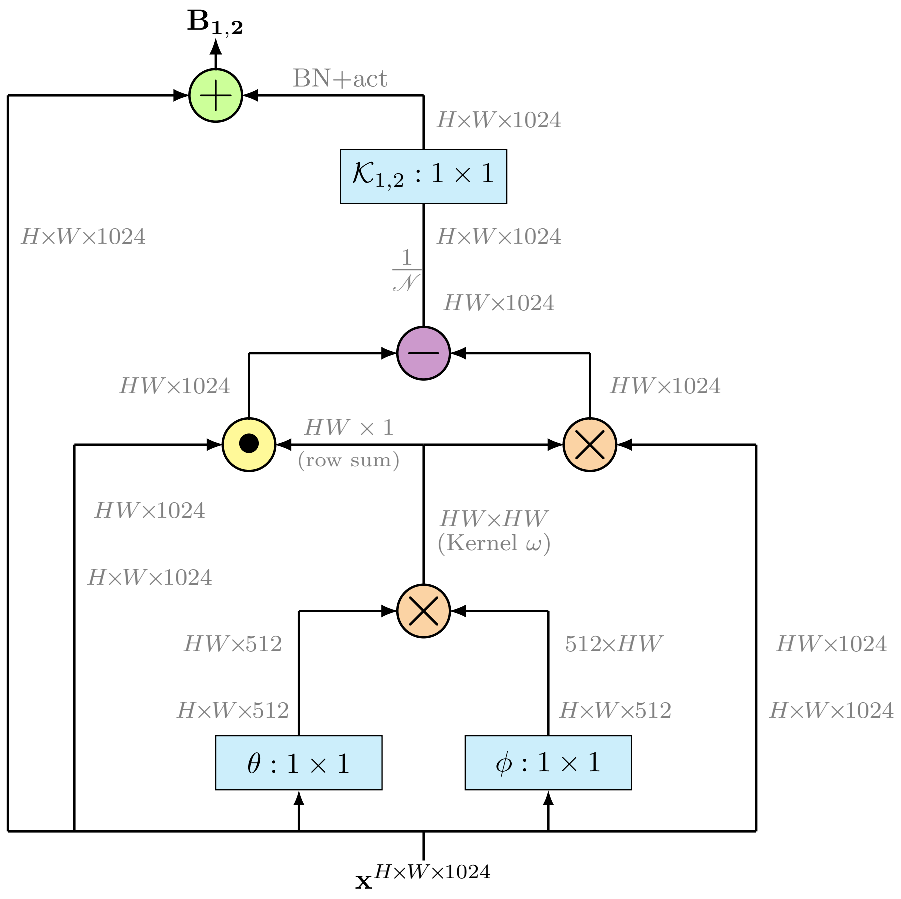

Figure 4.4: Forward propagation of tensors in a Nonlocal Block with the nonlocal diffusion operator $\mathcal{L}$.

In Figure 4.4, $\otimes$ represents matrix multiplication, $\oplus$ and $\ominus$ denote element-wise addition and subtraction respectively, and $\odot$ denotes 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 (4.4) and (4.5), and the dot product is used to compute the second part of equations (4.4) and (4.5).

Subsampled Nonlocal Blocks

As we will see later in section 6.1, this global computation can be very expensive. In order to minimize the floating-point operations, several remedies exist. Firstly, all the convolutions in the Nonlocal Block are 1×1 convolutions. This itself reduced 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 4.4. 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 4.5. Compare this figure to Figure 4.2.

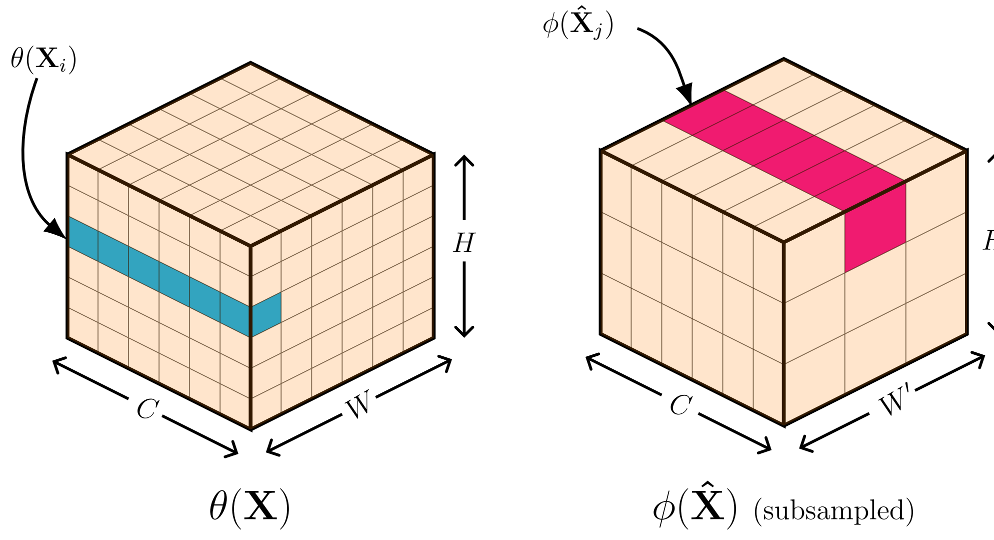
$\theta(\mathbf{X}_i)$
$\phi(\hat{\mathbf{X}}_j)$
H
H'
C
W
C
W'
$\theta(\mathbf{X})$
$\phi(\hat{\mathbf{X}})$ (subsampled)

Figure 4.5: Pixel strips of embedded versions of $\mathbf{X}$ with C channels, which are used to compute the subsampled version of the kernel entry $\omega(\mathbf{X}_i, \hat{\mathbf{X}}_j)$.

Figure 4.5 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 performance 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 [WGG+18] are discussed in section 6.1. In terms of the entire Nonlocal Block itself, Figure 4.6 shows how the feature maps are transformed when we use the subsampling trick. The nonlocality is still preserved in this case, but the computations are sparser. For instance, if we use a 2×2 pooling operation, the number of pairwise computations reduces by a quarter.

4.2.2 Fractional Laplacian operator $(-\Delta)^s$

As discussed before, the fractional Laplacian operator (3.13) can be seen as a special case of the nonlocal diffusion operator with kernel $\omega(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}|^{-(n+2s)}$. This means

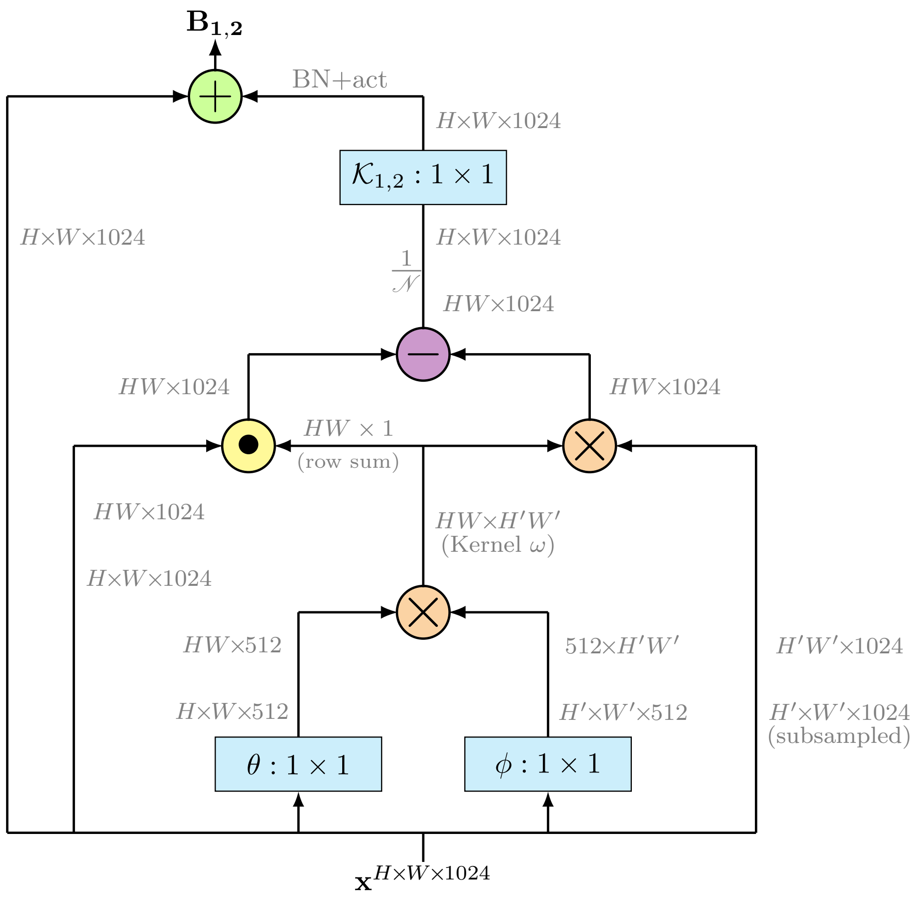

Figure 4.6: Forward propagation of tensors in a *subsampled* Nonlocal Block with the nonlocal diffusion operator $\mathcal{L}$.

that the discretization follows a similar procedure. In this case, we have the following two-stage forward propagation:

$$\begin{aligned}
[\mathbf{B}_1]_i &= \mathbf{X}_i + h\,\mathcal{K}_1\left[\frac{c_{n,s}}{\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)}\sum_j \frac{\lambda}{\|\theta(\mathbf{X}_i)-\phi(\mathbf{X}_j)\|_2^{n+2s}}(\mathbf{X}_i-\mathbf{X}_j)\right],\\
[\mathbf{B}_2]_i &= \mathbf{X}_i + h\,\mathcal{K}_2\left[\frac{c_{n,s}}{\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)}\sum_j \frac{\lambda}{\|\theta(\mathbf{X}_i)-\phi(\mathbf{X}_j)\|_2^{n+2s}}\Big([\mathbf{B}_1]_i-[\mathbf{B}_1]_j\Big)\right],
\end{aligned} \tag{4.6}$$

where θ and ϕ are again 1×1 convolutional embeddings, $\|\cdot\|_2$ is the L^2 norm, $0 < s < 1$, and $\lambda > 0$ is a scaling constant that is used to have greater control over the diffusion kernel. A small value of λ leads to a weak interaction between the different pixel strips, whereas a large value of λ damps the signal quickly. The rest of the expression is implemented in a similar fashion, with normalizing constant $\mathscr{N}$, etc. The tensors in the Nonlocal Block are computed similarly, as shown in Figure 4.4, and its subsampled version is shown in Figure 4.6. In the case of pseudo-differential operators, the $\otimes$ symbol for the kernel computation represents pair-wise distance computations. The other $\otimes$ symbol in the figures still stands for matrix multiplication. 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. The forward propagation of the features through this 2-stage process can be again seen as a discretization of an underlying PIDE (equation 4.2).

Besides, we have a singularity in the kernel if the embeddings of the pixel strips, i.e. $\theta(\mathbf{X}_i)$ and $\phi(\mathbf{X}_j)$, are the same. From section 3.2.1, we know that there is no net contribution to the integral when $\mathbf{y} \to \mathbf{x}$. Hence we just perform a *safe divide*, i.e. the kernel entry

$$\omega(\mathbf{X}_i, \mathbf{X}_j) = \frac{\lambda}{\|\theta(\mathbf{X}_i) - \phi(\mathbf{X}_j)\|_2^{n+2s}}$$

is replaced with 0 if the denominator turns out to be zero.

While computing the pairwise distances, if the mini-batch size is large, the tensors get replicated and thus consume a lot of memory. To overcome this problem, we can use the identity

$$\|\mathbf{x} - \mathbf{y}\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x} \cdot \mathbf{y} + \|\mathbf{y}\|^2.$$

This reduces the memory footprint by quite a bit. For instance, if we have matrices $A \in \mathbb{R}^{36\times 6}$ and $B \in \mathbb{R}^{9\times 6}$ that contain the values of 36 and 9 pixel strips of the two embeddings respectively (see Figure 4.5), then the square of the pairwise distance is roughly represented as follows

$$\begin{pmatrix} \vdots \end{pmatrix}_{36\times 1} - 2\big[A \cdot B^T\big] + \begin{pmatrix} \cdots \end{pmatrix}_{1\times 9},$$

with 36×1 and 1×9 representing the squared L^2 norms of the 36 and 9 pixel strips respectively. The addition and the subtraction is done row-wise and column-wise, not element-wise. For example, each of the 36 entries of the 36×1 vector is added to the corresponding row of the 36×9 matrix that results from the dot product operation.

4.2.3 Inverse fractional Laplacian operator $(-\Delta)^{-s}$

The inverse fractional Laplacian operator for $0 < s < n/2$, defined by equation (3.20), can be discretized in a similar fashion using a two-stage method with skip connections:

$$\begin{aligned}
[\mathbf{B}_1]_i &= \mathbf{X}_i + h\,\mathcal{K}_1\left[\frac{c_{n,-s}}{\sum\limits_j \omega(\mathbf{X}_i, \mathbf{X}_j)} \sum_j \frac{\lambda}{\|\theta(\mathbf{X}_i) - \phi(\mathbf{X}_j)\|_2^{n-2s}} \mathbf{X}_j\right], \\
[\mathbf{B}_2]_i &= \mathbf{X}_i + h\,\mathcal{K}_2\left[\frac{c_{n,-s}}{\sum\limits_j \omega(\mathbf{X}_i, \mathbf{X}_j)} \sum_j \frac{\lambda}{\|\theta(\mathbf{X}_i) - \phi(\mathbf{X}_j)\|_2^{n-2s}} [\mathbf{B}_1]_j\right].
\end{aligned} \tag{4.7}$$

For the case $s = n/2$, we have a log kernel (equation 3.21), which is implemented in the Nonlocal Block as

$$[\mathbf{B}_1]_i = \mathbf{X}_i + h\,\mathcal{K}_1\left[\frac{c_n}{\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)}\sum_j\Big(-2\lambda\log\Big(\big\|\theta(\mathbf{X}_i)-\phi(\mathbf{X}_j)\big\|_2\Big)-\gamma\Big)\,\mathbf{X}_j\right],$$
$$[\mathbf{B}_2]_i = \mathbf{X}_i + h\,\mathcal{K}_2\left[\frac{c_n}{\sum_j \omega(\mathbf{X}_i,\mathbf{X}_j)}\sum_j\Big(-2\lambda\log\Big(\big\|\theta(\mathbf{X}_i)-\phi(\mathbf{X}_j)\big\|_2\Big)-\gamma\Big)\,[\mathbf{B}_1]_j\right]. \tag{4.8}$$

For the case $0 < s < n/2$, the kernel entry is again replaced with zero, in case the denominator happens to be zero. For the case $s = n/2$, if the term $\log\big(\|\theta(\mathbf{X}_i)-\phi(\mathbf{X}_j)\|_2\big)$ turns out to be undefined, the term is then replaced by $-\dfrac{\gamma}{2\lambda}$ so that it does not contribute to the overall sum.

These two operators are different from the previous two discussed in this chapter. While for the diffusion-like operators, the kernel is integrated with the term $\mathbf{X}_j - \mathbf{X}_i$ (and $[\mathbf{B}_1]_j - [\mathbf{B}_1]_i$), for these two operators, we have the term $\mathbf{X}_j$ (and $[\mathbf{B}_1]_j$) only. This is reflected in the performance of the overall network, as we will see in section 5.3.1. The computations and forward propagations of the tensors for these two Nonlocal Blocks with subsampling are shown in Figure 4.7. The non-subsampled version can be found in Appendix A.1.

The PIDE-based Nonlocal Blocks discussed here have several advantages over other layers in the neural network. Firstly, it is clear that for all the operators introduced here, the pixel strips' values of the Nonlocal Block output depend on each pixel strip of the input tensor. This Block is also different from a fully-connected layer because it computes activations based on relationships between different regions of the image, and these long-range dependencies are a function of the input data. This is different in a fully-connected layer, where the relationship is established using trainable weights. Secondly, a fully-connected layer can usually only be added at the end of the network due to computational cost reasons, and the local information is lost by flattening the image. On the other hand, the Nonlocal Block can be added anywhere in the network, and it preserves the 2D structure of images. Besides, the output and input shapes of this implementation are the same. Therefore, this Block can be plugged into any neural network architecture associated with computer vision to accelerate the communication of information across pixels.

4.3 Regularization

Regularization is a process of adding information to the objective function in order to solve an ill-posed problem. In a neural network, it reduces the generalization error [GBC16], i.e. the error generated while using the model on the test data. Regularization is a technique that is also used to tackle underfitting and overfitting. In overfitting, the model performs well on the training data but poorly on the test data that it has not seen before. On the other hand, underfitting occurs when the model fails to learn and perform well on

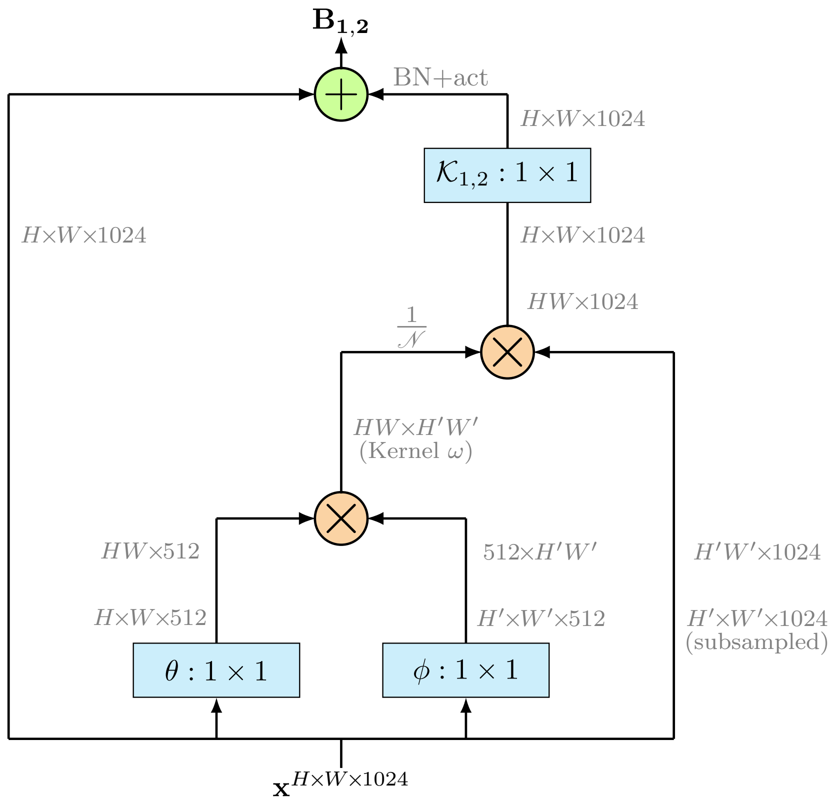

Figure 4.7: Forward propagation of tensors in a *subsampled* Nonlocal Block with the inverse Laplacian operator $(-\Delta)^{-s}$.

the training data itself because the model is not complex enough to learn the characteristics of the data. Several strategies have been used in the deep learning community in this regard, such as shake-shake regularization [Gas17]. Dropout [SHK+14], for instance, randomly drops nodes from the network during training to reduce the complexity of the model. Other techniques to reduce the generalization error, such as data augmentation, are also quite popular, which we will discuss in the next section.

The *weight-decay* or L^2 regularization (sometimes written as L_2 or $L2$) will be used in our experiments. It penalizes large weights in the hidden layers of the network and is equivalent to the well-known Tikhonov regularization. Let $\mathbf{K}$ denote some (convolutional) weights of the network, then the L^2 regularizer is given by

$$R(\mathbf{K}) = \frac{\alpha_1}{2}\|\mathbf{K}\|_F^2, \tag{4.9}$$

where $\|\cdot\|_F$ represents the Frobenius norm, and α_1 is the regularization hyperparameter that controls the trade-off between larger and smaller trainable weights. Very small values of α_1 lead to overfitting. Meanwhile, larger values of α_1 lead to underfitting because it incentivizes the network to push many network weights close to zero, which drastically reduces the expressive power and the complexity of the model. In fact, from an optimal control viewpoint, these regularization terms can be seen as *running costs* for the model, whereas the loss function S from equation (2.11) can be seen as the *terminal cost.*

From Theorem 2.7, we know that for a stable forward propagation, we would ideally want the weight matrices $\mathbf{K}(t)$ to change smoothly with time, i.e. we would like convolution weights to be piecewise smooth in time. To this end, as suggested in [CMH+18b], we use *weight smoothness decay* in combination with L^2 regularization. The smooth weight decay can be written as

$$R(\mathbf{K}) = \alpha_2 \, h \sum_{j=1}^{N-1} \sum_{k=1}^{2} \left\| \frac{\mathbf{K}_{j,k} - \mathbf{K}_{j+1,k}}{h} \right\|_F^2, \tag{4.10}$$

where h is the step size, α_2 is again a hyperparameter, and j represents each time step or each Block of the PDE-based neural network. Here it is assumed that there are two convolution layers/weights $\mathbf{K}_1$ and $\mathbf{K}_2$ in each Block/time step (in Hamiltonian Blocks, for example). This regularization favors weights that vary smoothly between adjacent layers of the network, i.e. piecewise smooth dynamics are favored. Because of the unique nature of Nonlocal Blocks, the weight smoothness decay is only applied to the normal Blocks and not to the Nonlocal Blocks, whereas L^2 weight decay is applied to all the convolutional network weights. Both the regularization terms are then added to the objective function (2.11), which the optimizer tries to then minimize.

4.4 Further implementation details

After the opening layer of 3×3 convolutions, the original Hamiltonian network consists of 3 Units [CMH+18b], with a pooling and a zero-padding layer between the Units, where the feature map size is halved, and the number of feature maps is increased by padding zeros (Figure 4.1). For our experiments, the Hamiltonian network is re-implemented. We will be using 3 Units with m Hamiltonian Blocks with average pooling with pool size 2 (window size over which the maximum or the average is taken) to decrease the size of the feature map, followed by 1×1 convolution (with ReLU) instead of zero-padding to increase the number of channels. The convolutions within each Block are all 3×3 convolutions with spatial zero-padding to maintain feature map size. For the image classification task, the number of nodes in the fully-connected layer is equal to the number of image classes in the dataset. Therefore, a normal Hamiltonian network has $12m + 2$ layers, with $4m$ layers in each Unit (2 convolutions and 2 transposed convolutions per Hamiltonian Block).

The Nonlocal Block is then added after the second Block in each Unit, and this network is trained and tested on several image classification benchmark datasets, such as CIFAR-10, CIFAR-100 [Kri09] and STL-10 [CNL11], in order to assess the performances and effects of each nonlocal operator in the network. The networks are also used for the semantic segmentation task in autonomous driving. To this end, the BDD100K dataset [YCW+20] is used, which is a large-scale dataset of visual driving scenes.

CIFAR-10 consists of 50,000 training images and 10,000 testing images in 10 classes, where the resolution of each RGB image is 32×32. CIFAR-100 contains the same images, with the same train-test split, but the images are categorized in 100 classes. STL-10 consists of 96×96 RGB images, and therefore the network needs a larger receptive field due to the higher resolution of the images. The training and test sets contain 5000 and 8000 images, respectively. Apart from the high resolution, the classification task for STL-10 is more challenging also because of the relatively small number of training samples. For the semantic segmentation task, the BDD100K dataset consists of 7000 and 1000 training and test images (RGB images) respectively, with each pixel belonging to one of the 20 classes. The resolution of each image is 1280×720. To demonstrate the performance for the segmentation task, we use small networks, and therefore we resize the images to a resolution of 160×90. This is done by removing every second row and column from each image and performing the same operation iteratively on the resulting image.

For training with CIFAR-10 and CIFAR-100, the mini-batch size is kept at 100. The per-pixel mean from the input image is subtracted before training [KSH12], and the means of the training data are used to perform per-pixel mean subtraction on the test data. Common data augmentation techniques [LXG+15] are performed, such as padding 4 zeros around the image, followed by random cropping and random horizontal flipping. This makes the network more robust and helps it to generalize better. At the end of the network, before the fully-connected layer, we have an average pooling layer with a pool size of 2 (Figure 4.1). For subsampling within the Nonlocal Block, max-pooling is performed with a pool size of 2, i.e. 2×2 patches of the image are used to compute the affinity kernel ω.

For the STL-10 dataset, the mini-batch size is 50. The same preprocessing and data augmentation techniques are used but with a padding of 12 zeros around the image before cropping, instead of 4, because the images in the STL-10 dataset are significantly larger. At the end of the network, before the fully-connected layer, we have an average pooling layer with pool size 8. For subsampling within the Nonlocal Block, max-pooling is performed with a pool size of 4, i.e. in this case, 4×4 patches of the image or feature maps are used to compute the affinity kernel ω.

For the segmentation task with the BDD100K dataset, the mini-batch is of size 8. This value is intentionally kept small to avoid a large memory footprint due to the kernel computation and the relatively high image resolution. For the data augmentation, the per-pixel mean subtraction is performed, and then the image is randomly flipped horizontally. In a usual neural network for semantic segmentation tasks, the image is subsampled several times and then upsampled again to get a high-dimensional output. We use a simple architectural design here, i.e. the pooling layers that are shown in Figure 4.1 are left out. This way, the spatial dimensions of the image are maintained. At the end of the network, instead of a fully-connected layer, the output of the last unit is passed through 1×1 convolutional layer with 20 filters, which gives us the prediction of each of the pixels in the image. For subsampling within the Nonlocal Block, max-pooling is performed with a pool size of 3.

All the networks are implemented and trained using the Tensorflow library [AAB+16] with a single Nvidia Tesla P100 GPU. The deeper networks (section 5.3.4) and the networks for the semantic segmentation task (section 5.5) are trained using an Nvidia Tesla V100 GPU. All the trainable weights in the neural network are initialized based on the suggestions in [HZR+15]. 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. For a more in-depth analysis on the effects and advantages of the parameter h, and for the right choice of the value of h, see [ZHW+19].

The loss function S in equation (2.11) is chosen to be the cross-entropy loss function (also called the log-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 decay 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 [CMH+18b]. As discussed before, the convolutional weights in the Nonlocal Block are only regularized by weight decay (2×10^{-4} for all the datasets–CIFAR-10/100, STL-10, BDD100K) and not by the weight smoothness decay. The scaling factor λ is 0.1 for all the nonlocal operators with the dimensional constant $n = 2$.

Chapter 5

Numerical Experiments

All the four PIDE-based Nonlocal Blocks given by equations (4.1), (4.6), (4.7) and (4.8) are added to the Hamiltonian network. Their performances on several benchmark datasets are then compared to ResNets of a similar size and to the original Hamiltonian network without the Nonlocal Block. We look at the effects of architectural changes on the performance and the robustness of the networks to noise. Finally, the neural networks are used for the semantic segmentation task in autonomous driving. We will call the networks with the Nonlocal Blocks as Nonlocal Hamiltonian networks, and the one without the nonlocality is just called Hamiltonian network. Each experiment is run five times with different random seeds, and the median test accuracy is reported in each case.

5.1 Architecture of the Hamiltonian network

In [CMH+18b], the smallest Hamiltonian network proposed by Chang, Meng, Haber et al. is a network with 74 layers, with 6 Hamiltonian Blocks in each of the three Units (6-6-6). The initial convolutional layer has 32 filters, and the number of channels in each Unit is {32, 64, 112}. Such architectures with three Units have been used in ResNets and RevNet, and have become a common blueprint for a well-performing CNN. Now, we try to change the number of Blocks in each Unit and see the corresponding effects on performance. Note that the majority of the network weights come from the third Unit because of the higher number of channels, which considerably increases the trainable convolutional weights in a network. For example, while testing this network on the CIFAR-10 database, the Hamiltonian network has 0.50M parameters (M stands for million). Therefore, if we want to have a different Hamiltonian network with a given number of trainable weights, say around 0.50M parameters, then one way to achieve this is by removing a lot of the Hamiltonian Blocks in earlier Units, and then in return adding a Block or two in the third Unit to maintain the total number of weights in the network. Or conversely, we can have a lot of extra Blocks in the earlier Units and remove a couple of Blocks from the last Unit. In order to see what effect these architectural changes have on the Hamiltonian network, these networks are tested on the benchmark datasets for image classification (Table 5.1) and compared to the original Hamiltonian network proposed in [CMH+18b].

The (6-8-5) architecture results are comparable to the (6-6-6) architecture, and at times worse. The (2-3-7) architecture performs significantly worse than the other two

Units	**Channels**	**CIFAR-10**		**CIFAR-100**		**STL-10**	
		Params (M)	**Acc. (%)**	**Params (M)**	**Acc. (%)**	**Params (M)**	**Acc. (%)**
6-6-6	32-64-112	0.50	92.75	0.67	70.38	0.50	79.95
2-3-7	32-64-112	0.49	92.02	0.65	69.53	0.48	79.25
6-8-5	32-64-112	0.49	92.78	0.65	70.31	0.48	79.90

Table 5.1: Test accuracy for different number of Blocks in each Unit of the Hamiltonian network.

networks. This confirms the fact that the earlier layers are very important to the networks. In [HSL+16], Huang, Sun, Liu et al. randomly drop certain layers while training, but the probability of the later layers being dropped is higher than the ones for the earlier layers. This is because the earlier layers extract low-level features that will be used by later layers. Therefore, we do not drop any Blocks in the earlier Units of the network. Since we see no increase in test accuracy for different number of Blocks in each Unit, we stick to the simple (6-6-6) architecture for our experiments. This way, for each feature map size, i.e. for each Unit, the network has equal expressive power. Later on, in section 5.3.4, the performance of deeper networks (18-18-18) is discussed, where the improvement in accuracy is mainly driven by the extra Blocks in the earlier Units.

5.2 Q-tips dataset with nonlocal dependence

We first test the four networks on a synthetic dataset called Q-tips that was introduced by Haber, Lensink, Treister et al. in [HLT+19] in order to test the extent of the receptive field of a network in semantic segmentation. For purposes of image classification, the dataset is slightly modified. Each image consists of a black midsection with a colored square at the end of the *stick*. The stick belongs to one of 15 classes, depending on the colored markers (red, green, blue, yellow and pink) that are present on each end, as shown in Figure 5.1. Hence, in order to classify the image properly, the network needs to facilitate long-range communication between the regions of the image, and without information about both ends of the stick, the image will be classified wrongly by the network.

The dataset consists of 512 training and validation examples. Each of the images is a 64×64 3-channel image with a stick of length l, width w and rotational orientation r, where the values are chosen from a discrete uniform distribution with $l \in \mathcal{U}\{32, 60\}$, $w \in \mathcal{U}\{4, 8\}$, $r \in \mathcal{U}\{-180, 180\}$. The pixel values are normalized to $[0, 1]$, and then a Gaussian noise $\eta \sim \mathcal{N}(0, 0.3)$ is added to make the classification task more challenging.

The number of Hamiltonian Blocks in each Unit of the network is equal to 6. The initial convolutional layer has 32 filters, and the number of channels in each Unit is {32, 64, 112}. This way, the original Hamiltonian network has 0.50M trainable parameters for the Q-tips dataset. Then the Nonlocal Block with each of the four operators is added, and this results in a network with 0.56M trainable parameters. ResNet-44 [HZR+16a] is chosen for comparison because it is a network of a similar size, in terms of the number of trainable parameters (0.66M).

The training is performed with a mini-batch size of 50. The learning rate for SGD is

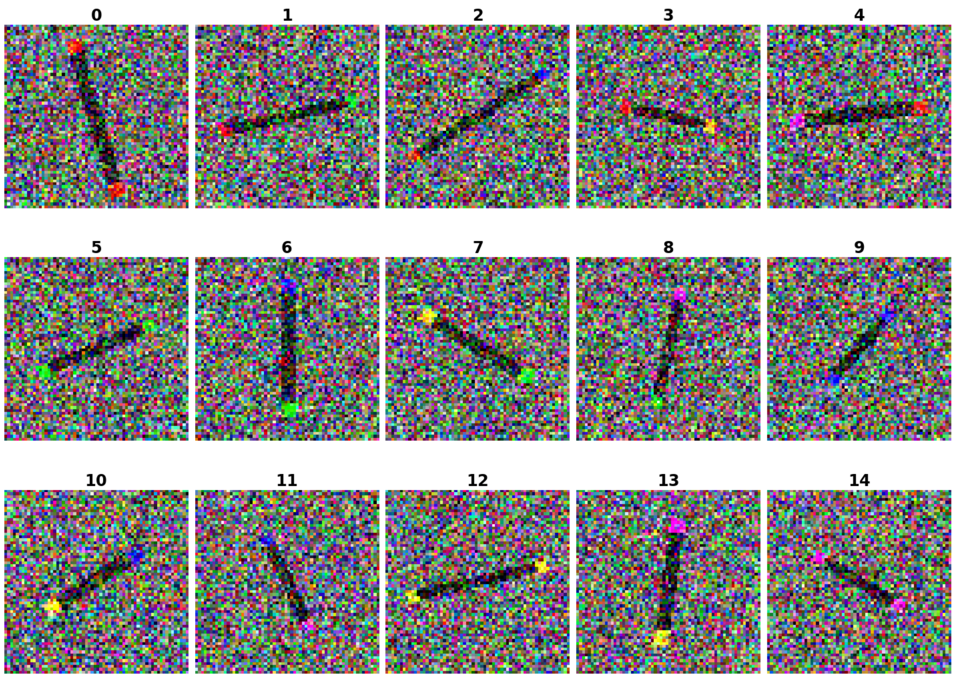

Figure 5.1: Image classes for the Q-tips dataset with Gaussian noise.

kept at 0.01, and the training is done for 30 epochs. All the other hyperparameters are the same as in the CIFAR-10 setting discussed in the last chapter.

The training curves can be seen in Figure 5.2 for the four nonlocal operators, with $s = 1/2$ for the pseudo-differential operators. The networks start the training pretty equally, and at times ResNet is better than the Nonlocal Hamiltonian networks, but after a few epochs, the nonlocal interactions come to the fore, and we see that the Nonlocal Hamiltonian networks accelerate the exchange of information between far away pixel in the images due to the networks' wider field-of-view, and therefore we have faster convergence of the training. On the other hand, the training curve for ResNet stagnates after a while, which suggests that ResNets need a deeper network with more convolutional layers to learn more about the dataset. The test curves are not considered in this case because the data is not complex enough. Therefore, there is a lot of fluctuation in the test accuracy when the network is training.

5.3 Nonlocal Hamiltonian networks on image classification benchmark datasets

5.3.1 Training on benchmark datasets

The Nonlocal Hamiltonian networks, with the different PIDE-based Nonlocal Blocks given by (4.1), (4.6), (4.7) and (4.8), are inserted in each Unit of the network and tested on the benchmark datasets CIFAR-10, CIFAR-100 and STL-10. The network architecture

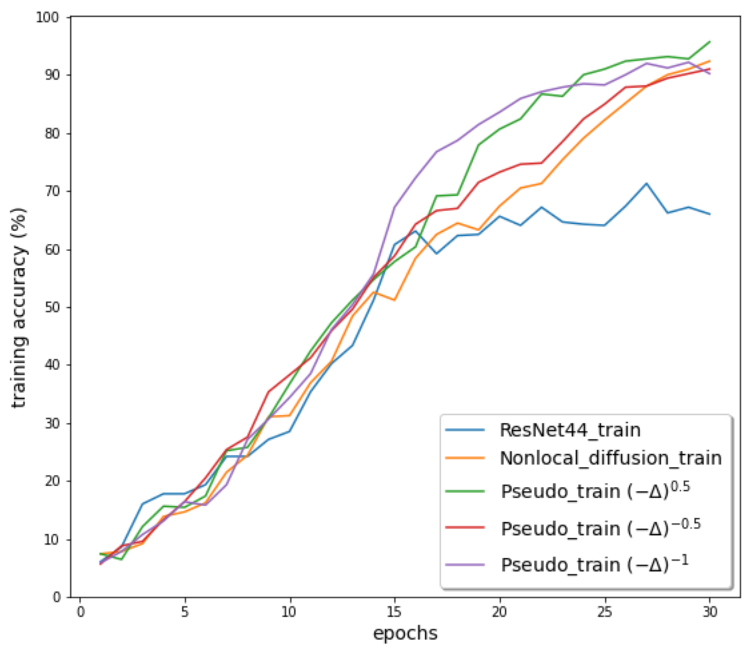

Figure 5.2: Training accuracy for Nonlocal Hamiltonian networks and ResNet-44 on the Q-tips dataset with Gaussian noise.

(6-6-6) is used, with {32, 64, 112} being the number of channels in the three Units. The value of s for the pseudo-differential operators is $1/2$. In the next section, the effect of s on the performance is explored. The rest of the implementation details have been discussed in the previous chapter. The original Hamiltonian network, ResNet-44 and PreResNet-20 [HZR+16b] are used as baseline networks for comparison and are trained on each of the three datasets. The main results with the test accuracies are shown in Table 5.2. The best result for each benchmark dataset is marked in boldface.

Network	**CIFAR-10**		**CIFAR-100**		**STL-10**	
	Params (M)	**Acc. (%)**	**Params (M)**	**Acc. (%)**	**Params (M)**	**Acc. (%)**
ResNet-44	0.66	92.64	0.66	69.51	0.66	75.79
PreResNet-20	0.57	92.35	0.59	71.07	0.57	77.39
Hamiltonian-74	0.50	92.75	0.67	70.38	0.50	79.95
Nonlocal diffusion $\mathcal{L}$	0.56	**93.27**	0.72	71.81	0.55	**82.62**
Pseudo-differential $(-\Delta)^{1/2}$		93.21		**71.84**		81.88
Pseudo-differential $(-\Delta)^{-1/2}$		93.08		71.24		81.56
Pseudo-differential $(-\Delta)^{-1}$		92.88		71.25		80.73

Table 5.2: Test accuracies on benchmark datasets for different Nonlocal Hamiltonian networks.

The figures in Table 5.2 show that all of the Nonlocal Hamiltonian networks perform better on the benchmark datasets. They also have fewer parameters than the baseline networks of ResNet and PreResNet networks (for CIFAR-10 and STL-10). The Nonlocal diffusion $\mathcal{L}$ network performs the best, closely followed by the Pseudo-differential $(-\Delta)^{1/2}$ network, which can be seen as a special case of the nonlocal diffusion operator, as pointed out in section 3.2.1. The inverse Laplacian operator $(-\Delta)^{-1}$ with the log kernel performs the worst, but it nevertheless outperforms the baseline networks. Besides, the accuracies of the networks are rather comparable to that of ResNet-56 and ResNet-110 and not to ResNet-44 (see [HZR+16a]). This shows that the presence of nonlocal interactions greatly reduces the need for deeper networks to achieve a certain level of accuracy. The better performance of the proposed neural networks in comparison to ResNets is quite encouraging because, at the time of writing this thesis, on CIFAR-10 and CIFAR-100 datasets, the BiT-L network [KBZ+19], which is considerably larger than the networks discussed here, has the best test accuracy, namely 99.37% and 93.51% respectively. It is based on the ResNet architecture, and it uses transfer learning to produce very compelling results.

The Nonlocal diffusion network performs better than the one suggested in [TSD+18], which did not have any skip connections in the Nonlocal Block. This suggests that introducing skip connections within the Nonlocal Block makes sure that the network learns only the update (or the residual) made to the identity function, and hence avoids any impedance when the information travels through the Nonlocal Block.

The Nonlocal diffusion $\mathcal{L}$ network and the Pseudo-differential $(-\Delta)^{1/2}$ network perform comparatively better than the other two proposed neural networks. This is possibly because of the difference term $[u(\mathbf{x}) - u(\mathbf{y})]$ (and $[u(\mathbf{y}) - u(\mathbf{x})]$). When computing each pixel in the output feature map, this term compares and computes the relative differences between the neighboring pixel values of the input feature map, whereas the operator $(-\Delta)^{-1/2}$ just computes a weighted sum of $u(\mathbf{y})$ (see Definition 3.11). Also, we see the biggest increase in accuracy for the STL-10 dataset. This is due to the fact that the images in this dataset are relatively larger than the ones in CIFAR-10/100, and therefore the nonlocality plays a more vital role in widening the receptive field of the network, which enables direct communication between pixels on opposite ends of the image. The test accuracy curves (till the 160th epoch) for the Nonlocal diffusion network with each of the three datasets are shown in Figure 5.3. The Hamiltonian-74 is used for comparison since it has the best performance out of all the three baseline networks. The test accuracy curves for the other Nonlocal Hamiltonian networks can be found in Appendix A.2.

The Nonlocal Hamiltonian networks can actually achieve better results for the STL-10 dataset. With a larger step size h, around 84%-84.5% test accuracy on STL-10 can be attained. The value of h is intentionally kept low because, as pointed out in [ZHW+19], a small h encourages the model to have smaller weights. A model with larger weights tends to suffer from overfitting. Such models are also vulnerable to adversarial attacks. For example, the Pseudo-differential $(-\Delta)^{1/2}$ network was trained with a higher value of h, namely $h = 0.15$, and it achieved a median test accuracy of 84.12%. In this case, the real parts of the eigenvalues of the weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$ in the Nonlocal Block, along with the eigenvalues of the symmetric parts of the weight matrices (see section 6.2), can be found in Appendix A.3. Furthermore, while running the experiments, with a higher value of h like 0.2, the network often ended up being untrainable due to exploding gradients.

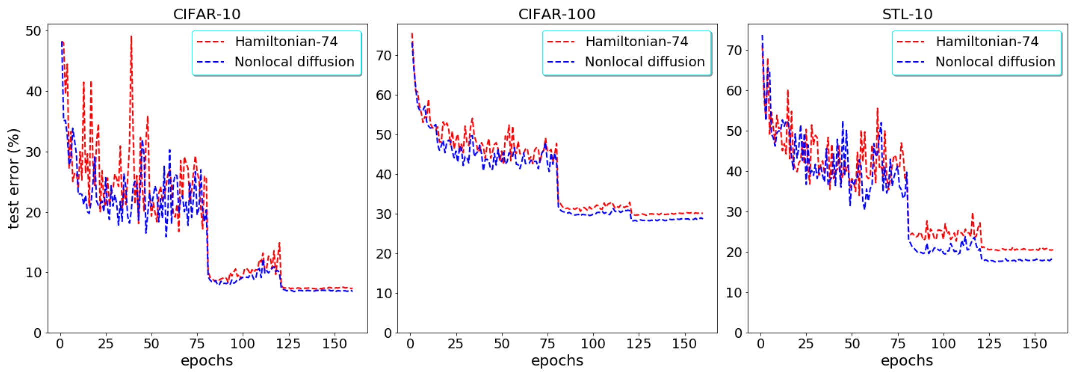

Figure 5.3: Test accuracy curves for the Nonlocal diffusion $\mathcal{L}$ network in comparison to the original Hamiltonian network.

5.3.2 Effect of the parameter s

The pseudo-differential operators, given by Definitions 3.7 and 3.11 (for $n = 2$), hold only for $0 < s < 1$. In the previous sections, the experiments were conducted using $s = 1/2$. In this section, we vary the power of the fractional Laplacian and the inverse fractional Laplacian operator and see if it has any effect on the performance of the corresponding networks.

Nonlocal Network	s	**CIFAR-10**		**CIFAR-100**		**STL-10**	
		Params (M)	**Acc. (%)**	**Params (M)**	**Acc. (%)**	**Params (M)**	**Acc. (%)**
$(-\Delta)^s$	1/2	0.56	93.21	0.72	71.84	0.55	81.88
	1/4		93.17		71.56		81.70
	3/4		93.24		71.73		81.72
$(-\Delta)^{-s}$	1/2		93.08		71.24		81.56
	1/4		93.11		71.29		81.43
	3/4		93.02		71.32		81.54

Table 5.3: Test accuracies on benchmark datasets for Nonlocal Blocks containing pseudo-differential operators with different values of s.

The two pseudo-differential operators are implemented in the PIDE-based Nonlocal Block with $s = 1/4$ and $s = 3/4$. The results of the tests can be seen in Table 5.3. We see very minimal changes to the performance of the network for each of the three datasets. This indicates 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.

5.3.3 Multi-stage Nonlocal Blocks

The discretizations of the nonlocal operators have a 2-stage computation (Figure 4.3). To see the effect of the number of stages in the Nonlocal Block, the Nonlocal diffusion network with four stages is taken as an example and is trained on the datasets (Table 5.4).

# Stages	CIFAR-10		CIFAR-100		STL-10	
	Params (M)	Acc. (%)	Params (M)	Acc. (%)	Params (M)	Acc. (%)
2	0.56	93.27	0.72	71.81	0.55	82.62
4	0.59	93.09	0.76	71.25	0.59	81.36

Table 5.4: Test accuracies on benchmark datasets for the Nonlocal diffusion $\mathcal{L}$ network with different number of stages in the Nonlocal Block.

The figures show that too many stages in the Nonlocal Block damp the signal in the network, and hence, the network performs worse or at least does not perform any better than the network with a 2-stage Nonlocal Block. This is in line with the observations made in [TSD+18] and is also discussed in Lemma 3.4. Operators similar to the inverse fractional Laplacian also suffer from instabilities if the number of stages is increased (see [TSD+18]). Therefore, although a nonlocal operation of this kind improves performance when the number of stages is kept low, overusing the nonlocal operation can lead to a lossy network. Also, the affinity kernel ω is computed once at the beginning of each Nonlocal Block to save computational cost. The features change so much after the first few stages that the kernel ω cannot depict the affinities accurately anymore after several stages.

Besides, although the Nonlocal Block has only a small number of trainable parameters, and the increase in the number of parameters is small for a 4-stage network, the nonlocal operation has a computational cost (due to pairwise distance calculations, etc.), and one should try to keep this to a minimum. Too many stages lead to a large number of floating-point operations, with very little or no gain in performance.

5.3.4 Deeper Nonlocal Hamiltonian networks

In this section, we show that the PIDE-based Nonlocal Blocks also work for deeper architectures. To this end, we use the Hamiltonian-218 network with 18 Blocks in each Unit (18-18-18) with {32, 64, 128} number of channels in the three Units. Then we add the Nonlocal Block just before the last Hamiltonian Block in each Unit, as proposed in [WGG+18]. These networks are then trained and tested on the CIFAR-10 dataset, with ResNet-110, ResNet-164 and the original Hamiltonian-218 as baselines.

Table 5.5 shows that the proposed neural networks compete well and at times outperform the state-of-the-art networks of a similar size. In comparison to the original Hamiltonian-218 network, all the Nonlocal Hamiltonian networks show accuracy improvements in the presence of the Nonlocal Block in their Units. Three out of the four Nonlocal Hamiltonian networks outperform the ResNet-164 network, although the ResNet-164 network has substantially more trainable parameters. This reiterates the fact that the nonlocal connections and their larger receptive fields somewhat compress the ResNet-like architectures. These connections partially alleviate the necessity to have very deep networks, and hence, we can save training time by training shallower networks with Nonlocal Blocks in them.

Network	CIFAR-10	
	Params (M)	**Acc. (%)**
ResNet-110	1.73	93.65
ResNet-164	2.61	93.79
Hamiltonian-218	1.78	93.64
Nonlocal diffusion $\mathcal{L}$	1.85	93.96
Pseudo-differential $(-\Delta)^{1/2}$		**94.04**
Pseudo-differential $(-\Delta)^{-1/2}$		93.86
Pseudo-differential $(-\Delta)^{-1}$		93.77

Table 5.5: Test accuracies on benchmark datasets for different Nonlocal Hamiltonian networks with 18 Hamiltonian Blocks in each Unit.

5.4 Robustness to noise and training data subsampling

The nonlocality of a network has a couple of knock-on effects, such as robustness to noise and to training data subsampling, which makes the networks more appealing in comparison to several traditional neural architectures. For the experiments in this section, we use the shallower networks (6-6-6) because the aim is to check how robust each network is and not to challenge the state-of-the-art networks by achieving better test accuracies.

5.4.1 Robustness to noise

Neural networks are vulnerable to adversarial examples, which leads to misclassification of examples that have been slightly perturbed [SZS+14; GSS15]. In fact, in [TSE+19], it is claimed that there is always a trade-off between adversarial robustness and accuracy. Therefore, the aim of each network should be to keep the sensitivity of the features to perturbations in the inputs in check, and if possible, offer relatively high performance at the same time. To examine this property, the Nonlocal Hamiltonian networks are tested on images with Gaussian noise, similar to the experiments in [ZS19]. The network is at first trained on uncorrupted training sets. After that, some noise is added to the test examples, and these examples are used to assess the predictions made by the network.

For a given test image $\mathbf{x}$, the corrupted image is obtained via $\mathbf{x} \mapsto \mathbf{x} + \eta$, where $\eta \sim \mathcal{N}(0, \sigma^2)$, and σ is chosen to be 0.02. The test accuracies are shown in Table 5.6, with the best result marked in boldface. In most cases, we see that the proposed networks perform better than the baseline neural networks. Interestingly, although the Pseudo-differential $(-\Delta)^{1/2}$ network performed slightly worse than the Nonlocal diffusion network on the benchmark datasets (section 5.3.1), it seems to be at times more robust to noise than the Nonlocal diffusion network. This shows that a higher test accuracy of a network does not automatically mean more robustness to noise. The results also show that the Nonlocal Hamiltonian networks are better equipped to keep the sensitivity of the network to perturbations in check.

Network	**CIFAR-10**	**STL-10**
	Acc. (%)	**Acc. (%)**
ResNet-44	89.48	75.23
PreResNet-20	88.12	74.04
Hamiltonian-74	89.50	79.75
Nonlocal diffusion $\mathcal{L}$	89.81	**80.52**
Pseudo-differential $(-\Delta)^{1/2}$	**90.25**	79.94
Pseudo-differential $(-\Delta)^{-1/2}$	89.62	80.10
Pseudo-differential $(-\Delta)^{-1}$	89.55	79.43

Table 5.6: Test accuracies on benchmark datasets for different Nonlocal Hamiltonian networks in the presence of Gaussian noise.

5.4.2 Robustness to data subsampling

Often, the challenge in deep learning is to acquire enough labeled data. Therefore, it is advantageous if a neural network can generalize relatively well, even when only a fraction of the entire dataset is used during training. To verify this property for the proposed Nonlocal Hamiltonian networks, the networks are trained on a fraction of the entire training data, and the accuracy of their predictions on all the test examples is observed (Table 5.7).

For CIFAR-10, the networks are trained with 5%, 10% and 15% of the training data, but for STL-10, it is 10%, 20% and 40% because of the smaller size of the dataset. It is to be noted that with lesser training data, there is generally a larger variance of the test accuracy over different random seeds. In most cases, the proposed networks have a better test accuracy in comparison to the baseline networks. The pseudo-differential operators $(-\Delta)^{1/2}$ and $(-\Delta)^{-1/2}$ work relatively better than the other two operators. The results show that the Nonlocal Hamiltonian networks, on average, need lesser training data to learn the features in the dataset and to achieve a certain generalization power.

Network	**CIFAR-10**			**STL-10**		
	5%	**10%**	**15%**	**10%**	**20%**	**40%**
ResNet-44	59.23	74.60	81.52	42.71	55.27	67.23
PreResNet-20	67.04	76.58	81.20	46.06	58.96	68.44
Hamiltonian-74	68.28	77.07	81.31	44.80	59.07	68.61
Nonlocal diffusion $\mathcal{L}$	68.94	77.20	81.66	46.26	58.76	**69.95**
Pseudo-differential $(-\Delta)^{1/2}$	68.61	77.19	**81.76**	**47.28**	58.49	69.58
Pseudo-differential $(-\Delta)^{-1/2}$	**69.12**	**77.53**	81.34	46.65	**59.47**	68.88
Pseudo-differential $(-\Delta)^{-1}$	68.47	77.33	81.68	45.58	59.07	68.96

Table 5.7: Test accuracies on benchmark datasets with decreased training data for different Nonlocal Hamiltonian networks.

5.5 Semantic segmentation on BDD100K

The semantic segmentation task can be viewed as a multi-dimensional classification problem, where each pixel is assigned a particular label. The aim of the learning problem is to predict the class of each pixel and thereby partition the image into different image classes or objects. This gives us an idea of where the object is present in the image and which pixels belong to it. This is of paramount importance, for example, for self-driving cars because the position and type of the object in front are necessary inputs to any self-driving algorithm. Other applications of segmentation include disease diagnosis and prevention, where one can perform tumor segmentation from medical images [HDWF+17]. The pixel-level, fine-grained control is also critical in the field of robotics and image search engines.

The Nonlocal Hamiltonian networks are used to segment images from the BDD100K dataset [YCW+20], which consists of images with everyday driving scenarios. An example of an image, along with its segmentation mask, is provided in Figure 5.4. Our aim here is to show the increase in performance due to the presence of nonlocal connections and not to challenge the state-of-the-art networks. Therefore, we have simple architectures with no pooling layers to maintain the spatial dimension of the feature maps, and we work with smaller image resolutions, as mentioned in section 4.4.

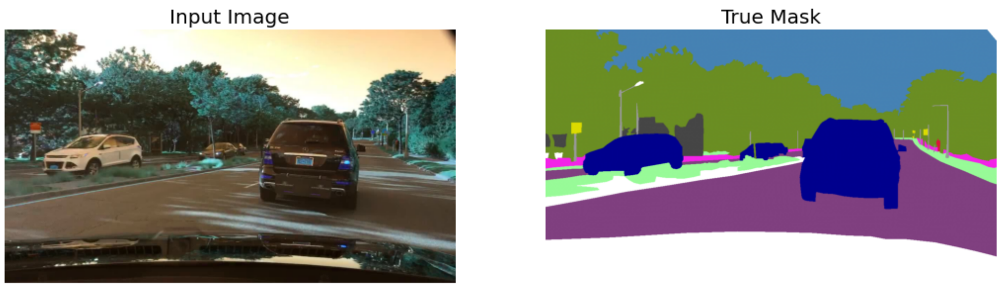

Figure 5.4: Input images and corresponding labels from Berkeley DeepDrive (BDD100K) dataset for the semantic segmentation task [YCW+20].

To evaluate the performance, two metrics will be used. The first one is the pixel accuracy, which simply reports the fraction of pixels in the image that are classified correctly. We can represent pixel accuracy as

$$\text{pixel accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{5.1}$$

where, for a class c, TP (true positive) is the number of pixels classified correctly as c, FP (false positive) is the number of pixels classified incorrectly as c, TN (true negative) is the number of pixels classified correctly as not c, and FN (false negative) is the number of pixels classified incorrectly as not c. This metric can be misleading at times when we have an imbalanced representation of classes in the images.

The second metric that we use is the Intersection-over-Union (IoU) metric (or the *Jaccard index*). It tries to estimate the overlap between the ground truth and the prediction, and it maintains equal weights for all the classes. The IoU metric is given by

$$\mathrm{IoU} = \frac{TP}{TP + FP + FN}. \tag{5.2}$$

This is calculated on a per-class basis, and the mean is then taken over all the classes, which gives us the meanIoU (mIoU) metric. If the denominator is zero, i.e. if the class does not show up in a batch of images, then the value is set to 1 since the network correctly predicts the absence of that particular class.

Network	**BDD100K**		
	Params (M)	**P. Acc. (%)**	**mIoU (%)**
ResNet-44	0.66	83.50	63.42
PreResNet-20	0.57	77.86	52.12
Hamiltonian-74	0.49	84.47	64.40
Nonlocal diffusion $\mathcal{L}$	0.54	84.80	**67.01**
Pseudo-differential $(-\Delta)^{1/2}$		**85.24**	66.71
Pseudo-differential $(-\Delta)^{-1/2}$		85.21	66.52
Pseudo-differential $(-\Delta)^{-1}$		84.95	66.26

Table 5.8: Pixel accuracies and meanIoU for the semantic segmentation task on the BDD100K dataset using the Nonlocal Hamiltonian networks.

The results for the proposed neural networks (with the 6-6-6 and {32, 64, 112} architecture) can be seen in Figure 5.8. When we consider the pixel accuracy and the mIoU metric, the Nonlocal Hamiltonian networks perform better than the baseline networks. The Nonlocal Hamiltonian networks' metric values are very close to each other. In terms of pixel accuracy, the Pseudo-differential $(-\Delta)^{-1/2}$ network and the Pseudo-differential $(-\Delta)^{1/2}$ network perform slightly better than the other two Nonlocal Hamiltonian networks. Surprisingly, the PreResNet-20 network performs relatively poorly in comparison to the other networks. Since the pixel accuracy metric can sometimes be misleading, we can look at the mIoU metric to have a better understanding of each network's performance. In this regard, the Nonlocal diffusion network performs marginally better than the other Nonlocal Hamiltonian networks. We see that the Nonlocal Hamiltonian networks have a performance gain of around 2–2.5 mIoU percentage points when we compare it to the original Hamiltonian network and around 3–3.5 mIoU percentage points when we compare it to ResNet-44.

To make sure that the networks actually learn to segment the images from driving scenes into semantically meaningful parts, we look at the predictions made by the networks. Figure 5.5 shows the true label of a test image (with a taxi) and the different predictions made by some of the networks discussed above. ResNet-44 has several pixels that are wrongly labeled 'black'. The Hamiltonian network, on the other hand, partly classifies the sky as a car. There are, of course, images where the baseline networks have better predictions, but Table 5.8 suggests that, on average, the Nonlocal Hamiltonian networks have better predictions.

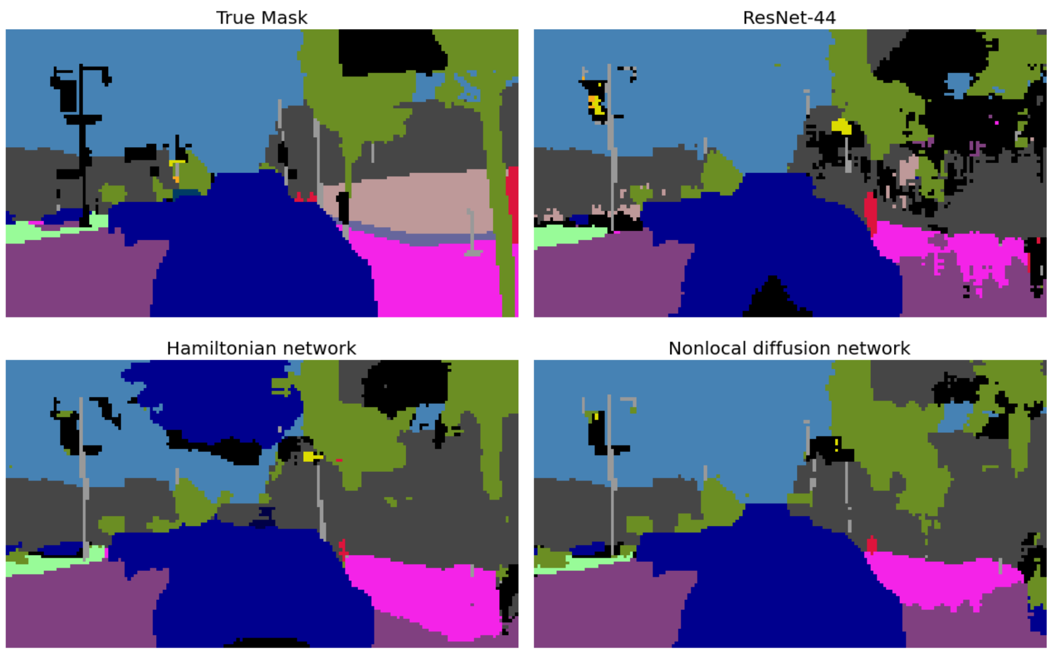

Figure 5.5: The true labels/masks of a test image from the BDD100K dataset and the predictions made by the different networks.

The results discussed here can be improved by performing segmentation using traditional encoder-decoder models that subsample the image several times and then *upsample* it back to get the predictions, i.e. the full-resolution segmentation map. Common architectures in this regard include the *Fully Convolutional Network* (FCN) [LSD15] and *U-Net* [RFB15], which contains skip connections for the upsampling layers. The results can also be improved by using other data augmentation techniques that have not been used here, or by using a deeper network, which might be necessary because of the high resolution of the images in the BDD100K dataset.

Chapter 6

Computational cost and forward propagation stability

6.1 Computational cost of the Nonlocal Blocks

The Nonlocal Blocks come with a computational cost that needs to be taken into consideration while inserting it into any neural network. Several strategies to reduce the number of floating-point computations have been discussed in section 4.2.1. The number of FLOPs (multiply-adds) associated with the proposed networks for the image classification task are shown in Table 6.1.

Network	**CIFAR-10**	**CIFAR-100**	**STL-10**
	FLOPs (M)	**FLOPs (M)**	**FLOPs (M)**
ResNet-44	98.3	98.3	879.5
Hamiltonian-74	159.6	159.9	1432.5
Nonlocal diffusion $\mathcal{L}$	192.9	193.2	2003.7
Pseudo-differential $(-\Delta)^{1/2}$	193.4	193.7	2012.5
Pseudo-differential $(-\Delta)^{-1/2}$	193.2	193.5	2011.0
Pseudo-differential $(-\Delta)^{-1}$	193.6	193.9	2019.5

Table 6.1: Number of floating-point operations (multiply-adds) for each of the networks when trained on the image classification benchmark datasets.

To ensure the stability of the forward propagation, we can observe that the Hamiltonian-74 network has roughly 1.5 times more FLOPs than the traditional ResNet-44. Also, the ResNets generally have only two convolutional layers in a Residual Block. In comparison, Hamiltonian Blocks have 2 convolutions and 2 transposed convolutions, which explains the higher number of FLOPs. The Nonlocal Blocks by themselves add very little extra floating-point operations for CIFAR-10 and CIFAR-100 but nevertheless help the networks achieve better accuracy on the benchmark datasets. Therefore, instead of stacking more convolutional layers to improve the performance of a given network, adding a Nonlocal Block can be an effective and efficient way of widening the field-of-view and thereby improving performance.

When it comes to datasets like STL-10 (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 that was mentioned in section 4.2.1. 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 6.2 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. As mentioned before, for the experiments in the previous chapter, the subsampling in the Nonlocal Block is performed using max pooling with a pool size of 2 and 4 for CIFAR-10/100 and STL-10, respectively.

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

Table 6.2: Number of floating-point operations (multiply-adds) depending on the subsampling pool size in the Nonlocal Blocks while training the Nonlocal diffusion network 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. Instead of comparing individual pixel strips, we compare the affinity between large sections of the image. The nonlocal interactions are therefore not sensitive and strong enough. Consequently, this would lead to very little increase in performance when the PIDE-based Nonlocal Block is inserted into any neural network. 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 in the dataset.

Independent of the choice of the pool size, the presence of Nonlocal Blocks in a neural network does not worsen the performance of the network. It rather improves performance,

[1]Not applicable because the original Hamiltonian network has no Nonlocal Blocks

and the increase in performance is dependent on the pool size of the subsampling in the Nonlocal Block. Therefore, for a given network, adding Nonlocal Blocks is a more effective way of improving performance than adding another normal Block (Hamiltonian, Residual, etc.), which would increase the number of trainable parameters and would make the network deeper and would make the optimization more challenging. On the other hand, one should remember that, as pointed out before, too many Nonlocal Blocks can damp the information that gets propagated down the layers and therefore wipe out any relevant differences and variations in the feature maps. That is to say, the normal Blocks are also necessary, and the Nonlocal Blocks should only be placed sporadically between the normal Blocks of the network due to damping effects and computational cost reasons.

Note that, the number of FLOPs increases roughly quadratically when the pool size is decreased by half. For example, as we halve the pool size from 8 to 4, 4 to 2 and 2 to 0, the increase in FLOPs is roughly 400M (1636.8M to 2003.7M), 1500M (2003.7M to 3471.4M) and 6000M (3471.4M to 9341.2M) respectively. The number of FLOPs roughly quadruples when the pool size is halved.

6.2 Stability of the PIDE-based forward propagation

It is crucial that the newly introduced PIDE-based Nonlocal Block does not introduce any instabilities in the forward propagation. If it did, then it would nullify the advantage of the ODE/PDE-based Hamiltonian network having the stability of the forward propagation. This would, as discussed before, lead to oversensitivity or insensitivity of the output on the input data. The depth of the network and the spectral norms of the weight matrices have a major role to play in this regard. To investigate this, we look at the spectral nature of the weights in the Nonlocal Block.

Other than the embeddings, the Nonlocal Block has two 1×1 convolutional layers $\mathcal{K}_1$ and $\mathcal{K}_2$ for the two stages. For the four Nonlocal Hamiltonian networks used in sections 5.2 and 5.3.1, the weights of the convolution layers have the dimensions $\mathcal{K}_{1,2} \in \mathbb{R}^{32\times32}$, $\mathcal{K}_{1,2} \in \mathbb{R}^{64\times64}$, $\mathcal{K}_{1,2} \in \mathbb{R}^{112\times112}$, and they belong to Unit 1, Unit 2 and Unit 3 of the network respectively. The outputs of these transformations are the feature maps that are used later on in other Blocks of the network. Thus, we look at the eigenvalues of these matrices to see how the features are transformed by these trainable weights. Figures 6.1 and 6.2 show the real parts of the eigenvalues of the weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in each Unit of the Nonlocal diffusion network while training on CIFAR-10 and STL-10, respectively. The real parts of the eigenvalues for the other Nonlocal Hamiltonian networks can be found in Appendix A.4.

The plots show that the real parts of the eigenvalues of the trainable weights are mostly very close to zero. The plus sign in each plot indicates the fraction of eigenvalues that have positive real parts. In the ideal case, this value should stay close to 0.5. Too many eigenvalues with positive real parts would lead to amplification of the signal leading to instabilities in the forward propagation. On the other hand, too many eigenvalues with negative real parts would lead to a lossy network. The plots for the Nonlocal diffusion network are pretty symmetric. The plots for the Pseudo-differential networks $(-\Delta)^{1/2}$, $(-\Delta)^{-1/2}$ and $(-\Delta)^{-1}$ (Figures A.7, A.8, A.9, A.10, A.11 and A.12) show that the real parts are slightly more asymmetrically distributed, which partly explains their relatively

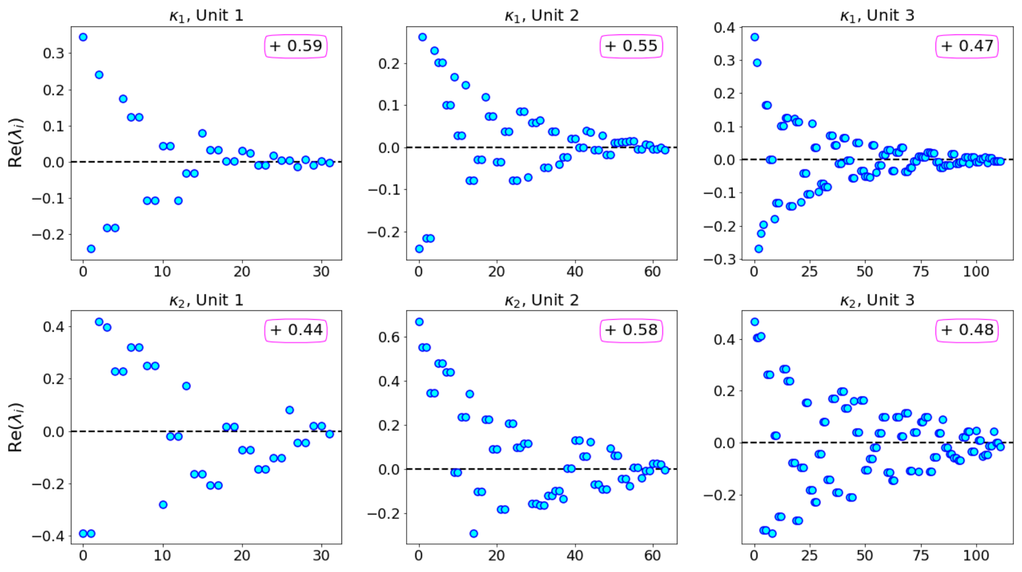

Figure 6.1: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Nonlocal diffusion $\mathcal{L}$ network while training on CIFAR-10.

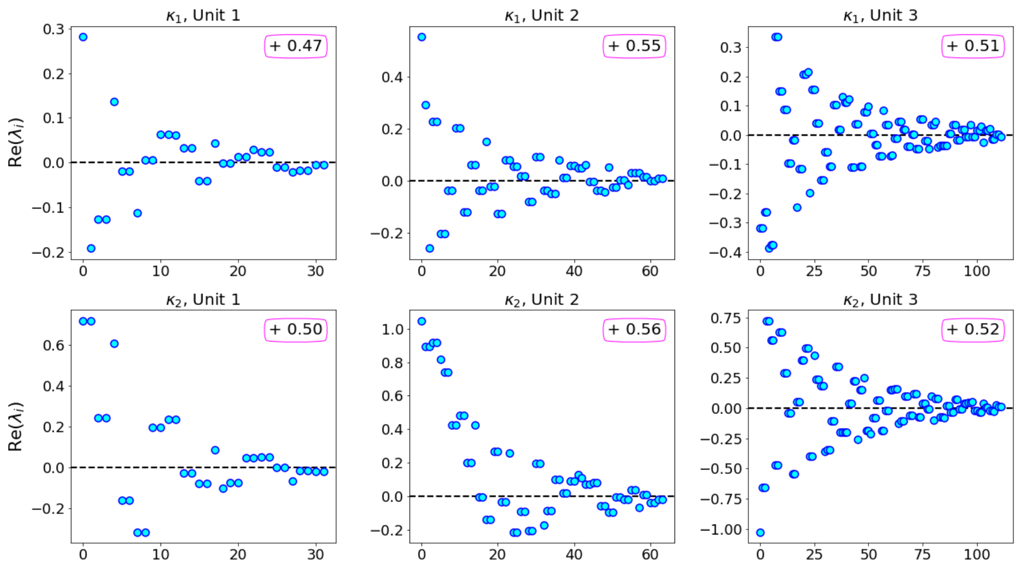

Figure 6.2: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Nonlocal diffusion $\mathcal{L}$ network while training on STL-10.

worse performance.

There is another way of looking at the properties of the transformations carried out by the trainable weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$. Let $\mathcal{K} \in \{\mathcal{K}_1, \mathcal{K}_2\}$. Then the amplification or damping of the features by $\mathcal{K} \in \mathbb{R}^{n\times n}$ is mainly determined by the associated quadratic form $\mathbf{x}^T\mathcal{K}\mathbf{x} \in \mathbb{R}$, for $\mathbf{x} \in \mathbb{R}^n$. We can decompose $\mathcal{K}$ into a symmetric and an antisymmetric part

$$\mathcal{K} = \frac{\mathcal{K} + \mathcal{K}^T}{2} + \frac{\mathcal{K} - \mathcal{K}^T}{2} = \mathcal{K}_s + \mathcal{K}_{as}.$$

Then the quadratic form of $\mathcal{K}$ is given by

$$\mathbf{x}^T \mathcal{K} \mathbf{x} = \mathbf{x}^T \Big(\frac{\mathcal{K} + \mathcal{K}^T}{2}\Big)\mathbf{x} + \mathbf{x}^T \Big(\frac{\mathcal{K} - \mathcal{K}^T}{2}\Big)\mathbf{x} = \mathbf{x}^T \mathcal{K}_s \mathbf{x},$$

where the quadratic form of the antisymmetric matrix $\mathcal{K}_{as}$ is zero because, for any antisymmetric matrix M, we have

$$\mathbf{x}^T M \mathbf{x} = (\mathbf{x}^T M \mathbf{x})^T = \mathbf{x}^T M^T \mathbf{x} = -\mathbf{x}^T M \mathbf{x}.$$

This implies that $2\,\mathbf{x}^T M \mathbf{x} = 0$ or $\mathbf{x}^T M \mathbf{x} = 0$. Therefore, the quadratic form of the symmetric part of $\mathcal{K}$, i.e. the quadratic form of $\mathcal{K}_s$, determines the spectral properties of the transformations. Since $\mathcal{K}_s$ is symmetric, the eigenvalues are all real.

Figures 6.3 and 6.4 show the eigenvalues of the symmetric parts of the weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in each Unit in the Nonlocal diffusion network while training on CIFAR-10 and STL-10 respectively. The eigenvalues of the symmetric parts for the other Nonlocal Hamiltonian networks can be found in Appendix A.5.

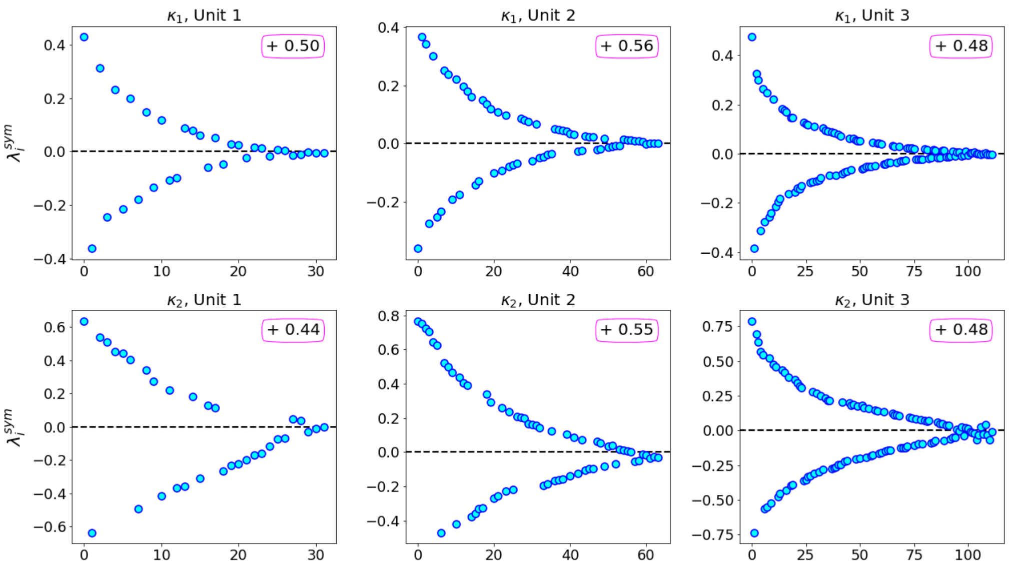

Figure 6.3: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Nonlocal diffusion $\mathcal{L}$ network while training on CIFAR-10.

The plots show that most of the eigenvalues are well-bounded and symmetric unlike the network suggested in [WGG+18], where the eigenvalues of the symmetric part are significantly larger when multiple stages are added to the Nonlocal Block, as shown in [TSD+18]. This well-boundedness of the eigenvalues makes the task of the optimizer easier, which leads to faster convergence. The plus sign with the number next to it indicates the fraction of positive eigenvalues. The values are relatively high for the Pseudo-differential $(-\Delta)^{-1}$ network ($\mathcal{K}_2$ in Figure A.18), despite having a small value of h, which partially explains why this network performs relatively worse than all the other three proposed neural networks.

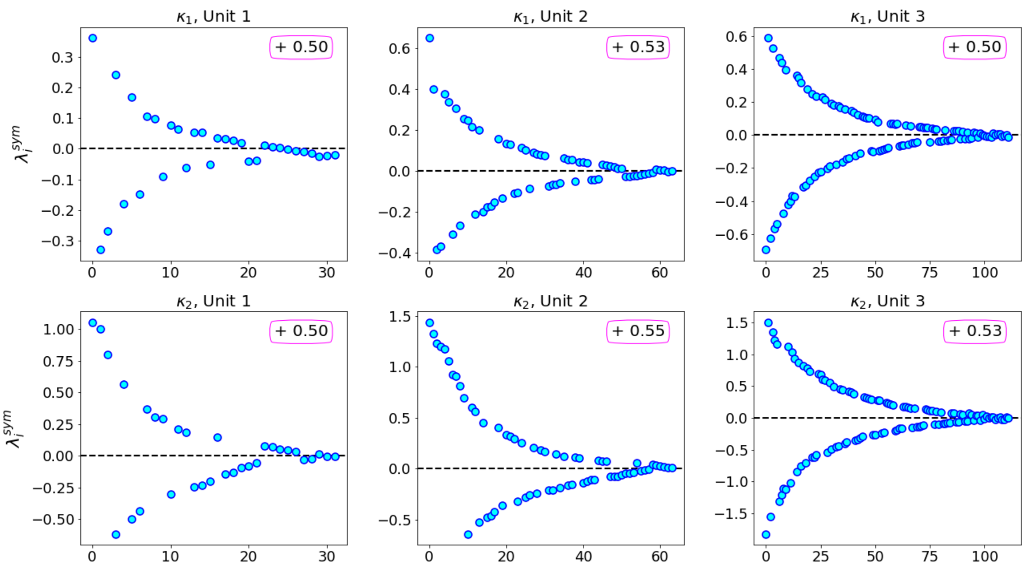

Figure 6.4: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Nonlocal diffusion $\mathcal{L}$ network while training on STL-10.

Chapter 7

Conclusion and Outlook

The aim of this thesis was to combine the concepts of forward propagation stability and widened receptive fields of neural networks through spatially nonlocal integral operators. To this end, four global integral operators are proposed, which are inspired by the weighted Laplacians and by the pseudo-differential operators, namely the fractional Laplacian and the inverse fractional Laplacian. The operators are implemented in Nonlocal Blocks, where the forward propagation of the features through these Blocks is governed by the discretization of an underlying PIDE. These Nonlocal Blocks are then added to each Unit of the stable Hamiltonian network, which is inspired by Hamiltonian systems.

To improve a network's performance, the recent trend is to have deeper or wider networks. Instead, this thesis proposes a network design involving Nonlocal Blocks that introduce nonlocal connectivities between pairs of regions of the image that accelerate the communication between far-away pixels and thereby widen the receptive field of the network. Due to an appropriate combination of local and nonlocal information, the network gets a more comprehensive view of each example during training. As a consequence, this helps the network to generalize better on unseen test examples. For instance, there can be images in a dataset, where a person is wearing a shirt with a dog on it. If the network does not have a wide enough receptive field, it would fail to realize that the image actually shows a person, and would rather classify the image as a dog. Such cases can be avoided if the network allows interactions between different regions of the image.

These nonlocal operators, although dense, are discretized in an efficient way using 1×1 convolutions and pooling such that we have a relatively low extra computational cost and very little increase in the number of trainable parameters. Similar to ResNets, the Nonlocal Blocks contain skip connections, which make sure that their presence does not hinder the flow of information in the network.

The nonlocality tackles the dilemma of network depth versus accuracy, which was mentioned before in the first chapter. It reduces the need for deeper networks in order to achieve a certain level of accuracy and therefore can be very beneficial when one wants to train a deep network but has computational constraints. Instead, one can have a slightly shallower network with the nonlocal operators inserted into the networks and thereby possibly save training time. The proposed neural networks achieve better accuracy than the original Hamiltonian model and the widely-used state-of-the-art ResNets. The Nonlocal diffusion $\mathcal{L}$ network and the Pseudo-differential $(-\Delta)^s$ network perform slightly better than the other two proposed networks. Moreover, the PIDE-based Nonlocal Block maintains the 2D structure of the images and has the same output shape as the input shape. Hence, they can be seamlessly inserted into any network. This can be useful if a neural network is to be trained and tested on a dataset that warrants nonlocal connections between the regions of the image, such as the synthetic Q-tips dataset.

The Nonlocal Blocks come with a computational cost, which can be adjusted based on the pool size of the subsampling in the Nonlocal Block. For most pooling sizes, the increase in computational cost is within acceptable limits. Moreover, just the presence of nonlocality, irrespective of the subsampling pool size, improves the performance of the network, especially for larger images.

Using spectral properties of the transformation matrices, we show that the Nonlocal Blocks do not destabilize the original Hamiltonian network in any way, and consequently, the stability of the forward propagation is still preserved. This stability provides an added benefit, i.e. the proposed networks are more robust to noise and perturbations of the input features and therefore more robust to adversarial attacks. The networks' global properties help them tackle local perturbations. Furthermore, the proposed nonlocal neural networks perform better than the baseline networks when the amount of training data is reduced. This makes them more appealing in applications, where there is a dearth of labeled data.

For future work, the PIDE-based Nonlocal Hamiltonian networks can be applied to other computer vision tasks, such as object recognition, where the nonlocal interactions can be crucial, and the output of the network is high-dimensional, or it can be employed in unsupervised learning tasks. While using the nonlocal operators, often, the mini-batch size is reduced to keep the memory footprint in check, although this has an adverse effect on the training time. Other strategies, such as reducing the channels of the embeddings (Figures 4.6 and 4.7) from 1024 to 256, can reduce the memory costs while computing the kernel. The effects on performance for such a drastic reduction in the number of channels in the embeddings need to be explored. Besides, the nonlocal diffusion operator $\mathcal{L}$ can be seen as a global weighted graph Laplacian. So it would be interesting to see if it finds any use in Graph Convolutional Networks (GCNs) [KW17]. Moreover, the nonlocal nature could be tested in combination with higher-order discretizations [ZCF18] or implicit discretizations [HLT+19; RW19] of the underlying PDE/ODE of the neural network. Implicit discretizations are actually known to broaden the receptive field of the network [HLT+19]. Hence, a connection to nonlocal operators is an interesting topic for further research. Furthermore, one could try out other nonstandard kernels (periodic, polynomial, etc.) [DMS09], kernels such as rational quadratic or locally stationary kernels [Gen01], and see their effects on the learning problem. Some kernels, such as the Gaussian and the Laplacian kernel, show no significant improvements in performance. Some of them might be harder to implement without large computational costs if the dimension of the input features is high. It also needs to be investigated if the performance of the Nonlocal Hamiltonian networks is dependent on the optimizer used during training. One could try second-order methods while training [BCN+11] to see if the networks generalize better.

The interpretations of ResNets and very deep networks, in general, are still an on-going research topic and hopefully, the dynamical system view coupled with spatially nonlocal operators will shed some light on the understanding and interpretability of these networks. This thesis hopes to contribute to the research on *structural diversity* [ZLCL+17] by providing alternative network designs, which could inspire further work in this direction in the future.

References

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado et al. ‘TensorFlow: Large-scale machine learning on heterogeneous distributed systems’. In: *arXiv preprint arXiv:1603.04467* (2016).

[AB17] Gabriel Acosta and Juan Pablo Borthagaray. ‘A fractional Laplace equation: Regularity of solutions and finite element approximations’. In: *SIAM Journal on Numerical Analysis* 55.2 (2017), pp. 472–495.

[AK06] Gilles Aubert and Pierre Kornprobst. *Mathematical problems in image processing: partial differential equations and the calculus of variations, 2nd Edition.* Vol. 147. Springer Science & Business Media, 2006.

[AP98] Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations.* Vol. 61. Society for Industrial and Applied Mathematics (SIAM), 1998.

[App09] David Applebaum. *Lévy processes and stochastic calculus.* 2nd ed. Cambridge University Press, 2009.

[Asc08] Uri M Ascher. *Numerical methods for evolutionary differential equations.* Society for Industrial and Applied Mathematics (SIAM), 2008.

[BCM05a] Antoni Buades, Bartomeu Coll and J-M Morel. ‘A non-local algorithm for image denoising’. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05).* Vol. 2. IEEE. 2005, pp. 60–65.

[BCM05b] Antoni Buades, Bartomeu Coll and Jean-Michel Morel. ‘A review of image denoising algorithms, with a new one’. In: *Multiscale Modeling & Simulation* 4.2 (2005), pp. 490–530.

[BCN+11] Richard H Byrd, Gillian M Chin, Will Neveitt and Jorge Nocedal. ‘On the use of stochastic hessian information in optimization methods for machine learning’. In: *SIAM Journal on Optimization* 21.3 (2011), pp. 977–995.

[BCN18] Léon Bottou, Frank E Curtis and Jorge Nocedal. ‘Optimization methods for large-scale machine learning’. In: *SIAM Review* 60.2 (2018), pp. 223–311.

[BCV13] Yoshua Bengio, Aaron Courville and Pascal Vincent. ‘Representation learning: A review and new perspectives’. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828.

[BDPM18] Juan Pablo Borthagaray, Leandro M Del Pezzo and Sandra Martínez. ‘Finite element approximation for the fractional eigenvalue problem’. In: *Journal of Scientific Computing* 77.1 (2018), pp. 308–329.

[BST+95] Jan Beran, Robert Sherman, Murad S Taqqu and Walter Willinger. ‘Long-range dependence in variable-bit-rate video traffic’. In: *IEEE Transactions on Communications* 43.2/3/4 (1995), pp. 1566–1579.

[Cap67] Michele Caputo. ‘Linear models of dissipation whose Q is almost frequency independent—II’. In: *Geophysical Journal International* 13.5 (1967), pp. 529–539.

[CCH+19] Bo Chang, Minmin Chen, Eldad Haber and Ed H Chi. ‘AntisymmetricRNN: A dynamical system view on recurrent neural networks’. In: *International Conference on Learning Representations (ICLR).* 2019.

[Ces10] Bruno Cessac. ‘A view of neural networks as dynamical systems’. In: *International Journal of Bifurcation and Chaos* 20.06 (2010), pp. 1585–1629.

[CG97] Fan RK Chung and Fan Chung Graham. *Spectral graph theory.* American Mathematical Society, 1997.

[CKZ+15] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler and Raquel Urtasun. ‘3d object proposals for accurate object class detection’. In: *Advances in Neural Information Processing Systems.* 2015, pp. 424–432.

[CL06] Ronald R Coifman and Stéphane Lafon. ‘Diffusion maps’. In: *Applied and Computational Harmonic Analysis* 21.1 (2006), pp. 5–30.

[CMH+18a] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung and David Begert. ‘Multi-level residual networks from dynamical systems view’. In: *International Conference on Learning Representations (ICLR).* 2018.

[CMH+18b] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert and Elliot Holtham. ‘Reversible architectures for arbitrarily deep residual neural networks’. In: *Thirty-Second AAAI Conference on Artificial Intelligence.* 2018.

[CNL11] Adam Coates, Andrew Ng and Honglak Lee. ‘An analysis of single-layer networks in unsupervised feature learning’. In: *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics.* Vol. 15. 2011, pp. 215–223.

[Con05] Rama Cont. 'Long range dependence in financial markets'. In: *Fractals in Engineering*. Springer, 2005, pp. 159–179.

[COO+18] Pratik Chaudhari, Adam Oberman, Stanley Osher, Stefano Soatto and Guillaume Carlier. 'Deep relaxation: partial differential equations for optimizing deep neural networks'. In: *Research in the Mathematical Sciences* 5.30 (2018).

[CRB+18] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt and David K Duvenaud. 'Neural ordinary differential equations'. In: *Advances in Neural Information Processing Systems*. 2018, pp. 6571–6583.

[CW08] Ronan Collobert and Jason Weston. 'A unified architecture for natural language processing: Deep neural networks with multitask learning'. In: *Proceedings of the 25th International Conference on Machine Learning (ICML)*. 2008, pp. 160–167.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. 'Imagenet: A large-scale hierarchical image database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ieee. 2009, pp. 248–255.

[DFK+07] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik and Karen Egiazarian. 'Image denoising by sparse 3-D transform-domain collaborative filtering'. In: *IEEE Transactions on Image Processing* 16.8 (2007), pp. 2080–2095.

[DG13] Marta D'Elia and Max Gunzburger. 'The fractional Laplacian operator on bounded domains as a special case of the nonlocal diffusion operator'. In: *Computers & Mathematics with Applications* 66.7 (2013), pp. 1245–1260.

[DGL+12] Qiang Du, Max Gunzburger, Richard B Lehoucq and Kun Zhou. 'Analysis and approximation of nonlocal diffusion problems with volume constraints'. In: *SIAM Review* 54.4 (2012), pp. 667–696.

[DMS09] Stefano De Marchi and Robert Schaback. 'Nonstandard kernels and their applications'. In: *Dolomites Research Notes on Approximation* 2.1 (2009).

[DNPV12] Eleonora Di Nezza, Giampiero Palatucci and Enrico Valdinoci. 'Hitchhiker's guide to the fractional Sobolev spaces'. In: *Bulletin des sciences mathématiques* 136.5 (2012), pp. 521–573.

[EEVG+15] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn and Andrew Zisserman. 'The pascal visual object classes challenge: A retrospective'. In: *International Journal of Computer Vision (IJCV)* 111.1 (2015), pp. 98–136.

[FCA+18] Chris Finlay, Jeff Calder, BILAL Abbasi and ADAM Oberman. 'Lipschitz regularized deep neural networks generalize and are adversarially robust'. In: *arXiv preprint arXiv:1808.09540* (2018).

[Gas17] Xavier Gastaldi. 'Shake-Shake regularization of 3-branch residual networks'. In: *International Conference on Learning Representations (ICLR)*. 2017.

[GB10] Xavier Glorot and Yoshua Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. In: *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256.

[GBC16] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[Gen01] Marc G Genton. 'Classes of kernels for machine learning: A statistics perspective'. In: *Journal of Machine Learning Research* 2 (2001), pp. 299–312.

[GO07] Guy Gilboa and Stanley Osher. 'Nonlocal linear image regularization and supervised segmentation'. In: *Multiscale Modeling & Simulation* 6.2 (2007), pp. 595–630.

[Gra06] Leo Grady. 'Random walks for image segmentation'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.11 (2006), pp. 1768–1783.

[GRU+17] Aidan N Gomez, Mengye Ren, Raquel Urtasun and Roger B Grosse. 'The reversible residual network: Backpropagation without storing activations'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 2214–2224.

[GSS15] Ian J Goodfellow, Jonathon Shlens and Christian Szegedy. 'Explaining and harnessing adversarial examples'. In: 2015.

[HDWF+17] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal et al. 'Brain tumor segmentation with deep neural networks'. In: *Medical Image Analysis* 35 (2017), pp. 18–31.

[HGD+17] Kaiming He, Georgia Gkioxari, Piotr Dollár and Ross Girshick. 'Mask r-cnn'. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988.

[HLT+19] Eldad Haber, Keegan Lensink, Eran Treister and Lars Ruthotto. 'IMEXnet: A forward stable deep neural network'. In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Vol. 97. 2019, pp. 2525–2534.

[HLVDM+17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten and Kilian Q Weinberger. 'Densely connected convolutional networks'. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269.

[HR17] Eldad Haber and Lars Ruthotto. ‘Stable architectures for deep neural networks’. In: *Inverse Problems* 34.1 (2017), p. 014004.

[HRH+18] Eldad Haber, Lars Ruthotto, Elliot Holtham and Seong-Hwan Jun. ‘Learning across scales—Multiscale methods for Convolution Neural Networks’. In: *Thirty-Second AAAI Conference on Artificial Intelligence.* 2018.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. ‘Long short-term memory’. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[HSL+16] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra and Kilian Q Weinberger. ‘Deep networks with stochastic depth’. In: *European Conference on Computer Vision (ECCV).* Springer. 2016, pp. 646–661.

[HW62] David H Hubel and Torsten N Wiesel. ‘Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex’. In: *The Journal of Physiology* 160.1 (1962), pp. 106–154.

[HZR+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian 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.

[HZR+16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. ‘Deep residual learning for image recognition’. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, pp. 770–778.

[HZR+16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. ‘Identity mappings in deep residual networks’. In: *European Conference on Computer Vision (ECCV).* Springer. 2016, pp. 630–645.

[Hör15] Lars Hörmander. *The analysis of linear partial differential operators I: Distribution theory and Fourier analysis.* Springer, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. ‘Batch normalization: Accelerating deep network training by reducing internal covariate shift’. In: *arXiv preprint arXiv:1502.03167* (2015).

[JSZ+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman et al. ‘Spatial transformer networks’. In: *Advances in Neural Information Processing Systems.* 2015, pp. 2017–2025.

[KBZ+19] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly and Neil Houlsby. ‘Big transfer (BiT): General visual representation learning’. In: *arXiv preprint arXiv:1912.11370* (2019).

[Kri09] Alex Krizhevsky. *Learning multiple layers of features from tiny images.* Tech. rep. University of Toronto, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. ‘Imagenet classification with deep convolutional neural networks’. In: *Advances in Neural Information Processing Systems.* 2012, pp. 1097–1105.

[KW17] Thomas N Kipf and Max Welling. ‘Semi-supervised classification with graph convolutional networks’. In: *International Conference on Learning Representations (ICLR).* 2017.

[Kwa17] Mateusz Kwaśnicki. ‘Ten equivalent definitions of the fractional Laplace operator’. In: *Fractional Calculus and Applied Analysis* 20.1 (2017), pp. 7–51.

[Lan72] Naum S Landkof. *Foundations of modern potential theory.* Grundlehren der mathematischen Wissenschaften. Springer, 1972.

[LBB+98] Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner. ‘Gradient-based learning applied to document recognition’. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[LBD+89] 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’. In: *Neural computation* 1.4 (1989), pp. 541–551.

[LC09] Wei Liu and Shih-Fu Chang. ‘Robust multi-class transductive learning with graphs’. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE. 2009, pp. 381–388.

[LCT+18] Qianxiao Li, Long Chen, Cheng Tai and E Weinan. ‘Maximum principle based algorithms for deep learning’. In: *Journal of Machine Learning Research* 18.165 (2018), pp. 1–29.

[LHP19] Keegan Lensink, Eldad Haber and Bas Peters. ‘Fully hyperbolic convolutional neural networks’. In: *arXiv preprint arXiv:1905.10484* (2019).

[LLH+19] Chenkuan Li, Changpin Li, Thomas Humphries and Hunter Plowman. ‘Remarks on the generalized fractional Laplacian operator’. In: *Mathematics* 7.4 (2019), p. 320.

[LLU+16] Wenjie Luo, Yujia Li, Raquel Urtasun and Richard Zemel. ‘Understanding the effective receptive field in deep convolutional neural networks’. In: *Advances in Neural Information Processing Systems.* 2016, pp. 4898–4906.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár et al. ‘Microsoft COCO: Common objects in context’. In: *European Conference on Computer Vision (ECCV).* Springer. 2014, pp. 740–755.

[LMS16] Gustav Larsson, Michael Maire and Gregory Shakhnarovich. 'Fractalnet: Ultra-deep neural networks without residuals'. In: *arXiv preprint arXiv:1605.07648* (2016).

[LP16] Qianli Liao and Tomaso Poggio. 'Bridging the gaps between residual learning, recurrent neural networks and visual cortex'. In: *arXiv preprint arXiv:1604.03640* (2016).

[LS17a] Zhen Li and Zuoqiang Shi. 'A flow model of neural networks'. In: *arXiv preprint arXiv:1708.06257* (2017).

[LS17b] Zhen Li and Zuoqiang Shi. 'Deep residual learning and pdes on manifold'. In: *arXiv preprint arXiv:1708.05115* (2017).

[LSD15] Jonathan Long, Evan Shelhamer and Trevor Darrell. 'Fully convolutional networks for semantic segmentation'. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pp. 3431–3440.

[LXG+15] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang and Zhuowen Tu. 'Deeply-supervised nets'. In: *Artificial Intelligence and Statistics.* 2015, pp. 562–570.

[LZL+18] Yiping Lu, Aoxiao Zhong, Quanzheng Li and Bin Dong. 'Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations'. In: *International Conference on Machine Learning (ICML).* 2018, pp. 3276–3285.

[LZO+10] Yifei Lou, Xiaoqun Zhang, Stanley Osher and Andrea Bertozzi. 'Image recovery via nonlocal operators'. In: *Journal of Scientific Computing* 42.2 (2010), pp. 185–197.

[MDFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi and Pascal Frossard. 'Deepfool: A simple and accurate method to fool deep neural networks'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, pp. 2574–2582.

[MKK+18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama and Yuichi Yoshida. 'Spectral normalization for generative adversarial networks'. In: *International Conference on Learning Representations (ICLR).* 2018.

[MS12] James Martens and Ilya Sutskever. 'Training deep and recurrent networks with hessian-free optimization'. In: *Neural networks: Tricks of the trade: Second Edition.* Springer, 2012, pp. 479–535.

[MS89] David Bryant Mumford and Jayant Shah. 'Optimal approximations by piecewise smooth functions and associated variational problems'. In: *Communications on Pure and Applied Mathematics* 42.5 (1989), pp. 577–685.

[NH10] Vinod Nair and Geoffrey E Hinton. ‘Rectified linear units improve restricted boltzmann machines’. In: *International Conference on Machine Learning (ICML).* 2010.

[PHM+17] Tobias Pohlen, Alexander Hermans, Markus Mathias and Bastian Leibe. ‘Full-resolution residual networks for semantic segmentation in street scenes’. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017, pp. 3309–3318.

[PM90] Pietro Perona and Jitendra Malik. ‘Scale-space and edge detection using anisotropic diffusion’. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.7 (1990), pp. 629–639.

[PMB13] Razvan Pascanu, Tomas Mikolov and Yoshua Bengio. ‘On the difficulty of training recurrent neural networks’. In: *International Conference on Machine Learning (ICML).* Vol. 28. 2013, pp. 1310–1318.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang et al. ‘Imagenet large scale visual recognition challenge’. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

[RFB15] Olaf Ronneberger, Philipp Fischer and Thomas Brox. ‘U-net: Convolutional networks for biomedical image segmentation’. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI).* Springer. 2015, pp. 234–241.

[RH20] Lars Ruthotto and Eldad Haber. ‘Deep neural networks motivated by partial differential equations’. In: *Journal of Mathematical Imaging and Vision* 62 (2020), pp. 352–364.

[RHG+15] Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun. ‘Faster r-cnn: Towards real-time object detection with region proposal networks’. In: *Advances in Neural Information Processing Systems.* 2015, pp. 91–99.

[ROS14] Xavier Ros-Oton and Joaquim Serra. ‘The Dirichlet problem for the fractional Laplacian: Regularity up to the boundary’. In: *Journal de Mathématiques Pures et Appliquées* 101.3 (2014), pp. 275–302.

[RR06] Michael Renardy and Robert C Rogers. *An introduction to partial differential equations.* Springer Science & Business Media, 2006.

[Rub96] Boris Rubin. *Fractional integrals and potentials.* Vol. 82. CRC Press, 1996.

[RW19] Viktor Reshniak and Clayton Webster. ‘Robust learning with implicit residual networks’. In: *arXiv preprint arXiv:1905.10479* (2019).

[SGS15a] Rupesh K Srivastava, Klaus Greff and Jürgen Schmidhuber. ‘Training very deep networks’. In: *Advances in Neural Information Processing Systems.* 2015, pp. 2377–2385.

[SGS15b] Rupesh Kumar Srivastava, Klaus Greff and Jürgen Schmidhuber. 'Highway networks'. In: *arXiv preprint arXiv:1505.00387* (2015).

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. 'Dropout: A simple way to prevent neural networks from overfitting'. In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[Sil07] Luis Silvestre. 'Regularity of the obstacle problem for a fractional power of the Laplace operator'. In: *Communications on Pure and Applied Mathematics* 60.1 (2007), pp. 67–112.

[SIV+17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke and Alexander A Alemi. 'Inception-v4, inception-resnet and the impact of residual connections on learning'. In: *Thirty-first AAAI Conference on Artificial Intelligence.* 2017.

[SKM+13] Tara N Sainath, Brian Kingsbury, Abdel-rahman Mohamed, George E Dahl, George Saon, Hagen Soltau, Tomas Beran et al. 'Improvements to deep convolutional neural networks for LVCSR'. In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding.* IEEE. 2013, pp. 315–320.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan et al. 'Going deeper with convolutions'. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pp. 1–9.

[SM17] Sho Sonoda and Noboru Murata. 'Double continuum limit of deep neural networks'. In: *ICML Workshop Principled Approaches to Deep Learning.* 2017.

[SS18] Justin Sirignano and Konstantinos Spiliopoulos. 'DGM: A deep learning algorithm for solving partial differential equations'. In: *Journal of Computational Physics* 375 (2018), pp. 1339–1364.

[ST10] Pablo Raúl Stinga and José Luis Torrea. 'Extension problem and Harnack's inequality for some fractional operators'. In: *Communications in Partial Differential Equations* 35.11 (2010), pp. 2092–2122.

[Ste70] Elias M Stein. *Singular integrals and differentiability properties of functions.* Vol. 2. Princeton University Press, 1970.

[Sti10] Pablo Raúl Stinga. 'Fractional powers of second order partial differential operators: Extension problem and regularity theory'. PhD thesis. Universidad Autónoma de Madrid, 2010.

[Sti19] Pablo Raúl Stinga. 'User's guide to the fractional Laplacian and the method of semigroups'. In: *Fractional Differential Equations* (2019), pp. 235–266.

[SV14] Raffaella Servadei and Enrico Valdinoci. ‘On the spectrum of two different fractional operators’. In: *Proceedings of the Royal Society of Edinburgh Section A: Mathematics, Physical and Engineering Sciences* 144.4 (2014), pp. 831–855.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens and Zbigniew Wojna. ‘Rethinking the inception architecture for computer vision’. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826.

[SZ15] Karen Simonyan and Andrew Zisserman. ‘Very deep convolutional networks for large-scale image recognition’. In: *International Conference on Learning Representations (ICLR)*. 2015.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow and Rob Fergus. ‘Intriguing properties of neural networks’. In: *International Conference on Learning Representations (ICLR)*. 2014.

[TG18] Matthew Thorpe and Yves van Gennip. ‘Deep limits of residual neural networks’. In: *arXiv preprint arXiv:1810.11741* (2018).

[TSD+18] Yunzhe Tao, Qi Sun, Qiang Du and Wei Liu. ‘Nonlocal neural networks, nonlocal diffusion and nonlocal modeling’. In: *Advances in Neural Information Processing Systems*. 2018, pp. 496–506.

[TSE+19] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner and Aleksander Madry. ‘Robustness may be at odds with accuracy’. In: *International Conference on Learning Representations (ICLR)*. 2019.

[VAG17] Aravind Vasudevan, Andrew Anderson and David Gregg. ‘Parallel multi channel convolution using general matrix multiplication’. In: *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2017, pp. 19–24.

[VBG+17] Rene Vidal, Joan Bruna, Raja Giryes and Stefano Soatto. ‘Mathematics of deep learning’. In: *arXiv preprint arXiv:1712.04741* (2017).

[VWB16] Andreas Veit, Michael J Wilber and Serge Belongie. ‘Residual networks behave like ensembles of relatively shallow networks’. In: *Advances in Neural Information Processing Systems*. 2016, pp. 550–558.

[WA05] Olaf Weckner and Rohan Abeyaratne. ‘The effect of long-range forces on the dynamics of a bar’. In: *Journal of the Mechanics and Physics of Solids* 53.3 (2005), pp. 705–728.

[Wei17] E Weinan. ‘A proposal on machine learning via dynamical systems’. In: *Communications in Mathematics and Statistics* 5.1 (2017), pp. 1–11.

[Wei98] Joachim Weickert. *Anisotropic diffusion in image processing.* Vol. 1. Teubner Stuttgart, 1998.

[WGG+18] Xiaolong Wang, Ross Girshick, Abhinav Gupta and Kaiming He. ‘Non-local neural networks’. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 2018, pp. 7794–7803.

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun et al. ‘Google’s neural machine translation system: Bridging the gap between human and machine translation’. In: *arXiv preprint arXiv:1609.08144* (2016).

[XGD+17] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu and Kaiming He. ‘Aggregated residual transformations for deep neural networks’. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017, pp. 5987–5995.

[Yam12] Masakazu Yamamoto. ‘Asymptotic expansion of solutions to the dissipative equation with fractional Laplacian’. In: *SIAM Journal on Mathematical Analysis* 44.6 (2012), pp. 3786–3805.

[YCW+20] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan et al. ‘BDD100K: A diverse driving dataset for heterogeneous multitask learning’. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2020.

[YWC+14] Dingjun Yu, Hanli Wang, Peiqiu Chen and Zhihua Wei. ‘Mixed pooling for convolutional neural networks’. In: *International Conference on Rough Sets and Knowledge Technology.* Springer. 2014, pp. 364–375.

[YWL+19] Yibo Yang, Jianlong Wu, Hongyang Li, Xia Li, Tiancheng Shen and Zhouchen Lin. ‘Dynamical system inspired adaptive time stepping controller for residual network families’. In: *arXiv preprint arXiv:1911.10305* (2019).

[ZAG18] Daniel Zügner, Amir Akbarnejad and Stephan Günnemann. ‘Adversarial attacks on neural networks for graph data’. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 2018, pp. 2847–2856.

[ZC88] Yi-Tong Zhou and Rama Chellappa. ‘Computation of optical flow using a neural network’. In: *IEEE 1988 International Conference on Neural Networks.* 1988, 71–78 vol.2.

[ZCF18] Mai Zhu, Bo Chang and Chong Fu. ‘Convolutional neural networks combined with runge-kutta methods’. In: *arXiv preprint arXiv:1802.08831* (2018).

[Zho20] Ding-Xuan Zhou. 'Universality of deep convolutional neural networks'. In: *Applied and Computational Harmonic Analysis* 48.2 (2020), pp. 787–794.

[ZHW+19] Jingfeng Zhang, Bo Han, Laura Wynter, Kian Hsiang Low and Mohan Kankanhalli. 'Towards robust resnet: A small step but a giant leap'. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 4285–4291.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. 'Wide residual networks'. In: *Proceedings of the British Machine Vision Conference (BMVC)* (2016).

[ZKL+15] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva and Antonio Torralba. 'Object detectors emerge in deep scene CNNs'. In: *International Conference on Learning Representations (ICLR)*. 2015.

[ZLCL+17] Xingcheng Zhang, Zhizhong Li, Chen Change Loy and Dahua Lin. 'Polynet: A pursuit of structural diversity in very deep networks'. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3900–3908.

[ZS19] Linan Zhang and Hayden Schaeffer. 'Forward stability of ResNet and its variants'. In: *Journal of Mathematical Imaging and Vision* 62 (2019), pp. 328–351.

Appendix

A.1 Non-subsampled Nonlocal Block for the inverse Laplacian operator

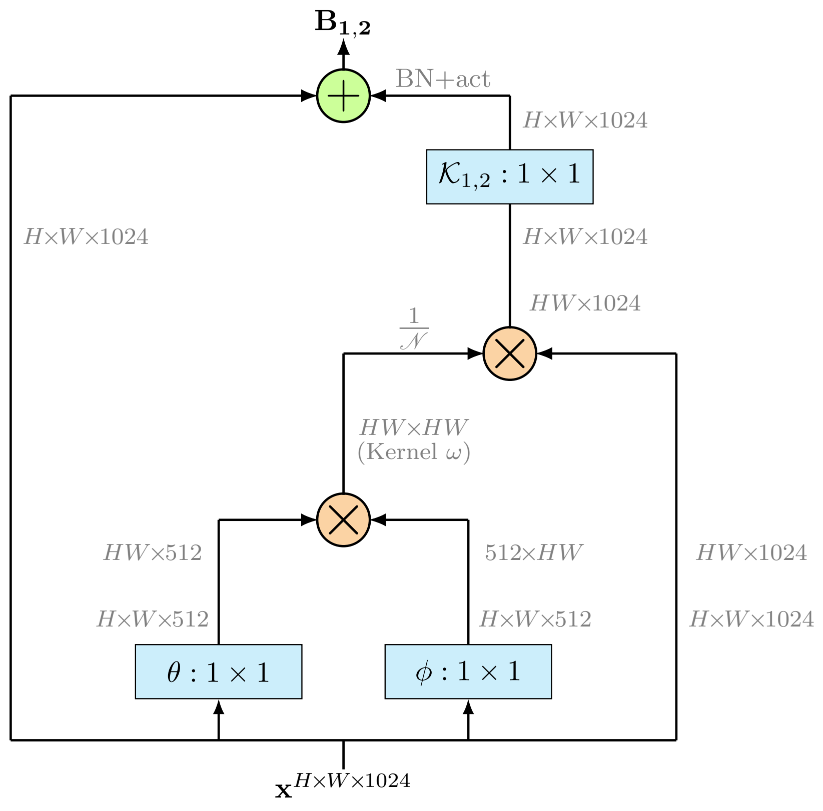

Figure A.1: Forward propagation of tensors in a non-subsampled Nonlocal Block with the inverse Laplacian operator $(-\Delta)^{-s}$.

A.2 Test accuracy curves for the Nonlocal networks

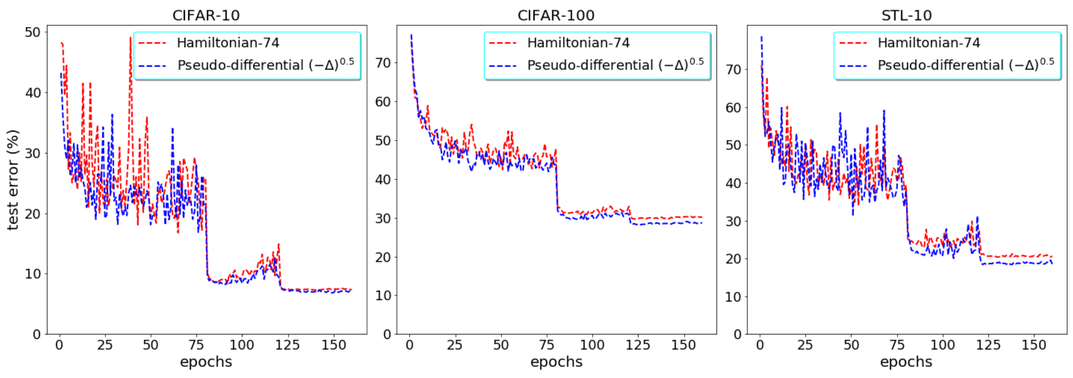

Figure A.2: Test accuracy curves for the Pseudo-differential $(-\Delta)^{1/2}$ network in comparison to the original Hamiltonian network.

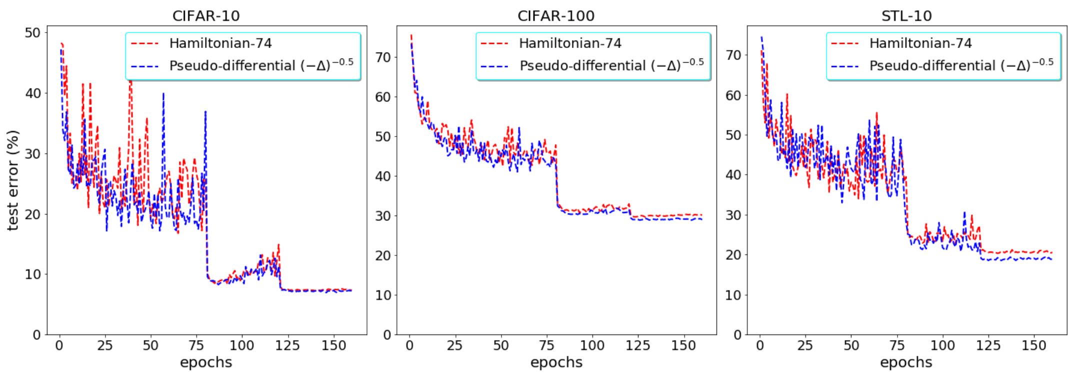

Figure A.3: Test accuracy curves for the Pseudo-differential $(-\Delta)^{-1/2}$ network in comparison to the original Hamiltonian network.

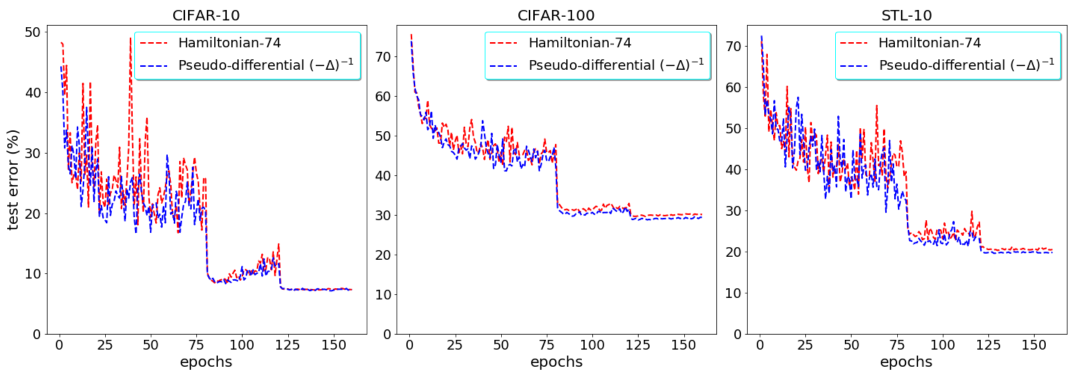

Figure A.4: Test accuracy curves for the Pseudo-differential $(-\Delta)^{-1}$ network in comparison to the original Hamiltonian network.

A.3 Spectral properties of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$ for larger h

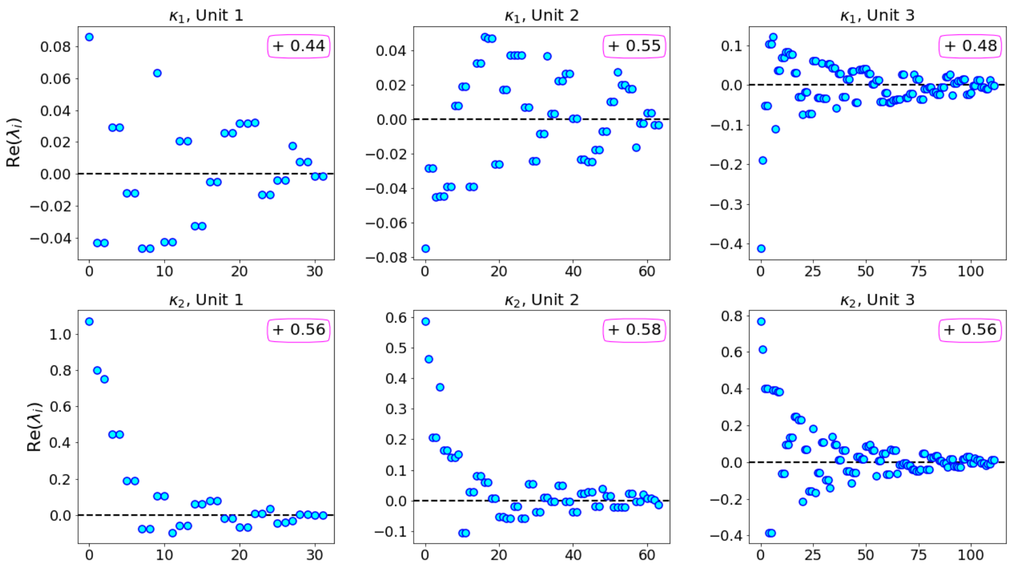

Figure A.5: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{1/2}$ network while training on STL-10 with larger values of h.

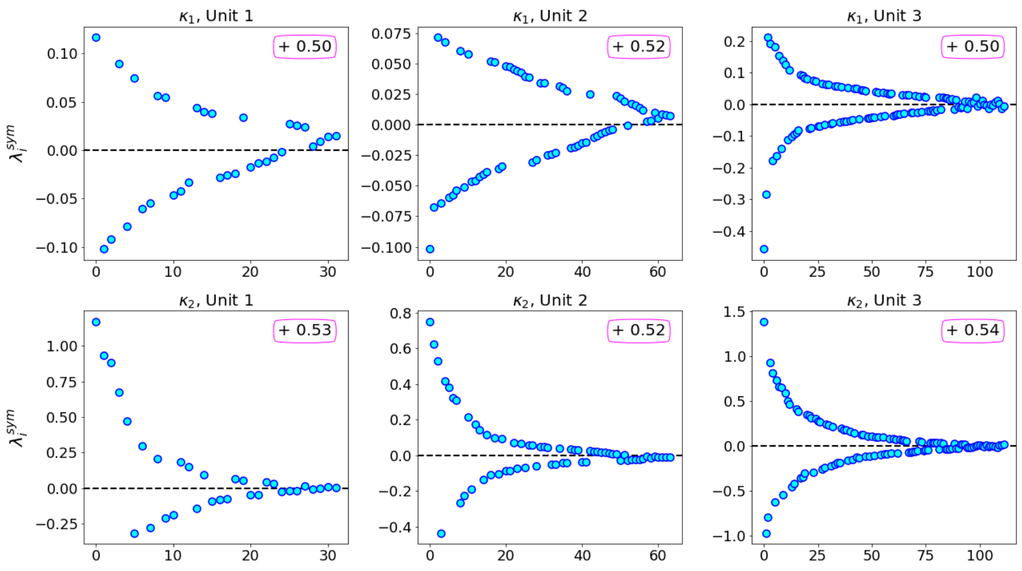

Figure A.6: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{1/2}$ network while training on STL-10 with larger values of h.

A.4 Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$

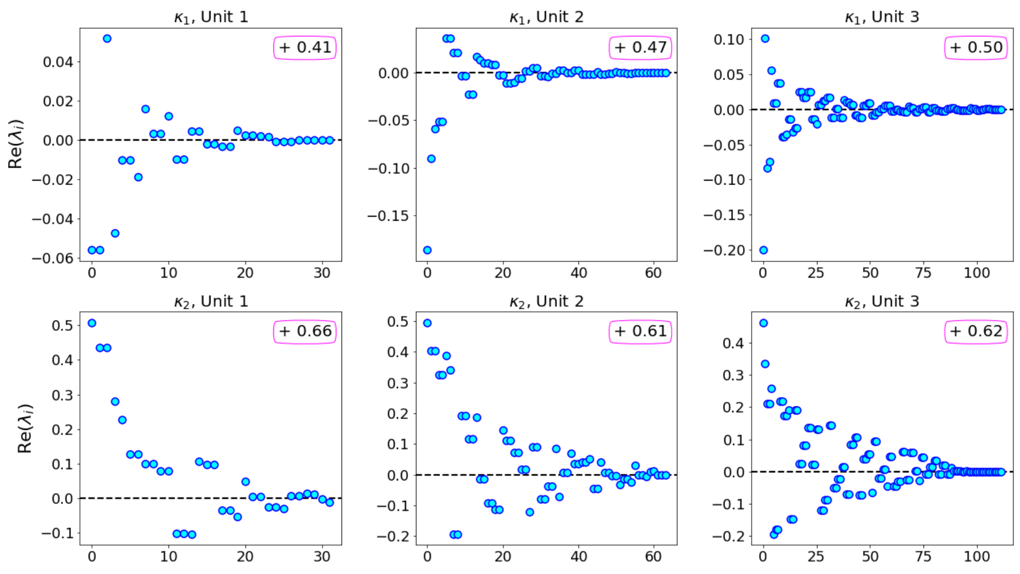

Figure A.7: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{1/2}$ network while training on CIFAR-10.

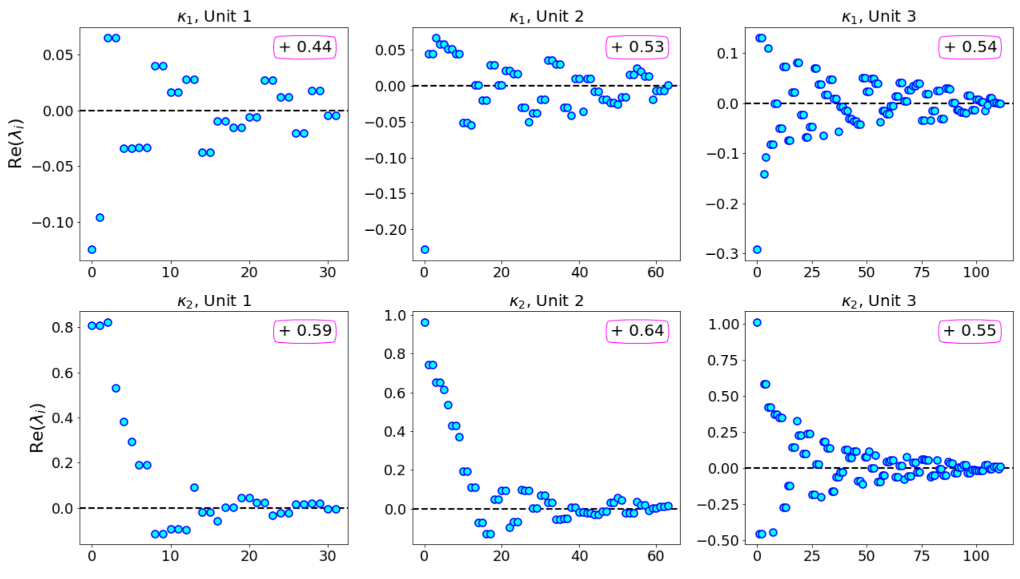

Figure A.8: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{1/2}$ network while training on STL-10.

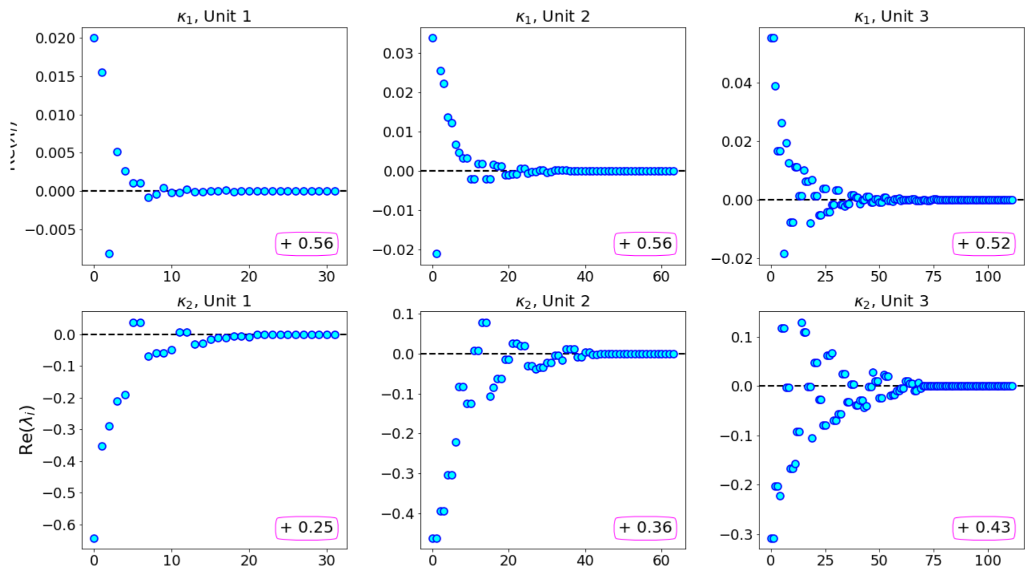

Figure A.9: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1/2}$ network while training on CIFAR-10.

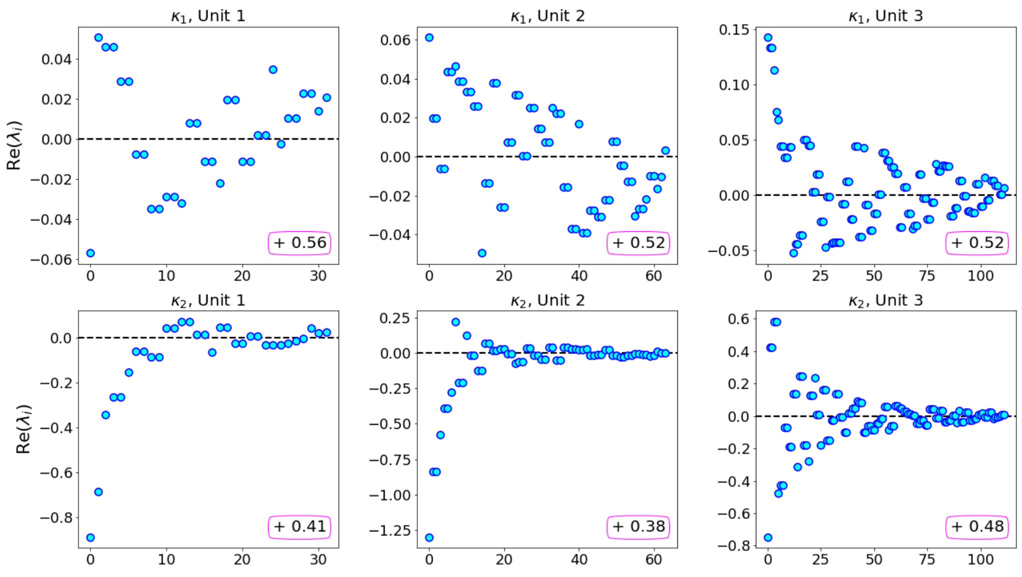

Figure A.10: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1/2}$ network while training on STL-10.

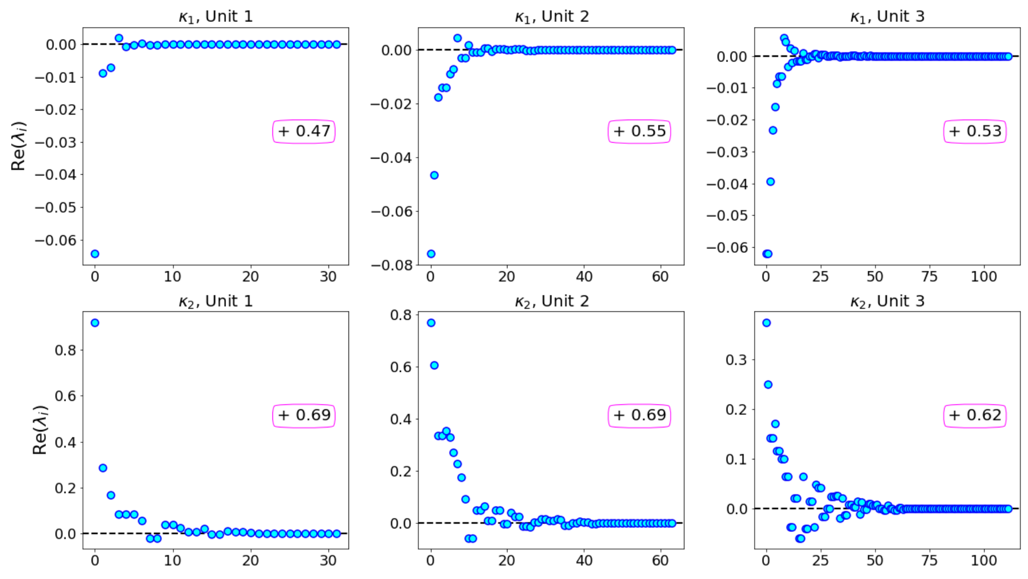

Figure A.11: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1}$ network while training on CIFAR-10.

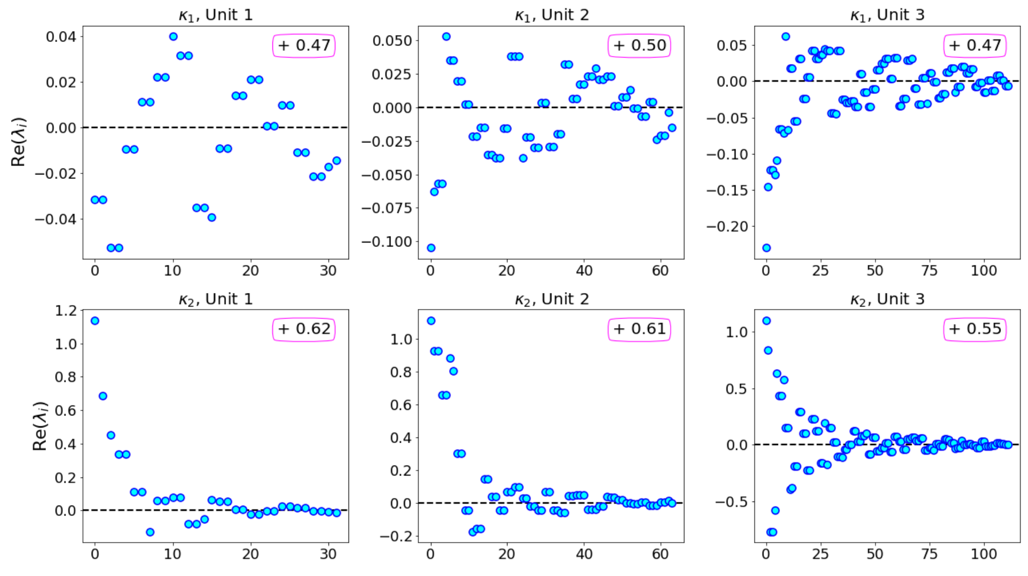

Figure A.12: Real parts of the eigenvalues of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1}$ network while training on STL-10.

A.5 Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$, $\mathcal{K}_2$

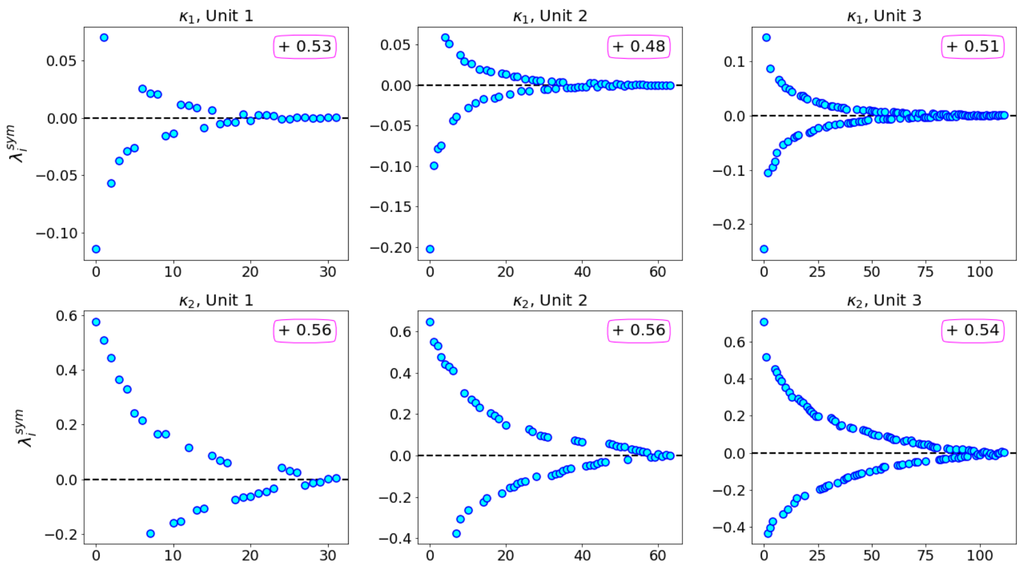

Figure A.13: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{1/2}$ network while training on CIFAR-10.

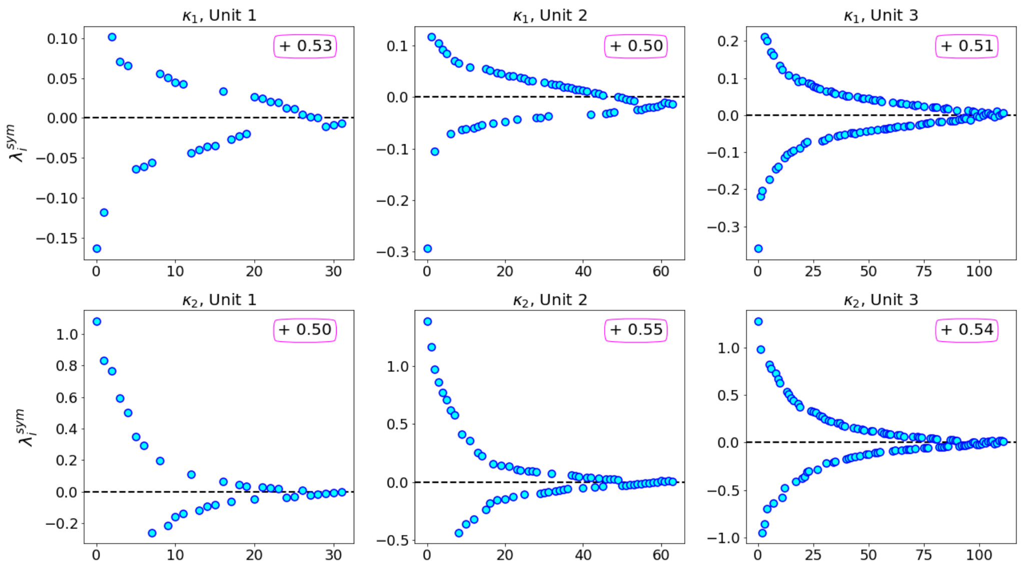

Figure A.14: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{1/2}$ network while training on STL-10.

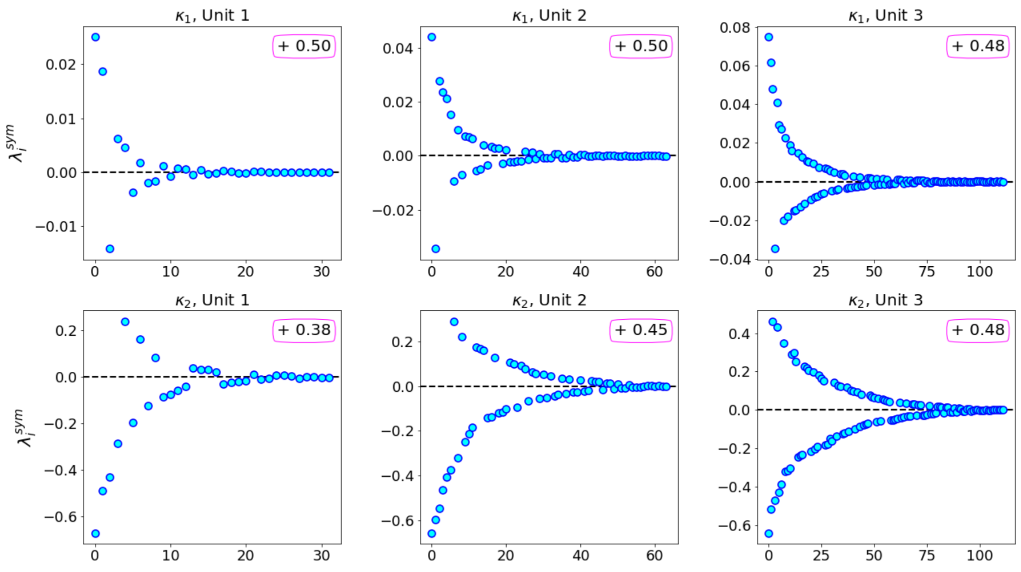

Figure A.15: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1/2}$ network while training on CIFAR-10.

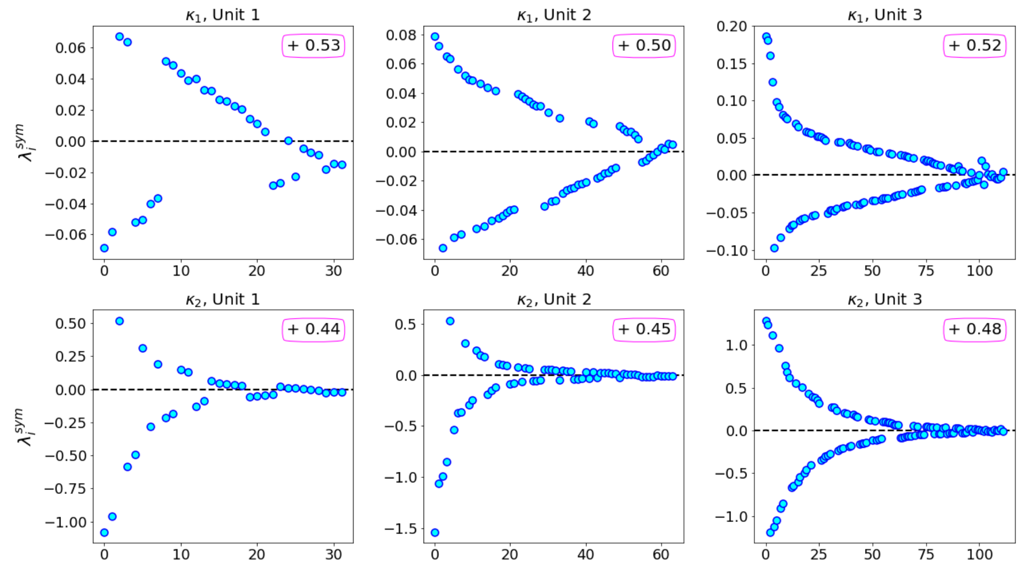

Figure A.16: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1/2}$ network while training on STL-10.

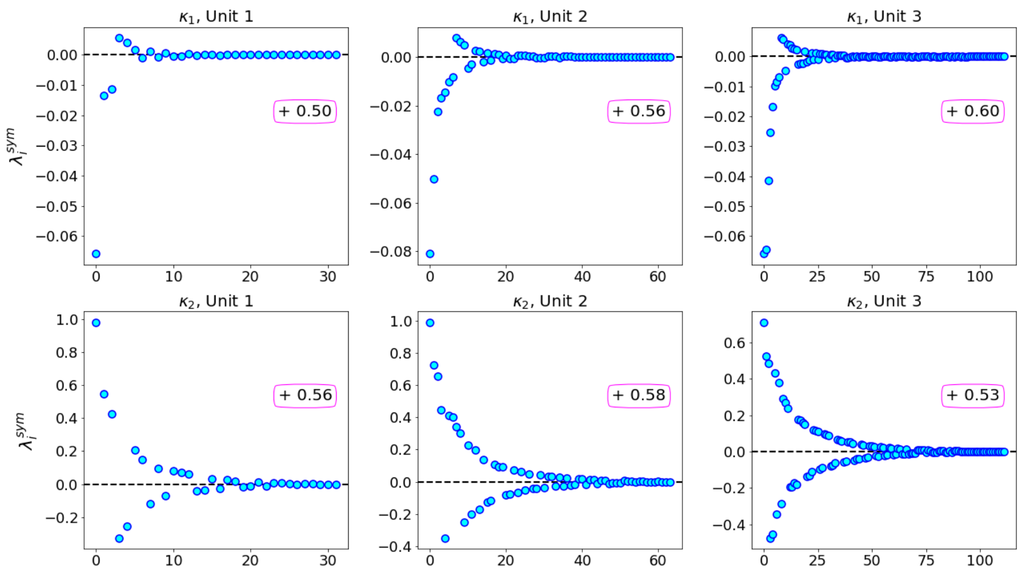

Figure A.17: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1}$ network while training on CIFAR-10.

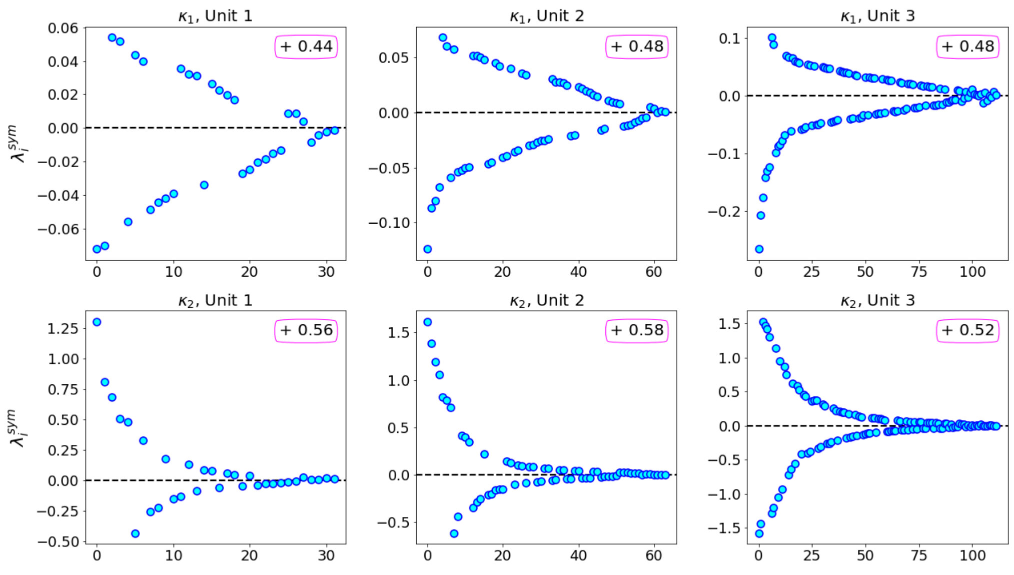

Figure A.18: Eigenvalues of the symmetric parts of weight matrices $\mathcal{K}_1$ and $\mathcal{K}_2$ in the Nonlocal Block placed in each Unit of the Pseudo-differential $(-\Delta)^{-1}$ network while training on STL-10.