Multigrid Methodologies in Deep Learning

From ResNet Analogies to Spectral Analysis of Convolutions



BERGISCHE
UNIVERSITÄT
WUPPERTAL

Dissertation

University of Wuppertal
School of Mathematics and Natural Science

submitted by
Antonia Maria van Betteray, M. Sc.
for the degree of Doctor of Natural Sciences (Dr. rer. nat.)

---

supervisor	Prof. Dr. Karsten Kahl
co-supervisor	Prof. Dr. Matthias Rottmann

Wuppertal, 2026

Acknowledgments

First, I would like to express my sincere gratitude to my supervisors Karsten Kahl and Matthias Rottmann, for making this project possible. Your continuous support throughout my doctoral studies allowed me to delve deeply into the subject. I am grateful for the knowledge and experience you shared, and I truly enjoyed our frequent conversations and discussions. Thank you for your endless patience and belief in me.

Furthermore, this work would not have been possible without financial support. This work is supported by the German Federal Ministry for Economic Affairs and Climate Action, within the project "KI Delta Learning" (grant no. 19A19013Q). I greatly appreciate the work and discussions within this project, as well as the opportunity to establish connections with various companies and universities.

I wish to thank Andreas Frommer for enabling me to do my doctorate in the "Applied Computer Science" working group. My thanks also goes to my colleagues in this group, despite spatial separation, I always felt welcomed and enjoyed the time with you. I would like to express my gratitude to my colleagues at ACM-AI lab and the former BUW-KI lab for making my time in the office so enjoyable. I appreciate your openness and curiosity about my work and our discussions. I would especially like to thank Annika and Svenja for their consistent support and camaraderie, which extended beyond the confines of the office. Moreover, I am grateful to everyone who has proofread this thesis in any way.

Last but not least I am deeply grateful to my friends and family for their constant love and support throughout along my academic journey. Johanna and Rebecca, thank you for always being there to listen. My husband, Matthias W., I am grateful for your steadfast support, your faith in me and your encouragement to keep moving forward. All my friends and family not mentioned by name, please know that your support and sympathy have meant a great deal to me. Finally, I cherish Chewie's loyal presence during this time.

Foreword

Parts of the work presented in this thesis have previously been published in the following publications:

- A. Van Betteray, M. Rottmann, and K. Kahl, *MGiaD: Multigrid in all dimensions. Efficiency and robustness by weight sharing and coarsening in resolution and channel dimensions*, in IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2023, pp. 1284–1293

- ——, *Poly-MgNet: Polynomial building blocks in multigrid-inspired ResNets*, in Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods (ICPRAM), SCITEPRESS - Science and Technology Publications, 2025, pp. 181–191

- ——, *LFA applied to CNNs: Efficient singular value decomposition of convolutional mappings by local Fourier analysis*, in Proceedings of the European Conference on Artificial Intelligence (ECAI), vol. 413 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2025, pp. 90–97

Contents

Acknowledgments III

Foreword V

1 Introduction 1

2 Fundamentals of Numerical Linear Algebra 9
2.1 Matrix Decompositions . . . 10
2.1.1 Eigenvalue Problems . . . 11
2.1.2 Singular Value Decomposition . . . 12
2.2 Crystalline Structures . . . 14
2.2.1 Lattices and Crystals . . . 14
2.2.2 Function Spaces on Crystals . . . 18
2.2.3 Fourier Functions on Lattice and Crystal Tori . . . 19
2.3 Discrete Fourier Transforms . . . 24
2.3.1 Fast Discrete Fourier Transforms . . . 25
2.3.2 Local Fourier Analysis . . . 26
2.4 Iterative Methods . . . 29
2.4.1 Boundary Value Problems . . . 29
2.4.2 Classical Iterative Methods . . . 31
2.5 Multigrid Methods . . . 37
2.5.1 Coarse-Grid Correction . . . 38
2.5.2 Two-Grid Method . . . 41
2.5.3 From Two-Grid to Multigrid Methods . . . 42

2.5.4 Full Approximation Scheme 44

3 Fundamentals of Machine Learning 45
3.1 Supervised Learning 45
3.1.1 Empirical Risk Minimization 46
3.1.2 The Bias-Complexity Trade-off 50
3.2 Deep Neural Networks 53
3.2.1 Feedforward Neural Networks 53
3.2.2 Minimization of Loss Functions 58
3.2.3 Parameter Estimation 60
3.2.4 Convolutional Neural Networks 64
3.2.5 Residual Neural Networks 75
3.2.6 Initialization of Convolutional Layers 76

4 Improving the Accuracy-Weight Trade-off 79
4.1 Introduction 79
4.1.1 Improved Scaling with Respect to all Relevant Dimensions 82
4.1.2 Building Blocks Inspired by Polynomial Smoothers 82
4.2 Related Work 83
4.2.1 Reducing the Number of Channels 83
4.2.2 Modified Layers 84
4.2.3 Multigrid-Inspired Architectures 84
4.2.4 Polynomial Neural Networks 85
4.3 Residual Learning and Multigrid Methods 86
4.3.1 Revisiting ResNet and Multigrid 86
4.3.2 Resolution Coarsening 87
4.3.3 Full Approximation Scheme (FAS) for Resolution Coarsening . . . 88
4.4 Multigrid in all Dimensions (MGiaD) 89
4.4.1 Channel Coarsening 89
4.4.2 Multigrid in Resolution and Channel Dimensions 90
4.4.3 Experimental Setup and Evaluation 92
4.4.4 Conclusion 99
4.5 Poly-MgNet: Polynomial Building Blocks in Multigrid-Inspired ResNets . 100
4.5.1 Polynomial (Smoothers) in Residual Blocks 100
4.5.2 Experimental Setup and Evaluation 106
4.5.3 Conclusion 110

5 Efficient SVD by Local Fourier Analysis 111
5.1 Introduction 111
5.2 Related Work 113
5.3 SVD of Convolutional Mappings by LFA 115
5.4 Numerical Results 118
5.5 Conclusion 123

6 Discussion & Outlook **125**

List of Notations **132**

Bibliography **134**

Appendix **145**

1 Introduction

Artificial neural networks (NNs) are a class of machine learning methods inspired by the structure and function of the human brain. They learn decision rules from a finite set of training data and aim to generalize these rules to correctly separate previously unseen data points.

The basic building blocks are *artificial neurons*, which mimic how biological neurons process input signals and generate outputs. One of the earliest realizations of this idea was the *perceptron*, proposed by Rosenblatt in 1958 [92]. For linearly separable data, the perceptron learning rule provably finds a linear decision boundary [86].

However, despite its groundbreaking nature, Minsky and Papert observed that the perceptron was limited in expressiveness, as it can only model linearly separable problems [82]. This limitation was addressed by introducing additional layers of neurons, leading to the development of *multilayer perceptrons* (MLPs) in 1967 by Ivakhnenko [34, 62]. By combining multiple hidden layers with non-linear activation functions, multilayer perceptrons became *universal approximators*. In fact, even a single hidden layer suffices to approximate any continuous function on a compact domain to arbitrary accuracy, as established by the *Universal Approximation Theorem* (1989) [25, 35, 57]. While deep neural networks, which are composed of many hidden layers, are not required for universality, they can provide significantly more efficient representations for certain classes of functions [31, 103].

Under the assumption that learnable features are shift-invariant, i.e., that the output changes consistently with shifts in the input, LeCun et al. developed LeNet, one of the first *convolutional neural networks* (CNNs) in the late 1980s and early 1990s [47, 73]. CNNs exploit the spatial structure of data through convolutional filters and pooling operations and thereby enable hierarchical feature extraction. These properties make them particularly well suited for image recognition tasks, where local patterns such as edges and textures are combined to form higher-level features.

In 2012, Krizhevsky et al. introduced AlexNet [71], which won the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC 2012) [94], a benchmark competition based on the *ImageNet* dataset for large-scale image recognition tasks. AlexNet demonstrated the effectiveness of GPU-accelerated training for deep CNNs. Since then, CNNs have consistently proven to be highly effective for image recognition tasks [53, 79, 120].

Deeper neural networks generally provide greater representational capacity and thereby enabling the modeling of more complex patterns – crucial to advances in computer vision performance. However, training very deep neural networks was constrained by issues such as vanishing or exploding gradients. Skip connections introduced with *residual networks*, e.g., ResNets, realized as additive mappings, facilitate information flow across layers and thereby stabilize gradient propagation [53, 54]. This development enabled the training of substantially deeper architectures. In particular, ResNets were the first neural networks composed of more than 100 layers to be trained successfully. At the same time, such architectures tend to be large, often containing millions of trainable parameters, which makes both training and deployment computationally demanding. Formally, a neural network is constructed by stacking layers, where each layer applies a linear mapping parameterized by a weight matrix and a bias vector, followed by a non-linear activation function. When applied successively, these transformations map an input to an output, which in classification tasks, is often interpreted as a probability distribution. Training a neural network consists of adjusting its parameters to minimize a task-specific loss on a finite training dataset using gradient-based optimization. The theoretical foundations of gradient-based methods in the neural network context were established by Amari and Yang in 1997 [123]. In practice, efficient gradient computation was enabled by backpropagation, introduced and popularized in the 1980s [93, 117].

The methods developed in this thesis focus exclusively on image classification, a central task in computer vision. Nonetheless, deep neural networks are also widely applied to related tasks such as image segmentation and object detection, as well as to other input domains including point cloud data, language and text.

A central challenge in deep neural networks is balancing *capacity*, the ability to represent complex patterns, and *complexity*, the difficulty of optimization and generalization. This challenge constitutes a fundamental problem in machine learning, referred to as the *bias-complexity trade-off* [100]. Closely related is the issue of *overparameterization*, where neural networks contain more weights than necessary for the task at hand, resulting in excessive complexity without a proportional benefit in performance. The tendency to build increasingly complex models has amplified this phenomenon. Overparameterization also increases the risk of overfitting, where a model learns the training data too closely and fails to generalize to unseen data. Moreover, both training and inference with large models are resource- and time-intensive, making them less suitable for applications with strict hardware or memory constraints.

Early efforts toward efficiency, such as the design of neural network architectures in MobileNets [59, 98], enabled competitive performance on edge devices such as mobile

Figure 1.1: Hierarchy of one-dimensional (1D) grids from fine to coarse discretizations.

phones or embedded systems. More recent approaches rely on neural network architecture search [33, 77, 97], quantization [115] and pruning [56, 75]. While these techniques often produce efficient architectures for inference, their during or post-training nature incurs substantial computational overhead. This motivates the design of efficient neural network architectures prior to training, a central challenge addressed in this thesis.

These considerations motivate the study of hierarchical methods, such as multigrid techniques, which naturally align with the layered structure of CNNs, as a means of better understanding and efficiently handling neural network structures. In this thesis, we address the problem of overparameterization from a multigrid perspective, aiming to exploit redundancies in neural network architectures to reduce complexity and thereby computational effort while preserving their capacity to represent complex patterns.

Multigrid (MG) methods are efficient hierarchical solvers for large, sparse systems of (non-)linear equations, which commonly arise from the discretization of partial differential equations (PDEs) [9, 10, 45, 107]. That is, instead of determining the continuous solution of a PDE, it is approximated by discretizing the PDE on a finite number of regularly arranged points, e.g., on a grid.

Each grid point represents an unknown of the resulting system. While direct methods are generally infeasible for such large-scale systems, classical iterative methods often suffer from slow convergence. multigrid methods address these limitations by reformulating the original problem as a hierarchy of grids at different resolutions (cf. Figure 1.1). On each grid, the system is approximated by an iterative scheme operating at its own scale, with coarser grids providing a more global perspective. Intuitively, the goal of multigrid methods is to reduce the residual error in the approximate solution. A useful perspective is to decompose the error into frequency components: iterative relaxation methods (smoothers) rapidly reduce high-frequency (oscillatory) error components on the fine grid but leave behind smooth, low-frequency errors. Crucially, these smooth errors appear oscillatory when represented on a coarser grid (cf. Figure 1.2), which makes them easier to eliminate there.

This observation leads to two fundamental, complementary components in multigrid:

(a) *Smoothing* – iterative relaxation to remove oscillatory errors and

(b) *Coarse-grid correction* – updating the fine grid solution using information from a coarser grid.

These two steps are applied recursively across multiple levels in the so-called V-cycle, where the problem is successively transferred to coarser grids and approximately solved on the coarsest grid before the solution is interpolated back to the finer grids, with relaxation applied at each level. Together, these components explain the effectiveness of

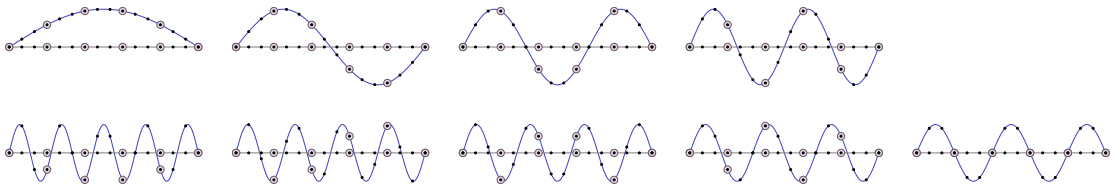


Figure 1.2: One-dimensional example of how error components (blue lines) of different frequencies appear on fine and coarse grids. The fine grid is indicated by small dots and the coarse grid by large dots. Slowly oscillating components (top row) remain distinct on the coarse grid, whereas high-frequency components (bottom row) coincide with certain low-frequency modes (aligned vertically above each other). Relative to the coarse grid, the originally slowly oscillating components now appear as high-frequency modes. As a result, nine distinct modes reduce to five distinct modes on the coarse grid.

multigrid methods, achieving optimal complexity with computational cost proportional to the number of unknowns.

Interestingly, as noted by He et al. [53], multigrid methods exhibit structural analogies with modern deep learning architectures. Similar to multigrid methods, CNNs operate across multiple resolution levels, building hierarchical feature representations.

The constant coefficients arising from the discretization of a PDE on a grid are assembled into the operator A, which acts as a convolutional kernel by locally applying shift-invariant operations across the grid. Likewise, restricting to a coarse grid in multigrid closely corresponds to pooling operations in CNNs.

Further research from the multigrid perspective has yielded methods to reduce the number of weights and computational resources of CNN architectures. He and Xu, for instance, justified reusing weight tensors across multiple layers in their *MgNet* framework [50], which can be interpreted as the left leg of a V-cycle, focusing on smoothing and restriction while omitting coarse-grid correction and prolongation. Moreover, Eliasof et al. extended the multigrid perspective to the channel dimension, namely *multigrid in channels (MGIC)* and thereby sparsified the interaction between channels [32]. These approaches highlight the potential of multigrid principles to address redundancy and overparameterization in neural network design, a perspective that is central to this thesis. In addition, the multigrid viewpoint provides powerful analytical tools. In this thesis, we employ *local Fourier analysis* (LFA) [9, 66], a classical technique for studying multigrid convergence, to analyze convolutional operators in neural networks. LFA replaces multiplication operators by their Fourier symbols, which are easier to handle analytically while still capturing the essential properties of the underlying operators.

So far, the spectral properties of convolutional operators in neural networks have been exploited mainly to improve training stability, for example through regularization [42,84, 99, 124] or to enhance robustness against adversarial attacks [19, 24], where input data are deliberately perturbed to induce misclassification. These applications underscore

the potential of spectral analysis as a tool to better understand and guide the behavior of deep CNNs.

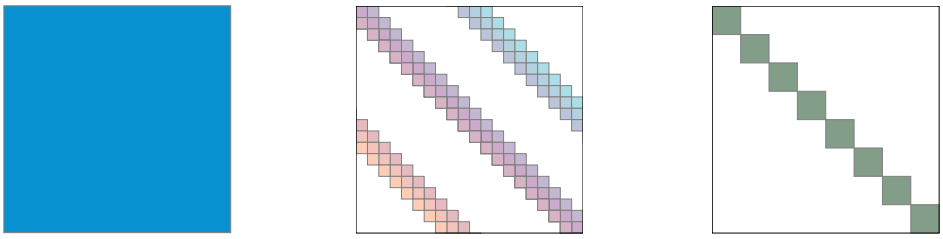
In this thesis, we build on these insights to explore multigrid analogies in deep residual networks to address the trade-off between representational capacity and model complexity. Specifically, we present three main contributions:

(a) In *MGiaD: Multigrid in all dimensions* [109], we improve efficiency and robustness along with respect to the resolution and channel dimension in residual networks by weight sharing [50] and coarsening along both dimensions.

(b) With *Poly-MgNet: Polynomial building blocks in residual networks* [111], we establish the use of polynomial building blocks inspired by multigrid smoothers as a natural extension of MgNet [50]. Thereby, we drastically reduce the number of weights with only a negligible loss in accuracy. We initialize the polynomial coefficients by exploiting the spectrum of the corresponding convolutional mapping. Our third contribution facilitates an efficient computation of this spectrum.

(c) In *LFA applied to CNNs: Efficient singular value decomposition* [110], we employ LFA to develop a scalable algorithm for the efficient computation of the singular value and eigenvalue decompositions of convolutional mappings by exploiting their shift-invariance.

Multigrid in all relevant dimensions. The majority of weights in an neural network is found in linear mappings. Without further assumptions, these linear maps are represented by dense (weight) matrices, i.e., fully connected layers, with corresponding sparsity pattern illustrated in Figure 1.3a. Intuitively, the number of weights depends on the spatial dimensions of the input and output and quickly becomes intractable as the input dimension increases. On the other hand, the weight complexity of convolutional layers becomes independent of the input dimensions due to their local action. The corresponding structure can be represented as blocked, banded matrix, as depicted in Figure 1.3b. However, the number of weights still scales quadratically with respect to the number of channels. Unfortunately, no assumption such as translation invariance can be made for the channel dimension. Typically, attempts to manually sparsify connectivity in CNN layers are met with significant loss in performance. To achieve improved scaling behavior with respect to all relevant dimensions, i.e., resolution and channels, while maintaining classification accuracy, we pick up recent developments in MG-inspired architectures and present a new efficient CNN architecture. By combining elements introduced in [50] and [32] in a natural, albeit subtle, manner, our architecture can be completely explained using multigrid terminology and thereby achieves a drastic reduction in the number of weights. Compared to ResNet, MgNet and MGIC, our approach achieves superior accuracy, under certain conditions.



(a) Fully connected (b) Convolutional mapping (c) Fourier block matrix

Figure 1.3: Sparsity patterns of mappings in neural network layers represented in matrix form. (a) Fully connected layer, (b) convolutional mapping, (c) block matrix of a convolutional mapping after the Fourier transform.

Polynomial building blocks. The solution u of a linear system $Au = f$ can be iteratively approximated by

$$u \leftarrow u + B(f - Au),$$

where f is the right-hand side, A denotes the system matrix and $B \approx A^{-1}$ serves as smoothing operator (smoother). The objective of such an iteration is to refine the solution u while reducing the error. He and Xu [50] observed that this smoothing step corresponds to a residual block, the fundamental building block of ResNet models, where both operators A and B are implemented as trainable convolutional operators. This interpretation justifies reusing the operator A across multiple ResNet-blocks, thereby reducing the total number of weights. However, the operator B is still realized as a fully coupled convolution, with distinct weights in each block. To further decrease the weight count, we also seek to express B in terms of A. Inspired by polynomial smoothers from multigrid, we replace $B \approx p(A)$ with a polynomial in A. Due to the fact that $p(A)$ has a low-degree, this operation now only consists of a handful of weights. In particular, the simplest case is the Richardson iteration [107], where $B = \omega I$. Here I denotes the identity and ω is a scalar. Clearly, we do not want to limit ourselves to such simple cases and explore in this thesis to what extent smoothing iterations using polynomial approximations are suitable within MgNet. In order to reduce the weight count in MG-inspired CNNs such as MgNet, we introduce a new building block. Our methods are inspired by multigrid smoothers, which yield layer modules with a reduced weight count. We implement these building blocks in MgNet and show that we can drastically reduce the overall weight count, while almost maintaining classification accuracy. Our studies yield further insights into the inherent connection between multigrid and ResNets and demonstrate that the multigrid methodology is useful for the construction of weight-count-efficient CNNs. To initialize the polynomial coefficients spectral properties of the system matrix A can be used, facilitating an improved performance with respect to classification accuracy.

Spectral analysis of convolutional layers. The singular values of convolutional mappings capture interesting spectral properties, which can be exploited, for instance, to

initialize the coefficients of polynomial building blocks. A convolutional mapping can be represented as a large, sparse, banded matrix, whose dimensions scale with the spatial size of the input as well as the number of channels. Consequently, the explicit computation of singular values quickly becomes infeasible due to memory constraints.

Existing methods [99] overcome this issue by employing the *Fast Fourier Transform* (FFT) to transfer the convolutional mapping into the frequency domain, where the matrix representation decomposes into smaller block matrices (cf. Figure 1.3c), whose singular values can be computed independently. For a fixed number of channels, this FFT-based approach computes the N singular values, where N denotes the number of spatial locations, with an asymptotic complexity of $\mathcal{O}(N \log N)$. We further reduce this computational complexity by a factor of $\log(N)$ to $\mathcal{O}(N)$ by exploiting the translation invariance of convolutional operators, which enables the use of LFA. Originally developed to study iterative methods, in particular multigrid schemes, LFA is employed here to efficiently analyze convolutional mappings. For this task, we propose an LFA-based algorithm and provide a theoretical analysis of its computational complexity, showing that it is asymptotically optimal with respect to the input dimension. This improvement not only establishes a theoretically efficient approach but also enhances the scalability of spectral analysis in large-scale neural network architectures.

Roadmap of the thesis. This thesis is organized into a total of six chapters, structured as follows. In Chapter 2 and Chapter 3, we introduce the foundations of numerical linear algebra and machine learning, which form the basis for the methods developed in this thesis. In Section 2.1 we discuss the eigenvalue decomposition and singular value decomposition, the latter of which is applied to convolutional mappings in later chapters. We focus on crystalline structures, introducing lattices and crystals and developing Fourier functions on crystal tori in Section 2.2. Building on these concepts, we define discrete Fourier transforms in Section 2.3 and develop the idea of LFA, which forms the foundation for efficient singular value decomposition of convolutional mappings. Finally, we examine iterative methods in Section 2.4 followed by multigrid methods, which are used to develop efficient neural network architectures later in this thesis. In Chapter 3, we summarize theoretical foundations in machine learning relevant to this thesis. In this chapter we briefly discuss statistical learning theory (Section 3.1), to provide the theoretical foundation for supervised learning methods. Within the statistical learning framework, the trade-off between bias and complexity is discussed, which is a fundamental problem in machine learning. In this thesis, we address this trade-off from an multigrid perspective exploiting hierarchical structures to reduce complexity while preserving representational capacity. Next, we explain deep neural networks as well as their learning algorithms (Section 3.2). Here, the focus is on CNNs, in particular residual networks, whose specific structure exhibits analogies to multigrid methods. These structural analogies form the foundation for our methods developed in this thesis. The notation introduced in these foundational chapters is used consistently throughout.

We dedicate Chapters 4 and 5 to the presentation and explanation of our methods. In Chapter 4, we introduce our approaches to address overparameterization in CNNs for image classification by exploiting structural analogies between residual networks and multigrid methods. We present two families of CNNs, namely MGiaD and Poly-MgNet, which share the same theoretical foundations and are therefore discussed together. We provide their motivation, review related work, present theoretical background and architectures, and evaluate their performance experimentally.

In Chapter 5, we present an algorithm for the efficient computation of the singular values of convolutional mappings using the LFA framework. In particular, we describe our algorithm, analyze its computational complexity and evaluate its runtime for different lattice sizes.

Ultimately, we conclude this thesis with a summary of our contributions and an outlook on future research in Chapter 6.

2 Fundamentals of Numerical Linear Algebra

The direct calculation of the solution of given large systems of (non-)linear equations that arise from boundary value problems, is often time-consuming and computationally expensive. In particular, direct methods typically do not exploit the sparsity of the system matrix, as the factorization process of direct solvers introduce additional nonzero elements. To obtain feasible solutions, the solution is approximated by *iterative methods*, where an initial guess is successively refined to approximate the solution. Ideally, the approximated solution converges to the exact solution.

In Section 2.4, we introduce basic iterative solvers, especially the Richardson iteration and its convergence properties. However, iterative methods still suffer from slow convergence, particularly for large-scale problems. A convergence rate independent of the system size can be achieved by using *multigrid methods* (MG methods). Multigrid methods are based on a hierarchy of grids, where each grid is a coarser representation of the problem and contributes to the approximate solution [10, 107]. In Section 2.5, we study multigrid methods, where iterative methods act as smoothers.

To analyze the different multigrid components of translation-invariant problems, *local Fourier analysis* (LFA) is used [9, 64]. It provides a frequency-domain approach to evaluate and optimize components of iterative and multigrid methods.

Another branch of this work focuses on the singular values of large matrices to gain insights into the spectral properties of the iteration matrix. Similar to solving linear systems, the direct calculation of the *singular value decomposition* (SVD) of large (sparse) matrices is prohibitively expensive in terms of both floating-point operations (FLOPs) and memory consumption. Since singular values in the frequency domain are equal to those in the spatial domain, leveraging the *Fast Fourier Transform* (FFT) enables the efficient computation of the singular value decomposition whenever applicable.

LFA is generally used in the construction and analysis of multigrid methods to obtain a precise approximation of the asymptotic convergence rate [9]. The main idea is to

leverage the connection between position space and frequency space via the Fourier transform so that translation-invariant operators, i.e., stencils in the position space, can be described as so-called symbols. A fundamental concept for applying LFA is the notion of lattices and crystals. An (ideal) crystal is characterized by the infinite repetition of a basic structural unit, the unit cell, arranged in a regular pattern defined by a lattice.

The remainder of this section is structured as follows. In Section 2.1, we focus on matrix decomposition methods, followed by an introduction to crystalline structures in Section 2.2 as a basis for defining Fourier functions on these structures and subsequently developing the discrete Fourier transforms on lattices in Section 2.3. Finally, we discuss basic iterative methods in Section 2.4 and multigrid methods in Section 2.5.

Notation and preliminaries. For the discussion, let $A \in \mathbb{C}^{m\times n}$ be a matrix, i.e., an $m \times n$ array of entries

$$a_{ij},\ i = 1, \dots, m,\ j = 1, \dots, n.$$

The j-th column of A is denoted by $a_{*j} = (a_{1j}, a_{2j}, \dots, a_{mj})^{\mathrm{T}} \in \mathbb{C}^m$, and the i-th row of A is accordingly denoted by $a_{i*} = (a_{i1}, a_{i2}, \dots, a_{in}) \in \mathbb{C}^n$. For simplicity, we write a_j instead of a_{*j} whenever no ambiguity arises. Unless explicitly needed otherwise, we write $A = [a_1|a_2|\dots|a_n]$ as a matrix of columns.

This row–column notation is convenient because it allows us to interpret a matrix both as a collection of column vectors and as a collection of row vectors. In particular, many matrix decompositions can be written in terms of outer products of column and row vectors. This representation makes the structure of the matrix more transparent.

A real *square* matrix $A \in \mathbb{R}^{m\times m}$ is *symmetric* if $a_{ij} = a_{ji}$, that is, its entries below the diagonal are equal to the entries above the diagonal and $A = A^{\mathrm{T}}$, where A^{T} denotes the transpose if A. An analogous property hold for a complex matrix $A \in \mathbb{C}^{m\times m}$, in the sense that A is *Hermitian* if $a_{ij} = \bar{a}_{ji}$, i.e., $A = A^*$. Here A^* denotes the conjugate transpose of A. The entries on the diagonal are real and the entries below the diagonal are the complex conjugates of the corresponding entries above the diagonal. Furthermore, if A is hermitian, we have $x^*Ay = \overline{y^*Ax}$ for all $x, y \in \mathbb{C}^m$ and in particular x^*Ax is real for $x \in \mathbb{C}^m$. If $x^*Ax > 0$ for all nonzero $x \in \mathbb{C}^m$, then A is *symmetric positive definite* [105].

2.1 Matrix Decompositions

One of the central results in numerical linear algebra is that any matrix can be decomposed into the product of two unitary matrices and a diagonal matrix. This factorization, known as *singular value decomposition*, provides a powerful framework for analyzing the structure and properties of matrices. To motivate its existence, we first turn to the *eigenvalue decomposition* of square matrices. The eigenvalue decomposition expresses

a matrix in terms of its eigenvalues and eigenvectors and in the symmetric (Hermitian) case yields an orthonormal basis of eigenvectors, i.e., a unitary diagonalization. This result forms the theoretical foundation of the singular value decomposition as the singular values arise as the square roots of the eigenvalues of an associated symmetric (Hermitian) matrix [39, 105].

We begin by reviewing the mathematics of eigenvalues and eigenvectors, then develop a geometric motivation for the singular value decomposition before turning to its algebraic formulation, following [105].

2.1.1 Eigenvalue Problems

Eigenvalue decomposition. Let $A \in \mathbb{C}^{m\times m}$ be a square matrix. The *characteristic polynomial* of A is defined as a polynomial of degree m,

$$p_A(\lambda) = \det(\lambda I_m - A) \quad \text{with} \quad \lambda \in \mathbb{C},$$

where I_m is the identity matrix.[1] A scalar $\lambda \in \mathbb{C}$ is called an *eigenvalue* of A if and only if it is a root of the characteristic polynomial, i.e., $p_A(\lambda) = 0$. Associated with each eigenvalue is at least one nonzero vector $\gamma \in \mathbb{C}^m$ satisfying

$$A\gamma = \lambda\gamma. \tag{2.1}$$

Such a vector is called an *eigenvector* corresponding to λ.

The set of eigenvectors belonging to an eigenvalue λ, together with the zero vector, forms a subspace of $\mathbb{C}^m$ called the *eigenspace* E_λ associated with λ. Its dimension $\dim(E_\lambda)$ is called the *geometric multiplicity* of λ. The *algebraic multiplicity* of λ is its multiplicity as a root of the characteristic polynomial. It holds that $1 \leq \dim(E_\lambda) \leq$ algebraic multiplicity. The set of all eigenvalues λ of A is called the *spectrum* of A, denoted by

$$\operatorname{spec}(A) = \{\lambda \in \mathbb{C} \mid \exists\, \gamma \neq 0 : A\gamma = \lambda\gamma\}.$$

The *spectral radius* $\rho(A)$ of A is the largest modulus of its eigenvalues, i.e.,

$$\rho(A) = \max_{\lambda \in \operatorname{spec}(A)} |\lambda|. \tag{2.2}$$

If A possesses m linearly independent eigenvectors $\gamma_1, \ldots, \gamma_m$ we may collect them into a nonsingular[2] matrix $X = [\gamma_1|\gamma_2|\ldots|\gamma_m]$. Let $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$ be the diagonal matrix of the corresponding eigenvalues. A matrix $A \in \mathbb{C}^{m\times m}$ is called *diagonalizable* if and only if it can be written as

$$A = X\Lambda X^{-1}.$$

[1] The $m \times m$ identity matrix I_m is defined by $I_m = (\delta_{i,j})_{i,j=1}^m$, where $\delta_{i,j}$ is the Kronecker symbol.
[2] A square matrix A with $\det(A) = 0$ is called singular.

This factorization is known as the eigenvalue decomposition of A. Equivalently, $AX = X\Lambda$ shows that the j-th column of X is an eigenvector of A with corresponding eigenvalue given by the j-th diagonal entry of Λ, i.e., $A\gamma_j = \lambda_j\gamma_j$. The eigenvalue decomposition can also be interpreted as a change of basis to the eigenbasis of A. For example, consider the linear system $A\gamma = f$. Multiplying from the left by X^{-1} yields $\Lambda X^{-1}\gamma = X^{-1}f$. In this eigenbasis, solving the system reduces to independent scalar equations [96]. Note, that a matrix $A \in \mathbb{C}^{m\times m}$ is diagonalizable if and only if, for every eigenvalue λ, the dimension of the eigenspace $|\dim(E_\lambda)|$ equals the algebraic multiplicity of λ.

If a matrix $A \in \mathbb{C}^{m\times m}$ is normal, i.e., $A^*A = AA^*$, then A is *unitarily diagonalizable*. In particular, there exists a unitary matrix[3] Q such that $A = Q\Lambda Q^*$.

Schur decomposition. Not every matrix is diagonalizable. However, every square matrix can be reduced to an upper triangular matrix via a unitary similarity transformation.

In particular, for any $A \in \mathbb{C}^{m\times m}$, there exists a unitary matrix $Q \in \mathbb{C}^{m\times m}$ such that

$$A = QTQ^*,$$

where $T \in \mathbb{C}^{m\times m}$ is upper triangular with eigenvalues of A as its diagonal entries. Thus, while diagonalization is not always possible, the Schur decomposition guarantees that any square matrix is unitarily similar to an upper triangular matrix, making the eigenvalues immediately visible.[4]

2.1.2 Singular Value Decomposition

The singular value decomposition is another spectral decomposition that, in contrast to the eigenvalue decomposition, can also be formulated for rectangular matrices $A \in \mathbb{C}^{m\times n}$. It is applicable to real and complex matrices alike. Its geometric motivation arises from the fact that the image of the unit sphere in the Euclidean norm[5] under the linear mapping induced by $A \in \mathbb{C}^{m\times n}$ is a hyperellipsoid. More precisely, if $\mathrm{S} \subset \mathbb{C}^n$ denotes the unit sphere, then $A\mathrm{S} \subset \mathbb{C}^m$ is a hyperellipsoid E. The *singular values* σ_j of A correspond to the lengths of the semi-axes of E, oriented along the *left singular vectors* $u_1, u_2, \ldots, u_m$. By collecting the *right singular vectors* as the columns of a unitary matrix $V = [v_1|v_2|\ldots|v_n] \in \mathbb{C}^{n\times n}$ and the left singular vectors as the columns of a unitary matrix $U = [u_1|u_2|\ldots|u_m] \in \mathbb{C}^{m\times m}$, and the singular values σ_i as diagonal entries of a diagonal matrix $\Sigma \in \mathbb{R}^{m\times n}$, the singular value decomposition of A is

$$A = U\Sigma V^*.$$

[3] $Q \in \mathbb{C}^{m\times m}$ is unitary if $Q^{-1} = Q^*$.

[4] The proof proceeds by induction and can be found in [105, Theorem 24.9].

[5] The Euclidean norm (or 2-norm) of a vector $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ is defined as $||v||_2 := \sqrt{v_1^2 + v_2^2 + \ldots v_n^2}$.

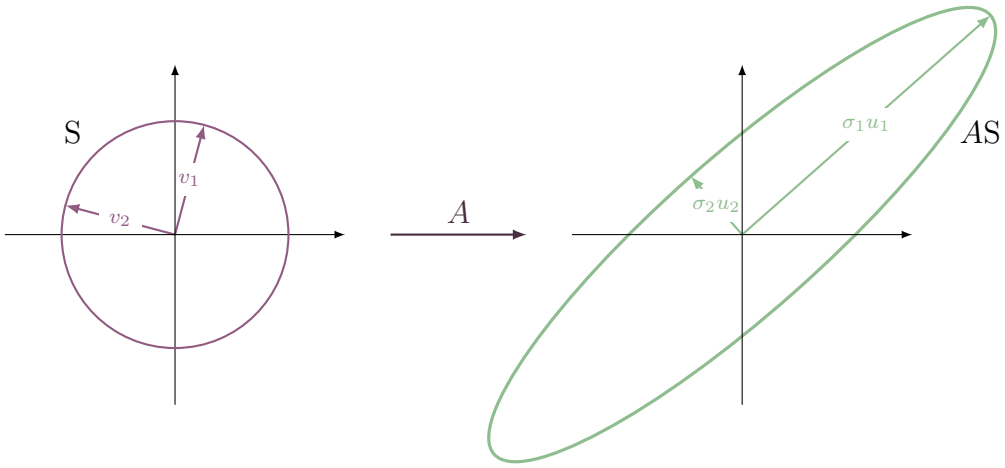


Figure 2.1: Geometric interpretation of the singular value decomposition of $A = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}$. The action of A maps the unit circle to an ellipse whose semi-axes have lengths given by the singular values σ_1 and σ_2 with directions determined by the left singular vectors u_1 and u_2.

Here, V^* denotes the conjugate transpose of V. The singular values σ_j are conventionally ordered non-increasingly, i.e., $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\mathtt{r}} \geq 0$ with $\mathtt{r} = \min(m, n)$. The number of positive singular values equals the rank of A, i.e., $\operatorname{rank}(A)$.

As illustrated in Figure 2.1, the left singular vectors u_i are mutually orthogonal. Together, they form an orthonormal basis of the codomain of A. Similarly, the right singular vectors v_j from an orthonormal basis of the domain of A. Thus, multiplication by V^* transforms an input vector into a basis aligned with the directions along which A exhibits minimal or maximal stretching, as determined by the singular values σ_j. In this basis, each singular value σ_j scales its associated direction.

The first $\mathtt{r}$ columns of U form an orthonormal basis for the image of A. They describe how the action of A maps scaled input directions into new directions in the output space. The fundamental relation between the orthonormal bases[6] U and V is $Av_j = \sigma_j u_j$ with $1 \leq j \leq \mathtt{r}$. It is important to note that every matrix has a singular value decomposition[7] whereas the class of matrices that can be factorized by an eigenvalue decomposition is limited to non-defective matrices [105].[8]

Given a rectangular matrix $A \in \mathbb{C}^{m \times n}$, we can compute its *reduced singular value decomposition* $A = \hat{U}\hat{\Sigma}V^*$. For $m \geq n$, the diagonal matrix $\hat{\Sigma} \in \mathbb{R}^{n \times n}$ contains the singular values. Both $\hat{U} \in \mathbb{C}^{m \times n}$ and $V \in \mathbb{C}^{n \times n}$ have orthonormal columns. If required, the full singular value decomposition can be obtained by padding $\hat{\Sigma}$ with $m - n$ zero rows

[6] This does not hold for the reduced representation.
[7] The theorem and its proof can be found in [105, Theorem 4.1].
[8] An eigenvalue whose algebraic multiplicity exceeds its geometric multiplicity is a *defective eigenvalue*. A matrix is called *defective* if it contains one or more defective eigenvalues [105].

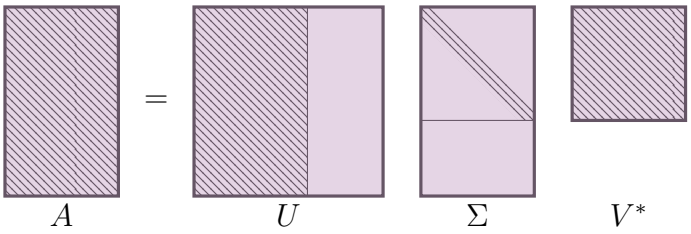


Figure 2.2: Illustration of the singular value decomposition for $m \geq n$. The full singular value decomposition includes both shaded and unshaded purple regions, whereas the reduced singular value decomposition comprises only the shaded purple regions corresponding to the nonzero singular values.

at the bottom to form $\Sigma \in \mathbb{R}^{m \times n}$, and extending $\hat{U}$ to a full unitary matrix by adding $(m - n)$ orthonormal columns on the the right, as illustrated in Figure 2.2. For $m \leq n$ the situation is analogous with the dimensions adjusted accordingly.

2.2 Crystalline Structures

In order to provide a mathematically concise framework for the later discussion of DFTs and the convolution theorem, we introduce the concepts of crystalline structures and quotient spaces in this section. An (ideal) crystal is characterized by the infinite repetition of a basic structure element, the unit cell, arranged in a regular pattern determined by a lattice. This section is based on [66, 69].

2.2.1 Lattices and Crystals

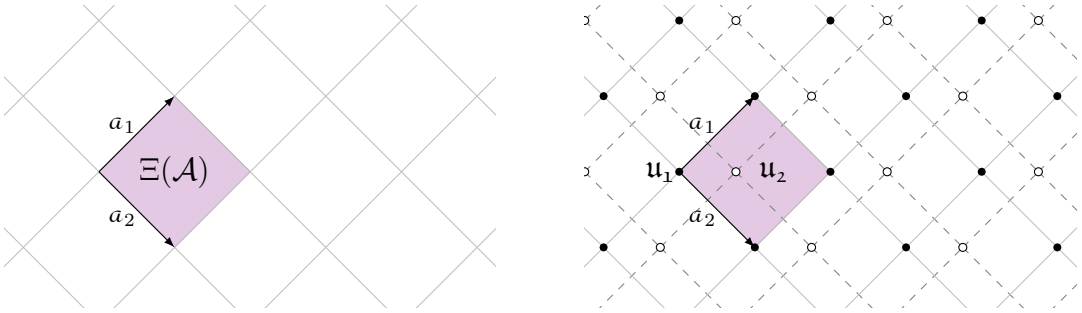
Consider linearly independent vectors $\mathcal{A} = [a_1|a_2|\dots|a_n] \in \mathbb{R}^{n \times n}$. The set of points

$$\mathbb{L} = \{x = \sum_{i=1}^{n} \mu_i a_i \in \mathbb{R}^n : \mu_1, \dots, \mu_n \in \mathbb{Z}\}$$

defines an n-dimensional lattice generated by $\mathcal{A}$. The vectors $a_1, a_2, \dots, a_n$ are referred to as the *primitive vectors* and together they form the *lattice basis* $\mathcal{A} \in \mathbb{R}^{n \times n}$. In matrix notation, we write the lattice $\mathbb{L}$ compactly as

$$\mathbb{L}(\mathcal{A}) = \{x = \mathcal{A} \cdot \mu \ : \ \mu \in \mathbb{Z}^n\} := \mathcal{A}\mathbb{Z}^n.$$

Observe that $(\mathbb{L}(\mathcal{A}), +)$ forms an abelian group. That is, $\mathbb{L}(\mathcal{A})$ is closed under addition, so that for any $x, y \in \mathbb{L}(\mathcal{A})$ we also have $x + y \in \mathbb{L}(\mathcal{A})$. Moreover, the zero vector 0 belongs to $\mathbb{L}(\mathcal{A})$ and for every $x \in \mathbb{L}(\mathcal{A})$ the additive inverse element $-x \in \mathbb{L}(\mathcal{A})$ exists.



(a) Unit cell $\Xi(\mathcal{A})$ of lattice $\mathbb{L}(\mathcal{A})$ (b) Crystal $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$

Figure 2.3: (a) Two-dimensional lattice $\mathbb{L}(\mathcal{A})$ with lattice basis $\mathcal{A} = [a_1\ a_2]$ and unit cell $\Xi(\mathcal{A})$ (shaded in purple). (b) A crystal $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$ consists of copies of $\mathcal{A}$, each translated by the lattice shifts $\mathfrak{u}_1$ and $\mathfrak{u}_2$, specifying the relative positions within the unit cell (shaded in purple).

With the lattice $\mathbb{L}(\mathcal{A})$ defined, we now introduce the notion of a unit cell. A unit cell $\Xi(\mathcal{A}) \subseteq \mathbb{R}^n$ is a connected set such that its translates by the lattice vectors of $\mathbb{L}(\mathcal{A})$ form a partition of $\mathbb{R}^n$:

$$\dot{\bigcup}_{x \in \mathbb{L}} \{x + \xi : \xi \in \Xi(\mathcal{A})\} = \mathbb{R}^n. \tag{2.3}$$

Specifically, the union over all lattice elements *tiles* $\mathbb{R}^n$ without overlap or gaps. A canonical choice of a unit cell is given by

$$\Xi(\mathcal{A}) = \mathcal{A}[0,1)^n = \{y = \mathcal{A} \cdot \varphi : \varphi \in [0,1)^n\}.$$

Although there exists various possible primitive cells, this natural and simple choice is the parallelotope spanned by the primitive vectors $a_1, a_2, \ldots, a_n$. Figure 2.3a illustrates a two-dimensional lattice with its corresponding unit cell.

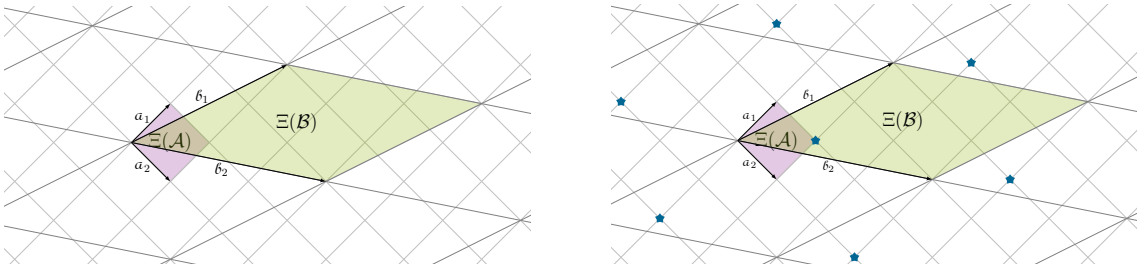
Sublattices. Let $\mathbb{L}(\mathcal{A})$ and $\mathbb{L}(\mathcal{B})$ be two lattices. We call $\mathbb{L}(\mathcal{B})$ a sublattice of $\mathbb{L}(\mathcal{A})$, i.e., $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$ if and only if $Z := \mathcal{A}^{-1}\mathcal{B} \in \mathbb{Z}^{n \times n}$.

Assume first that $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$. The j-th primitive vector of $\mathcal{B}$ is given by $\mathcal{B}e_j$, where e_j denotes the j-th unit vector in $\mathbb{R}^n$. Since $\mathcal{B}e_j \in \mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$, we can express $\mathcal{B}e_j$ as an integer linear combination of the primitive vectors of $\mathcal{A}$. That is, there exists $z_j \in \mathbb{Z}^n$ such that $\mathcal{B}e_j = \mathcal{A}z_j$. Equivalently, $\mathcal{A}^{-1}\mathcal{B}e_j = z_j \in \mathbb{Z}^n$. Since this holds for all j we obtain $Z := \mathcal{A}^{-1}\mathcal{B} = [z_1|z_2|\ldots|z_n] \in \mathbb{Z}^{n \times n}$.

Conversely, suppose $Z = \mathcal{A}^{-1}\mathcal{B} \in \mathbb{Z}^{n \times n}$. Let $x \in \mathbb{L}(\mathcal{B})$. Then there exists $y \in \mathbb{Z}^n$ such that $x = \mathcal{B}y$ and it follows that

$$x = \mathcal{B}y = \mathcal{A}(\mathcal{A}^{-1}\mathcal{B})y = \mathcal{A}Zy.$$

Since $Zy \in \mathbb{Z}^n$, we conclude that $x \in \mathbb{L}(\mathcal{A})$. Hence, $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$ [69]. Figure 2.4a depicts a lattice and a corresponding sublattice. A sublattice $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$ with $\mathcal{B} =$



(a) Sublattice $\mathbb{L}(\mathcal{B})$ of $\mathbb{L}(\mathcal{A})$ generated by the basis vectors $[b_1, b_2]$ with the corresponding unit cells $\Xi(\mathcal{A})$ shaded in purple and $\Xi(\mathcal{B})$ shaded in green

(b) Periodicity of the crystal torus $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$, illustrated by blue stars

Figure 2.4: Sublattice $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$ and periodicity of the corresponding crystal torus $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$.

$\mathcal{A} \cdot Z$ is called *aligned* with $\mathbb{L}(\mathcal{A})$ if and only if

$$Z = \operatorname{diag}(\mu_1, \mu_2, \ldots, \mu_n).$$

In particular, for the lattice bases $\mathbb{L}(\mathcal{A})$ and $\mathbb{L}(\mathcal{B})$ it is

$$b_\ell = a_\ell \cdot \mu_\ell \quad \text{for} \quad \ell = 1, 2, \ldots, n.$$

Crystals. A *crystal* structure generalizes a lattice by allowing the infinite repetition of a fixed structure element attached to each lattice point. In the following, we restrict our discussion to primitive cells of the lattice, without loss of generality.

Let $\mathbb{L}(\mathcal{A})$ be a lattice and $\mathfrak{u} = [\mathfrak{u}_1, \mathfrak{u}_2, \ldots, \mathfrak{u}_\nu] \in \Xi(\mathcal{A})^\nu$ the structure element consisting of $\nu \in \mathbb{N}$ points located within the unit cell $\Xi(\mathcal{A})$. The associated crystal is given by the point set

$$\mathbb{L}^{\mathfrak{u}}(\mathcal{A}) = \{x = \mathcal{A} \cdot \mu + \mathfrak{u}_\ell : \mu \in \mathbb{Z}^n, \ell = 1, \ldots, \nu\}.$$

Specifically, each lattice point $x \in \mathbb{L}(\mathcal{A})$ is shifted by the vectors $\mathfrak{u}_\ell$, such that $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$ is the union of ν translated copies of the lattice, with each point of the structure element $\mathfrak{u}_\ell$ repeated at every lattice point, as depicted in Figure 2.3b. Note that every lattice can be regarded as a crystal if the corresponding structure element consists of a single point.

A crystal $\mathbb{L}^{\mathfrak{v}}(\mathcal{B})$ is a subcrystal of $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$ satisfying $\mathbb{L}^{\mathfrak{v}}(\mathcal{B}) \subseteq \mathbb{L}^{\mathfrak{u}}(\mathcal{A})$.

To focus on finite-dimensional periodic structures and to deepen the understanding of the relationships between lattices and their sublattices, we introduce lattice tori as quotient groups.

Quotient spaces and lattice tori. Let $\mathbb{L}(\mathcal{A}), \mathbb{L}(\mathcal{B}) \subseteq \mathbb{R}^n$ be lattices such that $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$. Since $(\mathbb{L}(\mathcal{A}), +)$ forms a group, $(\mathbb{L}(\mathcal{B}), +)$ constitutes a subgroup of $(\mathbb{L}(\mathcal{A}), +)$, and the *quotient group*

$$\mathbb{T}_{\mathcal{B}}(\mathcal{A}) = \mathbb{L}(\mathcal{A}) \Big/ \mathbb{L}(\mathcal{B})$$

is well defined. The *lattice torus* $\mathbb{T}_{\mathcal{B}}(\mathcal{A})$ represents the periodicity of the lattice $\mathbb{L}(\mathcal{A})$ with respect to the sublattice $\mathbb{L}(\mathcal{B})$, capturing the finite pattern that repeats throughout the space. While the lattice $\mathbb{L}(\mathcal{A})$ defines the periodic arrangement of points in space, a crystal refines the periodicity by specifying the relative positions $\mathfrak{u}$ within each unit cell. Accordingly, given a crystal $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$, the corresponding *crystal torus* is defined by

$$\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}} := \mathbb{L}^{\mathfrak{u}}(\mathcal{A}) \Big/ \mathbb{L}(\mathcal{B}).$$

An element of $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ is an equivalence class $[x]_{\mathcal{B}}$ with representative $x \in \mathbb{L}^{\mathfrak{u}}(\mathcal{A})$. Specifically, for $[x]_{\mathcal{B}}, [y]_{\mathcal{B}} \in \mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ we have $[x]_{\mathcal{B}} = [y]_{\mathcal{B}}$ if and only if $x - y \in \mathbb{L}(\mathcal{B})$. Recall that $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$, so there exists a matrix $Z \in \mathbb{Z}^{n\times n}$ such that $\mathcal{B} = \mathcal{A} \cdot Z$. Furthermore, $|\mathfrak{u}|$ denotes the number of degrees of freedom in each unit cell of $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$. Then, for the crystal torus $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ the number of degrees of freedom is given by $|\det(Z)| \cdot |\mathfrak{u}|$. This construction can be interpreted as a periodic box of the infinite crystal.

To illustrate this concept, consider a two-dimensional lattice $\mathbb{L}(\mathcal{A})$ with $\mathcal{A} \in \mathbb{Z}^{2\times 2}$ and the lattice torus $\mathbb{T}_{\mathcal{A},\mathcal{B}}$, where

$$\mathcal{B} = \mathcal{A} \cdot Z = \mathcal{A} \cdot \begin{bmatrix} 3 & 2 \\ 1 & 3 \end{bmatrix} \in \mathbb{Z}^{2\times 2}.$$

The corresponding quotient space, shown in Figure 2.4b, consists of $|\mathbb{T}_{\mathcal{A},\mathcal{B}}| = |\det(Z)| = 7$ elements.

From a geometric perspective, the lattice torus is obtained by identifying opposite boundaries of a domain having the size of a single unit cell of the sublattice $\mathbb{L}(\mathcal{B})$, resulting in *periodic boundary conditions* in all dimensions.

The dual lattice. Let $\mathbb{L}(\mathcal{A})$ be a lattice. We define its *dual lattice* $\mathbb{L}(\mathcal{A})^*$ as

$$\mathbb{L}(\mathcal{A})^* = \{k \in \mathbb{R}^n : \langle x, k\rangle \in \mathbb{Z} \text{ for all } x \in \mathbb{L}(\mathcal{A})\},$$

where $\langle x, k\rangle = x^{\mathrm{T}}k$ denotes the Euclidean inner product. The lattice basis of $\mathbb{L}(\mathcal{A})^*$ is given by $\mathcal{A}^{-\mathrm{T}}$, i.e., $\mathbb{L}(\mathcal{A})^* = \mathbb{L}(\mathcal{A}^{-\mathrm{T}})$. Furthermore, the dual of the dual lattice is the lattice itself. The elements of the dual lattice $\mathbb{L}(\mathcal{A})^*$ are also referred to as *wave vectors*.

The concept of sublattices naturally extends to dual lattices. For two lattices $\mathbb{L}(\mathcal{A})$ and $\mathbb{L}(\mathcal{B})$ with $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$ the corresponding dual lattices satisfy

$$\mathbb{L}(\mathcal{A})^* \subseteq \mathbb{L}(\mathcal{B})^*.$$

Since $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$, there exists an integer matrix $Z \in \mathbb{Z}^{n\times n}$ such that $\mathcal{B} = \mathcal{A} \cdot Z$. Due to the fact that $\mathcal{A}$ and $\mathcal{B}$ are nonsingular we find

$$\mathcal{B}^{-\mathrm{T}} = \mathcal{A}^{-\mathrm{T}} Z^{\mathrm{T}} \iff \mathcal{A}^{-\mathrm{T}} = \mathcal{B}^{-\mathrm{T}} Z^{\mathrm{T}}.$$

Consequently, $\mathbb{L}(\mathcal{A})^* = \mathbb{L}(\mathcal{A}^{-\mathrm{T}}) = \mathbb{L}(\mathcal{B}^{-\mathrm{T}}Z^{\mathrm{T}}) \subseteq \mathbb{L}(\mathcal{B}^{-\mathrm{T}}) = \mathbb{L}(\mathcal{B})^*$. Furthermore, if $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ is the crystal torus associated with the lattice-sublattice pair $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$, then its dual crystal torus $\mathbb{T}^*_{\mathcal{A},\mathcal{B}}$ is given by

$$\mathbb{T}^*_{\mathcal{A},\mathcal{B}} = \Big(\mathbb{L}(\mathcal{A}) \Big/ \mathbb{L}(\mathcal{B})\Big)^* = \mathbb{L}(\mathcal{B})^* \Big/ \mathbb{L}(\mathcal{A})^*.$$

The number of elements in the dual lattice torus equals the number of elements the of the original lattice torus:

$$|\mathbb{T}_{\mathcal{A},\mathcal{B}}| = \frac{|\det(\mathcal{B})|}{|\det(\mathcal{A})|} = |\det(Z)| = \frac{|\det(\mathcal{A}^{-\mathrm{T}})|}{|\det(\mathcal{B}^{-\mathrm{T}})|} = |\mathbb{T}^*_{\mathcal{A},\mathcal{B}}|.$$

2.2.2 Function Spaces on Crystals

Building on the concepts of lattices and crystalline structures, we now lay the foundation for DFTs on crystalline structures by defining the corresponding function space.

Function spaces. The function space of a crystal torus $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ is defined by

$$\mathfrak{L}(\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}) = \left\{ f = (f_1, \ldots, f_{|\mathfrak{u}|}) : \mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}} \longrightarrow \mathbb{C}^{|\mathfrak{u}|} \right\}, \tag{2.4}$$

which is equipped with the inner product

$$\langle f, g \rangle := \frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x \in \mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} \langle f(x), g(x) \rangle_2 \tag{2.5}$$

where the Euclidean inner product on $\mathbb{C}^{|\mathfrak{u}|}$ is given by $\langle f(x), g(x)\rangle_2 := \sum_{l=1}^{|\mathfrak{u}|} f_l(x)\overline{g_l(x)}$. For each lattice point x, the component $f_j(x)$ corresponds to the values associated with the translated lattice point $x + \mathfrak{u}_j$. Furthermore, for $k \in \mathbb{T}^*_{\mathcal{A},\mathcal{B}}$ the wave functions $\varphi_{1,k}, \varphi_{2,k}, \ldots, \varphi_{|\mathfrak{u}|,k}$ form an orthonormal basis of $\mathfrak{L}(\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}})$. They are defined by

$$(\varphi_{l,k}(x))_j = \begin{cases} e^{2\pi \mathtt{i} \langle k, x \rangle} & \text{if } j = l \\ 0 & \text{else .} \end{cases}$$

This orthonormal basis decomposes into subspaces indexed by the wave vectors k as $\mathfrak{L}(\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}) = \bigoplus_{k \in \mathbb{T}^*_{\mathcal{A},\mathcal{B}}} \operatorname{span}\{\varphi_{1,k}, \varphi_{2,k}, \ldots, \varphi_{|\mathfrak{u}|,k}\}$.

The orthogonality of the basis functions reduces to the orthogonality of the underlying scalar exponential functions, since the basis functions differ only in the position of their

nonzero component. Let $\varphi(x) = e^{2\pi \mathrm{i}\langle k,x\rangle}$ and $g(x) = e^{2\pi \mathrm{i}\langle k',x\rangle}$. Using the inner product defined in Equation (2.5), we obtain

$$\langle \varphi(x), g(x)\rangle = \frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} \varphi(x)\overline{g(x)} = \frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k,x\rangle} e^{-2\pi \mathrm{i}\langle k',x\rangle} \tag{2.6}$$

$$= \frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x\rangle}. \tag{2.7}$$

For $k = k'$ we find $e^{2\pi \mathrm{i}\langle k-k',x\rangle} = 1$ in Equation (2.7). For $k \neq k'$ there exists $x' \in \mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ such that $e^{2\pi \mathrm{i}\langle k-k',x'\rangle} \neq 1$. Since $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ is a group, the map $x \mapsto x + x'$ is bijective, and we can rewrite the sum accordingly, which yields

$$\frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x+x'\rangle} = e^{2\pi \mathrm{i}\langle k-k',x'\rangle} \left(\frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x\rangle} \right)$$

$$\iff \frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x+x'\rangle} - e^{2\pi \mathrm{i}\langle k-k',x'\rangle} \left(\frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x\rangle} \right) = 0$$

$$\iff \frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x+x'\rangle} \cdot (1 - \underbrace{e^{2\pi \mathrm{i}\langle k-k',x'\rangle}}_{\neq 1}) = 0$$

Since $e^{2\pi \mathrm{i}\langle k-k',x'\rangle} \neq 1$ it follows that $\frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x+x'\rangle} = 0$. Combining both cases we conclude that

$$\frac{1}{|\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}|} \sum_{x\in\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}} e^{2\pi \mathrm{i}\langle k-k',x\rangle} = \begin{cases} 1, & k = k', \\ 0, & k \neq k'. \end{cases}$$

2.2.3 Fourier Functions on Lattice and Crystal Tori

This section is mainly based on [65].

For function spaces on crystals, the orthonormal wave functions can be identified with *Fourier basis functions*, also referred to as *Fourier modes*. A Fourier function is a complex-valued function that assigns to each position $x \in \mathbb{R}^n$ and a wave vector $k \in \mathbb{R}^n$ the complex exponential determined by their inner product, i.e.,

$$\varphi : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{C} \quad \text{with} \quad (x,k) \longmapsto e^{2\pi \mathrm{i}\langle k,x\rangle}. \tag{2.8}$$

In particular, a variation in x rotates the exponential in the complex plane, whereas larger values of k produce a faster oscillation and hence corresponding to higher frequencies. Furthermore, the Fourier function is multiplicative in both arguments and is

trivial on the integer lattice. Moreover, it lies on the unit circle in the complex plane, since it has a unit-moduls, i.e. $|\varphi(x,k)| = 1$. More formally, Fourier functions satisfy the following properties

(i) $\varphi(x+y, k+\ell) = \varphi(x,k)\cdot\varphi(y,k)\cdot\varphi(x,\ell)\cdot\varphi(y,\ell)$ for all $x, y, k, \ell \in \mathbb{R}^n$

(ii) $\varphi(x,k) = 1$ for all $(x,k) \in \mathbb{R}^n \times \mathbb{R}^n$ with $\langle x, k\rangle \in \mathbb{Z}$

(iii) $|\varphi(x,k)| = 1$ for all $(x,k) \in \mathbb{R}^n \times \mathbb{R}^n$.

These properties underlie the periodicity and orthogonality of Fourier functions on a lattice.

According to the interpretation of k as a wave vector, a Fourier mode can be viewed as a wave function

$$\varphi_k : \mathbb{R}^n \longrightarrow \mathbb{C} \;\text{ with }\; x \longmapsto e^{2\pi i\langle k,x\rangle}.$$

Its value at position x depends on the fixed wave vector k.

Fourier functions on a lattice. The Fourier function can be restricted in the spatial dimension to a lattice $\mathbb{L}(\mathcal{A}) \subseteq \mathbb{R}^n$, which results in

$$\varphi : \mathbb{L}(\mathcal{A}) \times \mathbb{R}^n \longrightarrow \mathbb{C} \quad \text{with} \quad (x,k) \longmapsto e^{2\pi\mathrm{i}\langle k,x\rangle}.$$

For each fixed wave vector k, this defines a wave function

$$\varphi_k = e^{2\pi\mathrm{i}\langle k,x\rangle} \;\text{ with }\; x \in \mathbb{L}(\mathcal{A}).$$

Every Fourier function on $\mathbb{L}(\mathcal{A})$ is uniquely represented by a wave vector $\hat{k}$ contained in the unit cell $\Xi(\mathcal{A}^{-\mathrm{T}})$ of the dual lattice $\mathbb{L}(\mathcal{A})^*$. Accordingly,

$$\varphi : \mathbb{L}(\mathcal{A}) \times \Xi(\mathcal{A}^{-\mathrm{T}}) \longrightarrow \mathbb{C} \quad \text{with} \quad (x,\hat{k}) \longmapsto e^{2\pi\mathrm{i}\langle \hat{k},x\rangle}.$$

Equivalently, any $k \in \mathbb{R}^n$ admits a unique representation

$$k = \hat{k} + \mathcal{A}^{-\mathrm{T}}\zeta, \;\text{ with }\; \zeta \in \mathbb{Z}^n.$$

For $x \in \mathbb{L}(\mathcal{A})$, i.e., $x = \mathcal{A}\mu$ with $\mu \in \mathbb{Z}^n$ one obtains by periodicity:

$$\begin{aligned}\varphi(x,k) =& e^{2\pi\mathrm{i}\langle k,x\rangle} = e^{2\pi\mathrm{i}\langle \hat{k}+\mathcal{A}^{-\mathrm{T}}\zeta,\, \mathcal{A}\mu\rangle}\\ =& e^{2\pi\mathrm{i}\langle \hat{k},\, \mathcal{A}\mu\rangle}\cdot e^{2\pi\mathrm{i}\langle \mathcal{A}^{-\mathrm{T}}\zeta,\, \mathcal{A}\mu\rangle}\\ =& e^{2\pi\mathrm{i}\langle \hat{k},\, \mathcal{A}\mu\rangle}\cdot \underbrace{e^{2\pi\mathrm{i}\langle \zeta,\, \mu\rangle}}_{=1} = \varphi(x,\hat{k}).\end{aligned}$$

Consequently, two Fourier functions defined on $\mathbb{R}^n$ are indistinguishable on $\mathbb{L}(\mathcal{A})$ whenever their wave vectors differ by an element of the dual lattice $\mathbb{L}(\mathcal{A})^*$. This effect, known as *aliasing*, reflects the periodicity induced in the frequency domain by sampling the spatial domain, so that wave vectors are effectively identified modulo $\mathbb{L}(\mathcal{A})^*$.

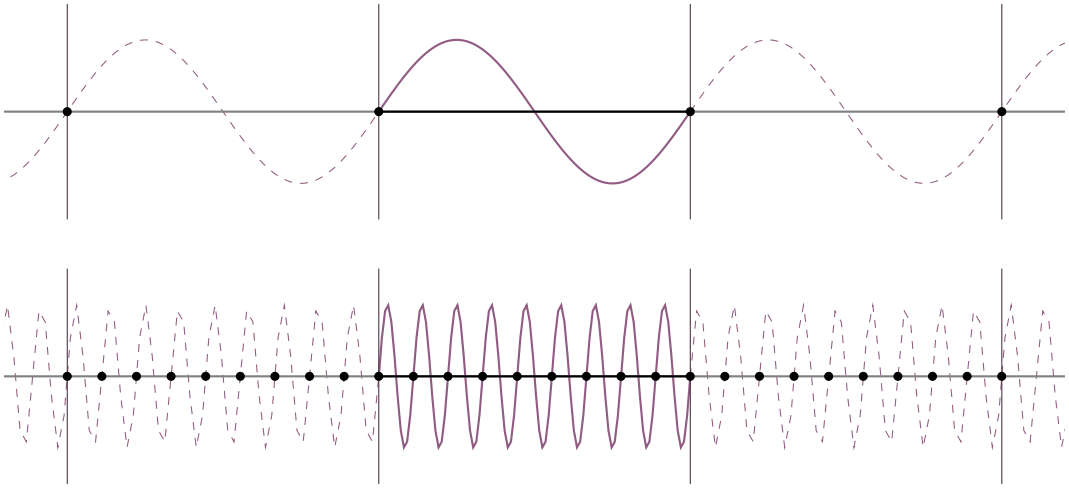


Figure 2.5: Fourier modes on $\mathbb{R}$ with different wave vectors. Unit cell boundaries are indicated by vertical bars, while lattice points are marked by black dots. The Fourier modes within one unit cell are highlighted by solid lines and periodically extended by dashed lines. At the top, the wavelength equals the size of the unit cell, corresponding to the lowest nonzero Fourier mode. At the bottom, multiple wavelengths fit into one unit cell, corresponding to higher-frequency Fourier modes.

Fourier functions on a torus. Let $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A})$ be a lattice-sublattice pair and restrict the spatial domain of the Fourier function to the finite box $\Xi(\mathcal{B})$. This gives

$$\varphi : \Xi(\mathcal{B}) \times \mathbb{R}^n \longrightarrow \mathbb{C} \quad \text{with} \quad (\hat{x}, k) \longmapsto e^{2\pi i \langle k, \hat{x} \rangle}.$$

The box $\Xi(\mathcal{B})$ defines the period for the periodic boundary conditions by

$$\varphi(\hat{x}, k) = \varphi(\hat{x} + \mathcal{B} \cdot \eta, k) \text{ for all } \eta \in \mathbb{Z}^n.$$

Consequently, at least one full oscillation of every admissible wave must fit into the box, providing a lower bound on the nonzero wavelength as depicted in Figure 2.5. By periodicity, every Fourier function restricted to $\Xi(\mathcal{B})$ can be identified by a wave vector k from the dual lattice $\mathbb{L}(\mathcal{B})^*$. Accordingly, the set of Fourier functions on the torus is

$$\varphi : \Xi(\mathcal{B}) \times \mathbb{L}(\mathcal{B}^{-\mathrm{T}}) \longrightarrow \mathbb{C} \quad \text{with} \quad (\hat{x}, k) \longmapsto e^{2\pi \mathrm{i} \langle k, \hat{x} \rangle}.$$

Finally, the partition property of the unit cell (cf. Equation (2.3)) ensures that restricting to $\Xi(\mathcal{B})$ together with periodic boundary conditions faithfully represents functions on the torus $\mathbb{T}_{\mathcal{A},\mathcal{B}}$. Consequently, two Fourier functions on $\mathbb{R}^n$ restrict to the same function on the torus $\mathbb{T}_{\mathcal{A},\mathcal{B}}$ if and only if their wave vectors differ by an element of the dual lattice $\mathbb{L}(\mathcal{B})^*$.

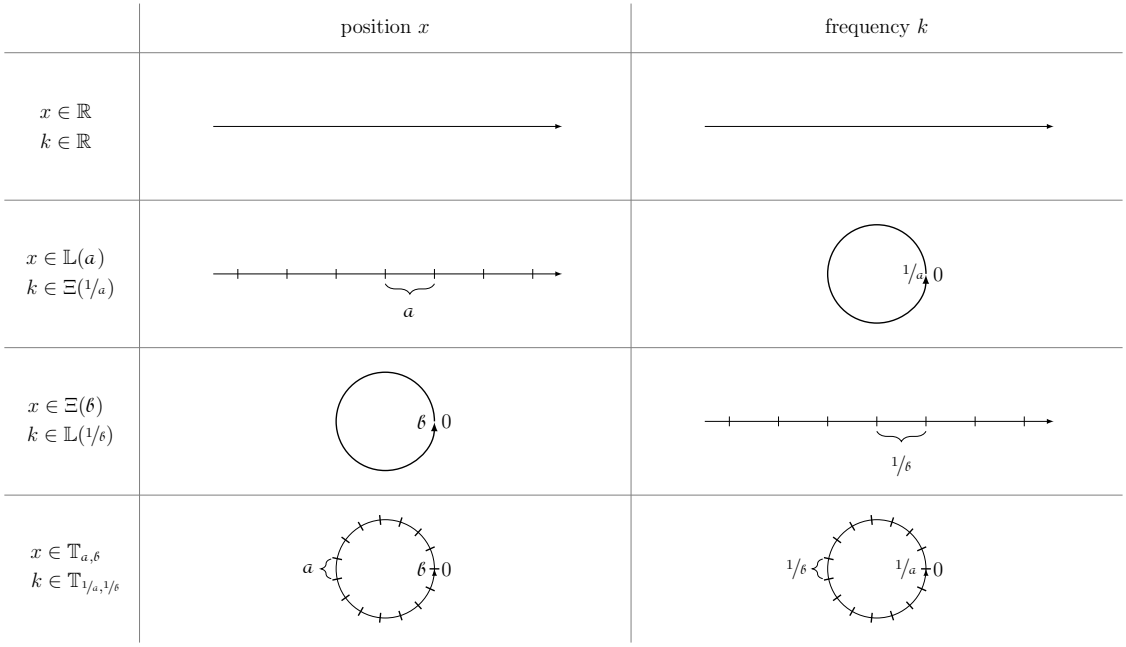


Figure 2.6: Illustration of position and frequency domains under restriction in the one-dimensional case (based on [65]). Consider a one-dimensional lattice-sublattice pair $\mathbb{L}(\mathcal{b}) \subseteq \mathbb{L}(a) \subseteq \mathbb{R}$. Without restricting either variable, i.e., $x, k \in \mathbb{R}$, both domains are continuous. Discretizing the position domain to the lattice $x \in \mathbb{L}(a)$ results in periodicity of the frequency domain with unit cell $\Xi(1/a)$. Conversely, discretizing the frequency domain to the lattice $k \in \mathbb{L}(1/\mathcal{b})$ yields periodicity of the position domain with unit cell $\Xi(\mathcal{b})$. Restricting the position domain to the lattice torus $x \in \mathbb{T}_{\mathcal{A},\mathcal{B}}$ makes the position domain discrete with respect to the lattice $\mathbb{L}(a)$, while imposing periodicity with respect to the sublattice $\mathbb{L}(\mathcal{b})$. Accordingly, the frequency domain becomes discrete on the dual lattice $\mathbb{L}(\frac{1}{a})$ due to the periodicity of the position domain and remains periodic with respect to the dual sublattice $\mathbb{L}(1/\mathcal{b})$.

Fourier functions on a lattice torus. Let $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A}) \subseteq \mathbb{R}^n$ be a lattice-sublattice pair. The corresponding lattice torus is $\mathbb{T}_{\mathcal{A},\mathcal{B}}$ whose elements are equivalence classes

$$[x]_{\mathcal{B}} = \{x + \mathcal{B} \cdot \zeta \,|\, \zeta \in \mathbb{Z}^n\} \quad \text{with} \quad x \in \mathbb{L}(\mathcal{A}).$$

For convenience, we identify each class $[x]_{\mathcal{B}}$ with its canonical representative $\hat{x} \in [x]_{\mathcal{B}} \cap \Xi(\mathcal{B})$. Since the Fourier functions must be well-defined on the torus, their values cannot depend on the particular representative chosen for the equivalence class $[x]_{\mathcal{B}}$. We therefore adopt the convention (with slight abuse of notation) that $[x]_{\mathcal{B}}$ is identified with its canonical representative $\hat{x}$, and use equivalence classes and their representatives interchangeably. This is consistent, since each Fourier mode φ_k is $\mathcal{B}$-periodic and the elements of $\mathbb{T}_{\mathcal{A},\mathcal{B}}$ can be identified by any chosen representative $\hat{x}$ so that $\varphi_k(x + \mathcal{B} \cdot \zeta) = \varphi_k(x)$ with $\zeta \in \mathbb{Z}^n$. Figure 2.6 summarizes the periodicity of Fourier functions restricted in position and/or frequency space. Restricting the frequencies to the dual lattice torus

$\mathbb{T}^*_{\mathcal{A},\mathcal{B}}$ gives rise to the admissible Fourier modes

$$\varphi : \mathbb{T}_{\mathcal{A},\mathcal{B}} \times \mathbb{T}^*_{\mathcal{A},\mathcal{B}} \longrightarrow \mathbb{C} \;\text{ with }\; ([x]_{\mathcal{B}}, [k]_{\mathcal{A}^{-\mathrm{T}}}) \longmapsto e^{2\pi i \langle \hat{k}, \hat{x} \rangle},$$

where $\hat{x} \in [x]_{\mathcal{B}} \cap \Xi(\mathcal{B})$ and $\hat{k} \in [k]_{\mathcal{A}^{-\mathrm{T}}} \cap \Xi(\mathcal{A}^{-\mathrm{T}})$ denote the canonical representatives. The periodicity constraint induced by the sublattice cut-off implies that the minimal wavelength of nonzero modes is bounded by the spacing induced by $\mathbb{T}^*_{\mathcal{A},\mathcal{B}}$. The corresponding Fourier modes are normalized by

$$\widehat{\varphi}_{\hat{k}} : \mathbb{T}_{\mathcal{A},\mathcal{B}} \longrightarrow \mathbb{C} \;\text{ with }\; \hat{x} \longmapsto \frac{1}{\sqrt{m}} \varphi_k(\hat{x}), \quad m = |\mathbb{T}_{\mathcal{A},\mathcal{B}}| = |\mathbb{T}^*_{\mathcal{A},\mathcal{B}}|. \tag{2.9}$$

The normalized Fourier modes form an orthonormal basis of $\mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{B}})$ with respect to $\langle \cdot, \cdot \rangle$. Accordingly, the Fourier transform on the lattice torus is represented by the unitary Fourier matrix

$$\mathbf{F} = \begin{bmatrix} \widehat{\varphi}_{\hat{k}_1} & \widehat{\varphi}_{\hat{k}_2} & \dots & \widehat{\varphi}_{\hat{k}_m} \end{bmatrix} \in \mathbb{C}^{m \times m}.$$

Fourier functions on a crystal torus. Let $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A}) \subseteq \mathbb{R}^n$, where $\mathcal{B} = \mathcal{A} \cdot Z$ for some $Z \in \mathbb{Z}^{n \times n}$, and let $m = |\det(Z)|$. The corresponding crystal torus is $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}}$ with structure element $\mathfrak{u} = [\mathfrak{u}_1, \mathfrak{u}_2, \dots, \mathfrak{u}_\nu] \in \Xi(\mathcal{A})^\nu$. Recall that a crystal torus can be viewed as a collection of ν lattice tori $\mathbb{T}_{\mathcal{A},\mathcal{B}}$ each shifted by one of the structure elements $\mathfrak{u}_\ell \in \Xi(\mathcal{A})$ for $\ell = 1, 2, \dots, \nu$. The Fourier functions on the shifted components $\mathbb{T}^{\mathfrak{u}_\ell}_{\mathcal{A},\mathcal{B}}$ are defined as independent copies of lattice Fourier modes:

$$\varphi^{\mathfrak{u}_\ell}_{\hat{k}} : \mathbb{T}_{\mathcal{A},\mathcal{B}} \longrightarrow \mathbb{C} \;\text{ with }\; \hat{x} \longmapsto e^{2\pi i \langle \hat{k}, \hat{x} + \mathfrak{u}_\ell \rangle} \cdot e_\ell \text{ for } \ell = 1, 2, \dots, \nu \tag{2.10}$$

where $\hat{x} \in \mathbb{T}_{\mathcal{A},\mathcal{B}}$, $\hat{k} \in \mathbb{T}^*_{\mathcal{A},\mathcal{B}}$ and e_ℓ is the ℓ-th unit vector in $\mathbb{C}^\nu$.

The relative positioning of the structure element $\mathfrak{u}_\ell$ within the unit cell $\Xi(\mathcal{A})$ manifests as a *phase adjustment factor*. Indeed, each Fourier function can be decomposed into a lattice mode $\varphi_{\hat{k}}$ and a crystal dependent phase shift, i.e.,

$$\varphi^{\mathfrak{u}_\ell}_{\hat{k}}(\hat{x}) = \underbrace{e^{2\pi \mathrm{i} \langle \hat{k}, \hat{x} \rangle}}_{\varphi_{\hat{k}}} \cdot \underbrace{e^{2\pi \mathrm{i} \langle \hat{k}, \mathfrak{u}_\ell \rangle_2}}_{\text{phase adjustment}} \cdot e_\ell \quad \text{for} \quad \ell = 1, \dots, \nu. \tag{2.11}$$

Collecting these Fourier functions yields the complex $\nu \times \nu$ matrix

$$F^{\mathfrak{u}}_{\hat{k}} = \left[\varphi^{\mathfrak{u}_1}_{\hat{k}}, \varphi^{\mathfrak{u}_2}_{\hat{k}}, \dots, \varphi^{\mathfrak{u}_\nu}_{\hat{k}} \right] = \mathrm{diag}(e^{2\pi i \langle \hat{k}, \mathfrak{u}_1 \rangle_2}, e^{2\pi i \langle \hat{k}, \mathfrak{u}_2 \rangle_2}, \dots, e^{2\pi i \langle \hat{k}, \mathfrak{u}_\nu \rangle_2}) \cdot \varphi_{\hat{k}}(\hat{x}).$$

The orthonormal basis of $\mathfrak{L}(\mathbb{T}^{\mathfrak{u}_\ell}_{\mathcal{A},\mathcal{B}})$ is given by the set of scaled wave functions

$$\widehat{\varphi}^{\mathfrak{u}_\ell}_{\hat{k}} = \frac{1}{\sqrt{m}} \cdot \varphi^{\mathfrak{u}_\ell}_{\hat{k}} \text{ for } \ell \in \{1, 2, \dots, \nu\}, \quad \hat{k} \in \mathbb{T}^*_{\mathcal{A},\mathcal{B}}.$$

For any ordering $\hat{k}_1, \hat{k}_2, \ldots, \hat{k}_m$ of the m canonical representatives in $\mathbb{T}^*_{\mathcal{A},\mathcal{B}}$, the wave functions can be expressed as the product of the regular Fourier functions in $\mathbb{T}_{\mathcal{A},\mathcal{B}}$ and the corresponding phase adjustment factor $e^{2\pi \mathrm{i}\langle \hat{k}, \mathrm{u}_\ell \rangle}$. Collecting these functions for all $\hat{k} \in \mathbb{T}^*_{\mathcal{A},\mathcal{B}}$ yields a block-structured matrix $\mathbf{F}^{\mathrm{u}} \in \mathbb{C}^{\nu m \times \nu m}$.

In the following, we focus on translation-invariant operators and therefore omit the distinction between lattice element x and its canonical representative $\hat{x}$ in the quotient space. The same convention applies for the dual lattice torus, i.e., we identify $\hat{k}$ with k.

2.3 Discrete Fourier Transforms

Fourier functions on lattice tori encode spatial periodicity together with internal phase relations induced by the crystal structure. So far, we have constructed them as a natural orthonormal basis for functions defined on a finite periodic lattice with internal structure, i.e., a crystal torus. To make this basis functional for spectral analysis, we require a transform that represents arbitrary functions in terms of their Fourier coefficients.

The *discrete Fourier transform* (DFT) provides precisely this mechanism. It expresses a function on the crystal torus as a linear combination of discrete Fourier basis functions and thereby establishes an explicit mapping between position space and frequency space.

Built on this, the Fast Fourier Transform exploits the underlying lattice hierarchy to compute Fourier coefficients with reduced computational complexity. We introduce the FFT for crystal structures in Section 2.3.1.

At the same time, the DFT reveals structural simplifications of operators. In particular, multiplication operators become diagonal or, in the presence of structure elements, block-diagonal when expressed in the Fourier basis. This property is central to local Fourier analysis, where applying the DFT to these operators yields so-called symbols. We study LFA in Section 2.3.2. Throughout this section, we follow [65, 66, 69].

The discrete Fourier transform on lattice and crystal tori. Consider the finite-dimensional vector space $\mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{B}}) \cong \mathbb{C}^{|\det(Z)|}$ associated with the lattice-sublattice pair $\mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A}) \subseteq \mathbb{R}^n$, where $\mathcal{B} = \mathcal{A} \cdot Z$ for some $Z \in \mathbb{Z}^{n \times n}$. The number of elements within a torus $\mathbb{T}_{\mathcal{A},\mathcal{B}}$ is denoted by $m = |\mathbb{T}_{\mathcal{A},\mathcal{B}}| = |\det(Z)|$. The DFT on $\mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{B}})$ is a bijective linear mapping

$$\mathcal{F} : \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{B}}) \longrightarrow \mathfrak{L}(\mathbb{T}^*_{\mathcal{A},\mathcal{B}})$$

defined for $g \in \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{B}})$ by

$$\hat{g}_k = \mathcal{F}(g)(k) = \langle g, \varphi_k \rangle = \frac{1}{\sqrt{|\mathbb{T}_{\mathcal{A},\mathcal{B}}|}} \sum_{x \in \mathbb{T}_{\mathcal{A},\mathcal{B}}} g(x) e^{-2\pi \mathrm{i}\langle k, x \rangle},$$

where $k \in \mathbb{T}^*_{\mathcal{A},\mathcal{B}}$ and φ_k is the associated Fourier basis function, cf. Equation (2.9). Collecting the orthonormal Fourier basis functions as columns yields the unitary matrix

$$\mathbf{F} = \begin{bmatrix} \varphi_{k_1} & \varphi_{k_2} & \cdots & \varphi_{k_m} \end{bmatrix} \in \mathbb{C}^{m \times m}.$$

For completeness, note that the *inverse Fourier transform* is given by

$$\mathcal{F}^{-1}(\hat{g}) = \mathbf{F}^* \cdot \hat{g} \quad \text{for} \quad \hat{g} \in \mathfrak{L}(\mathbb{T}^*_{\mathcal{A},\mathcal{B}}),$$

so that forward and inverse DFT differ only in the sign of the complex exponential appearing in the basis functions.

For vector-valued functions $g = (g_1, \ldots, g_{|\mathfrak{u}|}) \in \mathfrak{L}(\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{B}})$, the DFT extends naturally by acting componentwise. Each component satisfies $g_l \in \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{B}})$, and

$$\mathcal{F}g := (\mathcal{F}g_1, \ldots, \mathcal{F}g_{|\mathfrak{u}|}).$$

In particular, the $|\mathfrak{u}|$ shifted copies of the lattice torus associated with the structure element $\mathfrak{u} = [\mathfrak{u}_1, \mathfrak{u}_2, \ldots, \mathfrak{u}_{|\mathfrak{u}|}]$ give rise to the diagonal phase adjustment, cf. Equation (2.11).

2.3.1 Fast Discrete Fourier Transforms

The DFT is commonly implemented via FFT. By employing a divide-and-conquer strategy the FFT reduces the computational complexity from $\mathcal{O}(m^2)$ to $\mathcal{O}(m \log(m))$ by recursively decomposing the transform into a sequence of smaller DFTs, which can then be combined efficiently to yield the full result [43]. On lattice tori, the presence of intermediate sublattices provides a natural framework for such factorization. In particular, let $\mathbb{L}(\mathcal{Z}) \subseteq \mathbb{L}(\mathcal{B}) \subseteq \mathbb{L}(\mathcal{A}) \subseteq \mathbb{R}^n$ be a hierarchy of lattices with corresponding dual lattices $\mathbb{L}(\mathcal{A}^{-\mathrm{T}}) \subseteq \mathbb{L}(\mathcal{B}^{-\mathrm{T}}) \subseteq \mathbb{L}(\mathcal{Z}^{-\mathrm{T}}) \subseteq \mathbb{R}^n$. The lattice $\mathbb{L}(\mathcal{Z})$ defines the domain cut-off, i.e., the underlying problem is $\mathcal{Z}$-periodic with respect to $\mathbb{L}(\mathcal{Z})$. For $\mathbb{L}(\mathcal{Z}) \subseteq \mathbb{L}(\mathcal{A})$ is $\mathcal{Z} = \mathcal{A} \cdot Z$ with $Z \in \mathbb{Z}^{n \times n}$ and the corresponding lattice torus is

$$\mathbb{T}_{\mathcal{A},\mathcal{Z}} = \mathbb{L}(\mathcal{A}) \Big/ \mathbb{L}(\mathcal{Z}) \;\cong\; \mathbb{L}(\mathcal{B})^{\mathbb{L}(\mathcal{A})/\mathbb{L}(\mathcal{B})} \Big/ \mathbb{L}(\mathcal{Z}). \tag{2.12}$$

That is, the intermediate lattice $\mathbb{L}(\mathcal{B})$ partitions the torus $\mathbb{T}_{\mathcal{A},\mathcal{Z}}$ into disjoint blocks of size $|\mathbb{T}_{\mathcal{A},\mathcal{B}}|$. Every spatial representative $x \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}$ decomposes into an in-block representative $y \in {}^{\mathbb{L}(\mathcal{B})}/_{\mathbb{L}(\mathcal{Z})}$ and an across-block component $z \in {}^{\mathbb{L}(\mathcal{A})}/_{\mathbb{L}(\mathcal{B})}$ such that $x = y + z$. Similarly, each frequency representative $k \in {}^{\mathbb{L}(\mathcal{Z}^{-\mathrm{T}})}/_{\mathbb{L}(\mathcal{A}^{-\mathrm{T}})}$ can be decomposed into an across frequency $\ell \in {}^{\mathbb{L}(\mathcal{B}^{-\mathrm{T}})}/_{\mathbb{L}(\mathcal{A}^{-\mathrm{T}})}$ and an in-block frequency $a \in {}^{\mathbb{L}(\mathcal{Z}^{-\mathrm{T}})}/_{\mathbb{L}(\mathcal{B}^{-\mathrm{T}})}$, such that $k = \ell + a$. In this way, the lattice hierarchy itself encodes the in–block and across–block components that drive the factorization of the Fourier transform. With this decomposition, the Fourier modes factorize as

$$\varphi(x,k) = e^{2\pi \mathrm{i}\langle k,x\rangle} = e^{2\pi \mathrm{i}\langle \ell+a, y+z\rangle} = \varphi(y,\ell) \cdot \varphi(y,a) \cdot \varphi(z,\ell) \cdot \varphi(z,a).$$

For $y = \mathcal{B} \cdot \mu$ and $\ell = \mathcal{B}^{-\mathrm{T}} \cdot \nu$ with $\mu, \nu \in \mathbb{Z}^n$ and $\nu^{\mathrm{T}}\mu \in \mathbb{Z}$ we obtain

$$\varphi(y, \ell) = e^{2\pi \mathrm{i}\langle \ell, y\rangle} = e^{2\pi \mathrm{i}\langle \mathcal{B}\cdot\mu, \mathcal{B}^{-\mathrm{T}}\cdot\nu\rangle} = e^{2\pi \mathrm{i}\langle \mu, \nu\rangle} = e^{2\pi \mathrm{i}\nu^{\mathrm{T}}\mu} = 1.$$

Furthermore, this decomposition consists of FFTs on $\mathbb{L}(\mathcal{B}) \,/\, \mathbb{L}(\mathcal{Z})$ given by $\varphi(y, a)$ (in-block transforms) and $\mathbb{L}(\mathcal{A})/\mathbb{L}(\mathcal{B})$, given by $\varphi(z, \ell)$ (across-block transforms), followed by phase adjustments $\varphi(z, a)$. These *twiddle factors* [37] reflect that the transform on $\mathbb{T}_{\mathcal{B},\mathcal{Z}}$ has degrees of freedom spread across the lattice rather than confined to a unit cell.

2.3.2 Local Fourier Analysis

We now turn to the application of the DFT to multiplication operators on crystal tori. Such operators are translationally invariant with respect to the underlying lattice and can be interpreted as convolution-like operators. Expressed in the Fourier basis, they diagonalize or, in the presence of a structure element, block-diagonalize, which makes their spectral properties directly accessible. Note this fundamental property also underlies the application of the FFT to multiplication operators. However, LFA exploits this property more directly by working with Fourier symbols, which are inherently free of phase adjustments.

Lattice and crystal operators. Let $\mathbb{L}(\mathcal{Z}) \subseteq \mathbb{L}(\mathcal{A}) \subseteq \mathbb{R}^n$ be a lattice-sublattice pair such that $\mathcal{Z} = \mathcal{A} \cdot Z$ with $Z \in \mathbb{Z}^{n\times n}$. A lattice operator is a linear mapping

$$A : \mathfrak{L}_(\mathbb{T}_{\mathcal{A},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{Z}}),$$

acting on scalar functions in the function space $\mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{Z}})$, cf. Equation (2.4).

The crystal torus $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{Z}}$ with structure element $\mathfrak{u} = [\mathfrak{u}_1, \ldots, \mathfrak{u}_{|\mathfrak{u}|}]$ represents a finite periodic structure with $|\mathfrak{u}|$ degrees of freedom per lattice site. A crystal function is then a vector-valued function $f = (f_1, \ldots, f_{|\mathfrak{u}|})$, with $f_j \in \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{Z}})$.

A crystal operator maps vector-valued functions to vector-valued functions. More generally, for domain a torus $\mathbb{T}^{\mathfrak{d}}_{\mathcal{A},\mathcal{Z}}$ and a co-domain torus $\mathbb{T}^{\mathfrak{c}}_{\mathcal{A},\mathcal{Z}}$, we define

$$A = (A_1, \ldots, A_{|\mathfrak{c}|}) : \mathfrak{L}(\mathbb{T}^{\mathfrak{d}}_{\mathcal{A},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}^{\mathfrak{c}}_{\mathcal{A},\mathcal{Z}}),$$

where each component operator is a linear map

$$A_j : \mathfrak{L}(\mathbb{T}^{\mathfrak{d}}_{\mathcal{A},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}^{\mathfrak{c}_j}_{\mathcal{A},\mathcal{Z}}), \quad f \longmapsto A_j(f) = A_j(f_1, f_2, \ldots, f_{|\mathfrak{d}|}), \quad j = 1, \ldots, |\mathfrak{c}|.$$

In particular, each output component of the crystal function in the co-domain is obtained by applying the operator A_j to all components $f_1, \ldots, f_{|\mathfrak{d}|}$ of the domain function.

Multiplication operators. Let $A : \mathfrak{L}(\mathbb{T}^{\mathfrak{d}}_{\mathcal{A},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}^{\mathfrak{c}}_{\mathcal{A},\mathcal{Z}})$ be a crystal operator. We call A a multiplication operator if there exist multiplication matrices $M_\delta \in \mathbb{C}^{|\mathfrak{c}|\times|\mathfrak{d}|}$ with $\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}$ such that for each $x \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}$ and $f = (f_1, \ldots, f_{|\mathfrak{d}|})^{\mathrm{T}} \in \mathfrak{L}(\mathbb{T}^{\mathfrak{d}}_{\mathcal{A},\mathcal{Z}})$ we find

$$(Af)(x) = \sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta \cdot f(x+\delta). \tag{2.13}$$

Operators of this type are convolution-like (stencil operators[9]) and are characterized by the translation invariance with respect to the underlying lattice. For each primitive vector $\boldsymbol{a} \in \mathbb{L}(\mathcal{A})$ the translation operator is defined as

$$T_{\boldsymbol{a}} : \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{Z}}) \text{ , so that } T_{\boldsymbol{a}} f(x) = f(x+\boldsymbol{a}).$$

Since any lattice vector $\boldsymbol{a} = \sum_{\ell=1}^{n} j_\ell \boldsymbol{a}_\ell$ with $j_\ell \in \mathbb{Z}$ is a linear combination of primitive vectors, it follows that $T_{\boldsymbol{a}}$ can be expressed as composition of translations by the primitive vectors, i.e., $T_{\boldsymbol{a}} = T^{j_1}_{\boldsymbol{a}_1} \circ T^{j_2}_{\boldsymbol{a}_2} \circ \cdots \circ T^{j_n}_{\boldsymbol{a}_n}$. A multiplication operator A is translation-invariant with respect to $\mathcal{A}$ if A commutes with every translation by a primitive vector, i.e., $AT_{\boldsymbol{a}} - T_{\boldsymbol{a}} A = 0$ for all primitive vectors $\boldsymbol{a} \in \mathbb{L}(\mathcal{A})$.

Indeed, let A be such an operator with multiplication matrices $M_\delta \in \mathbb{C}^{|\mathfrak{c}|\times|\mathfrak{d}|}$. Then for any $f \in \mathfrak{L}(\mathbb{T}^{\mathfrak{d}}_{\mathcal{A},\mathcal{Z}})$ and $x \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}$ we apply

$$(T_{\boldsymbol{a}} A f)(x) = (Af)(x+\boldsymbol{a}) = \sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta \cdot f(x+\boldsymbol{a}+\delta)$$

and

$$(AT_{\boldsymbol{a}} f)(x) = \sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta (T_{\boldsymbol{a}} f)(x+\delta) = \sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta f(x+\boldsymbol{a}+\delta).$$

Consequently, $T_{\boldsymbol{a}} A = AT_{\boldsymbol{a}}$ for all $\boldsymbol{a} \in \mathbb{L}(\mathcal{A})$.

Convolutional theorem. An $\mathcal{A}$-translationally invariant operator as defined in Equation (2.13) can be interpreted as a *discrete cross-correlation*, a structure frequently used in machine learning applications in place of classical convolution. Equivalently, it acts via discrete *convolution*

$$(A * f)(x) = \sum_{\delta \in \mathbb{T}(\mathcal{A},\mathcal{Z})} M_\delta \cdot f(x-\delta),$$

with convolutional kernel $M_\delta \in \mathbb{C}^{|\mathfrak{c}|\times|\mathfrak{d}|}$, indexed by the lattice displacements $\delta \in \mathbb{L}(\mathcal{A})$. The role of convolutions and cross-correlation in artificial neural networks is discussed in detail in Section 3.2.4.

[9] Stencil operators compute each output value as a weighted sum of neighboring input values, using a fixed pattern that is typically applied uniformly across the domain.

In the crystal case, let φ_k be a vector-valued function associated with a fixed wave vector

$$\varphi_k(x) = e^{2\pi \mathrm{i}\langle k,x\rangle} \cdot v \quad \text{with} \quad v \in \mathbb{C}^{|\mathfrak{d}|} \quad \text{and } k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}}$$

that is, φ_k is a linear combination of scalar Fourier modes associated with wave vector k. Applying the multiplication operator A yields

$$(A\varphi_k)(x) = \sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta \cdot \varphi_k(x+\delta) = \left(\sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta \cdot e^{2\pi \mathrm{i}\langle k,\delta\rangle} \right) \underbrace{e^{2\pi \mathrm{i}\langle k,x\rangle} \cdot v}_{\varphi_k(x)}.$$

We thus define the matrix-valued *symbol* of A as

$$A_k = \sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta \cdot e^{2\pi \mathrm{i}\langle k,\delta\rangle} \in \mathbb{C}^{|\mathfrak{c}|\times|\mathfrak{d}|}.$$

Hence, for each wave vector $k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}}$, the subspace

$$\varepsilon_k = \{e^{2\pi \mathrm{i}\langle k,x\rangle} \cdot v | v \in \mathbb{C}^{|\mathfrak{d}|}\}$$

is invariant under A and the operator acts on this subspace by multiplication by A_k. At each wave vector k, the operator acts as

$$(A\varphi_k)(x) = \left(\sum_{\delta \in \mathbb{T}_{\mathcal{A},\mathcal{Z}}} M_\delta \cdot e^{2\pi \mathrm{i}\langle k,\delta\rangle} \right) \varphi_k(x) = A_k \varphi_k(x).$$

According to Equation (2.10), for each frequency k the sets $\mathbf{F}^{\mathfrak{d}}_k = \left[\varphi_k^{\mathfrak{d}_1} \varphi_k^{\mathfrak{d}_2} \dots \varphi_k^{\mathfrak{d}_{|\mathfrak{d}|}}\right]$ and $\mathbf{F}^{\mathfrak{c}}_k = \left[\varphi_k^{\mathfrak{c}_1} \varphi_k^{\mathfrak{c}_2} \dots \varphi_k^{\mathfrak{c}_{|\mathfrak{c}|}}\right]$ form an orthonormal basis of all Fourier modes associated with the domain and co-domain respectively. With this notation, the action of the operator at frequency k can be written as

$$A\mathbf{F}^{\mathfrak{d}}_k = \mathbf{F}^{\mathfrak{c}}_k A_k. \tag{2.14}$$

Since each k indexes a different diagonal block, we say that A is (block-)diagonalized by the DFT. For the lattice case, i.e., $\mathfrak{d} = \mathfrak{c} = 1$, the multiplication operator corresponds to a pointwise multiplication in the frequency domain and Equation (2.14) reduces to

$$A\varphi_k = A_k \varphi_k$$

and since each Fourier mode is already an eigenfunction, the scalar-valued symbol $A_k \in \mathbb{C}$ is an eigenvalue,cf. Equation (2.1).

The spectrum of A on the torus can be given by the union of the spectra of its symbols

$$\operatorname{spec}_{\mathcal{Z}}(A) = \bigcup_{k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}}} \operatorname{spec}(A_k).$$

2.4 Iterative Methods

In this section, we provide an overview of classical iterative methods for solving sparse linear systems arising from discretizing partial differential equations (PDE). We begin by reviewing boundary value problems to motivate the approximation of a continuous problem by a system of equations. This is followed by an introduction to classical iterative methods with a particular focus on the Richardson iteration. This section is mainly based on [96, 105].

2.4.1 Boundary Value Problems

The following discussion is mainly based on [10] and [96]. Boundary value problems are fundamental in the mathematical modelling of physical phenomena, including heat distribution, fluid dynamics, and electrostatics. To motivate the application of iterative methods, we first consider the following two-dimensional boundary value problem as a foundational example [10]. Consider the unit square domain $\Omega = (0,1)^2$ together with the boundary value problem[10]

$$-u_{x,x} - u_{y,y} = f(x,y) \quad \text{for} \quad (x,y) \in \Omega$$

with Dirichlet boundary condition on the boundary $\partial\Omega$

$$u(x,y) = 0 \quad \text{for} \quad (x,y) \in \partial\Omega.$$

To discretize the domain Ω, each spatial dimension, x and y, is partitioned into N equally sized subintervals. This yields a uniform, equidistant grid with mesh size $h_x = h_y = \frac{1}{N}$ and $M = (N-1)^2$ interior grid points (unknowns) defined as

$$(x_i, y_j) = (ih_x, jh_y), \qquad i,j = 1,2,\ldots,N-1.$$

The resulting computational grid is denoted by Ω_h and illustrated in Figure 2.7.

Finite-difference discretization. Applying the finite difference method[11], at each grid point the differential equation is approximated by the second-order difference scheme

$$\frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h_x^2} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{h_y^2} = f_{i,j}, \quad f_{i,j} = f(ih_x, jh_y) \tag{2.15}$$

for $i,j = 1,\ldots,N-1$. The homogeneous boundary condition reads $u_{i,0} = u_{i,N} = u_{0,j} = u_{N,j} = 0$. Stacking the unknowns in *lexicographic order* (first i, then j) yields

$$u = (u_{1,1},\ldots,u_{1,N-1},u_{2,1},\ldots,u_{2,N-1},\ldots,u_{N-1,1},\ldots,u_{N-1,N-1})^{\mathrm{T}}$$

[10] This boundary value problem is known as the Poisson equation on the unit square.
[11] The derivatives in the PDE are replaced by difference approximations [10].

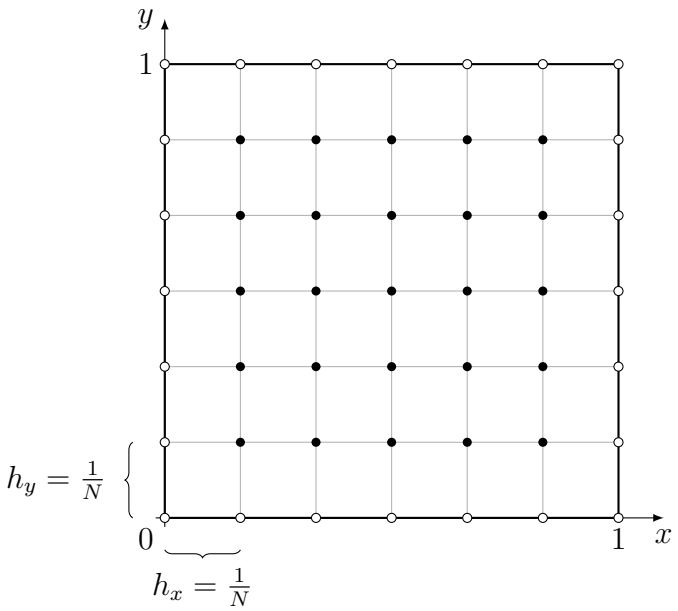


Figure 2.7: Uniform discretization of the unit square $\Omega = (0,1)^2$ into N subintervals in each coordinate direction, resulting in the mesh size $h_x = h_y = 1/N$. Interior grid points are shown as black dots, while boundary grid points are shown as white dots.

and likewise f, Equation (2.15) can be represented as a linear system

$$Au = f \tag{2.16}$$

where, with $h := h_x = h_y$, A is the block-tridiagonal matrix

$$A = \frac{1}{h^2}\begin{pmatrix} B & -I & & & \\ -I & B & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & B & -I \\ & & & -I & B \end{pmatrix} \text{ with } B = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}.$$

Since only neighboring grid points interact, A is often represented as a 5-point stencil

$$A = \frac{1}{h^2}\begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix}. \tag{2.17}$$

This stencil compactly encodes the coupling between a grid point and its four direct neighbors in the two-dimensional case.

Discretized operators on the lattice torus. The discretized operator can be formulated on a lattice by neglecting boundary conditions, since placing the problem on a

lattice torus imposes periodic boundary conditions. The local structure of the operator "in the bulk" remains entirely unaffected by this modification.

In particular, the value at each grid point on the lattice torus is obtained as a fixed linear combination of its neighboring values, independent of its global position. Consequently, the discretized operator is a convolution operator. This viewpoint highlights the translation-invariant structure of the operator, which extends naturally to higher-dimensional lattices.

A discretized unit square with periodic boundary conditions can be represented by the lattice torus $\mathbb{T}_{\mathcal{A},\mathcal{Z}}$ with lattice bases

$$\mathcal{Z} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathcal{A} = \frac{1}{h}\begin{bmatrix} a_1 & a_2 \end{bmatrix} = \frac{1}{h}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

On this lattice torus we define the discretized operator $A_h : \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{Z}})$ by

$$(A_h f)(x) := \sum_{\delta \in \mathcal{N}} M_A^{(\delta)} f(x+\delta), \quad x \in \mathbb{T}_{\mathcal{A},\mathcal{Z}},$$

where $\mathcal{N} = \{0, \pm a_1, \pm a_2\}$ denotes the set of lattice shifts with the five-point stencil. Explicitly, $M_A^{(0)} = 4$ and $M_A^{(a_1)} = M_A^{(-a_1)} = M_A^{(a_2)} = M_A^{(-a_2)} = -1$.

The representation of the discretized operator as a linear combination of shift operators highlights its local structure, since each grid point depends only on its nearest neighbors. While this locality leads to sparse system matrices, the direct solution of the resulting linear system, Equation (2.16), remains computationally expensive for large problem sizes. For example, Gaussian elimination requires $\mathcal{O}(M^3)$ arithmetic operations. In addition, for large systems, i.e., $M > 10^4$ memory requirements quickly become prohibitive [105]. This motivates the use of *iterative methods*. Instead of solving the problem in one finite sequence of operations, iterative methods exploit the local stencil structure to generate successive approximations that ideally converge to the true solution.

2.4.2 Classical Iterative Methods

In this section, we introduce classical iterative methods, following [96], for solving large sparse systems of linear equations $Au = f$, cf. Equation (2.16), where $A \in \mathbb{C}^{n\times n}$ is the system matrix and $f \in \mathbb{C}^n$ denotes the right-hand side and u is the unique solution. Given an initial guess $u^{(0)} \in \mathbb{C}^n$, an iterative method produces a sequence of iterates $u^{(1)}, u^{(2)}, \ldots$ which ideally converges to the exact solution u. A *stationary linear iterative method* is defined by the iteration rule

$$u^{(\tau+1)} = u^{(\tau)} + B(f - Au^{(\tau)}),$$

where $B \in \mathbb{C}^{n \times n}$ typically serves as an approximation to A^{-1}. The method is called stationary since the update rule does not depend on the iteration index τ. In the τ-th iteration, the *iteration error* is defined by

$$e^{(\tau)} = u - u^{(\tau)}.$$

That is, the iteration converges to u, i.e., $u^{(\tau)} \to u$ if and only if $e^{(\tau)} \to 0$. However, both the exact solution u and the true error $e^{(\tau)}$ are generally inaccessible. Instead, one monitors the *residual*

$$r^{(\tau)} = f - Au^{(\tau)},$$

as a computational measure of how well the current iterate $u^{(\tau)}$ approximates the exact solution. Residual and error are related through the *residual equation*

$$Ae^{(\tau)} = r^{(\tau)}. \tag{2.18}$$

Thus, given an iterate $u^{(\tau)}$, $\tau \geq 0$, the residual $r^{(\tau)}$ can be computed and solving the residual equation for $e^{(\tau)}$ would yield the exact correction

$$Ae^{(\tau)} = r^{(\tau)} \Leftrightarrow e^{(\tau)} = A^{-1}r^{(\tau)}.$$

Since this is typically unfeasible, one instead computes an approximate error $\hat{e}^{(\tau)} = Br^{(\tau)} \approx e^{(\tau)}$, which then is used to improve the iterate

$$u^{(\tau+1)} = u^{(\tau)} + Br^{(\tau)} = u^{(\tau)} + \hat{e}^{(\tau)}. \tag{2.19}$$

Most stationary iterative methods are based on this *residual correction* principle.

Convergence of iterative methods. The residual at iteration τ can be written as

$$r^{(\tau+1)} = (I - AB)r^{(\tau)} = (I - AB)^{\tau}r^{(0)}.$$

Since the error satisfies Equation (2.18), we equivalently obtain the *error propagator*

$$e^{(\tau)} = (I - BA)^{\tau}e^{(0)}.$$

An iterative method converges to a solution u for any initial guess if and only if

$$\lim_{\tau \to \infty} e^{(\tau)} = 0 \Longleftrightarrow \lim_{\tau \to \infty} (I - BA)^{\tau}e^{(0)} = 0 \Longleftrightarrow \lim_{\tau \to \infty} (I - BA)^{\tau} = 0. \tag{2.20}$$

Thus, convergence is entirely determined by the powers of the iteration matrix $I - BA$. Furthermore, the iteration matrix $I - BA$ can be viewed as a degree-one matrix polynomial in BA[12]. More generally, after τ iterations the error satisfies

$$e^{(\tau)} = (I - BA)^{\tau}e^{(0)} = p_{\tau}(BA)e^{(0)}, \tag{2.21}$$

[12] An entirely analogous argument applies to the residual with the iteration matrix $I - AB$.

where p_τ is a polynomial of degree τ with $p_\tau(0) = 1$. We refer to p_τ as the *error polynomial*. Let $\lambda_j \in \text{spec}(BA)$ be an eigenvalue of BA with corresponding eigenvector γ_j, i.e., $BA\gamma_j = \lambda_j\gamma_j$. Then

$$p_\tau(\lambda_j) = (1 - \lambda_j)^\tau \quad \text{and} \quad (I - BA)^\tau\gamma_j = p_\tau(\lambda_j)\gamma_j. \tag{2.22}$$

Hence, the convergence behavior of the iteration is determined by the values of the error polynomial at the eigenvalues of BA. Moreover, factorizing the error polynomial yields

$$p_\tau(BA) = \prod_{i=1}^{\tau}(I - \frac{1}{\zeta_i}BA) \quad \text{with} \quad p_\tau(\zeta_i) = 0, \qquad \zeta_i \in \mathbb{C}.$$

Using Equation (2.22), we obtain

$$p_\tau(BA)\gamma_j = p_\tau(\lambda_j)\gamma_j = \left(\prod_{i=1}^{\tau}(1 - \frac{\lambda_j}{\zeta_i})\right)\gamma_j.$$

Writing the initial error as a linear combination of the eigenvectors yields

$$e^{(0)} = \sum_j c_j\gamma_j, \quad \text{with} \quad c_j \in \mathbb{C}.$$

Hence, the error after τ iterations is

$$e^{(\tau)} = \sum_j \left(\prod_{i=1}^{\tau}\left(1 - \frac{\lambda_j}{\zeta_i}\right)\right) c_j\gamma_j,$$

which is small, if the roots ζ_i reflect the distribution of the eigenvalues of BA correctly. The error decay in each direction γ_j is controlled by the scalar factor

$$p_\tau(\lambda_j) = \prod_{i=}^{\tau}\left(1 - \frac{\lambda_j}{\zeta_i}\right).$$

Thus, convergence is guaranteed if

$$\max_{\lambda \in \text{spec}(BA)} |p_\tau(\lambda)| < 1,$$

and the *convergence rate* is dictated by how small this maximum can be made. More fundamentally, recall from Equation (2.20) that convergence requires $\lim_{\tau\to\infty} e^{(\tau)} = 0$. Thus, all eigenvalues λ_j of BA must lie strictly inside the open unit disk centered at 1. Equivalently the spectral radius of the iteration matrix must satisfy

$$\rho(I - BA) < 1.$$

Figure 2.8 displays error polynomials of different degrees τ, illustrating that for eigenvalues inside the unit disk the error components decay, while for eigenvalues outside the unit disk, the error components grow without bound.

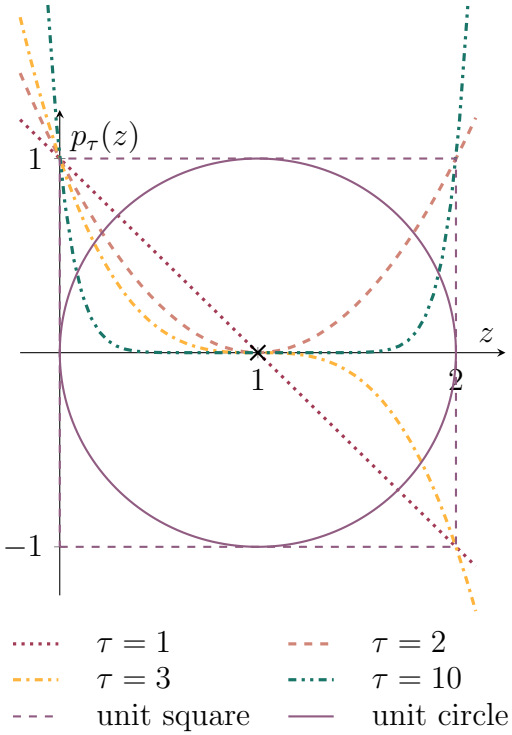


Figure 2.8: Error polynomials $p_\tau(z) = (1-z)^\tau$ for $\tau \in \{1, 2, 3, 10\}$ evaluated on the real axis. The root of multiplicity τ is located at $z = 1$ and marked by $\times$. At $z = 0$, we always have $p_\tau(0) = 1$, whereas at $z = 2$, the values alternate between -1 and 1.

Error bounds. For any consistent matrix norm, the error is bounded by

$$||e^{(\tau)}|| \leq ||(I - BA)^\tau||\,||e^{(0)}|| \leq ||I - BA||^\tau\,||e^{(0)}||.$$

In particular, if $||I - BA|| < 1$, the error decays at least geometrically, i.e., $e^{(\tau)} \to 0$ as $\tau \to \infty$. This condition is stronger than necessary, but it already guarantees convergence and provides an explicit estimate on the asymptotic reduction factor.

Equivalently, using the error polynomial p_τ from Equation (2.21)

$$||e^{(\tau)}|| \leq ||p_\tau(BA)||\,||e^{(0)}||.$$

If $BA = X\Lambda X^{-1}$ is diagonalizable, then

$$||e^{(\tau)}|| \leq ||X(I - \Lambda)^\tau X^{-1}||\,||e^{(0)}|| \leq \kappa(X) \max_j |1 - \lambda_j|^\tau ||e^{(0)}||,$$

where $\kappa(X) = ||X||\,||X^{-1}||$ denotes the (relative) *condition number* of X.

Note that the eigenvalues do not completely determine convergence since a poorly conditioned eigenbasis X can lead to transient growth of the error. In this case, some components of $e^{(\tau)}$ can initially increase in norm before eventually decaying, even though

the asymptotic convergence condition $\rho(I - BA) < 1$ is satisfied.[13] In contrast, if BA is normal, i.e., it is unitarily diagonalizable, then $\kappa(X) = 1$ in the 2-norm. The error bound simplifies to

$$||e^{(\tau)}|| \leq \max_j |1 - \lambda_j|^\tau ||e^{(0)}||_2,$$

and, more generally, for any polynomial p

$$||p(BA)||_2 = \max_{\lambda \in \text{spec}(BA)} |p(\lambda)|. \tag{2.23}$$

Thus, for normal matrices, the spectral radius completely characterizes convergence, since no hidden growth can occur.

Richardson iteration. A particularly simple stationary iterative scheme results from choosing $B = \omega I$, with *relaxation parameter* $\omega > 0$, yielding the *Richardson iteration*

$$u^{(\tau+1)} = u^{(\tau)} + \omega(f - Au^{(\tau)}).$$

In this case, the iteration matrix reduces to $I - \omega A$.

The corresponding error polynomial $p_\tau(z) = (1 - \omega z)^\tau$ is of degree τ and has a root at $z = \frac{1}{\omega}$ of multiplicity τ. Assume that A is diagonalizable with real spectrum in $[\lambda_{\min}, \lambda_{\max}]$, so that the eigenvalues satisfy $0 \leq \lambda_{\min} \leq \lambda_i \leq \lambda_{\max}$. Since the iteration matrix is $I - \omega A$, convergence depends on the eigenvalues of A. In particular, Richardson iteration converges if and only if there exists $\omega > 0$ such that $\rho(I - \omega A) < 1$. Formally, we require

$$|1 - \omega\lambda_j| < 1 \quad \text{for} \quad \lambda_j \in [\lambda_{\min}, \lambda_{\max}] \Longleftrightarrow 0 < \omega < \frac{2}{\lambda_{\max}}.$$

According to Equation (2.23), the worst case reduction factor after one step is

$$q := \max_{\lambda_j \in [\lambda_{\min}, \lambda_{\max}]} |1 - \omega\lambda_j|.$$

Since $||e^{(\tau)}|| \leq q^\tau ||e^{(0)}||$, a smaller reduction factor results in faster convergence. In order to find a parameter ω that minimizes the maximum deviation of $1 - \omega\lambda$ over the interval of eigenvalues, we solve the minimax problem

$$\min_{\omega > 0} \max_{\lambda_j \in [\lambda_{\min}, \lambda_{\max}]} |1 - \omega\lambda_j|.$$

Thus, we require

$$|1 - \omega\lambda_{\min}| = |1 - \omega\lambda_{\max}|.$$

[13] Here, "poorly conditioned" means that the eigenvectors of BA are nearly linearly dependent, so that the eigenbasis matrix X has a large condition number $\kappa(X) = ||X||\,||X^{-1}|| \gg 1$, resulting in numerical instability in the eigendecomposition.

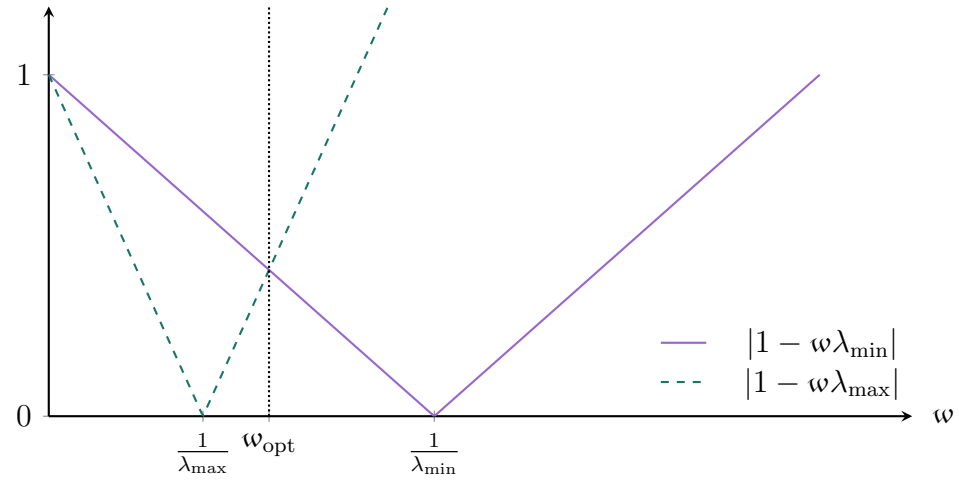


Figure 2.9: $|1-\omega\lambda_{\min}|$ and $|1-\omega\lambda_{\max}|$ as functions of the relaxation parameter ω. The optimal parameter ω_{opt} is located at the intersection of the two functions.

Solving this yields the optimal relaxation parameter

$$\omega_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}.$$

Figure 2.9 illustrates $|1-\omega\lambda_{\min}|$ and $|1-\omega\lambda_{\max}|$ as functions of ω, and their intersection corresponds to ω_{opt}.

The corresponding optimal asymptotic convergence factor for Richardson iteration is

$$q_{\text{opt}} = |1 - \omega_{\text{opt}}\lambda_{\min}| = |1 - \omega_{\text{opt}}\lambda_{\max}| = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{\kappa - 1}{\kappa + 1}, \text{ with } \kappa = \frac{\lambda_{\max}}{\lambda_{\min}}.$$

However, in practice the optimal ω can be difficult to use. Instead, we find an upper bound on the spectral radius, i.e., $\rho(A) \leq \zeta$.[14] Then a sufficient condition for convergence is $0 < \omega \leq 2/\zeta$. A common choice is $\omega = 1/\zeta$ since it satisfies $1/\zeta \leq 1/\rho(A) \leq 2/\rho(A)$.

Smoothing property of Richardson iteration. For the eigenpairs $A\gamma_j = \lambda_j\gamma_j$ the error propagator with relaxation parameter ω is

$$e^{(\tau)} = \sum_j (1 - \omega\lambda_j)^\tau c_j \gamma_j.$$

If only an upper bound $\zeta \approx \lambda_{\max}$ on the spectrum is available, a common choice is $\omega = 1/\zeta$. Each eigencomponent is then reduced by the factor $(1 - \lambda_j/\zeta)^\tau$. The slowest decaying component corresponds to the smallest eigenvalue $\lambda_{\min}$. The corresponding

[14] For instance using the power iteration method.

reduction factor per step is $1 - {}^{\lambda_{\min}}/_{\zeta}$, which is close to 1 whenever $|{}^{\lambda_{\min}}/_{\zeta}| \ll 1$. Using the Taylor expansion $\log(1-x) \approx -x$ for small x, we obtain after τ steps

$$(1 - \frac{\lambda_{\min}}{\zeta})^{\tau} = \exp\left(\tau \cdot \log(1 - \frac{\lambda_{\min}}{\zeta})\right) \approx \exp\left(-\tau \frac{\lambda_{\min}}{\zeta}\right).$$

Hence, reducing this component by a constant factor requires approximately

$$\tau \approx \frac{\zeta}{\lambda_{\min}}$$

iterations. Since $\zeta \sim \lambda_{\max}$, this gives $\tau \sim {}^{\lambda_{\max}}/_{\lambda_{\min}} = \kappa(A)$. This ratio effectively acts as condition number of the spectrum since convergence is slow if this ratio is large.

2.5 Multigrid Methods

In Section 2.4, we have seen that iterative methods exhibit smoothing properties by quickly reducing error components in the directions of eigenvectors associated with large eigenvalues. These correspond to high-frequency components of the error. Conversely, low-frequency components associated with small eigenvalues are only weakly damped. Multigrid methods address this deficiency by complementing a *smoother* $\mathcal{S}$, typically a basic iterative method, with a *coarse-grid correction* that eliminates the low-frequency error components which remain after smoothing. The key idea is that error components that appear as low-frequency on the fine grid become high-frequency on a coarser grid and therefore can be reduced effectively there. We first formulate multigrid on lattice tori and then present the coarse-grid correction from a matrix perspective. Then we introduce the multigrid V-cylce by generalizing the two-grid algorithm to multiple grids. This section is mainly based on [65, 96, 107].

Aliasing of error components on different scales. In this setting, two-dimensional grids can be naturally represented by a lattice-sublattice pair. The fine grid is represented by a lattice $\mathbb{L}(\mathcal{A})$ with lattice basis $\mathcal{A} \in \mathbb{R}^{2\times 2}$. The coarse grid corresponds to the sublattice $\mathbb{L}(2 \cdot \mathcal{A}) \subseteq \mathbb{L}(\mathcal{A})$, containing, for example, only every second grid point of the fine grid $\mathbb{L}(\mathcal{A})$. In practice, the problem typically is restricted to a finite computational grid. Rather than considering the infinite lattice $\mathbb{L}(\mathcal{A})$, we restrict the problem to the finite quotient space

$$\mathbb{T}_{\mathcal{A},\mathcal{Z}} = \mathbb{L}(\mathcal{A}) \Big/ \mathbb{L}(\mathcal{Z}),$$

where $\mathbb{L}(\mathcal{Z}) \subseteq \mathbb{L}(\mathcal{A})$ is the sublattice with primitive vectors $\mathfrak{z}_1 = N \cdot a_1$ and $\mathfrak{z}_2 = N \cdot a_2$, so that the finite torus has N points in each lattice direction and is periodic with period N.

On this torus, the error can be expanded into Fourier modes $\varphi_k = e^{2\pi \mathrm{i}\langle k,x\rangle}$ for $x \in \mathbb{L}(\mathcal{A})$ and $k \in \mathbb{L}(\mathcal{A})^*$, where $\mathbb{L}(\mathcal{A})^*$ is the dual of $\mathbb{L}(\mathcal{A})$, cf. Section 2.2.3. Since the fine grid

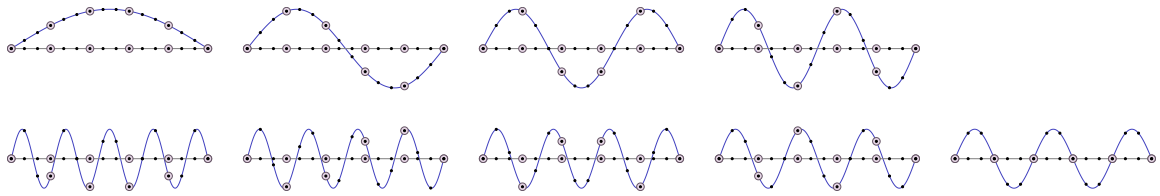


Figure 2.10: One-dimensional illustration of the scale separation of error components. The coarse grid is represented by large dots. Slowly oscillatory components remain distinguishable on the coarser scale, whereas highly oscillatory (high-frequency) components become indistinguishable due to aliasing.

is represented by the finite torus $\mathbb{T}_{\mathcal{A},\mathcal{Z}}$, only finitely many modes occur, indexed by the wave vectors elements of the dual torus

$$k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}} = \mathbb{L}(\mathcal{Z}^{-\mathrm{T}}) \Big/ \mathbb{L}(\mathcal{A}^{-\mathrm{T}}).$$

As discussed in Section 2.2.3 Fourier modes on the lattice torus can naturally be separated into low- and high frequencies. Low-frequency (smooth) modes vary slowly across the fine grid and are concentrated near the origin of the frequency domain. High-frequency (oscillatory) modes oscillate rapidly between neighboring grid points, cf. Figure 2.5. Restricting the fine grid $\mathbb{L}(\mathcal{A})$ to a coarse grid, e.g., $\mathbb{L}(2 \cdot \mathcal{A})$, induces the dual lattice hierarchy $\mathbb{L}(2 \cdot \mathcal{A}^{-\mathrm{T}}) \subseteq \mathbb{L}(\mathcal{A})^*$. In this setting, the frequency domain is "folded" along this dual sublattice. In particular, two fine grid frequencies that differ by an element of the coarse dual lattice become indistinguishable on the coarse grid, cf. Section 2.2.3. As a result, the low-frequency components on the fine grid are mapped into relatively higher frequencies with respect to the coarser grid. This effect is illustrated in Figure 2.10 for different frequencies in $\mathbb{R}$.

2.5.1 Coarse-Grid Correction

The central idea of multigrid is to solve the problem on a hierarchy of grids, each of which represents the same underlying problem. Such a hierarchy may arise either by discretizing the original problem on progressively coarser grids or by constructing coarse-grid operators using the Galerkin condition. Regardless of how the hierarchy is constructed, operating on different scales requires transfer operators between the fine and coarse grids. Transfer operators for restriction and prolongation, respectively, are linear mappings between function spaces on different grids. Both operators are linear transferring operators that either reduce or increase spatial resolution, while preserving the underlying lattice structure.

Transfer operations. Let $\mathbb{L}(\mathcal{Z}) \subseteq \mathbb{L}(B) \subseteq \mathbb{L}(A)$ be a hierarchy of lattices. The coarse grid is represented by the lattice torus $\mathbb{T}_{\mathcal{B},\mathcal{Z}}$. The transfer to the coarse grid is given by

a linear mapping, acting on the function space of the coarse grid torus $\mathbb{T}_{\mathcal{B},\mathcal{Z}}$,

$$R : \mathfrak{L}(\mathbb{T}^{\mathfrak{a}}_{\mathcal{B},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}^{(0)}_{\mathcal{B},\mathcal{Z}}). \tag{2.24}$$

Here, the structure element $\mathfrak{a} = (0, a_1, a_2, a_1 + a_2) \cong \mathbb{T}_{\mathcal{A},\mathcal{B}}$ consists of the four representatives of the lattice torus $\mathbb{T}_{\mathcal{A},\mathcal{B}}$. They determine the contributions of the fine-grid points to each coarse-grid point. A simple choice is the *injection operator*, which samples every second grid point. In this case the lattice basis of the coarse grid is given by $\mathcal{B} = 2 \cdot \mathcal{A}$ as illustrated in the one-dimensional example in Figure 2.10. A more common choice is the *full weighting* operator that averages neighboring lattice points, over a 9-point neighborhood[15]. In the setting of lattice operators, the restricting operator is a multiplication operator

$$(Rf)(x) = \sum_{\delta \in \mathcal{N}} M_R^{(\delta)} f(x + \delta)$$

with nonzero multipliers $M_R^{(\delta)} \in \mathbb{R}^{1 \times |\mathbb{T}_{\mathcal{A},\mathcal{B}}|}$ and $\delta \in \mathcal{N} = \{0, -b_1, -b_2, -b_1 - b_2\}$. Here $\mathcal{B} = [b_1 \mid b_2] = 2 \cdot [a_1 \mid a_2]$. The corresponding nonzero multipliers are given by

$$M_R^{(-b_1)} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{4} \end{bmatrix} \qquad M_R^{(0)} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

$$M_R^{(-(b_1+b_2))} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \qquad M_R^{(-b_2)} = \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

Figure 2.11 illustrates such a weighted restriction.

The transfer from the coarse to the fine grid is performed by the *prolongation* operator

$$P : \mathfrak{L}(\mathbb{T}^{(0)}_{\mathcal{B},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}^{\mathfrak{a}}_{\mathcal{B},\mathcal{Z}}).$$

For symmetric matrices A, the prolongation operator is commonly chosen as $P = R^{\mathrm{T}}$ and the corresponding multiplication operator reads

$$(Pf)(x) = \sum_{\delta \in \mathcal{N}} M_P^{(\delta)} f(x + \delta).$$

The multiplication matrices for the prolongation arise as $M_P^{(\delta)} = (M_R^{(-\delta)})^{\mathrm{T}}$. The prolongation is illustrated in Figure 2.12. With the transfer operators defined, we can define a suitable coarse-grid operator using the *Galerkin* condition

$$A_c : \mathfrak{L}(\mathbb{T}_{\mathcal{B},\mathcal{Z}}) \longrightarrow \mathfrak{L}(\mathbb{T}_{\mathcal{B},\mathcal{Z}}) \quad \text{with} \quad A_c = RAP.$$

[15] 9-point stencils are a common choice, but larger stencils, such as 25-point or 49-point are also possible.

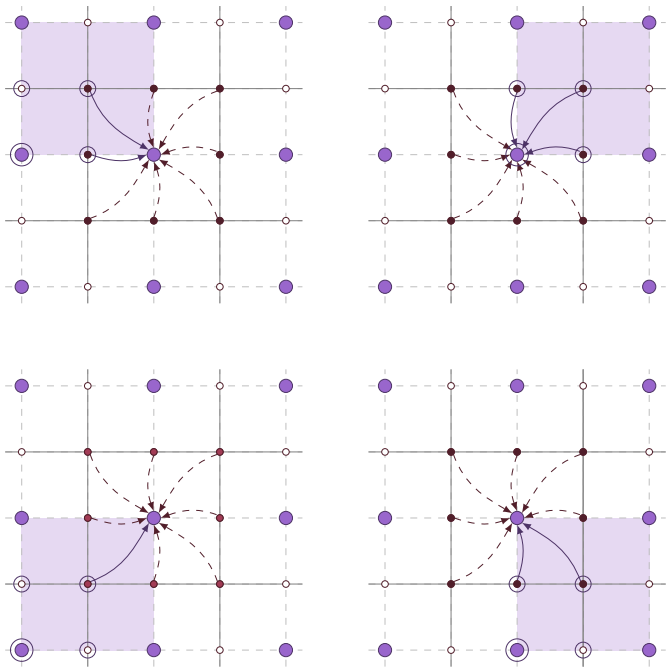


Figure 2.11: Illustration of a weighted restriction from a fine lattice (small dots) to a coarse lattice (large dots). Circled fine-grid points within a shaded coarse-grid unit cell act as structure elements. Solid arrows show their contributions to the coarse-grid point, weighted by $M_R^{(0)}$ (upper right), $M_R^{(-\delta_2)}$ (lower right), $M_R^{(-\delta_1)}$ (upper left) and $M_R^{(-\delta_1-\delta_2)}$ (lower left).

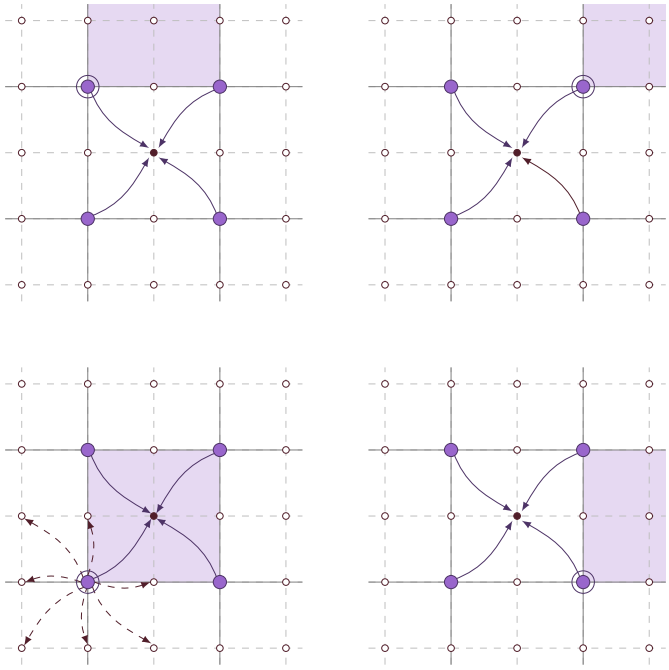


Figure 2.12: Prolongation from a coarse grid (large dots) to a fine grid (small dots). Solid arrows indicate the contributions of coarse-grid points to a fine-grid point. Dashed arrows show the contribution of the circled coarse-grid point, whose corresponding unit cell is shaded.

This operator represents the fine-grid operator A consistently on the coarse grid. In particular, the residual equation is solved on the coarse grid

$$A_c \hat{e}_c = r_c \quad \text{with } r_c = Rr = R(f - Au^{(\tau)}).$$

Here $\hat{e}_c$ is the coarse grid error approximation, which is interpolated back to the fine grid via prolongation $\hat{e} = P\hat{e}_c$.

Collecting these steps yields the coarse-grid correction operator for the fine grid residuals:

$$K := I - PA_c^{-1}RA = I - \Pi_A, \tag{2.25}$$

where Π_A denotes a projection onto a lower-dimensional coarse-grid space and is therefore not invertible. Consequently, K has a non-trivial null space, that is, there exist nonzero vectors $e \neq 0$ such that $KAe = 0$. This means that some error components, particularly high-frequency ones, remain unaffected by the coarse-grid correction scheme. Moreover, (Galerkin) coarse-grid correction is an A-orthogonal projection, satisfying $||I - \Pi_A(P)||_A = 1$.

Consequently, coarse-grid correction alone cannot eliminate all error components. In particular, high-frequency components remain essentially unchanged. Therefore, multigrid combines coarse-grid correction with smoothing to reduce all error components.

2.5.2 Two-Grid Method

As discussed above, coarse grid-correction alone cannot eliminate the high-frequency components of the error. To efficiently damp these error components a few iterations of a relaxation scheme, e.g., the Richardson iteration introduced in Section 2.4.2, are applied before and after the coarse-grid correction. The corresponding smoothing propagators are denoted by $\mathcal{S}^{\nu_{\text{pre}}}$ and $\mathcal{S}^{\nu_{\text{post}}}$, where ν_{pre} and ν_{post} denote the numbers of pre- and post-smoothing steps respectively.

A *two-grid cycle* consists of *pre-smoothing*, followed by *coarse-grid correction* and a subsequent *post-smoothing*, as summarized in Algorithm 1. The two-grid cycle is repeated until a prescribed stopping criterion is satisfied.

Algorithm 1 Two-grid-cycle $(A, f, u^{(0)})$

1: $u^{(\nu_{\text{pre}})} = \mathcal{S}^{\nu_{\text{pre}}}(A, f, u^{(0)})$ ▷ ν_{pre} pre-smoothing steps
2: $r_c = R(f - Au^{(\nu_{\text{pre}})})$ ▷ restrict residual
3: $e_c = A_c^{-1} r_c$ ▷ solve on coarse grid
4: $\tilde{u} = u^{(\nu_{\text{pre}})} + Pe_c$ ▷ prolongate and update
5: $u^{(\nu_{\text{post}})} = \mathcal{S}^{\nu_{\text{post}}}(A, f, \tilde{u})$ ▷ ν_{post} smoothing steps

The two-grid method defines a linear iterative scheme. Its error propagator combines smoothing and coarse-grid correction operators. Specifically, the error propagator is

$$E = \mathcal{S}^{\nu_{\text{post}}} K \mathcal{S}^{\nu_{\text{pre}}}$$

with coarse-grid operator from Equation (2.25). The corresponding iteration on the approximate solution $u^{(\tau)}$ is given by

$$\begin{aligned} u^{(\tau+1)} &= u^{(\tau)} + PA_c^{-1}R(f - Au^{(\tau)}) \\ &= (I - PA_c^{-1}RA)u^{(\tau)} + PA_c^{-1}Rf \\ &= Ku^{(\tau)} + PA_c^{-1}Rf. \end{aligned}$$

Since both smoothing operators $\mathcal{S}^{\nu_{\text{pre}}}$, $\mathcal{S}^{\nu_{\text{post}}}$, and the coarse-grid correction operator K are linear and consistent error propagators, their product E is again a linear error propagator [69].

Each component of the two-grid cycle targets a different part of the error spectrum. The smoothing operators effectively damp the high-frequency error components, whereas the coarse-grid correction reduces the remaining low-frequency components. This relies on the approximation property, namely, that the low-frequency error components on the fine grid can be represented accurately on the coarse grid and therefore corrected there. If the smoother, transfer operators and the approximation property satisfy suitable assumptions, their complementary action yields

$$\rho(E) < 1,$$

which guarantees convergence of the two-grid method.

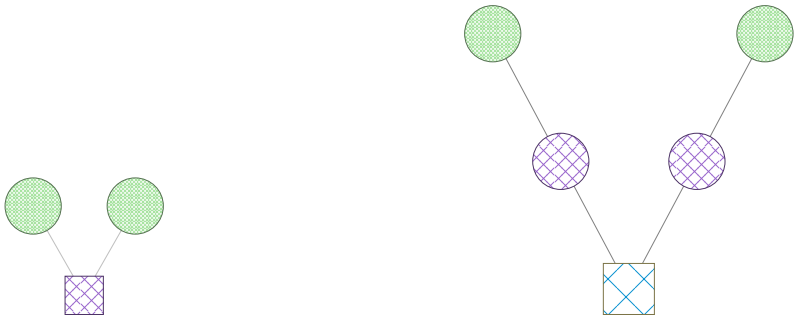
2.5.3 From Two-Grid to Multigrid Methods

To obtain a multigrid method, the principle of the two-grid construction is applied recursively. In particular the exact coarse grid solve in line 3 of Algorithm 1 is replaced by another two-grid cycle acting on the next coarser grid [107]. A multigrid method with L-levels is characterized by a lattice hierarchy

$$\mathbb{L}(\mathcal{B}_L) \subseteq \mathbb{L}(\mathcal{B}_{L-1}) \subseteq \ldots \subseteq \mathbb{L}(\mathcal{A})$$

where $\mathbb{L}(B_L)$ denotes the coarsest grid and $\mathbb{L}(A)$ the lattice with the finest grid. On level ℓ, let A_ℓ denote the system matrix and I_ℓ the corresponding identity operator, and S_ℓ the smoothing operator. The corresponding smoothing iterations are denoted by $\mathcal{S}_l^{\nu_{\text{pre}}}$ and $\mathcal{S}_l^{\nu_{\text{post}}}$ respectively. Transfer operators between levels l and $\ell + 1$ are $R_\ell^{\ell+1}$ and $P_{\ell+1}^{\ell}$, respectively. The multigrid error propagator is defined recursively by

$$E_0 = 0 \quad \text{and} \quad E_\ell = \mathcal{S}_\ell^{\nu_{\text{post}}} K_\ell \mathcal{S}_\ell^{\nu_{\text{pre}}} \text{ for } \quad \ell = 1, \ldots, L,$$



(a) V-cycle with two levels (b) V-cycle with three levels

Figure 2.13: Structure of multigrid V-cycle iterations with two (a) and three (b) grid levels. Grid spacing reflects the coarseness of the corresponding level. Circular nodes denote smoothing operations and rectangular nodes correspond to exact coarse-grid solves. Downward lines (\) indicate restriction and upward lines (/) indicate prolongation of the coarse-grid solution.

where the coarse-grid correction operator is given by

$$K_\ell = I_\ell - P_{\ell+1}^\ell \left[I_{\ell+1} - E_{\ell+1} \right] A_{\ell+1}^{-1} R_\ell^{\ell+1} A_\ell.$$

Substituting the recursive coarse-grid correction K_ℓ into the two-grid error propagator yields the multigrid error propagator E_ℓ. The resulting recursive algorithm is known as multigrid V-cycle. Multigrid V-cycles with two and three grid levels are illustrated in Figure 2.13. Variants such as W-cycles or F-cycles can be obtained by revisiting coarse-grid levels multiple times; but are not covered in this thesis (see e.g., [107] for details). To illustrate the recursive construction, we consider the two finest levels, $\ell = 0, 1$. For readability, we denote operators on the coarse grid $\ell = 1$ by the subscript c, i.e., $A_c = A_1$, $I_c = I_1$, $E_c = E_1$, $R = R_0 1$ and $P = P_1^0$. On the coarse grid, the error equation becomes

$$A_c e_c = R(f - Au).$$

Since the coarse-grid solve is itself replaced by another two-grid cycle, the coarse-grid correction is given by

$$e_c = (I_c - E_c) A_c^{-1} R(f - Au).$$

Consequently, the updated fine-grid iterate is $u \leftarrow u + Pe_c$ and therefore

$$\begin{aligned} \tilde{u} &= u + Pe_c \\ &= u + P(I_c - E_c) A_c^{-1} R(f - Au) \\ &= (I - P(I_c - E_c) A_c^{-1} RA)u + P(I_c - E_c) A_c^{-1} Rf. \end{aligned}$$

Thus, the coarse-grid correction operator on the fine grid is

$$K = I - P(I_c - E_c) A_c^{-1} RA,$$

and the corresponding two-level multigrid error propagator is given by $E = \mathcal{S}^{\nu_{\text{post}}} K \mathcal{S}^{\nu_{\text{pre}}}$.

While we have written out the update formulas explicitly for the two finest levels, these constructions naturally extend to any number of grid levels, as shown in Algorithm 2.

2.5.4 Full Approximation Scheme

So far, we have considered only linear problems. However, nonlinearities arise naturally, particularly when drawing analogies between multigrid methods and convolutional neural networks, where nonlinearities are intrinsic.

Both components of the multigrid method, smoothing and coarse-grid correction, can be generalized to nonlinear problems. Since an error equation is generally not available for nonlinear problems, the coarse-grid problem is formulated for the full approximation.

The key idea of the *full approximation scheme* (FAS) is that the coarse-grid solution is not initialized with zero, as in the linear case, but with the restricted fine grid approximation. Consequently, the coarse level inherits a consistent approximation, providing the starting point for the nonlinear correction. Thus, after ν smoothing steps we set

$$u_{\ell+1}^{(0)} = R_\ell^{\ell+1} u_\ell^{(\nu)}.$$

As a consequence the coarse-grid residual equation is modified to

$$A_{\ell+1}(u_{\ell+1}) = A_{\ell+1}(R_\ell^{\ell+1} u_\ell) + R_\ell^{\ell+1}(f_\ell - A_\ell u_\ell),$$

which incorporates the nonlinear structure of the problem. The corresponding fine grid correction then reads

$$\tilde{u}_\ell = u_\ell + P_{\ell+1}^{\ell}(u_{\ell+1} - R_\ell^{\ell+1} u_\ell).$$

In contrast to the linear case, where the coarse-grid problem is formulated for the error equation, FAS formulates the coarse-grid problem directly for the full approximation on the coarse level. A multigrid-V-cycle with FAS is given by Algorithm 2.

Algorithm 2 FAS V-Cycle$(A_\ell, f_\ell, u_\ell^{(0)})$

1: **if** $\ell = L$ **then**
2: $\quad u_\ell = A_\ell^{-1} f_\ell$ ▷ exact coarse-grid solve
3: **else**
4: $\quad u_\ell^{(\nu_{\text{pre}})} = \mathcal{S}^{\nu_{\text{pre}}}(A_\ell, f_\ell, u_\ell^{(0)})$ ▷ ν_{pre} pre-smoothing steps
5: $\quad r_{\ell+1} = R_\ell^{\ell+1}(f_\ell - A_\ell u_\ell^{(\nu_{\text{pre}})})$ ▷ restrict residual
6: $\quad u_{\ell+1}^{(0)} = R_\ell^{\ell+1} u_\ell^{(\nu_{\text{pre}})}$ ▷ restrict iterate
7: $\quad f_{\ell+1} = r_{\ell+1} + A_{\ell+1} u_{\ell+1}^{(0)}$ ▷ adjust coarse problem
8: $\quad u_{\ell+1} = \text{FAS } V\text{-cycle}(A_{\ell+1}, f_{\ell+1}, u_{\ell+1}^{(0)})$ ▷ recursive coarse-grid solve
9: $\quad u_\ell = u_\ell^{(\nu_{\text{pre}})} + P_{\ell+1}^{\ell}(u_{\ell+1} - u_{\ell+1}^{(0)})$ ▷ prolongate and update
10: $\quad u_\ell^{(\nu_{\text{post}})} = \mathcal{S}^{\nu_{post}}(A_\ell, f_\ell, u_\ell)$ ▷ ν_{post} smoothing steps
11: **end if**

3 Fundamentals of Machine Learning

Machine learning (ML) methods are statistical algorithms used to identify patterns and relationships in data to establish decision rules. These rules can be used to label unknown, previously unseen data points. In this context, *learning* refers to the process of optimizing the model parameters. This iterative process typically involves minimizing an objective function, often called the loss function, that measures the model's performance. Based on this objective, the model parameters are adjusted to improve the model's performance. Prior to optimization, the parameter space is typically restricted to a predefined set of models, known as the hypothesis space. This chapter introduces the key concepts of supervised learning, including the definition of the hypothesis space and the characterization of loss functions.

In Section 3.1 we introduce the general concept of a statistical learning framework, which provides a theoretical foundation for understanding supervised learning methods and for analyzing generalization to new, previously unseen data points. In Section 3.2 we present a specific class of machine learning methods: deep neural networks (NNs).

3.1 Supervised Learning

The core idea of supervised learning is to extract information from a given set of labeled data points, the training set, about the underlying distribution in order to generalize to unlabeled data points. The objective is to minimize the error the machine learning method makes while predicting the labels of new data points. This leads to the learning paradigm of empirical risk minimization, discussed in Section 3.1.1. The ability to generalize depends on the complexity of the chosen hypothesis class, which represents an inductive bias introduced before training. Larger hypothesis classes allow for more flexibility but also increase the risk of overfitting to the training set. Conversely, a small hypothesis class might limit the ability to capture complex features of the underlying

data distribution. This relationship is known as bias-complexity trade-off and discussed in Section 3.1.2. This section follows [100] and [40].

3.1.1 Empirical Risk Minimization

We begin by introducing the statistical learning framework, including definitions of the learner's input, its resulting output and measures of success. This provides a distilled version of the underlying theory, focusing on illustrating the core principles of statistical learning. We then discuss the empirical risk minimizer, introduce the notion of bias and a core idea of generalization. After that, we elaborate on how training on finite datasets leads to fundamental trade-offs and limitations of this theoretical point of view.

The statistical learning framework. Given an arbitrary set $\mathcal{X}$, which denotes the input domain, the goal is to estimate a function ψ that approximates a perfect but unknown labeling function ϱ. The true labeling function ϱ assigns each instance from the input domain $z_i \in \mathcal{X}$ a corresponding label $y_i \in \mathcal{Y}$ from the label set $\mathcal{Y}$, which contains all possible labels.[1] In formal terms, this can be expressed as $\varrho : \mathcal{X} \to \mathcal{Y}$. The instances $z_i \in \mathcal{X}$, are typically represented by a d-dimensional vector of features, i.e., attributes of the domain instances and drawn from an unknown probability distribution over $\mathcal{X}$. For classification tasks, the label set $\mathcal{Y}$ is typically finite. Let $S = \{(z_i, y_i)\}_{i=1}^{t} \sim \mathcal{D}$ be a finite dataset of t independently and identically distributed (i.i.d.) instances and corresponding labels drawn from a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, $y_i = \varrho(z_i)$. $\mathcal{D}$ is also referred to as *data-generating* distribution. We write $\mathbb{P}_{z,y}$ to denote the corresponding product probability measure on $(\mathcal{X} \times \mathcal{Y})^t$. Based on S the goal is to estimate ψ, such that the difference between $\psi(z_i)$ and $\varrho(z_i)$ is minimal. The probability that the function ψ does not predict the correct label on a random instance $x \sim \mathcal{D}$, i.e., $\varrho(z) \neq \psi(z)$ is the *generalization error* defined by

$$\mathcal{L}_{\mathcal{D},\varrho}(\psi) \stackrel{\text{def.}}{=} \mathop{\mathbb{P}}_{z\sim\mathcal{D}}[\psi(z) \neq \varrho(z)] \stackrel{\text{def.}}{=} \mathcal{D}(\{z : \psi(z) \neq \varrho(z)\}). \tag{3.1}$$

$\mathcal{L}_{\mathcal{D},\varrho}(\psi)$ is also called the *risk* or *true error* of ψ and can thus be viewed as the *loss* of the learner.

A function $\mathcal{L} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ is referred to as 0-1 loss, as it takes value "0" for correctly predicted labels for instances $f \in \mathcal{X} \sim \mathcal{D}$ and "1" for wrong predictions and can be written as

$$\mathcal{L}_{\mathcal{D},\varrho}(\psi) = \left\{ \begin{array}{ll} 1 & \text{if } \psi(z) \neq \varrho(z) \\ 0 & \text{if } \psi(z) = \varrho(z) \end{array} \right\} = \mathbb{1}_{\{\psi(z)\neq\varrho(z)\}}.$$

Taking the expectation over $\mathcal{D}$ yields the expected 0-1 loss

$$\mathcal{L}_{0-1} = \mathbb{E}_{z\sim\mathcal{D}}[\mathcal{L}(\psi(z), \varrho(z))] = \mathbb{E}_{z\sim\mathcal{D}}[\mathbb{1}_{\{\psi(z)\neq\varrho(z)\}}].$$

From now, the labeling function ϱ is fixed and the loss function is denoted by $\mathcal{L}_{\mathcal{D},\varrho} = \mathcal{L}_{\mathcal{D}}$.

[1] This use of y is unrelated to the spatial coordinates (x, y) introduced in Section 2.4.1.

The empirical risk minimizer. For a training set S, the *empirical risk* (or *empirical error*) of a prediction rule ψ is defined as the average loss over the training instances:

$$\mathcal{L}_S(\psi) = \frac{1}{t} \sum_{i=1}^{t} \mathbb{1}_{\{\psi(z_i) \neq y_i\}}. \tag{3.2}$$

A predictor ψ that minimizes the empirical loss $\mathcal{L}_S(\psi)$ is called an *empirical risk minimizer*(ERM) and the corresponding learning principle is known as empirical risk minimization .

Given the *hypothesis class* $\mathcal{H}$, i.e., the set of all candidate functions, the objective is to find a function $\psi : \mathcal{X} \to \mathcal{Y}$ from $\mathcal{H}$ that minimizes the empirical risk $\mathcal{L}_S(\psi)$ from Equation (3.2). The empirical risk minimizer over $\mathcal{H}$, denoted by $\mathrm{ERM}_{\mathcal{H}}$, returns a function $\psi \in \mathcal{H}$ with minimal error over a given training set S, specifically

$$\mathrm{ERM}_{\mathcal{H}}(S) \in \operatorname*{argmin}_{\psi \in \mathcal{H}} \mathcal{L}_S(\psi). \tag{3.3}$$

For the following discussion, we assume that the hypothesis class $\mathcal{H}$ is finite. As a result

$$\psi_S = \operatorname*{argmin}_{\psi \in \mathcal{H}} \mathcal{L}_S(\psi) \tag{3.4}$$

is the $\mathrm{ERM}_{\mathcal{H}}$ applied to S.

Realizability assumption. To introduce some fundamentals, we first assume that there exists a hypothesis $\psi^* \in \mathcal{H}$ that perfectly fits the underlying data distribution, i.e., $\mathcal{L}_{\mathcal{D}}(\psi^*) = 0$. This assumption simplifies the following discussion. Later, we relax the assumption, since generally the data may not be perfectly representable by the hypothesis class $\mathcal{H}$. Ultimately, our goal is to analyze how well the ERM performs on the true risk. Furthermore, under the *realizability assumption*, the hypothesis ψ^* agrees with the true labeling function, i.e., $\psi^*(z_i) = \varrho(z_i)$ for every training sample z_i.

PAC learnability. The ERM allows us to select a hypothesis that performs best on the training data. However, a low empirical error does not necessarily imply a low generalization error. To this end we aim to find probabilistic guarantees on how well a hypothesis learned from the finite training set S generalizes on previously unseen samples drawn from $\mathcal{D}$. Specifically, we seek conditions under which the learner reliably finds a hypothesis with small error after seeing a feasible number of training samples.

A hypothesis ψ is *approximately correct* if its error $\mathcal{L}_{\mathcal{D}}(\psi)$ is bounded by some error rate $\epsilon > 0$, i.e., $\mathcal{L}_{\mathcal{D}}(\psi) < \epsilon$. If a learning algorithm returns $\mathcal{L}_{\mathcal{D}}(\psi) < \epsilon$ with at least probability $(1 - \delta)$, then the algorithm is said to be confident. Here, $\delta \in (0, 1)$ is a confidence parameter. Specifically, if $\mathcal{L}_{\mathcal{D}}(\psi) \leq \epsilon$ the algorithm returns an approximately correct

hypothesis. On the other hand, if $\mathcal{L}_{\mathcal{D}}(\psi) > \epsilon$, the learning algorithm is considered to have failed. Assuming the realizability assumption holds and if there exist a function

$$t_{\mathcal{H}} : (0,1) \times (0,1) \longrightarrow \mathbb{N} \quad \text{with} \quad (\epsilon, \delta) \longmapsto t_{\mathcal{H}}(\epsilon, \delta),$$

such that $t \geq t_{\mathcal{H}}$, a hypothesis class $\mathcal{H}$ is said to be *Probably Approximately Correct* (PAC) learnable under the following conditions. For every $\epsilon, \delta > 0$, any distribution $\mathcal{D}$ and any $\varrho \in \mathcal{H}$, the learning algorithm returns a hypothesis ψ confidently and $t_{\mathcal{H}}$ is a threshold to the sample complexity. If the actual sample size is too small i.e., $t < t_{\mathcal{H}}$, there may be too few samples to learn representative features. As a result, the predictor ψ can adapt perfectly to the samples in the training set while exhibiting poor generalization to unseen data points, making ψ prone to overfitting. Conversely, if $t \geq t_{\mathcal{H}}$, the ERM finds, with high probability, a hypothesis whose true risk is close the best achievable in $\mathcal{H}$. However, if $\mathcal{L}_{\mathcal{D}}(\psi)$ remains high regardless of the amount of data, this indicates a limitation of the hypothesis class $\mathcal{H}$, resulting in underfitting. For a finite hypothesis class $\mathcal{H}$ the sample complexity can be bounded by minimal samples $t_{\mathcal{H}}$ that are required to achieve a desired accuracy and confidence:

$$t_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{\log(\frac{|\mathcal{H}|}{\delta})}{\epsilon} \right\rceil. \tag{3.5}$$

Consequently, the number of training samples needed to guarantee generalization depends on the complexity of the hypothesis class $\mathcal{H}$. To derive the upper bound on the sample complexity defined in Equation (3.5), we estimate the probability over training samples $S|_z = (z_1, \ldots, z_t) \sim \mathcal{D}^t$, such that the hypothesis ψ_S, learned from $S \sim \mathcal{D}^t$, has a generalization error greater than ϵ. Specifically, we seek an upper bound on

$$\mathbb{P}_{S \sim \mathcal{D}^t}[\mathcal{L}_{\mathcal{D},\varrho}(\psi_S) > \epsilon].$$

The set of hypotheses with generalization error exceeding ϵ is referred to as the set of "bad" hypotheses

$$\mathcal{H}_B = \{\psi \in \mathcal{H} : \mathcal{L}_{\mathcal{D}}(\psi) > \epsilon\}.$$

Conversely, hypotheses not contained in $\mathcal{H}_B$ are referred to as "good" hypotheses. A hypothesis $\psi \in \mathcal{H}$ selected by the ERM solely based on S, despite $\mathcal{L}_D(\psi) > \epsilon$, is called a *misleading hypothesis*. The set $\mathcal{M} \subseteq \mathcal{X}^t$ denotes the set of training samples for which a misleading hypothesis exists

$$\mathcal{M} = \{S|_z : \exists\, \psi \in \mathcal{H}_B \text{ such that } \mathcal{L}_S(\psi) = 0\}.$$

That is, $\mathcal{M}$ contains those samples $S|_z$ for which a bad hypothesis $\psi \in \mathcal{H}_B$ exists, that behaves identically to a good hypothesis evaluated on $S|_z$, i.e., achieves zero empirical error. Under the realizability assumption, the event of $\mathcal{L}_{\mathcal{D}}(\psi_S) > \epsilon$ can only occur if the empirical loss $\mathcal{L}_S(\psi) = 0$ is zero for a bad hypothesis $\psi \in \mathcal{H}_B$. Consequently, $\mathcal{M}$ can be expressed as the union over all bad hypotheses:

$$\mathcal{M} = \bigcup_{\psi \in \mathcal{H}_B} \{S|_z : \mathcal{L}_S(\psi) = 0\}.$$

This allows us to upper bound to the probability that the learning algorithm returns a hypothesis with true error exceeding ϵ by the probability that the sample, drawn i.i.d. from $\mathcal{D}^t$, is misleading, i.e.,

$$\mathbb{P}(S|_z : \mathcal{L}_{\mathcal{D}}(\psi_S) > \epsilon) \leq \mathbb{P}(\mathcal{M}) = \mathbb{P}\left(\cup_{\psi \in \mathcal{H}_B} \{S|_z : \mathcal{L}_S(\psi) = 0\}\right). \tag{3.6}$$

Applying the union bound[2] as follows:

$$\mathbb{P}(\{S|_z : \mathcal{L}_{\mathcal{D}}(\psi_S) > \epsilon\}) \leq \sum_{\psi \in \mathcal{H}_{\mathcal{B}}} \mathbb{P}(\{S|_z : \mathcal{L}_S(\psi) = 0\}). \tag{3.7}$$

Fix a particular hypothesis $\psi \in \mathcal{H}_B$. The probability that it achieves zero training error on an i.i.d. drawn sample is

$$\begin{aligned}\mathbb{P}[\{S|_z : \mathcal{L}_S(\psi) = 0\}] &= \mathbb{P}[\{S|_z : \psi(z_i) = \varrho(z_i)\, \forall\, i\}] \\ &= \prod_{i=1}^{t} \mathbb{P}(\{z_i : \psi(z_i) = \varrho(z_i)\}).\end{aligned}$$

For each element z_i of the training set it holds that

$$\mathbb{P}[\psi(z_i) = \varrho(z_i)] \overset{\varrho(z_i)=y_i}{=} 1 - \mathbb{P}[\psi(z_i) \neq y_i] = 1 - \mathcal{L}_{\mathcal{D},\varrho}(\psi) \leq 1 - \epsilon,$$

and since $\psi \in \mathcal{H}_B$, i.e., $\mathcal{L}_{\mathcal{D},\varrho}(\psi) > \epsilon$, the probability that a bad hypothesis correctly predicts the label of a single sample is at most $1 - \epsilon$, i.e.,

$$\mathbb{P}[\psi(z_i) = y_i] \leq 1 - \epsilon.$$

Since the samples are assumed to be independent, the probability that ψ correctly predicts samples in S satisfies

$$\mathbb{P}[\mathcal{L}_S(\psi) = 0] \leq (1 - \epsilon)^t \leq e^{-\epsilon t}.$$

Combining this result with Equation (3.7) yields that the probability of drawing a sample on which the predictor ψ_S fails is bounded by

$$\mathbb{P}[\mathcal{L}_{\mathcal{D}}(\psi_S) > \epsilon] \leq |\mathcal{H}_B| e^{-\epsilon t} \leq |\mathcal{H}| \cdot e^{-\epsilon t}.$$

To ensure that this failure is no greater than δ, we require that

$$|\mathcal{H}| \cdot e^{-\epsilon t} \leq \delta.$$

Solving for t yields

$$t \geq \frac{1}{\epsilon} \log\left(\frac{|\mathcal{H}|}{\delta}\right).$$

[2]The union bound states that for two sets A and B and a distribution $\mathcal{D}$, $\mathcal{D}(A \cup B) \leq \mathcal{D}(A) + \mathcal{D}(B)$. Equality only holds if $A \cap B = \emptyset$.

The sample complexity is defined as the smallest integer $t_{\mathcal{H}}$ satisfying these requirements. Consequently, for any sample size $t \geq t_{\mathcal{H}}(\epsilon, \delta)$ we find

$$\mathbb{P}[\mathcal{L}_D(\psi) \leq \epsilon] \geq 1 - \delta$$

and thus $\mathcal{H}$ is PAC-learnable with desired accuracy and confidence. In practice, however, the realizability assumption does not hold. We therefore let $\mathcal{D}(z, y) = \mathcal{D}(z)D_z(y)$ denote the true joint probability distribution over inputs z and labels y, with marginal distribution $\mathcal{D}(z)$ of the inputs $z \in \mathcal{X}$ and $\mathcal{D}_z(y) = \mathcal{D}(y|z)$ the conditional distribution of y given z.

The generalization loss (cf. Equation (3.1)) is then defined as

$$\mathcal{L}_D(\psi) \stackrel{\text{def}}{=} \mathbb{P}_{(z,y)\sim\mathcal{D}}[\psi(z) \neq y].$$

A hypothesis class $\mathcal{H}$ is *agnostic PAC learnable* if the learning algorithm returns a hypothesis ψ satisfying, with probability of at least $1 - \delta$,

$$\mathcal{L}_D(\psi) \leq \min_{\psi' \in \mathcal{H}} \mathcal{L}_D(\psi') + \epsilon. \tag{3.8}$$

Thus, agnostic PAC learnability guarantees an approximation to the optimal loss achievable with the given hypotheses class $\mathcal{H}$, even when no perfect hypothesis exist.

3.1.2 The Bias-Complexity Trade-off

A key insight of PAC learning is that the sample complexity, i.e., the number of training samples required to guarantee generalization, depends on the complexity of the hypothesis class $\mathcal{H}$. By restricting the learner to a specific hypothesis class $\mathcal{H}$, we introduce an *inductive bias*. Even if the true labeling function does not belong to $\mathcal{H}$, the learner is bound to choose a hypothesis from the predefined hypothesis class. The complexity of the hypothesis class determines the learner's capacity, i.e., its ability to fit diverse training data. Specifically, a hypothesis class with high capacity can approximate complex functions very well but is also more sensitive to noise in the training data. This increases the risk of "over-adapting", a phenomenon referred to as *overfitting*. Conversely, a hypothesis class with insufficient capacity may fail to capture the underlying patterns, resulting in *underfitting*. This trade-off between the ability of the hypothesis to learn from the training set and its ability to generalize to unseen data is known as the *bias-complexity trade-off* and is also commonly referred to as *bias-variance trade-off*. Furthermore, the *No Free Lunch Theorem* states that there is no universal hypothesis class or learning algorithm that achieves a better generalization error across all problems. In practical applications the bias-complexity trade-off can be influenced by explicitly constraining the complexity of the hypothesis space. This idea is known as *regularization*. It introduces a preference for simpler solutions, often reducing model variability and thereby the risk of overfitting.

Complexity of the hypothesis space. The principle of *Occams' razor* states, that among multiple hypothesis classes capable of approximating the true labeling function, the "simplest" class is usually preferable. In the context of this thesis, the simplicity of a hypothesis class is typically understood as its capacity, which can be informally measured by the number of degrees of freedom it allows.

When the hypothesis class $\mathcal{H}$ is too simple, the hypothesis returned by the ERM may exhibit high bias, as it fails to capture relevant patterns from the data, leading to underfitting. In this case, even the best ERM $\psi_S \in \mathcal{H}$ cannot represent the underlying data distribution. The fitted hypothesis shown on the left of Figure 3.1 illustrates underfitting in a binary classification setting, where the learned hypothesis fails to adequately separate the two classes.

Conversely, if the hypothesis class $\mathcal{H}$ is too complex, the chosen prediction hypothesis may have low bias but becomes highly sensitive to the training data, including its noise. The ERM may then return a predictor $\psi_S \in \mathcal{H}$ that predicts the labels of the training set S perfectly, achieving zero empirical risk. However, this hypothesis is likely to overfit, i.e., it fails to generalize to previously unseen data points. This case is shown in the middle of Figure 3.1, where the hypothesis perfectly fits the data points, but fails to capture the true decision boundary.

Therefore, the objective is to find a favorable trade-off between bias and complexity through an appropriate choice of the hypothesis class $\mathcal{H}$ [40, 100]. The hypotheses depicted on the right in Figure 3.1 represents a well-fitted hypothesis that achieves low training error while maintaining good generalization and thereby avoiding underfitting and overfitting. The simplest way to restrict the hypothesis space is to limit the number of possible predictors within the hypothesis space $\mathcal{H}$. In general, this is not a trivial task. For binary classification tasks, the *Vapnik-Chervonenkis dimension* (VC dimension) is commonly used. The VC dimension of a hypothesis class $\mathcal{H}$, denoted by $\text{VCdim}(\mathcal{H})$, is defined as the largest subset $\mathcal{V} \subset \mathcal{X}$, for which there exists a labeling function $\psi \in \mathcal{H}_\mathcal{V}$, such that the empirical error is zero, i.e., $\mathcal{L}_\mathcal{V}(\psi) = 0$. This concept is known as *shattering*. If $\mathcal{H}_\mathcal{V}$ contains all possible binary labelings of V, there exists a predictor that perfectly separates the data points. Furthermore, for $\mathcal{H}_\mathcal{V} : \mathcal{V} \to \{0, 1\}$ the number of predictors in the restricted subset is upper bounded by $|\mathcal{H}_\mathcal{V}| = 2^{|\mathcal{V}|}$. The $\text{VCdim}(\mathcal{H})$ of a finite hypothesis set $\mathcal{H}$ satisfies $\text{VCdim}(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$.

The *Natarajan dimension* is a combinatorial measure of a hypothesis space, that generalizes the VC dimension for multiclass problems with $\mathcal{C}$ classes; see [100] for more details.

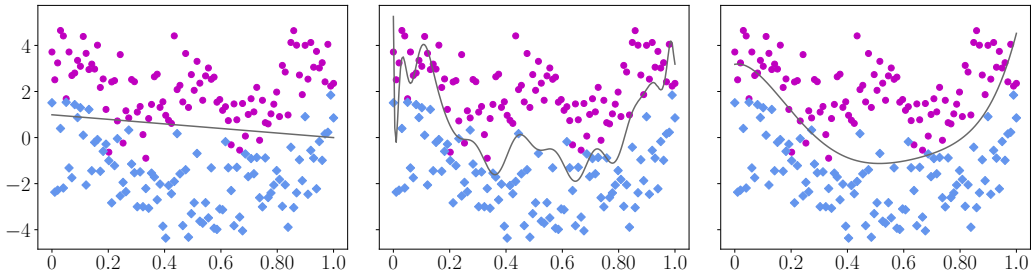


Figure 3.1: Illustration of hypotheses with varying complexities fitted to a toy dataset: underfitting (left), overfitting (middle), and a well-fitted hypothesis (right).

Error decomposition in ERM learning. For the hypothesis $\psi_S \in \mathcal{H}$ returned by $\text{ERM}_\mathcal{H}$ trained on S, the true error $\mathcal{L}_\mathcal{D}$ can be decomposed as

$$\mathcal{L}_\mathcal{D}(\psi_S) = \underbrace{\min_{\psi \in \mathcal{H}} \mathcal{L}_\mathcal{D}(\psi)}_{\epsilon_{\text{app}}} + \underbrace{\left(\mathcal{L}_\mathcal{D}(\psi_S) - \min_{\psi \in \mathcal{H}} \mathcal{L}_\mathcal{D}(\psi) \right)}_{\epsilon_{\text{est}}}$$

where

- ϵ_{app} is the *approximation error*, that measures the inductive bias caused by the restriction to a specific hypothesis class. It is possible that even the best hypothesis in $\mathcal{H}$ does not yield a good approximation of the target function.
- ϵ_{est} is the *estimation error*, that measures the difference between the approximation error and the error returned by the ERM predictor and thus measures the error caused by estimating the best function.

This error decomposition illustrates the bias-complexity trade-off: choosing a larger hypothesis class typically reduces the approximation error ϵ_{app} but increases the risk of overfitting and therefore the estimation error ϵ_{eps}. Reducing the complexity of the hypothesis class reduces the estimation error but increases the approximation error which might result in underfitting.

The No Free Lunch Theorem. Agnostic PAC learnability guarantees that it is possible to find prediction rules that correctly classify most data points with a certain probability. However, this does not imply the existence of a universal learner, that is, a learner capable of solving any task without prior knowledge. According to Wolpert, no learner is universally dominant over any other [118]. Formally, Wolpert introduced the *No Free Lunch Theorem*, which states that, averaged over all possible data-generating distributions, all learning algorithms have the same expected performance. Consequently, there is no universally optimal learning algorithm. Instead, the goal is to identify the learning algorithm that is best suited to the underlying data distribution by making reasonable

assumptions about the underlying data distribution of the task at hand, thereby enabling effective incorporation of prior knowledge. Furthermore, [100] showed that, for a learning algorithm trained on a binary classification task, even if there exists a hypothesis that perfectly represents the distribution of the underlying data, the learning algorithm may fail unless there are enough training samples or the hypothesis class is appropriately chosen.

3.2 Deep Neural Networks

(Artificial) neural networks (NNs) form a class of parameterized machine learning methods. With parameters $\theta \in \Theta \subseteq \mathbb{R}^d$, choosing a specific neural network model architecture corresponds to restricting the hypothesis class $\mathcal{H}$ to a set of parameterized functions, where each function is parameterized by a fixed number of trainable weights.

During the learning process these weights are adapted to minimize a loss function over the training data, thereby selecting the best function within the restricted hypothesis class $\mathcal{H}$ to approximate the true labeling function. This review is based on [100].

In line with the historical development of neural networks we begin by introducing multilayer perceptrons as the foundational class of feedforward neural networks. We then discuss convolutional neural networks (CNNs), which exploit the spatial structure in the input data through local connectivity and weight sharing. For this section, we follow [40]. Finally, we introduce residual neural networks (ResNets), based on [53], a specific CNN architecture designed to mitigate the vanishing gradient problem through identity skip connections.

While residual convolutional neural networks remain powerful for image classification tasks or as backbones for object detection and segmentation tasks, recent advances have incorporated self-attention mechanisms such as Transformer architectures [29,112]. However, such architectures are beyond the scope of this thesis.

3.2.1 Feedforward Neural Networks

A neural network computes its output by applying a series of transformations to the input data. In the feedforward (inference) phase, the input is successively multiplied with weight matrices and passed through non-linear activation functions. In essence, a neural network is a composition of linear transformations and pointwise non-linearities, enabling it to model complex, non-linear relationships.

The simplest form of a neural network is the *single-layer perceptron*. It models a binary classification problem where the input $z = (z_1, \ldots, z_n) \in \mathcal{X} \subseteq \mathbb{R}^n$ is mapped to an output $y \in \{0, 1\}$ using a parameterized function $\psi_\theta : \mathcal{X} \to \{0, 1\}$. The parameters

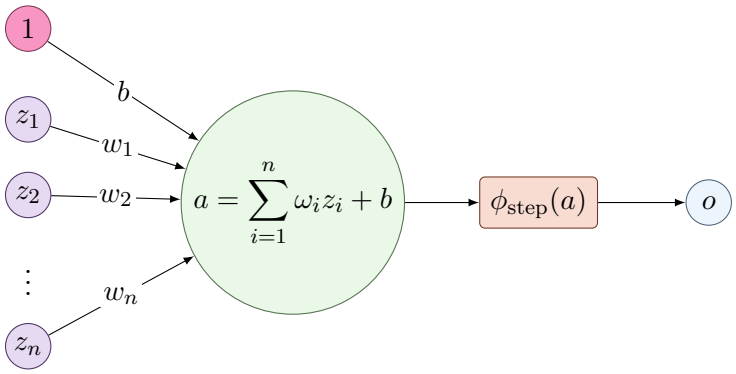


Figure 3.2: Schematic of a single-layer perceptron. The inputs $z \in \mathbb{R}^n$ are weighted and summed, together with a bias term, to form the pre-activation a. Applying the step activation function yields the output $o = \phi_{\text{step}}(a)$.

$\theta = (\omega, b)$ comprise a weight vector $\omega \in \mathbb{R}^n$ and a bias term $b \in \mathbb{R}$. This perceptron computes the weighted sum of the input values x_i with a bias b, i.e.,

$$a = \sum_{i=1}^{n} \omega_i z_i - b = \omega^T z - b$$

which is then passed through a non-linear activation function $\phi_{\text{step}} : \mathbb{R} \to \mathbb{R}$, defined as

$$\phi_{\text{step}}(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

This step function acts as a threshold on the output of the affine transformation. Thus, the output of the perceptron is given by

$$o = \psi_\theta(z) = \phi_{\text{a}} = \phi_{\text{step}}(\omega^T z - b),$$

which returns 1 if the weighted sum of the input exceeds the bias, and 0 otherwise. Figure 3.2 illustrates a single perceptron with a step activation function. The computational complexity for a single-layer perceptron is $\mathcal{O}(n)$ since it takes one dot product of length n to compute the output a of the affine transformation. Rosenblatt [92] showed that for linearly separable data, single-layer perceptrons can solve binary classification problems. However, single-layer perceptrons fail on tasks with non-linear decision boundaries.

Multilayer perceptron. To solve non-linearly separable problems, a single-layer perceptron can be extended by stacking multiple perceptrons, resulting in a *multilayer perceptron*. A multilayer perceptron typically consists of three types of layers: an *input layer*, one or more *hidden layers*, and an *output layer*. Each layer is represented by a parameterized mapping

$$\psi_\theta^{(\ell)} : \mathbb{R}^{n_{\ell-1}} \longrightarrow \mathbb{R}^{n_\ell},$$

where $\ell \in \{1, \dots, L\}$ denotes the layer index for a total of $L \in \mathbb{N}$ layers. The input layer $\psi_\theta^{(1)} : \mathcal{X} \to \mathbb{R}^{n_1}$ receives the input features and passes them to the subsequent hidden layers $\psi_\theta^{(2)}, \dots, \psi_\theta^{(L-1)}$. Each hidden layer consists of *neurons* that apply an affine transformation followed by a pointwise non-linear activation function, enabling the multilayer perceptron to capture increasingly abstract representations of the input. The final output layer $\psi_\theta^{(L)}$ produces the multilayer perceptron's output, which may be a scalar, a vector, or a probability distribution, depending on the task.

A multilayer perceptron can be viewed as an acyclic directed graph $\mathrm{G} = (V, E)$ where V is a set of nodes, representing the neurons, and E a set of directed edges, connecting the nodes. Each layer corresponds to a subset $V_\ell \subset V$ and every neuron in $V_{\ell-1}$ is connected to every neuron in V_ℓ, making the layers *fully connected* (fc). Each edge is assigned a weight $\omega : E \to \mathbb{R}$, which determines the strength of the connection and is adjusted during the training process. The total number of weights determines the model complexity with respect to learnable parameters. The *depth* of a multilayer perceptron refers to the total number of layers L, while the width is defined by the maximal number n_ℓ of neurons in the ℓ-th layer.

Formally, for each neuron $j \in \{1, \dots, n_\ell\}$ in layer ℓ, the pre-activation is given by

$$a_j^{(\ell)} = (\omega_j^{(\ell)})^{\mathrm{T}} z + b_j^{(\ell)}$$

where ω_j is the weight vector and $b_j \in \mathbb{R}$ the bias.

Stacking the weight vectors row-wise yields the weight matrix $W^{(\ell)} = ((\omega_1^{(\ell)}), \dots, (\omega_{n_\ell}^{(\ell)}))$, so that for an output $z \in \mathbb{R}^{n_{\ell-1}}$ of the previous layer, the linear transformation becomes

$$a^{(\ell)} = W^{(\ell)} z + b^{(\ell)} \in \mathbb{R}^{n_\ell}.$$

The computational complexity of such a layer is $\mathcal{O}(n_{\ell-1} \cdot n_\ell)$, since n_ℓ dot products of length $n_{\ell-1}$ must be computed. To model non-linear relationships, the affine transformation is followed by a non-linear activation function ϕ that is applied elementwise. Each layer is associated with its own set of parameters $\theta_\ell = (W^{(\ell)}, b^{(\ell)})$ and activation function ϕ_ℓ, and we define for every layer ℓ:

$$\psi_{\theta_\ell}^{(\ell)} = \phi_\ell(W^{(\ell)} z + b^{(\ell)}) = \phi_\ell(a^{(\ell)}).$$

The parameters of the entire network are given by $\theta = (\theta_1, \dots, \theta_L)$. The resulting full multilayer perceptron defines the *forward pass* as the composition of L layer mappings

$$\begin{aligned} \psi_\theta : \mathbb{R}^{n_0} &\longrightarrow \mathbb{R}^{n_L} \\ z &\longmapsto \psi_\theta(z) = \psi_{\theta_L}^{(L)} \circ \psi_{\theta_{L-1}}^{(L-1)} \circ \dots \circ \psi_{\theta_1}^{(1)}. \end{aligned}$$

An example of such a forward pass is illustrated in Figure 3.3.

The *Universal Approximation Theorem* states that, under mild conditions on the activation function, neural networks with at least one hidden layer of sufficient width can

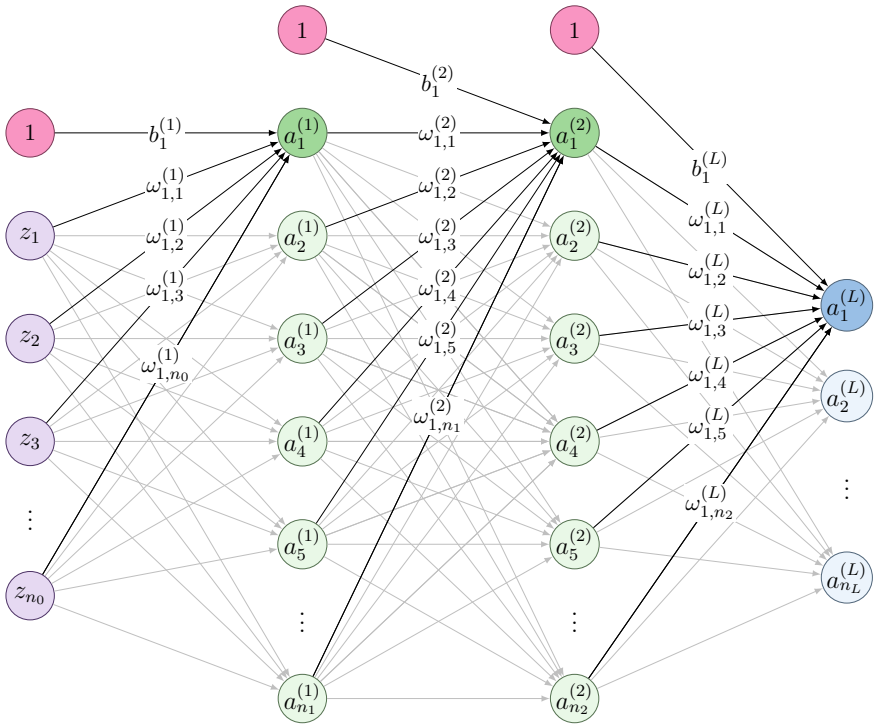


Figure 3.3: Schematic of an multilayer perceptron with depth 2, n_ℓ neurons per layer, and explicit bias terms. The input, hidden and output layers are shown in distinct colors. The connections contributing to the computation of the first neuron in each layer are highlighted.

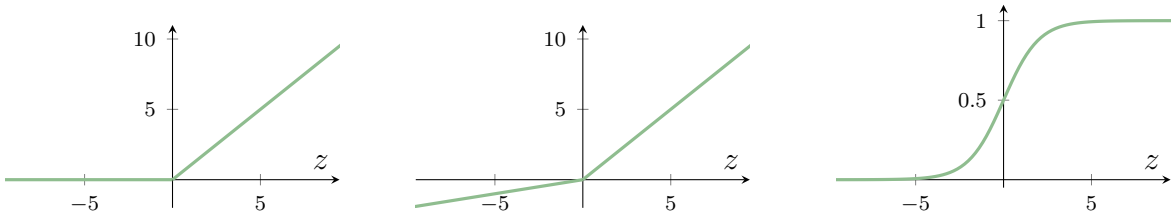
approximate any continuous function on a compact domain to arbitrary accuracy [25,57]. This foundational result justifies the use of multilayer perceptrons to model highly non-linear relationships in real-world data.

Activation functions. To enable the multilayer perceptron to learn complex and non-linear patterns and obtain universal approximation properties, each (hidden) layer is associated with a non-linear activation function $\phi : \mathbb{R} \longrightarrow \mathbb{R}$, which is typically applied elementwise to the output of layer ℓ.

One commonly used activation function is the logistic function, which is also referred to as the *sigmoid* function in the context of deep learning:

$$\phi_{\text{sigmoid}}(a) = \frac{1}{1 + e^{-a}} \quad \in (0, 1).$$

While ϕ_{sigmoid} has the desirable property of being differentiable, it saturates when the input a is either very large or very small. The derivatives corresponding to these regions approach zero and consequently the gradients become very small. The nested composition of the layers causes even smaller gradients to be multiplied repeatedly.



(a) ϕ_{ReLU} (b) $\phi_{\text{leaky ReLU}}$, slope $\alpha = 0.2$ (c) ϕ_{sigmoid}

Figure 3.4: Visualization of different activation functions.

Consequently, this effect compounds and results in *vanishing gradients*. The calculation of gradients and their role in training NNs and the issue of vanishing gradients are discussed in Section 3.2.2 in more detail.

To address the vanishing gradient problem associated with sigmoid activation functions, the *rectified linear unit* (ReLU) activation function is commonly used, defined as

$$\phi_{\text{ReLU}}(a) = \max(0, a) = \frac{a + |a|}{2} = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

The variable a denotes the input to the activation function and is therefore referred to as *pre-activation*. It is typically the output of the affine transformation of the current layer. Explicitly, ϕ_{ReLU} is a piecewise linear function that is 0 if $a \leq 0$ and equals the input a if a is positive. The popularity of ReLU is due to its computational efficiency, since it only requires a comparison and a maximum operation. The application of ϕ_{ReLU} in multilayer perceptrons also preserves properties which simplify optimization with gradient-based methods and are beneficial to the generalization of the multilayer perceptron [40].

The fact that negative values are set to zero bears the risk of dying ReLU. For the values on the negative half-axis the gradient is zero, so that the corresponding weights are not adjusted in the training process and remain "*dead*", which can cause a loss in the capacity of the multilayer perceptron. Allowing a small non-zero slope preserves activity in the neurons and ensures that the weights continue to be adjusted. This so-called leaky ReLU is given by

$$\phi_{\text{leaky ReLU}(a)} = \max(\beta \cdot a, a) = \begin{cases} a & \text{if } a > 0 \\ \beta \cdot a & \text{otherwise,} \end{cases} \tag{3.10}$$

where $0 < \beta < 1$ is a fixed parameter controlling the slope. The drawback of this variant is that negative activations are no longer suppressed completely, which may reduce the sparsity induced by ReLU. These common activation functions are illustrated in Figure 3.4. Further generalizations of ϕ functions or other activation functions are not discussed in this work.

The final output of a multilayer perceptron for classification tasks is probabilities estimates of class labels $y \in \mathcal{Y}$ for an input $z \in \mathcal{X}$. To that end, the output $a^{(L)}$ of the

output layer is transformed by a *softmax* function

$$\text{softmax}(a_i) = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \quad \in [0,1], \quad i = 1, \ldots, n \tag{3.11}$$

to normalize $a^{(L)}$ into a probability distribution. To obtain the final class prediction, a decision rule is applied to the output of the softmax layer Equation (3.11). This final decision rule is typically based on the maximum a-posteriori principle

$$\hat{y} = \underset{y' \in \mathcal{Y}}{\text{argmax}}\, \psi_\theta(z)_{y'}.$$

Multilayer perceptrons perform well when the input data space $\mathcal{X}$ is low-dimensional. However, they are affected by the *curse of dimensionality*, a phenomenon that occurs when dealing with high-dimensional data. In practical pattern recognition applications, the input space is composed of numerous variables, which complicates the distinction between classes [5].

3.2.2 Minimization of Loss Functions

The goal of learning is to find a parameterized hypothesis $\psi_\theta \in \mathcal{H}$ that minimizes the true loss $\mathcal{L}_\mathcal{D}(\psi) = \mathbb{E}_{z\sim\mathcal{D}}[\mathcal{L}(\psi, z)]$ over the data-generating distribution $\mathcal{D}$, where $\theta \in \Theta \subseteq \mathbb{R}^n$ denotes the trainable parameters. However, in practice the true distribution $\mathcal{D}$ is unknown. Instead, the empirical risk $\mathcal{L}_S(\psi)$ computed on a training set S is used to proxy the true risk.

Gradient descent is an iterative optimization method used to minimize the empirical risk. Given a parameterized hypothesis ψ_θ, in every optimization step the parameters θ are updated in the direction of the negative gradient of the loss function. Since the full data-generating distribution $\mathcal{D}$ is inaccessible, the gradient of the true loss $\nabla\mathcal{L}_\mathcal{D}(\psi_\theta)$, cannot be computed directly.

Instead, *stochastic gradient descent* (SGD) approximates the gradient using randomly sampled (mini-) batches of the training data to update the parameters towards a noisy but unbiased direction.

Throughout this section, we assume that ψ_θ is sufficiently smooth, i.e., differentiable with respect to its parameters θ. To formalize this, we consider a parameterized family of functions

$$\psi : \Theta \times \mathcal{X} \longrightarrow \mathcal{Y} \quad \text{with} \quad \theta \in \Theta \subseteq \mathbb{R}^n,$$

where θ denotes the parameter vector. For a fixed input $z \in \mathcal{X}$, we regard $\psi(\theta, z)$ as a function of θ, i.e., $\psi_\theta := \psi(\theta, z) : \mathbb{R}^n \to \mathcal{Y}$. In the context of stochastic gradient descent we are interested in evaluating the gradients with respect to θ. When the input is implicitly fixed, we may write $\psi(\theta)$ as shorthand for $\psi(\theta, z)$.

We begin by introducing gradient descent, then generalize it to the stochastic setting, and ultimately describe the learning process using stochastic gradient descent.

Gradient Descent. In this section we introduce the gradient descent algorithm. Let $\psi : \mathbb{R}^n \to \mathbb{R}$ be a differentiable convex function. The gradient of $\psi(\theta)$ at a parameter vector $\theta \in \mathbb{R}^n$ is defined as the vector of partial derivatives:

$$\nabla\psi(\theta) = \left[\frac{\partial\psi(\theta)}{\partial\theta_1}, \ldots, \frac{\partial\psi(\theta)}{\partial\theta_n}\right]^{\mathrm{T}}.$$

We initialize the parameter vector with an initial guess $\theta^{(1)}$ and choose a positive step size $\mu > 0$, referred to as the *learning rate*. Typically, the learning rate is decreased during training [26]. In each iteration $\tau = 1, \ldots, M$ the gradient is evaluated and the parameters are updated by stepping in the direction of the steepest descent, which is defined as

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \mu\nabla\,\psi(\theta^{(\tau)}). \tag{3.12}$$

Extensions, such as *momentum* methods with parameter $\beta \in [0, 1)$ can be used to mitigate stagnation and accelerate convergence [90]. The gradient $\nabla\psi(\theta)$ points in the direction of the greatest increase of ψ near θ. Since the goal is to minimize the function, i.e., to find at least a local minimum, the algorithm updates the parameters by subtracting the gradient, thereby reducing the function value at each step. After M iterations, the gradient descent algorithm returns the final iterate $\theta^{(M+1)}$.

The gradient of the empirical risk over a training set $S = (z_1, \ldots, z_t)$ is given by

$$\nabla\mathcal{L}_S(\theta) = \frac{1}{t}\sum_{i=1}^{t}\nabla_\theta\mathcal{L}(\psi_\theta, z_i). \tag{3.13}$$

Note that, for quadratic problems, a steepest gradient descent step (cf. Equation (3.12)) has the form of a Richardson iteration (cf. Equation (2.19)). For quadratic objective functions, the negative gradient coincides with the residual, so both methods update the current iterate along the residual direction. However, while the optimal step length μ in gradient descent may vary between iterations, the classical Richardson iteration uses a fixed relaxation parameter [96].

Stochastic Gradient Descent. The stochastic gradient descent algorithm extends the gradient descent algorithm to settings where the training set S is large and computing the full gradient at each iteration is computationally unfeasible.

In many practical applications, good generalization requires a large training set S with $t = |S|$. To avoid memory bottlenecks, stochastic gradient descent approximates the full gradient using only a *mini-batch*, i.e., a uniformly drawn subset $B \subseteq S$ of size $t' \ll t$.

Formally, instead of computing the complete gradient of the empirical risk as in Equation (3.13), stochastic gradient descent computes an unbiased estimate of the gradient

using the mini-batch B:

$$v = \frac{1}{t'} \nabla \sum_{z_i=1}^{t'} \mathcal{L}(\psi_\theta, z_i) \quad \text{with} \quad z_i \in B. \tag{3.14}$$

At each iteration τ, we sample a mini-batch $B^{(\tau)} \subseteq S$, such that the gradient depends on both the current parameters $\theta^{(\tau)}$ and the current batch. The vector $v^{(\tau)}$ is a random vector whose conditional expectation given the current parameters equals the full gradient of the empirical risk, i.e., $\mathbb{E}[v^{(\tau)}|\theta^{(\tau)}] = \nabla_\theta \mathcal{L}_S(\theta^{(\tau)})$. Replacing the full gradient in Equation (3.12) by the stochastic estimator (Equation (3.14)) yields the SGD update

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \mu \cdot v^{(\tau)}.$$

The stochastic gradient descent algorithm to minimize a function $\psi(\theta)$ is summarized in Algorithm 3.

Algorithm 3 Minimize $\psi(\theta)$ with SGD

Require: $\mu \in \mathbb{R}_{>0}, M \in \mathbb{N}$, initial parameters $\theta^{(1)}$
1: **for** $\tau = 1, 2, \ldots, M$ **do**
2: Sample $B^{(\tau)} \sim \mathcal{U}(S)$
3: Compute $v^{(\tau)}$ randomly, s.t. $\mathbb{E}[v^{(\tau)}|\theta^{(\tau)}] = \nabla_\theta \mathcal{L}_S(\theta^{(\tau)})$
4: $\theta^{(\tau+1)} = \theta^{(\tau)} - \mu v^{(\tau)}$
5: **end for**
6: **return** $\theta^{(M+1)}$

3.2.3 Parameter Estimation

In the context of supervised learning with deep neural networks, choosing the best hypothesis $\psi_\theta \in \mathcal{H}_\theta = \{\psi_\theta : \theta \in \Theta\}$ corresponds to learning the model parameters θ. These parameters are typically learned by maximizing the likelihood of the observed data under the model's predictive distribution. This principle is referred to as *maximum likelihood estimation*. Since the empirical risk induced by certain loss functions, such as the 0-1 loss in classification, is generally not differentiable, surrogate objective functions are commonly employed in practice.

Maximum likelihood estimation. Given a dataset S of t i.i.d. samples, a hypothesis ψ_θ induces a predictor

$$\hat{y}(z)_{\psi_\theta} := \operatorname*{argmax}_{y \in \mathcal{Y}} \psi_\theta(y|z),$$

with the optimal parameters θ estimated by *maximum likelihood estimation*. Formally, for a single observation (z, y) we want to solve

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmax}} \, \psi_\theta \left(y | z \right),$$

by maximum likelihood estimation. Let $S = \{(z_i, y_i)\}_{i=1}^t$ be the training set of t i.i.d. examples. The set of input features is denoted by $S|_z = \{z_1, \ldots, z_t\} \in \mathcal{X}$ and the corresponding labels are given by $S|_y = \{y_1, \ldots, y_t\} \in \mathcal{Y}$. Under the i.i.d. assumption, the likelihood of the observed labels given the inputs and model parameters θ is

$$\psi_\theta(S|_y, S|_z) = \prod_{i=1}^{t} \psi_\theta(y_i | z_i), \tag{3.15}$$

where $\psi_\theta(y|z)$ is the model's predicted conditional probability.

Negative log-likelihood. Maximizing Equation (3.15) is equivalent to minimizing the *negative log-likelihood*

$$\mathcal{L}_S(\psi_\theta) := -\log \left(\prod_{i=1}^{t} \psi_\theta(y_i | z_i) \right) = -\sum_{i=1}^{t} \log(\psi_\theta(y_i | z_i)).$$

Thus, taking the negative logarithm transforms the maximum likelihood estimation problem into

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{t} (-\log(\psi_\theta(y_i | z_i))).$$

As the true probability distribution $\mathcal{D}$ is unknown, the expected conditional loss

$$\mathcal{L}_{\mathcal{D}}(\psi_\theta) = \mathbb{E}_{(z,y)\sim\mathcal{D}}[-\log(\psi_\theta(y|z))] \tag{3.16}$$

cannot be evaluated directly. Instead, it is approximated using the training set.

Connection to cross-entropy and Kullback-Leibler divergence. We can interpret the term $-\log(\psi(y|z))$ as the log-loss under a model probability distribution $\mathcal{M}_\theta(z, y) \approx \mathcal{D}(z, y)$. Here $\mathcal{D}(z, y) = \mathcal{D}(z) D_z(y)$ denotes the true joint probability distribution over inputs z and labels y. $\mathcal{D}(z)$ is the marginal distribution of the input $z \in \mathcal{X}$ and $\mathcal{D}_z(y) = \mathcal{D}(y|z)$ is the conditional distribution of y given z. Analogously we write the model's joint distribution $\mathcal{M}_\theta(z, y) = \mathcal{D}(z)\mathcal{M}_{\theta,z}(y)$, where $M_{\theta,z}(y) = \psi_\theta(y|z)$ is the corresponding conditional distribution given z. Since $\mathcal{D}(z)$ uis independent of the model parameters, the model only estimates the conditional label distribution. For a fixed z, the expected conditional loss (cf. Equation (3.16)) can be written in the terms of the expected *cross-entropy* (CE)

$$\mathbb{E}_{(z,y)\sim\mathcal{D}}[-\log(\psi_\theta(y|z))] = \mathbb{E}_{z\sim\mathcal{D}}[\mathrm{CE}(D_z || M_{\theta,z})],$$

where

$$\mathrm{CE}(\mathcal{D}_z||\mathcal{M}_{\theta,z}) = \mathbb{E}_{y\sim\mathcal{D}_z}[-\log(\mathcal{M}_{\theta,z}(y))]. \tag{3.17}$$

Here "||" indicates that cross-entropy is viewed as a function that takes two probability distributions as its arguments, following the convention used in information theory.

Furthermore, the conditional entropy of the true label distribution is

$$\mathrm{H}(\mathcal{D}_z) = \mathbb{E}_{y\sim\mathcal{D}_z}[-\log(\mathcal{D}_z(y))],$$

which quantifies the uncertainty of the true label distribution at a fixed input z.

The (conditional) *Kullback-Leibler* (KL) divergence between the true conditional distribution $\mathcal{D}_z$ and the model's conditional distribution $\mathcal{M}_{\theta,z}$ is

$$\begin{aligned}
\mathrm{KL}(\mathcal{D}_z||\mathcal{M}_{\theta,z}) &= \mathbb{E}_{y\sim\mathcal{D}_z}\left[\log\left(\frac{\mathcal{D}(z)\mathcal{D}_z(y)}{\mathcal{D}(z)\mathcal{M}_{\theta,z}(y)}\right)\right] \\
&= \mathbb{E}_{y\sim\mathcal{D}_z}[\log(\mathcal{D}_z(y)) - \log(\mathcal{M}_{\theta,z}(y))] \\
&= \mathbb{E}_{y\sim\mathcal{D}_z}[-\log(\mathcal{M}_{\theta,z}(y))] - \mathbb{E}_{y\sim\mathcal{D}_z}[-\log(\mathcal{D}_z(y))] \\
&= \mathrm{CE}(\mathcal{D}_z||\mathcal{M}_{\theta,z}) - \mathrm{H}(\mathcal{D}_z).
\end{aligned}$$

The KL divergence therefore depends only on the conditional distributions and is independent of the marginal distribution of the input.

Averaging over all $z \sim \mathcal{D}$, the KL divergence is given by

$$\begin{aligned}
\mathrm{KL}(\mathcal{D}||\mathcal{M}_\theta) &= \mathbb{E}_{z\sim\mathcal{D}}\left[\mathrm{CE}(\mathcal{D}_z||\mathcal{M}_{\theta,z}) - \mathrm{H}(\mathcal{D}_z)\right] \\
&= \mathbb{E}_{(z,y)\sim\mathcal{D}}[-\log(\mathcal{M}_{\theta,z}(y))] - \mathbb{E}_{(z,y)\sim\mathcal{D}}[-\log(\mathcal{D}_z(y))] \\
&= \mathrm{CE}(\mathcal{D}||\mathcal{M}_\theta) - \mathrm{H}(\mathcal{D}).
\end{aligned}$$

Thus, minimizing the expected cross-entropy defined in Equation (3.17) is equivalent to minimizing the KL divergence, since $\mathrm{H}(\mathcal{D})$ is constant with respect to parameters θ.

Backpropagation. Gradient-based methods such as stochastic gradient descent are motivated by the optimization of convex functions. Although the loss surfaces of neural networks are generally non-convex, their simplicity and the availability of efficient gradient computations make gradient-based methods well-suited for training neural networks. In practice, these methods typically converge to a good approximate minimizer of the empirical risk, even though finding the ERM is, in general, NP-hard [6, 100].

In the learning process of a multilayer perceptron the gradients for stochastic gradient descent are calculated using the *backpropagation* algorithm [93]. In each iteration τ, the weights ω_τ are adjusted based on the current loss $\mathcal{L}(\hat{y}, y)$. The gradient is computed with respect to each weight separately by calculating the respective partial derivatives. When calculating the gradients for the weight matrix $W^{(\ell)}$, the weights in other layers

remain fixed. To formalize this, consider a subnetwork defined by layers $V_\ell, \dots, V_L$ as a function of neurons in layer ℓ. Its associated loss function is $\mathcal{L}^{(\ell)} : \mathbb{R}^{n_\ell} \to \mathbb{R}$.

Recall that the input to layer ℓ is the output of the previous layer $o^{(\ell-1)} \in \mathbb{R}^{n_{\ell-1}}$. After applying the affine transformation and non-linear activation ϕ, we have the pre-activation $a^{(\ell)} = W^{(\ell)} o^{(\ell-1)}$ and the output $o^{(\ell)} = \phi(a^{(\ell)})$. Thus, the loss restricted to the subnetwork starting at V_ℓ can be written as a function of $W^{(\ell)}$:

$$g^{(\ell)}(W^{(\ell)}) = \mathcal{L}^{(\ell)}(\phi^{(\ell)}(a^{(\ell)})) = \mathcal{L}^{(\ell)}(\phi(W^{(\ell)} o^{(\ell-1)} + b^{(\ell)})). \tag{3.18}$$

For simplicity, we omit the bias term in the following derivation. To vectorize the weight matrix, the rows of $W^{(\ell)}$ are concatenated as a column vector $\omega^{(\ell)} \in \mathbb{R}^{n_{\ell-1} \cdot n_\ell}$. That allows us to treat the loss of the subnetwork Equation (3.18) as a scalar function of a vector. We also define a block-diagonal matrix $O^{(\ell-1)} \in \mathbb{R}^{n_\ell \times (n_{\ell-1} \cdot n_\ell)}$ whose diagonal block entries are copies of $(o^{(\ell-1)})^{\mathrm{T}}$:

$$O^{(\ell-1)} = \begin{pmatrix} (o^{(\ell-1)})^{\mathrm{T}} & 0 & \dots & 0 \\ 0 & (o^{(\ell-1)})^{\mathrm{T}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (o^{(\ell-1)})^{\mathrm{T}} \end{pmatrix}.$$

This allows us to rewrite the affine transformation as $a^{(\ell)} = W^{(\ell)} o^{(\ell-1)} = O^{(\ell-1)} \omega^{(\ell)}$. Applying the chain rule to $g^{(\ell)}$ yields the nested composition $\omega^{(\ell)} \mapsto a^{(\ell)} \mapsto o^{(\ell)} \mapsto \mathcal{L}^{(\ell)}(o^{(\ell)})$. Therefore, the Jacobian matrix[3] J is given by

$$J_{\omega^{(\ell)}}(g^{(\ell)}) = J_{o^{(\ell)}}(\mathcal{L}^{(\ell)}) J_{a^{(\ell)}}(o^{(\ell)}) J_{\omega^{(\ell)}}(a^{(\ell)})$$

Substituting each term, we get

$$J_{\omega^{(\ell)}}(g^{(\ell)}) = \delta^{(\ell)} \mathrm{diag}(\phi'(a^{(\ell)})) O^{(\ell-1)},$$

where $J_{o^{(\ell)}}(\mathcal{L}^{(\ell)}) = \delta^{(\ell)} := \nabla_{o^{(\ell)}} \mathcal{L}^{(\ell)} \in \mathbb{R}^{1 \times n_\ell}$. Expanding the Jacobian row-wise yields

$$J_{\omega^{(\ell)}}(g^{(\ell)}) = \left[\delta_1^{(\ell)} \phi'(a_1^{(\ell)}) (o^{(\ell-1)})^{\mathrm{T}}, \dots, \delta_{n_\ell}^{(\ell)} \phi'(a_{n_\ell}^{(\ell)}) (o^{(\ell-1)})^{\mathrm{T}} \right].$$

In words: the j-th neuron in layer ℓ contributes a gradient term proportional to its error signal $\delta_j^{(\ell)}$, scaled by the derivative of its activation function $\phi'(a_j^{(\ell)})$ and multiplied by the activations of the previous layer $o^{(\ell-1)}$. Equivalently, for the j-th neuron in layer ℓ, the gradient with respect to its incoming weight vector $W_j^{(\ell)}$ is given by

$$\nabla_{W_j^{(\ell)}} g^{(\ell)} = \delta_j^{(\ell)} \phi'(a_j^{(\ell)}) o^{(\ell-1)}$$

and the gradient of the subnetwork loss with respect to $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ is

$$\nabla_{W^{(\ell)}} g^{(\ell)} = \left(\delta^{(\ell)} \odot \phi'(a^{(\ell)}) (o^{(\ell-1)})^{\mathrm{T}} \right),$$

[3]The Jacobian matrix contains the values of the first-order partial derivatives.

where $\odot$ denotes elementwise multiplication.

Now, the gradient of the loss $\mathcal{L}^{(\ell)}$ at $o^{(\ell)}$, i.e., $\delta^{(\ell)} = J_{o^{(\ell)}}(\mathcal{L}^{(\ell)})$ is calculated recursively for every layer ℓ. For the last layer L the loss is given by $\mathcal{L}^{(L)} = \mathcal{L}(\hat{y}, y)$ and the corresponding gradient is

$$\delta^{(L)} = J_{o^{(L)}}(\mathcal{L}^{(L)}) = \phi'(\hat{y}).$$

For each hidden layer $\ell \in \{1, \dots, L-1\}$, the loss of the corresponding subnetwork starting at V_ℓ, can be expressed as a function of the ℓ-th output:

$$\mathcal{L}^{(\ell)}(o^{(\ell)}) = \mathcal{L}^{(\ell+1)}(\phi(W^{(\ell)} o^{(\ell)})).$$

Applying the chain rule yields the recursive relation:

$$\delta^{(\ell)} = J_{o^{(\ell)}}(\mathcal{L}^{(\ell)}) = J_{o^{(\ell+1)}}(\mathcal{L}^{(\ell+1)}) J_{a^{(\ell+1)}}(o^{\ell+1}) J_{o^{(\ell)}}(a^{(\ell+1)}).$$

Here $J_o^{(\ell)}$ denotes the Jacobian with respect to the output $o^{(\ell)}$. Since $a^{(\ell+1)} = W^{(\ell)} o^{(\ell)}$ and $o^{(\ell+1)} = \phi(a^{(\ell+1)})$, this simplifies to

$$\delta^{(\ell)} = \delta^{(\ell+1)} \mathrm{diag}(\phi'(a^{(\ell+1)})) W^{(\ell)}.$$

The forward pass and the subsequent backpropagation for a single sample $(z, y) \in S$ in a multilayer perceptron represented by a layered graph (V, E) with weight matrices $W^{(\ell)}$ are summarized in Algorithm 4.

3.2.4 Convolutional Neural Networks

CNNs [73] are a specialized class of feedforward neural networks, designed to process data with grid-like topologies such as images, which can be represented as a two-dimensional grid of pixels. In a standard multilayer perceptron, every node of layer ℓ is connected to every node of layer $\ell + 1$. In contrast, CNNs exploit the spatial structure of the data by introducing local connectivity: each neuron in a convolutional layer interacts only with a local receptive field of the previous layer. This sparsity reduces the number of parameters while preserving the topology of the input. A CNN consists of various types of layers. Similar to multilayer perceptrons, CNNs begin with an input layer that receives raw image data, and end with an output layer, typically a fully connected layer followed by an argmax decision rule. However, the hidden layers are typically convolutional layers, based on discrete convolution operators, which process the data spatially. A convolution applies a *kernel* (or filter) that slides across the input, aggregating local information. A kernel is defined by its parameters (weights), while the convolutional layer usually includes a bias. The resulting *feature map* serves as the input to the subsequent layer. The total number of independent parameters is determined by the spatial and channel dimensions of the kernel, determines the effective degrees of freedom. Note that for certain types of convolutions, the degrees of freedom can be strictly less than the raw weight count.

Algorithm 4 Backpropagation((z, y), $\{W^{(\ell)}\}_{\ell=1}^{L}$, (V, E), ϕ)

Input: data point z, weight matrices $\{W^{(\ell)}\}_{\ell=1}^{L}$, activation ϕ,
$W_{i,j}^{(\ell)} = (v_j^{(\ell-1)}, v_i^{(\ell)})$
forward pass:
$o^{(0)} = z$
for $\ell = 1, \dots, L$ **do**
    for $i = 1, \dots, n_\ell$ **do**
        $a_i^{(\ell)} = \sum_{j=1}^{n_{\ell-1}} W_{i,j}^{(\ell)} o_j^{(\ell-1)}$
        $o_i^{(\ell)} = \phi(a_i^{(\ell)})$
    end for
end for
$\delta^{(L)} = \phi'(\hat{y})$
backward pass:
for $\ell = L-1, L-2, \dots, 1$ **do**
    for $i = 1, \dots, n_\ell$ **do**
        $\delta_i^{(\ell)} = \sum_{j=1}^{n_{\ell+1}} W_{i,j}^{(\ell)} \delta_j^{(\ell+1)} \phi'(a_j^{(\ell+1)})$
    end for
end for
return partial derivatives $\delta_i^{(\ell)} \phi'(a_i^{(\ell)}) o_j^{(\ell-1)}$ for each edge $(v_j^{(\ell-1)}, v_i^{(\ell)}) \in E$

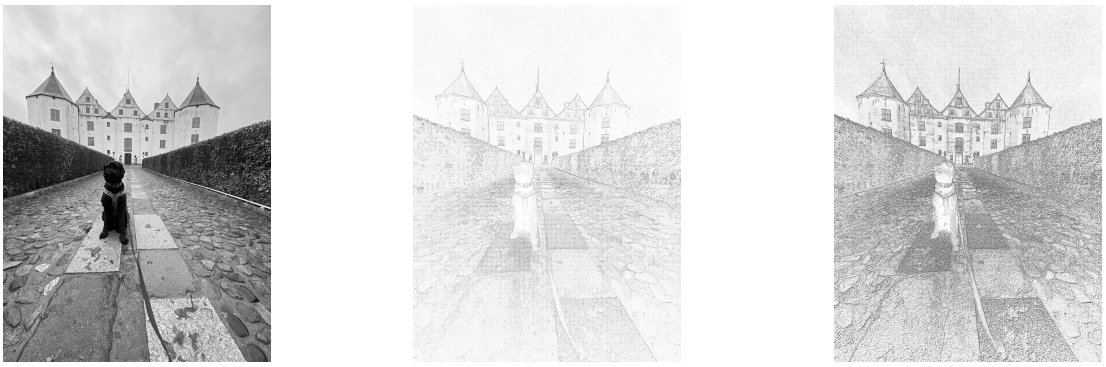
Convolutional layers are also equipped with non-linear activation functions, most commonly ReLU, which enables non-linear transformations. They are often combined with *normalization operations*, e.g., batch normalization, to mitigate the internal *covariate shift*, i.e., the change in the distribution of activations across mini-batches or between layers, thereby improving stability and robustness [61].

Typically, a stack of convolutional layers is followed by a *pooling layer* that downsamples the feature maps, reducing spatial resolution while retaining salient information. Pooling can be viewed as a special case of a convolutional operation [11,129]. A key property of CNNs is *translation (shift) equivariance*, arising in convolution layers through parameter sharing [20]. In particular, if the input shifts, the output shifts by the same amount. Convolutional operators act as multiplication operators in the Fourier domain (cf. Section 2.3.2) and thus naturally exhibit this property as they commute with translations.

Similar to multilayer perceptrons, universal approximation results hold for CNNs in general. By combining convolutional layers with non-linear activations CNNs are able to approximate continuous functions [128].

The following discussion is based on [5, 30, 40].

The convolution operation. The idea of extracting particular features, such as edges, corners or textures, from image data, has long been a fundamental approach in computer



(a) Original gray scaled image (b) 3×3 Laplacian filter (c) 5×5 Laplacian filter

Figure 3.5: Edge detection using a Laplacian filter (cf. Equation (2.17)) without normalization.

vision. Handcrafted filters like Laplacian operators are applied to the image to detect spatial variation, typically enhancing textures or the edges of objects as exemplarily demonstrated in Figure 3.5. In the context of CNNs, the characteristic operation is based on the mathematical operation. Instead of using predefined filters, the parameters are learned in order to extract the most relevant features for a given task.

An image can be represented as a three-dimensional tensor $f \in \mathbb{R}^{n \times m \times c}$, with spatial width m and height n and channel dimension c. The channel dimension for images specifies the color channels, i.e., a color image has $c = 3$ channels for red, green and blue, grayscale images are only composed of one color channel, i.e., $c = 1$. In order to simplify notation we start by considering a single channel, i.e., $c = 1$. Let $f \in \mathbb{R}^{n \times m}$ be a two-dimensional input image and $A \in \mathbb{R}^{s_{x_1} \times s_{x_2}}$ be a convolutional *kernel*, where $s_{x_1}, s_{x_2} \in \mathbb{N}$. In practice, kernels are often chosen to be centered and square. To ensure a unique center of the kernel, the kernel size $s = s_{x_1} = s_{x_2}$ is typically chosen to be odd, i.e., $s \in 2\mathbb{N} + 1$. Common choices include $s \in \{1, 3, 5, 7, 11\}$. The discrete two-dimensional convolutional operator is then defined by

$$(A * f)(x_1, x_2) = \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} f(x_1 + i, x_2 + j) \cdot A(i, j). \tag{3.19}$$

This corresponds to computing a dot product between the kernel and a local image patch by summing up the element-wise products. The procedure is depicted in Figure 3.6 for a two-dimensional convolution. Applying this computation locally and uniformly across the image produces the output feature map. After applying a non-linear activation function, the numerical values of the resulting feature map are called *activations*. However, in practice, both terms are often used interchangeably. Strictly speaking, Equation (3.19) defines the *cross-correlation*, which is commonly referred to as convolution in the deep learning literature.

As already discussed in Section 2.3.2, convolutions can naturally be formulated in terms of lattices. Let $\mathcal{A} := I_2 \in \mathbb{Z}^{2 \times 2} \subseteq \mathbb{Z}^2$ denote the lattice basis generating the two-

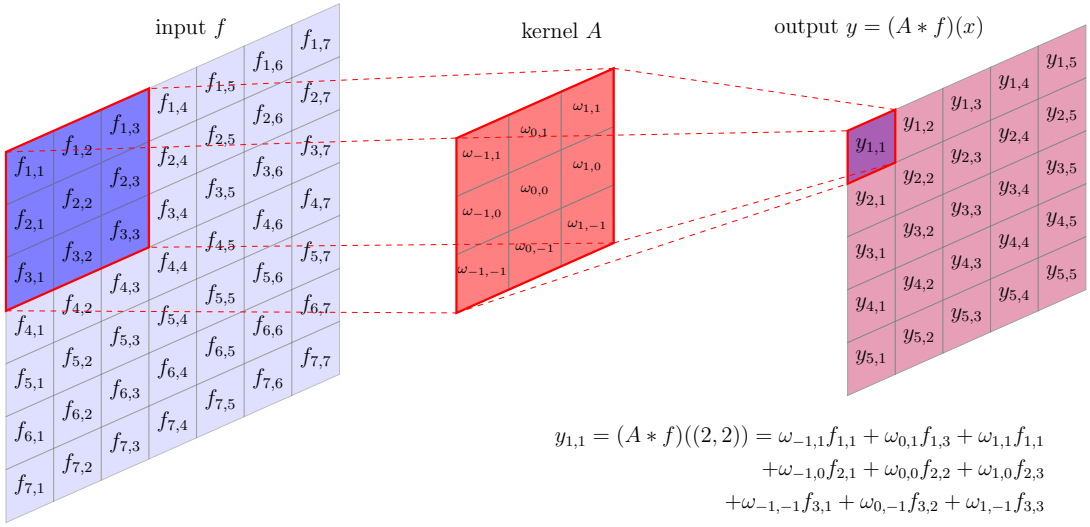


Figure 3.6: Two-dimensional convolution of a 7×7 input f with a 3×3 kernel, using a single input channel and producing a single output channel.

dimensional lattice

$$\mathbb{L}(\mathcal{A}) := \mathcal{A}\mathbb{Z}^2 = \{x = \mathcal{A} \cdot \mu : \mu \in \mathbb{Z}^2\} \subseteq \mathbb{Z}^2.$$

In the discrete setting, we consider a finite subset $\mathbb{L} \subseteq \mathbb{Z}^2$ of this lattice, which determines the spatial grid of the input. Let $f \in \mathbb{R}^{m \times n}$ denote a single-channel input and $A \in \mathbb{R}^{s_{x_1} \times s_{x_2}}$ a convolution kernel.

Typically, the kernel dimensions $s_{x_1} \times s_{x_2}$ are chosen to be odd to ensure symmetry around the center. The filter *neighborhood* $\mathcal{N}$ is then defined as

$$\mathcal{N} = \left\{(\delta_1, \delta_2) \in \mathbb{Z}^2 \,|\, \delta_1 \in \{-\lfloor \tfrac{s_{x_1}}{2} \rfloor, \ldots, \lfloor \tfrac{s_{x_1}}{2} \rfloor\}, \delta_2 \in \{-\lfloor \tfrac{s_{x_2}}{2} \rfloor, \ldots, \lfloor \tfrac{s_{x_2}}{2} \rfloor\}\right\}.$$

For example, a 3×3 kernel covers the neighborhood centered at a position x and spans all relative positions $\delta \in \{-1, 0, 1\}^2$, as illustrated in Figure 3.7.

In practice, multiple two-dimensional filters are applied to the input, allowing the CNN to capture different local features simultaneously. Suppose the input tensor to a convolution, given by $f \in \mathbb{R}^{m \times n \times c_{\text{in}}}$, has c_{in} input channels, e.g., the three color channels of an image. This input is mapped to c_{out} output channels by convolution. The corresponding kernel is a four-dimensional tensor $A \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}} \times s_{x_1} \times s_{x_2}}$, organized according to the channel dimensions c_{in} and c_{out}, and the spatial dimension s_{x_1} and s_{x_2} of the two-dimensional filters. Each slice $A_{:,j,:,:}$ corresponds to the set of filters producing the j-th output from all input channels.

Assuming a unit distance between two orthogonal adjacent spatial entries, the input tensor can be represented as a function defined on a lattice torus

$$\mathbb{T}^{c_{\text{in}}}_{n,m} = \mathbb{L}^{c_{\text{in}}}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \Big/ \mathbb{L}\left(\begin{bmatrix} m & 0 \\ 0 & n \end{bmatrix}\right).$$

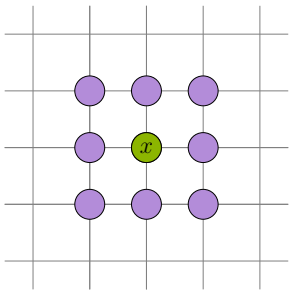


Figure 3.7: Extent of the neighborhood $\mathcal{N}$ of a 3×3 convolution operator A centered at x (green dot). Purple dots indicate the coordinates of the neighborhood $\mathcal{N}$.

The rows and columns are identified modulo m and n, respectively, so that shifts beyond the boundaries wrap around. As introduced in Section 2.2, the structure element $\mathfrak{c}_{\text{in}}$ reflects the presence of c_{in} co-located channel degrees of freedom at each spatial location of the input. Furthermore, $\mathfrak{c}_{\text{in}}$ denotes the position of the channel degrees of freedom while $c_{\text{in}} = |\mathfrak{c}_{\text{in}}|$ is their number. A similar structure can be defined for the output of a convolutional layer by

$$\mathbb{T}^{\mathfrak{c}_{\text{out}}}_{n,m} = \mathbb{L}^{\mathfrak{c}_{\text{out}}}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \Big/ \mathbb{L}\left(\begin{bmatrix} m & 0 \\ 0 & n \end{bmatrix}\right).$$

A linear mapping between $\mathbb{T}^{\mathfrak{c}_{\text{in}}}_{n,m}$ and $\mathbb{T}^{\mathfrak{c}_{\text{out}}}_{n,m}$ is a crystal operator

$$A : \mathfrak{L}(\mathbb{T}^{\mathfrak{c}_{\text{in}}}_{n,m}) \longrightarrow \mathfrak{L}(\mathbb{T}^{\mathfrak{c}_{\text{out}}}_{n,m}), \tag{3.20}$$

acting on the function spaces $\mathfrak{L}$ defined on each lattice torus. Collecting the weights of A associated with every relative position $\delta \in \mathcal{N}$ yields multiplication matrices $M_A^{(\delta)} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}$. As discussed in Section 2.3.2, using these multiplication matrices, the action of the crystal operator A (cf. Equation (3.20)) can be expressed as a multiplication operator acting on the input f as follows:

$$(A * f)(x) = \sum_{\delta \in \mathcal{N}} M_A^{(\delta)} \cdot f(x + \delta). \tag{3.21}$$

This operator can be interpreted as a multi-channel convolution evaluated at $x \in \mathbb{L}(\mathcal{A})$ defined by the mapping

$$A : \mathbb{R}^{m \times n \times c_{\text{in}}} \longrightarrow \mathbb{R}^{m' \times n' \times c_{\text{out}}},$$

where the convolution is evaluated using wrap-around at the spatial boundaries. This corresponds to the implicit assumption that the input is periodic in both spatial directions, ensuring that the convolution is well-defined at the edges of the input. Extending the input beyond its boundary by prescribed values is referred to as *padding*. Conceptually, padding plays a role analogous to boundary conditions in boundary value problems (cf. Section 2.4.1) and different padding modes are depicted in Figure 3.8.

In practice, padding is commonly employed to preserve the spatial dimensions of the input and to retain information near the boundaries. This is achieved by enlarging the input by $\mathtt{p} = (\mathtt{p}_{x_1}, \mathtt{p}_{x_2})$ pixels. While Equation (3.21) prescribes periodic padding, this assumption does not apply to all practical implementations of convolutions. The implications of periodic extension and its practical alternatives are discussed below.

The convolutional operation can be interpreted as a *sliding window* mechanism, where the kernel moves across the spatial extent of the input. The number of pixels by which the kernel shifts along each input dimension before computing the next output is referred to as the *stride*, denoted by $\mathtt{t} = (\mathtt{t}_{x_1}, \mathtt{t}_{x_2}) \in \mathbb{N}^2$. In most applications the stride is set to $\mathtt{t}_{x_1} = \mathtt{t}_{x_2} = 1$. Given a stride $\mathtt{t} = (\mathtt{t}_{x_1}, \mathtt{t}_{x_2})$ and padding $\mathtt{p} = (\mathtt{p}_{x_1}, \mathtt{p}_{x_2})$, the spatial dimensions m' and n' of the output feature map are computed as

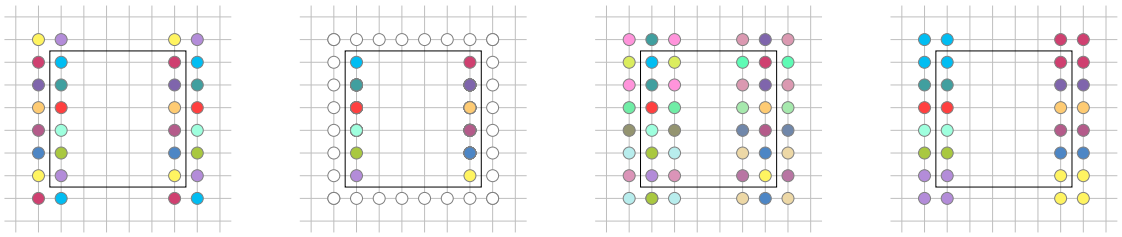
$$m' = \lfloor \frac{m - s_{x_1} + 2\mathtt{p}_{x_1} + \mathtt{t}_{x_1}}{\mathtt{t}_{x_1}} \rfloor \qquad\qquad n' = \lfloor \frac{n - s_{x_2} + 2\mathtt{p}_{x_2} + \mathtt{t}_{x_2}}{\mathtt{t}_{x_2}} \rfloor.$$

A convolution with stride $\mathtt{t} > 1$ is called *strided convolution*. This form of spatial subsampling alters the shift-equivariance property of the convolutional operator. This difficulty can be resolved by rewriting the operation with respect to the underlying shift-equivariance, which corresponds to an implicit increase in the number of channels. In this way, the strided convolution can be reformulated as a standard convolution as in Equation (3.21) but with respect to a sublattice. Specifically, instead of evaluating the convolution at every point x of the original lattice generated by $\mathcal{A}$, the output is evaluated on the corresponding sublattice (cf. Section 2.2.1), generated by $\mathcal{A} \cdot Z = \mathcal{A} \cdot \mathtt{t} \cdot I_2 = \mathtt{t} \cdot \mathcal{A}$ with equal stride $\mathtt{t}_{x1} = \mathtt{t}_{x2} = \mathtt{t}$ and $\mathtt{t} \in \mathbb{N}$. This operation combines convolutional filtering and subsampling and thereby reduces the output dimensions while preserving learned features at regularly spaced intervals. A *dilated* convolution is characterized by a patchy kernel, i.e., the relative positions around the kernel center are shifted by a factor $\mathtt{d}$. We define the dilated convolution $*_{\mathtt{d}}$ as

$$(A *_{\mathtt{d}} f)(x) = \sum_{\delta \in \mathcal{N}} M_A^{(\delta)} \cdot f(x + \delta \cdot \mathtt{d}).$$

Dilated convolutions thereby increase the receptive field without explicitly increasing the filter size, i.e., no extra weights are required [16,125]. Both, the strided and dilated convolution affect the receptive field. Without sufficient padding, the convolution operators considered so far reduce the spatial dimension of the input. Note that convolutions are typically implemented with an additional bias $b \in \mathbb{R}^{c_{\text{out}}}$, which is omitted here to simplify the notation.

Boundary conditions. For convolutions on crystalline structures we assume *periodic padding*. That is, the input is extended periodically by repeating its values, so that each index is interpreted modulo the corresponding dimension of the input shape. However, in the context of CNNs *zero padding* is commonly used, i.e., by enlarging the input with



(a) Periodic padding (b) Zero padding (c) Reflection padding (d) Replicant padding

Figure 3.8: Different padding modes for $\mathtt{p} = (1, 1)$. Colors indicate repeated values.

zeros, which corresponds to Dirichlet boundary conditions in boundary value problems. Other values besides zeros are possible for *constant padding*. Another option is to repeat the values directly located at the border, which is known as *replication padding*. Alternatively, the input can be mirrored at the boundary, which is known as *reflection padding* [50]. The padding modes are illustrated in Figure 3.8.

Matrix representation. The discrete convolution can also be expressed in matrix form. By unrolling the input and output arrays into vectors in row-major order, i.e., left to right and top to bottom, the convolution is represented by a sparse matrix. Its sparsity pattern is depicted in Figure 3.9. The nonzero entries of this matrix are given by the trainable kernel weights $\omega_{i,j}$, where i and j index the corresponding row and column of the convolution kernel, respectively [30].

Weight count. Assume a square kernel of size s with c_{in} and c_{out} channels, and an appropriate padding scheme to preserve the spatial resolution. The number of learnable weights in the convolutional layer is independent of the input feature map dimension but depends only on the channel dimensions and the kernel size s and is given by $s^2 \cdot c_{\text{in}} \cdot c_{\text{out}}$, i.e., the weight tensor scales as $\mathcal{O}(s^2 \cdot c_{\text{in}} \cdot c_{\text{out}})$. These corresponding convolutional matrices are sparse in the spatial (resolution) dimension, since the kernel covers only a local receptive field, but is dense in the channel dimension, meaning that every input channel contributes to every output channel. Consequently, the weight count scales quadratically with the number of channels. A large number of weights increases the risk of overfitting, motivating the development of alternative convolutional variants such as grouped or separable convolutions.

Grouped convolutions and separable convolutions. An effective way to reduce the number of weights in a convolutional layer is to partition the input channels into G groups of size $\mathtt{g}_s$. This technique is referred to as *grouped convolutions* [71]. In this setting, the convolution is applied independently to each group $\mathtt{g} \in \{1, \dots, G\}$, using its own set of filters $A_{\mathtt{g}} \in \mathbb{R}^{c_{\text{out}} \times \mathtt{g}_s \times s \times s}$, where $\mathtt{g}_s = \frac{c_{\text{in}}}{G}$. By restricting each kernel to operate only on its group, the connectivity within the convolution is limited. Consequently, the

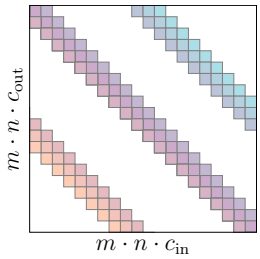


Figure 3.9: Matrix representation of the convolution of a 3×3 kernel with an $m \times n$ input.

corresponding multiplication matrix M_A^δ for each $\delta \in \mathcal{N}$ becomes sparse. In particular, $M_A^{(\delta)}$ is block-diagonal with G blocks, i.e.;

$$M_A^{(\delta)} = \begin{bmatrix} M_{A_1}^{(\delta)} & 0 & \cdots & 0 \\ 0 & M_{A_2}^{(\delta)} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & M_{A_G}^{(\delta)} \end{bmatrix} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}, \tag{3.22}$$

where each diagonal block $M_{A_\mathtt{g}}^{(\delta)} \in \mathbb{R}^{\mathtt{g}_s \times \mathtt{g}_s}$ encodes the convolution restricted to one group of channels. The convolutional mapping is then given by

$$(A * f)(x) = \sum_{\delta \in \mathcal{N}} M_A^{(\delta)} f(x + \delta).$$

Figure 3.10 illustrates the connectivity patterns induced by convolutions with different group sizes, highlighting how the degree of sparsity depends on the choice of G. The weight count of a grouped convolution is $\mathcal{O}(s^2 \cdot \frac{c_{\text{in}} \cdot c_{\text{out}}}{G})$.

A special case of grouped convolution is the *depth-wise convolution*, corresponding to $\mathtt{g}_s = 1$, where each input channel is convolved with its own dedicated 2D kernel [59]. In this case, each multiplication matrix $M_{A_\mathtt{g}}^{(\delta)} = \omega_\mathtt{g}^{(\delta)}$ is diagonal, i.e., $M_A^{(\delta)} = \text{diag}(\omega_1^{(\delta)}, \omega_2^{(\delta)}, \ldots, \omega_{c_{\text{in}}}^{(\delta)})$. Obviously, the connectivity pattern is maximally restrictive as no channel interaction takes place. A common strategy to mitigate this limitation is *channel shuffling* [126]. In this approach, the output channels of a convolutional layer are reshaped, transposed, and flattened before being passed to the next layer, thereby enabling interactions between different channels in subsequent convolutions.

Another approach to address this issue is the use of *depth-wise separable convolutions*. In this setting, the depth-wise convolution that operates across the spatial dimensions is followed by a pointwise convolution that mixes information across channels. The pointwise convolution is represented by a weight matrix $D \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}$, corresponding to 1×1 kernels that linearly project the output of the depth-wise convolution into a new channel space. Formally, the depth-wise separable convolution can be expressed as

$$(A * f)(x) = D \left(\sum_{\delta \in \mathcal{N}} M_A^{(\delta)} \cdot f(x + \delta) \right),$$

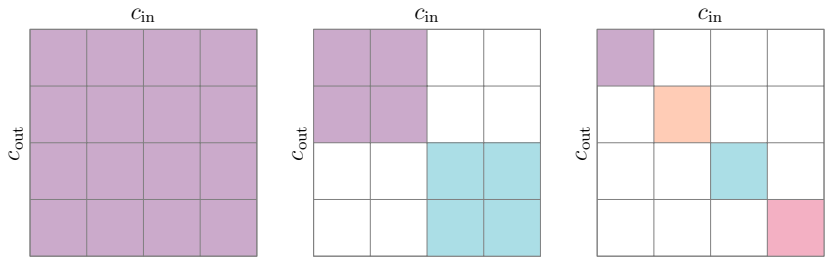


Figure 3.10: Connectivity patterns of convolution operations in matrix representation: standard convolution with $\mathtt{g}_s = c_{\text{in}}$ (left), grouped convolution with $\mathtt{g}_s = \frac{c_{\text{in}}}{\mathtt{g}}$ (middle), and depth-wise convolution with $\mathtt{g}_s = 1$ (right).

where the depth-wise convolution $\sum_{\delta\in\mathcal{N}} M_A^{(\delta)} \cdot f(x+\delta)$ is applied independently to each input channel, while the subsequent pointwise operator D performs a linear projection onto c_{out} output channels. This decomposition reduces the number of weights to $\mathcal{O}(s^2 c_{\text{in}} + c_{\text{in}} \cdot c_{\text{out}})$, while largely preserving the representational capacity of the standard convolution [21].

Pooling layers. *Pooling* operations are a special case of convolutional mapping that downsamples feature representations from layer ℓ to the subsequent layer $\ell+1$, typically applied after a sequence of standard convolutions. The restriction operation R acts independently on each channel and the corresponding mapping is defined by

$$R_\ell^{\ell+1} : \mathbb{R}^{m_\ell \times n_\ell \times c} \longrightarrow \mathbb{R}^{m_{\ell+1} \times n_{\ell+1} \times c},$$

where $n_\ell \times m_\ell$ and $n_{\ell+1} \times m_{\ell+1}$ are the spatial resolutions at layers ℓ and $\ell+1$ respectively. Analogous to restriction operations in multigrid (cf. Equation (2.24)), pooling transfers information from a fine grid to a coarser grid, enabling a hierarchical feature representation.

Similar to the convolution operation, the pooling window (or kernel) slides across the spatial dimensions of the input according to a specified stride. At each location $x \in \mathbb{L}$, the kernel aggregates information from the overlapping patch either by computing the mean or the maximum, i.e., compressing spatial information by summarizing the local neighborhood and thereby removing redundant information. Unlike convolutional layers, however, pooling layers are parameter-free, i.e., the kernel does not contain weights that are learnable weights. According to regular convolutions pooling windows are square with values $\mathtt{r} = (\mathtt{r}_{x_1}, \mathtt{r}_{x_2}) \in (2\mathbb{N}+1)^2$, i.e., a local centered neighborhood. For a pooling window of size $\mathtt{r}_{x_1} \times \mathtt{r}_{x_2}$, the dimensions of the output feature map is $n_{\ell+1} = \frac{n_{\ell+1}}{\mathtt{r}_{x_1}}$ and $m_{\ell+1} = \frac{m_{\ell+1}}{\mathtt{r}_{x_2}}$.

Although overlapping pooling windows, i.e., $\mathtt{t} < \mathtt{r}$ are possible (cf. Section 2.5), we focus on block-wise pooling operations, i.e., non-overlapping pooling windows. Recall

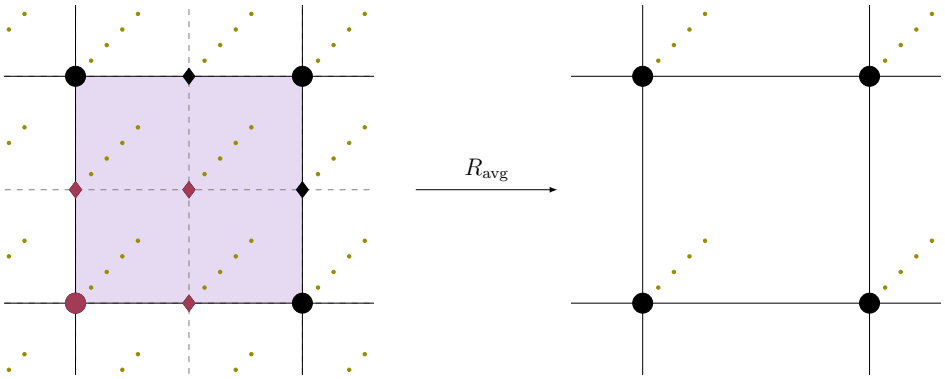


Figure 3.11: Schematic of average pooling on a crystal torus with structure elements representing the channel degrees of freedom (green dots). On the left, diamonds denote the fine-grid points. The highlighted diamonds indicate the fine-grid points that are averaged to obtain the corresponding coarse-grid point (large dot). This averaging is performed independently for each channel degree of freedom within the shaded coarse-grid unit cell.

from Section 2.5.1 that for a single channel $c = 1$, the coarse torus has structure element $\mathfrak{c} = (0)$ and the pooling is a linear restriction

$$R : \mathfrak{L}(\mathbb{T}^{\mathfrak{a}}_{n_{\ell+1},m_{\ell+1}}) \longrightarrow \mathfrak{L}(\mathbb{T}^{\mathfrak{c}}_{n_{\ell+1},m_{\ell+1}}),$$

where $\mathfrak{a} \cong {}^{\mathbb{T}_{m_\ell,n_\ell}}\!/_{\mathbb{T}_{m_{\ell+1},n_{\ell+1}}}$, indexes the fine grid points in the unit cell $\Xi({}^{\mathbb{T}_{m_\ell,n_\ell}}\!/_{\mathbb{T}_{m_{\ell+1},n_{\ell+1}}})^{|\mathfrak{a}|}$ (abbreviated as Ξ), that are averaged to the corresponding coarse grid point.[4] For multiple channels, i.e., $c > 1$, the fine and coarse lattice tori $\mathbb{T}^{\hat{\mathfrak{c}}}_{m_\ell,n_\ell}$ and $\mathbb{T}^{\mathfrak{c}}_{m_{\ell+1},n_{\ell+1}}$ carry structure elements $\hat{\mathfrak{c}} = (\hat{\mathfrak{c}}_1, \ldots, \hat{\mathfrak{c}}_c)$ and $\mathfrak{c} = (\mathfrak{c}_1, \ldots, \mathfrak{c}_c)$. The pre-image structure element is obtained by combining each channel degree of freedom $\hat{\mathfrak{c}}$ with the corresponding fine grid neighborhood indexed by $\mathfrak{a}$

$$\mathfrak{f} = \{\hat{\mathfrak{c}}_j + \mathfrak{a} \mid j = 1, \ldots, c, \text{ and } \mathfrak{a} \in \{0, a_1, a_2, a_1 + a_2\}\},$$

so that the restriction reads

$$R : \mathfrak{L}(\mathbb{T}^{\mathfrak{f}}_{n_{\ell+1},m_{\ell+1}}) \longrightarrow \mathfrak{L}(\mathbb{T}^{\mathfrak{c}}_{n_{\ell+1},m_{\ell+1}}).$$

This restriction is shift-invariant with respect to the coarse grid. For *average pooling*, such a restriction acts as a multiplication operator, equivalent to a convolution with a constant kernel over the unit cell Ξ with kernel values ${}^1\!/_{|\mathfrak{a}|}$ and $\mathfrak{a} \in \Xi$ thus preserving an aggregated representation of the activation. The corresponding operator is

$$(R_{\text{avg}} f)(x) = \frac{1}{|\mathfrak{a}|} \sum_{\mathfrak{a} \in \Xi} f(x + \mathfrak{a}),$$

for all $x \in \mathbb{T}_{m_{\ell+1},n_{\ell+1}}$. Figure 3.11 illustrates the average pooling operation on the lattice torus for a 2×2 pooling window.

[4] For a 2×2 window $\mathfrak{a} = (0, a_1, a_2, a_1 + a_2)$ with $[a_1 \mid a_2] \in I_2$ (cf. Section 2.5).

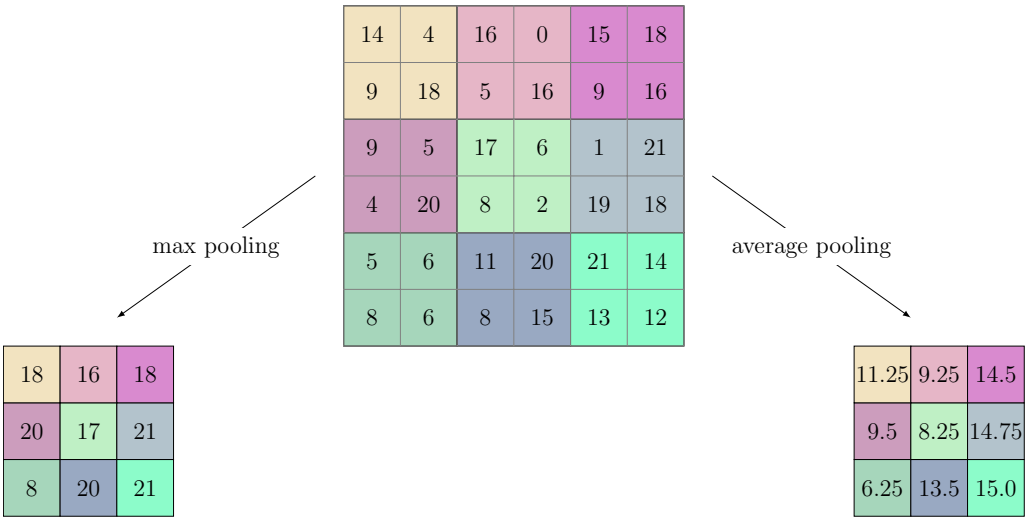


Figure 3.12: Example of max pooling and average pooling for a single channel and a 2×2 pooling window ($\mathtt{r} = 2$) applied to a 6×6 feature map. Each colored 2×2 block of the input feature map corresponds to a unit cell of the coarse torus.

In *max pooling*, the pooling operator returns the largest value in the local neighborhood

$$(R_{\max} f)(x) = \max_{\mathfrak{a} \in \Xi} f(x + \mathfrak{a}),$$

which highlights local dominant features. Both pooling types, depicted in Figure 3.12, reduce sensitivity to small translations of the input and thereby improving robustness of the learned representation. This property is particularly relevant in image classification, where the objective is to detect the presence of a specific object regardless of their exact position in the image. Pooling therefore introduces an implicit prior that the functions learned by the network should be less dependent on precise locations of features [40].

Transposed convolutions. The *transposed convolution*, commonly referred to as *deconvolution*, is frequently used to upsample feature maps within CNNs. Although, in general $A^{\mathrm{T}} \neq A^{-1}$, the transpose A^{T} is again a linear operator. Analogous to the prolongation operation introduced in Section 2.5, the transposed convolution to Equation (3.21) for some input $g \in \mathbb{R}^{m \times n \times c_{\text{out}}}$ evaluated at position $x \in \mathbb{Z}^2$ is given by

$$(A^{\mathrm{T}} * g)(x) = \sum_{\delta \in \mathcal{N}} M_{A^{\mathrm{T}}}^{(\delta)} \cdot g(x - \delta),$$

where the multiplication matrices $M_{A^{\mathrm{T}}}^{(\delta)} = (M_A^{(\delta)})^{\mathrm{T}} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}}}$ contain the transposed kernel weights. In this sense, the transposed convolutions is analogous to prolongation in multigrid methods, acting as the adjoint of the convolution operator and commonly used for learnable upsampling of feature maps.

Vanishing gradient problem. The backpropagation algorithm introduced in Section 3.2.3 enables efficient training of multilayer perceptrons and CNNs. However, deep neural networks tend to suffer from vanishing or exploding gradients. This problem arises as the gradients are propagated backward through many layers, repeated application of the chain rule may cause their magnitude to decrease (or increase) exponentially. Consequently, the gradients in the early layers may become extremely small, slowing down the learning process or even preventing effective learning, thereby limiting the ability of the neural network to learn. Activation functions such as the sigmoid, whose derivatives are bounded between 0 and 1, further worsen the problem of vanishing gradients. This issue can be mitigated by using ReLU activation functions. Nevertheless, training very deep neural networks remained challenging, since it was still unstable and prone to degradation of performance as the number of layers increased. The introduction of skip connections in *residual networks*, together with careful initialization of the weight matrices, helps preserve stronger gradients during backpropagation.

3.2.5 Residual Neural Networks

The residual network (ResNet) architecture was introduced by He et al. in 2015, directly addresses the vanishing gradient problem [53]. Assuming that a subsequent combination of non-linear layers $\psi(f)$ can approximate complex functions $\varrho(f)$, it can equivalently approximate the corresponding residual mapping

$$\psi(f) = \varrho(f) - f \iff \varrho(f) := \psi(f) + f.$$

A *residual block* (ResNet-block) is composed of a multiple convolutions mappings, typically combined with non-linearities and batch normalization. Its output is defined by

$$o = \psi(f, \{W^{(i)}\}) + f, \quad \text{for } i = 1, 2, \ldots$$

where $\{W^{(i)}\}$ denotes the collection of learnable weight matrices of the convolutional layers in the residual block. Formally, each $W^{(i)}$ corresponds to the weights of the i-th convolution within the block. Thus, $\psi(f, \{W^{(i)}\})$ represents the composite mapping of all layers in the block applied to the input f. Specifically, a ResNet-block with two successive layers is a composition of two convolutional mappings $A = W^{(1)}$ and $B = W^{(2)}$ with a ReLU activation function $\phi = \phi_{\text{ReLU}}$ given by $\psi(f) = B \circ \phi \circ A(f)$. The identity shortcut is an elementwise addition, followed by a second activation function ϕ. An example of such a two-layer ResNet-block is depicted in Figure 3.13. Note, that the convolutional mappings A and B have the same number of input and output channels with padding $\mathtt{p} = 1$, ensuring that f and $\psi(f)$ have identical dimensions. For different numbers of input and output channels, an additional mapping is applied to the shortcut connection to ensure matching dimensions.

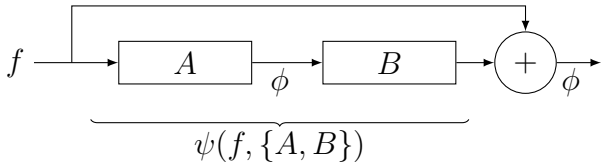


Figure 3.13: ResNet-block with two convolutional layers.

ResNet18. The focus in this thesis is on the ResNet18 architecture. The original ResNet18 is composed of 18 layers in total, an initial convolutional layer with 64 output channels, followed by 8 ResNet-blocks organized into four resolution levels. Each ResNet-block consists of two convolutional layers, as depicted in Figure 3.13. The two ResNet-blocks within a resolution level have the same number of input and output channels and preserve spatial dimensions. Specifically, the four resolution levels use $64, 128, 256$ and 512 channels, respectively. The ResNet-blocks are preceded by a 7×7 max-pooling and are followed by an average pooling layer. The pooled feature representation is then passed to a fully connected layer with 1,000 output neurons, followed by a softmax classification layer. The ResNet18 architecture is illustrated in Figure 3.14.

3.2.6 Initialization of Convolutional Layers

Poor initialization can result in vanishing or exploding gradients, slow convergence or yield suboptimal solutions. He et al. (2015) proposed an initialization method to improve the robustness of training deep CNNs with ReLU activation functions [52]. In this scheme, the weights of convolutional layers are drawn from Gaussian distributions [71]. However, with fixed standard deviation, CNNs with more than 8 layers can fail to converge, since in very deep networks improper scaling of the initial weights can still lead to unstable gradients.

Variance-preserving initialization. Let $o = \phi(W * f + b)$ denote a CNN layer with weight kernel $W \in \mathbb{R}^{h \times c \times s \times s}$, bias $b \in \mathbb{R}^h$ and ReLU activation function. The variance of its output depends on the variance of the weight kernel W and the input f. If the initial variance is too large, the activations can grow throughout the forward pass, increasing the risk of exploding gradients. Two commonly used initialization schemes address this problem by scaling the weight variance according to the number of connections. Assuming linear activations in CNNs, Glorot and Bengio (2010) proposed a properly scaled uniform distribution for initialization, i.e., the *Xavier* initialization

$$W^{(\ell)} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right)$$

with $n_{\text{in}} = c \cdot s^2$ denoting the effective input and $n_{\text{out}} = h \cdot s^2$ output connections for layer ℓ and $\mathcal{U}$ denotes the uniform distribution [38]. These assumptions are violated

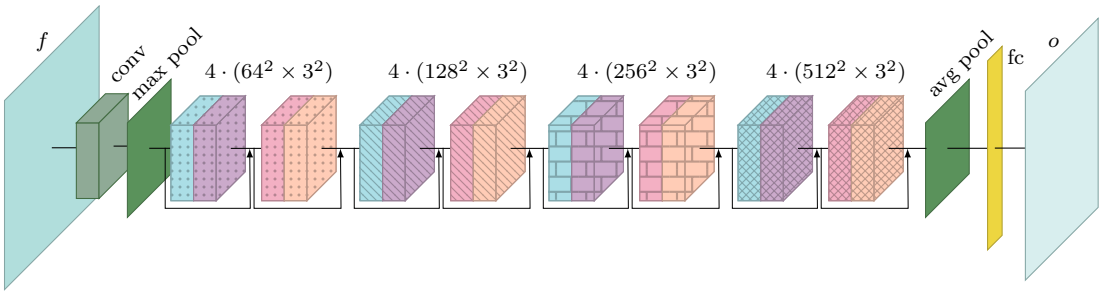


Figure 3.14: ResNet18 architecture. The four resolution levels are distinguished by different patterns, while colors indicate the four convolutional layers within each resolution level. The weight tensors of the convolutions within each resolution level have dimensions $c \times c \times 3 \times 3$, where $c \in \{64, 128, 256, 512\}$ denotes the number of channels. Non-linear activation functions are omitted for clarity.

for ReLU activation functions, since ReLU only passes non-negative outputs and thus discards half of the input signal. This is compensated by *Kaiming-He initialization*

$$W^{(\ell)} \sim \mathcal{G}\left(0, \frac{2}{n_{\text{in}}}\right)$$

where $\mathcal{G}$ denotes the normal distribution. See [52] for details.

4 Improving the Accuracy-Weight Trade-off

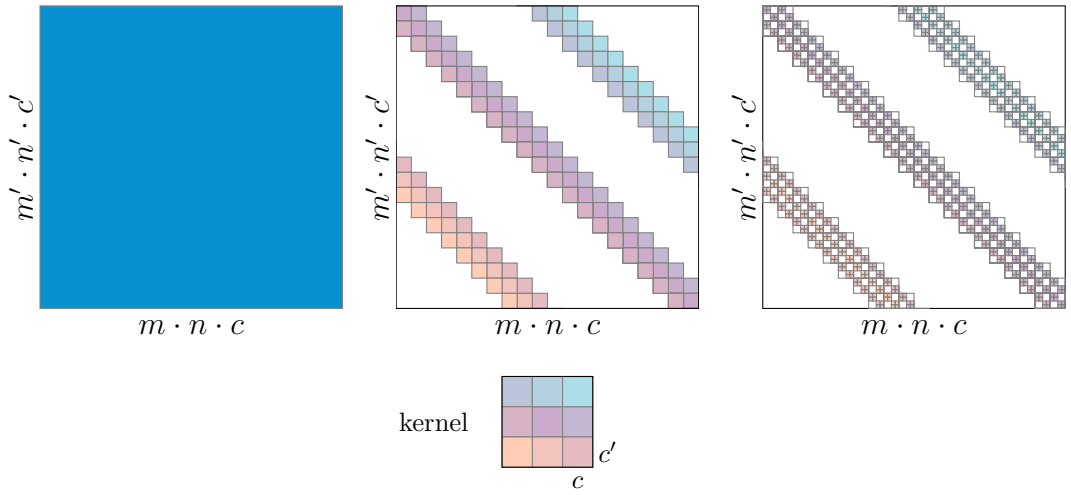
The following content is based closely on our previously published works [109] and [111]. Parts of the text are reproduced verbatim. In this chapter we unify the presentation of both approaches and begin with a joint introduction in Section 4.1. Although we aim to avoid unnecessary overlap with the concepts reviewed in Chapters 2 and 3, we briefly revisit selected points to provide context and motivation for our approaches.

4.1 Introduction

Deep CNNs are powerful models for image classification tasks [52, 71, 94]. ResNets, in particular, have become popular due to their ability to overcome the vanishing gradient problem [53]. CNN architectures comprise 10-100 million of learnable parameters, i.e., weights. Despite their high classification accuracy, these networks have a tendency to be heavily overparameterized, making a reduction in the number of weights clearly desirable. However, this reduction can introduce an unwanted bias. As discussed in Section 3.1, this trade-off, known as "bias-complexity trade-off", constitutes a fundamental problem in machine learning [100].

In this work, we address this challenge by introducing CNN architectures, that achieve a more favorable bias-complexity trade-off, in terms of an accuracy-weight trade-off, by exploiting multigrid-inspired ideas.

In Section 2.5 and Section 3.2.5, we reviewed multigrid methods and ResNet architectures, respectively, highlighting shared building blocks, most notably convolutions and pooling operations. This analogy was already pointed out by He et al. in 2016 [53]. Building on these insights, He and Xu proposed the MgNet framework, which integrates multigrid schemes into the design of ResNets [50]. They showed that a significant number of weights can be saved by appropriate weight sharing [50]. Moreover, Eliasof et al.



(a) Fully connected (b) Convolution (c) Grouped convolution

Figure 4.1: Sparsity patterns of neural network layers in matrix representation. (a) fully connected layer; (b) channel coupling of a regular convolution; and (c) channel coupling of a grouped convolution. The weights are colored according to the convolutional 3×3 kernel in the second row.

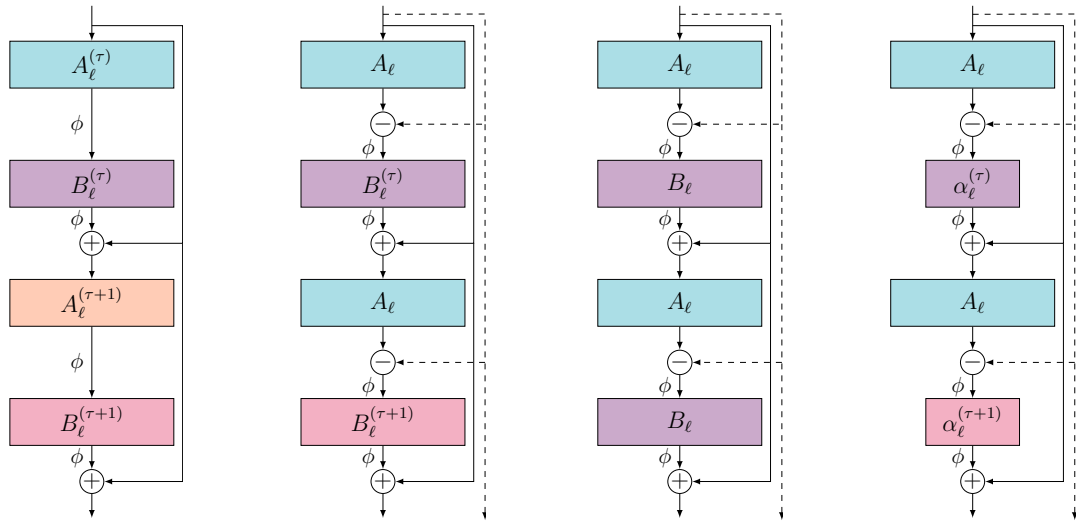
demonstrated that a hierarchical structure in the channel dimension can improve the weight complexity [32].

Utilizing these findings we introduce two families of CNN architectures, that exploit multigrid-ResNet analogies to achieve a beneficial accuracy-weight trade-off:

1. MGiaD [109] establishes multigrid structures in all relevant dimensions, i.e., in the resolution dimension as well as in the channel dimension. Appropriate weight sharing and a hierarchical structure in the channel dimension improve the weight complexity to linear scaling.

2. Poly-MgNet [111] extends the MgNet framework by incorporating polynomial smoother-inspired building blocks, resembling the error-propagation polynomials in classical multigrid methods. These models drastically reduce the weight complexity while incurring only negligible loss in expressive capacity.

For context, we shortly review some fundamentals already covered in detail in Section 3.2 and Section 2.5. The complexity of the number of weights in a neural network layer can be considered as a function of the channels and the spatial extent of the input and output of the layers. Consider spatial dimensions m, n, m', n' of input and output feature maps, respectively, and let c, c' be the corresponding channel dimensions. As noted in Section 3.2.1, a fully connected layer contains $\mathcal{O}((m \cdot n \cdot c) \cdot (m' \cdot n' \cdot c'))$ weights, which quickly becomes intractable for growing spatial dimensions [100] (cf. Figure 4.1a). In contrast, a convolutional layer (cf. Section 3.2.4) with a stencil of size $s \times s'$ requires only $\mathcal{O}(s \cdot s' \cdot c \cdot c')$, independent of the spatial dimensions. Nevertheless, the weight



(a) ResNet-block (b) Non-stationary (c) Stationary (d) Polynomial

Figure 4.2: Weight sharing in ResNet- and MgNet-blocks. (a) Two consecutive ResNet-blocks without weight sharing; (b) MgNet-blocks with shared weight tensor A_ℓ corresponding to a non-stationary iterative scheme; (c) MgNet-blocks with shared weight tensors A_ℓ and B_ℓ, corresponding to stationary iterative scheme; and (d) MgNet-blocks with a polynomial building block $B_\ell = p(A_\ell)$, corresponding to a polynomial iterative scheme, where the only learnable parameters are the scalar coefficients $a_\ell^{(\tau)}$ and $a_\ell^{(\tau+1)}$.

count scales quadratically with respect to the number of channels. The sparsity pattern of the corresponding matrix is illustrated Figure 1.3b.

While convolutions achieve significant weight savings, their local receptive filed come at the expense of slow information propagation. More efficient information exchange across layers is achieved by skip connections as employed in ResNets, cf. Section 3.2.5.

Multigrid methods, introduced in Section 2.5 as solvers for large sparse systems arising from discretizing PDEs, combine local relaxation with a coarse-grid correction to enable efficient long-range information exchange. Building on this principle, He and Xu introduced MgNet, where residual layers play the role of smoothers and pooling operations act as coarse-grid restrictions. In the multigrid setting, the discretized operators remain fixed between consecutive coarsening or pooling operations, justifying weight sharing across multiple residual layers and thereby reducing the overall weight count. The reuse of weight tensors within ResNet-blocks on one resolution level is depicted in Figure 4.2.

The remainder of this chapter is structured as follows. We begin by complementing this introduction with separate overviews of our approaches in Section 4.1.1 and Section 4.1.2. We then discuss related works in a unified manner in Section 4.2 and the analogies between residual learning and multigrid methods are presented in Section 4.3. Building on these theoretical foundations, we present technical details, experimental setups and

results for MGiaD in Section 4.4, while we provide the corresponding discussion for Poly-MgNet in Section 4.5. Each of these sections concludes with a summary.

4.1.1 Improved Scaling with Respect to all Relevant Dimensions

Although weight sharing induced by MgNet cuts down the number of weights in a ResNet-block, the weight count scales quadratically with respect to the number of channels. Unfortunately, no assumption analogous to shift-invariance in the spatial dimension can be made for the channel dimension and any attempt to manually sparsify its connectivity, i.e., by blocking or dropping connections, is typically met with significant performance degradation. Attempts to automatically reduce CNN weight count while preserving predictive performance include pruning [46, 55, 75, 116], neural architecture search [2, 41], and the development of resource-efficient architectural components [58,59,122,126]. A multigrid perspective on this sparsification problem is taken by [32], where the limited exchange of information between channels is addressed by introducing another hierarchical structure, termed multigrid in channels (MGIC). In this way, a linear scaling of the weight count can be achieved. Picking up on these recent developments we present an efficient ResNet-type architecture of multigrid-inspired CNNs that exhibit improved scaling behavior with respect to all relevant dimensions compared to recent architectures. Our architecture combines elements introduced in [50] and [32] in a natural, albeit subtle manner and can be fully explained by multigrid terminology. The resulting architecture substantially reduces overparameterization, yielding a considerably smaller weight count than comparable residual architectures.

In our experiments, we compare our architecture to ResNet, MgNet and MGIC on several datasets and achieve superior performance in terms of accuracy. For example, compared with ResNet18 on CIFAR-10, we reduce the weight count by a factor of 10 while sacrificing only up to 0.5 percentage points (pp) in accuracy. Even after reducing the weight count by a massive factor of 28, the loss in accuracy remains below 1 pp. The development of our method is presented in Section 4.4.2, followed by experimental results in Section 4.4.3 and concluding remarks in Section 4.4.4.

4.1.2 Building Blocks Inspired by Polynomial Smoothers

A typical ResNet-block consists of two convolutional mappings, denoted by A and B, gated by a residual connection. Let $Au = f$ be a system of linear equations. In multigrid smoothing applies an operator B to the residual $f - Au$, i.e.,

$$u \leftarrow u + B(f - Au),$$

in order to update the solution u, i.e., by iteratively reducing the residual, cf. Section 2.4.2. This operation has the property of smoothing the error associated with

the current approximation u. In MgNet, B is chosen as a learnable convolution operator, but here we consider a cheaper alternative that reuses the convolution A in its definition. Again motivated by the residual propagator polynomial preconditioner (cf. Equation (2.19)), we consider polynomials $p(A)$ for B with only a handful of learnable parameters, thereby reducing the number of learnable parameters by almost 50%.

In Section 2.4.2 we analyzed the Richardson iteration [107], which is a particularly simple instance of this idea. In this case, $B = \omega I$, where I is the identity matrix and ω is a scalar parameter. However, we aim to consider more general polynomials.

In this work, we explore the extent to which smoothing iterations based on polynomial approximations are suitable within MgNet, as depicted in Figure 4.2. Specifically, instead of learning an independent convolutional operator B, we represent it as a polynomial in the shared operator A, requiring only a small number of coefficients to be learned. To this end, we introduce a new building block in multigrid-inspired CNNs such as MgNet, yielding a further reduction in the weight count. These methods are inspired by multigrid smoothers (cf. Section 2.4.2), namely block, polynomial and Richardson smoothers, yielding layer modules of reduced weight counts. Incorporating this building block into MgNet resulted in a significant reduction of the total weight count while incurring only a minor loss in classification accuracy. Our approach yields further insight into the inherent connection between multigrid and ResNets and demonstrates that multigrid methodology is useful for the construction of weight-count-efficient CNNs.

We first explain the construction of our layer inspired by multigrid smoothers in Section 4.5.1, followed by the experimental results in Section 4.5.2. Finally, we conclude this section in Section 4.5.3.

4.2 Related Work

In this section, we provide an overview of related works for both approaches, which can be classified into four categories, ordered roughly from remotely to closely related. A fifth category discusses related exclusively to the Poly-MgNet approach.

4.2.1 Reducing the Number of Channels

Existing methods for reducing the weight count and existing methods for reducing the computational complexity of CNNs overlap considerably. One way to reduce the computational complexity is a reduction of the number of channels in convolutional layers. Consider the simplified case in which the number of input channels c equals the number of output channels of a given convolutional layer with filter size $s \times s$. Such a layer contains $s^2 \cdot c^2$ weights. Thus, a reduction of c results in a clear reduction of the number of weights. In [83] it was shown experimentally that there is redundancy in CNNs,

allowing connections between channels to be removed after the CNN has been trained. This process is known as CNN pruning [46, 49, 55, 75]. At the same time pruning and other sparsity-enhancing methods [13, 47] also reduce the model complexity in terms of the number of weights. While these approaches first train a CNN to convergence before pruning it, in [36] the CNN is pruned periodically during training. In [74] a saliency criterion to identify structurally important connections is proposed, which allows for pruning before training. Like the latter approach, we reduce the weight count before training. Unlike pruning-based methods, however, our approach is based on the inherent similarity of multigrid and ResNet. It derives weight-count-efficient layer modules directly from multigrid methodology, thereby providing an explanation for more weight count efficient layer modules. A related line of research is neural architecture search [12], whose goal is to search the space of architectures close to optimal ones with respect to specified optimization criteria. If the focus is on computational efficiency, tuning the channel hyperparameters is one possible approach [41], resulting in a reduced weight count. Whereas these approaches rely on optimization procedures to reduce the number of weights, our approach derives weight-count-efficient architectures directly from multigrid methodology, providing a principled explanation for the resulting efficiency.

4.2.2 Modified Layers

Another line of research focuses on the specific design of convolutional layers. Compared to the aforementioned approaches, the constructions outlined in this section rely on manually designed architectural modifications and conventional techniques to improve computational efficiency. As in the previous category, there is a close connection between computational efficiency and reducing the weight count. One approach to reduce the number of weights is the use of grouped convolutions [71, 122], where the channels are grouped into g decoupled subsets, each convolved with its own set of filters, thereby reducing the weight count to $s^2 \cdot (c/g)^2$. This decoupled structure impedes the exchange of information across channels. To overcome this limitation, ShuffleNet [126] combines grouped convolutions with channel shuffling. Another approach that also reduces the number of floating-point operations uses so-called depthwise separable convolutions [58, 59, 98], a key component or the MobileNet architecture. Here, a depthwise convolution, i.e., a grouped convolution with $g = c$, is followed by a pointwise 1×1 convolution to create a linear combination of the output of the former. This reduces the weight count to $s^2 + 2c$, effectively replacing each kernel slice of size c^2 with a rank-one approximation.

4.2.3 Multigrid-Inspired Architectures

In scientific computing, multigrid methods are algorithms based on hierarchical discretizations that efficiently solve systems of (non-)linear partial differential equations

(PDEs) [10, 64, 106, 107]. Some works utilize CNNs to solve PDEs [104], e.g., by estimating optimal preconditioners [44] or prolongation and restriction operators [67]. Another line of research focuses on architectures based on common computational components as well as similarities between CNNs and multigrid methods [53]. In [68] an architecture consisting of pyramid layers of convolutional layers at different scales, with each pyramid processing coarse- and fine-grid representations in parallel, is proposed. This multigrid architecture improves accuracy while remaining weight-count efficient. As already pointed out, He and Xu [50] introduced the MgNet framework, which finds a justification for sharing weight tensors across consecutive ResNet blocks with the same spatial resolution. By exploiting multigrid concepts in the spatial dimensions, compared to ResNets with the same number of layers, MgNet models have fewer weights while maintaining classification accuracy. An alternating stack of MgNet blocks and pooling layers can be viewed as the left leg of a multigrid V-cycle. Another hierarchical structure, however, is proposed in the channel dimension in [32]. The resulting architecture, termed multigrid-in-channels (MGIC), is built on grouped convolutions and coarsening via channel pooling. By introducing multigrid concepts in the channel dimensions, this approach improves the scaling of the number of weights with the number of channels from quadratic to linear. In our approach [109] we take a unified multigrid perspective in both the spatial and channel dimension. The introduced architecture, called multigrid in all dimensions (MGiaD), improves the trade-off between the number of weights and accuracy via full approximation schemes in the spatial and channel dimensions, including MgNet's weight sharing. Similar to these approaches, we exploit the inherent similarities between CNNs and multigrid to further reduce the weight count in [111]. Our focus in this work is on casting multigrid smoothers into CNN layer modules, an idea that has not been explored in previous work.

4.2.4 Polynomial Neural Networks

Polynomial neural networks are neural networks that represent their output as a polynomial of the input. This approach was first explored using the group method of data handling [87]. It determines the structure of a polynomial model through a combinatorial search and selects the best solution based on an external criterion, e.g., least squares. While these methods belong to self-organizing neural networks, another category models the network output as a higher-order polynomial of the input [22, 23, 101]. Our goal is to use polynomials as building blocks within CNNs rather than modeling the network itself as a polynomial approximator.

4.3 Residual Learning and Multigrid Methods

Multigrid consists of two complementary components, the smoother and the coarse-grid correction. In a recursive fashion, the coarse-grid correction is performed by restricting to a coarser grid, followed by smoothing on the next coarser scale and another coarse-grid correction. This structure is already reflected in typical CNNs, which alternate between convolutions and pooling operations. In both multigrid methods and their CNN counterparts, the smoothing operation requires considerable computational effort. A suitable choice of smoother is crucial for the efficiency of a multigrid method, and we will see that the underlying principles can be exploited for CNNs. In this section, we revisit smoothing and coarsening and unify the multigrid and CNN perspectives.

4.3.1 Revisiting ResNet and Multigrid

Given a data-feature relation $A(u) = f$, the right-hand side f represents the data space and u the features. In CNNs, A is learnable and $A(u) = f$ can be optimized [50]. The mappings between data $f \in \mathbb{R}^{m\times n\times c}$ and features $u \in \mathbb{R}^{m\times n\times h}$ are given by

$$\begin{aligned} A &: \mathbb{R}^{m\times n\times h} \mapsto \mathbb{R}^{m\times n\times c}, \quad \text{such that } A(u) = f \\ B &: \mathbb{R}^{m\times n\times c} \mapsto \mathbb{R}^{m\times n\times h}, \quad \text{such that } u \approx B(f), \end{aligned}$$

where m and n characterize the spatial resolution dimensions of the input, while c and h determine the number of input and output channels, respectively. Note that the number of input and output channels can differ. A can be considered a *feature-to-data map*, while B is applied to elements of the data space and is also referred to as a *feature extractor*. In multigrid, the property $u \approx B(f)$ is beneficial to the convergence of the method, which will be explained in the following. In the context of CNNs, convolutional mappings are usually combined with non-linear activation functions. For a clear presentation of the similarities between CNNs and multigrid, the non-linearities are omitted for now. Recall that a non-stationary iterative scheme is given by

$$u \leftarrow u + B^{(\tau)}(f - Au) \text{ for } \tau = 1, 2, \ldots \tag{4.1}$$

where B is an appropriate feature extractor.

Note that the feature extractor $B^{(\tau)}$ depends on τ, yet Equation (4.1) can be turned into a stationary scheme with a fixed B, i.e., a shared weight tensor. On the other hand, interpreting A as a data-feature mapping motivates a fixed A, i.e., also a shared weight tensor. As shown in [50], the structure of Equation (4.1) with activation functions added resembles a ResNet-block. Explicitly, let u^* denote the exact solution and $e^{(\tau)} = u^* - u^{(\tau)}$ the error at iteration τ. Then, the error is propagated according to

$$e^{(\tau+1)} = e^{(\tau)} - B^{(\tau)}Ae^{(\tau)} = (I - B^{(\tau)}A)e^{(\tau)}.$$

The operator $(I - B^{(\tau)}A)$ represents a ResNet-block, with I the identity mapping and $-B^{(\tau)}A$ the residual mapping (cf. Figure 4.2a). The main difference between MgNet and ResNet is the sharing of one or more weight tensors. The weight count of two such distinct blocks is given by $4 \cdot (s^2 \cdot c \cdot h)$. In MgNet, for the non-stationary case, i.e., only A is shared, the weight count is reduced to $3 \cdot (s^2 \cdot c \cdot h)$. For the stationary case, i.e., when both A and B are shared, the weight count for two blocks is only $2 \cdot (s^2 \cdot c \cdot h)$. The pseudoinverse of A is generally a dense matrix, which would have to be represented by a fully connected weight layer, conflicting with the required convolutional structure of the feature extractors $B^{(\tau)}$ (respectively B). Even with an optimal choice of $B^{(\tau)}$, the convergence of the iteration in Equation (4.1) is inevitably slow due to the local structure of the operator. However, convolutional operators $B^{(\tau)}$ are computationally inexpensive. A few iterations have a smoothing effect on features, and the resulting error can be accurately represented at a coarser resolution.

4.3.2 Resolution Coarsening

The restriction of the (residual) data to coarser scales, performed by mappings

$$R_\ell^{\ell+1} : \mathbb{R}^{m_\ell \times n_\ell \times c_\ell} \mapsto \mathbb{R}^{m_{\ell+1} \times n_{\ell+1} \times c_{\ell+1}}$$

yields a hierarchy of resolution levels $\ell = 1, \ldots, L$. On each level ℓ, the smoothing iteration (Equation (4.1)) can be applied using level-specific mappings A_ℓ and $B_\ell^{(\tau)}$, where initially $u_\ell^{(0)} = 0$. Note that the feature extractors can also be stationary. Analogous to restriction in multigrid, the spatial resolution in CNNs is reduced by pooling operations with a stride greater than 1. In CNNs, the number of channels is usually increased in the process. Corresponding to the coarsening leg of a standard multigrid V-cycle [107], the combination of ν smoothing iterations on each resolution level ℓ and $L - 1$ restrictions yields Algorithm 5.

Algorithm 5 \-MgNet(f_1)

Initialization: $u_1 = 0$
for $\ell = 1, \ldots, L$ **do**
    for $\tau = 1, \ldots, \nu$ **do**
        $u_\ell = u_\ell + B_\ell^{(\tau)}(f_\ell - A_\ell(u_\ell))$
    end for
    if $\ell < L$ **then**
        $u_{\ell+1} = 0$
        $f_{\ell+1} = R_\ell^{\ell+1}(f_\ell - A_\ell(u_\ell))$
    end if
end for

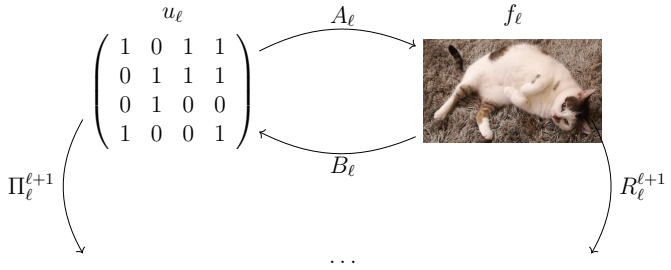


Figure 4.3: Data-feature relations A_ℓ and B_ℓ on resolution level ℓ, followed by transfer to the coarser resolution level $\ell+1$, with $R_\ell^{\ell+1}$ restricting the residual data and $\Pi_\ell^{\ell+1}$ projecting the feature approximation. The residual $r_\ell = f_\ell - A_\ell u_\ell$ is computed by applying A_ℓ to the features u_ℓ, after which the feature extractor B_ℓ is applied to r_ℓ.

4.3.3 Full Approximation Scheme (FAS) for Resolution Coarsening

So far, non-linear activation functions, potential non-linear poolings, and normalization operations, which are characteristic of CNNs, have been disregarded. However, if we now take activation functions into account, we obtain a non-linear structure for CNNs. As noted in Section 2.5, non-linearity in multigrid problems yields multiple possible solutions, so that the initial solution not only determines the solution but also has a crucial influence on the convergence rate. Consequently, choosing an initial guess $u_{\ell+1}^{(0)}$ at the coarser scale based on the current feature approximation u_ℓ is likely to improve upon $u_{\ell+1}^{(0)} = 0$. Therefore, the feature approximations of non-linear problems are also projected to the coarser scale by a mapping

$$\Pi_\ell^{\ell+1} : \mathbb{R}^{m_\ell \times n_\ell \times c_\ell} \mapsto \mathbb{R}^{m_{\ell+1} \times n_{\ell+1} \times c_{\ell+1}} \tag{4.2}$$

to initialize the solution $u_{\ell+1}^{(0)} = \Pi_\ell^{\ell+1} u_\ell^{(\nu)}$. Given this non-trivial initial solution on level $\ell+1$, the restricted residual data $f_{\ell+1}$ requires an adjustment by $A_{\ell+1}(u_{\ell+1}^{(0)})$. Accordingly, in Algorithm 5, lines 7 and 8 are changed to

$$\begin{aligned} u_{\ell+1}^{(0)} &= \Pi_\ell^{\ell+1} u_\ell^{(\nu)} \\ f_{\ell+1} &= \mathbb{R}_\ell^{\ell+1}(f_\ell - A_\ell(u_\ell^{(\nu)})) + A_{\ell+1}(u_{\ell+1}^{(0)}) \end{aligned}$$

In the CNN context, the projection operation in Equation (4.2) corresponds to another pooling operation, but it has no exact counterpart in the general ResNet architecture. Figure 4.3 summarizes the relevant mappings, including the role of B_ℓ as feature extractor and A_l as data-feature mapping, as well as the restriction and projection between resolution levels.

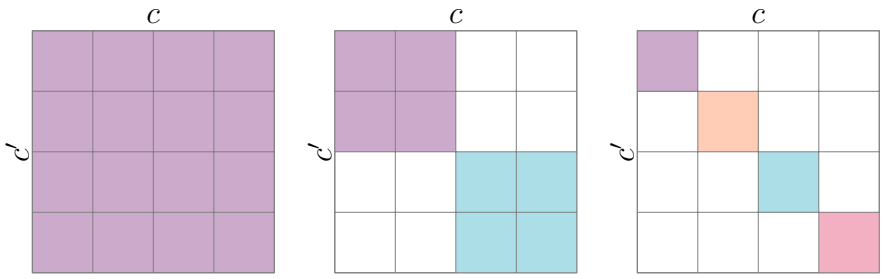


Figure 4.4: Connectivity patterns of convolution operations in matrix representation: regular convolution with $\mathsf{g}_s = c_{\text{in}}$ (left); $\mathsf{g}_s = c_{\text{in}}/\mathsf{g}$ (middle); depthwise, i.e., $\mathsf{g}_s = 1$ (right).

4.4 Multigrid in all Dimensions (MGiaD)

We start this section by establishing channel coarsening in Section 4.4.1 and then combining it with resolution coarsening in MgNet [50] to obtain multigrid in all dimensions in Section 4.4.2.

4.4.1 Channel Coarsening

While the convolutional and hierarchical structure of CNNs allows for an efficient treatment of the resolution dimensions, with linear scaling of the weight count with respect to these dimensions, the situation is completely different with respect to the channel dimension. Here, typically no restriction is imposed on the connectivity structure, i.e., the convolutional maps are dense with respect to the coupling of input and output channels. Clearly, the full coupling of channels enables efficient exchange of information but poses the problem of quadratic scaling in the weight count and could lead to a poor accuracy-weight trade-off due to redundancies. Assuming that the number of channels cannot be reduced, a decrease in the weight count is only possible by addressing their connectivity. Unfortunately, one cannot profit from an invariance assumption analogous to that underlying convolutional connections in the resolution dimensions. Thus, any reduction in the channel connectivity is of an ad hoc nature. The most straightforward way to limit the connectivity is to group channels into g groups such that information is exchanged only within each group (cf. Section 3.2.4). The corresponding connectivity pattern is depicted in Figure 4.4. Such a strategy reduces the weight count from $\mathcal{O}(s^2 \cdot c \cdot h)$ to $\mathcal{O}(s^2 \cdot c \cdot h/\mathsf{g})$ [71]. While replacing A_ℓ, B_ℓ, or both by grouped convolutions of group size g_s, with $\mathsf{g}_s < c$, significantly reduces the weight count, the lack of interaction between channels also decreases the accuracy (cf. Figure 4.5). To facilitate efficient channel interaction, a grouped restriction mapping was introduced in [32],

$$\widehat{R}_{\ell,\kappa}^{\kappa+1} : \mathbb{R}^{m_\ell \times n_\ell \times c_{\ell,\kappa}} \mapsto \mathbb{R}^{m_\ell \times n_\ell \times c_{\ell,\kappa+1}} \tag{4.3}$$

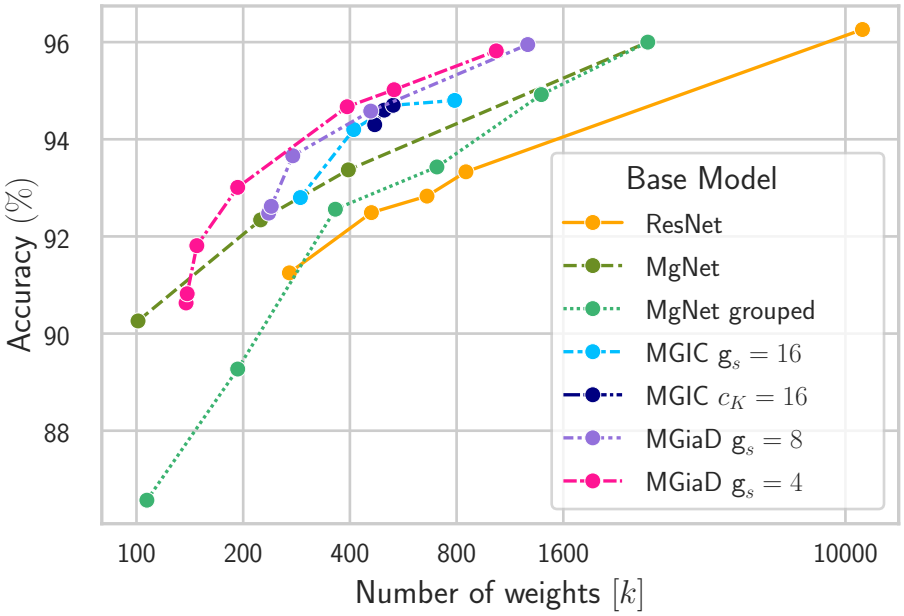


Figure 4.5: Accuracy-weight trade-off on CIFAR-10 for different ResNet, MgNet, MGIC and MGiaD models. For MGiaD, the group size $\mathtt{g}_s \in \{4, 8\}$ is fixed while the number of channels c_K varies.

with $c_{\ell,\kappa+1} = {}^{c_{\ell,\kappa}}/_2$, halving the number of channels, e.g., as depicted for two in-channel levels in Figure 4.6. Analogous to a multigrid V-cycle (cf.Algorithm 1) [107], another grouped mapping

$$\widehat{P}^{\kappa}_{\ell,\kappa+1} : \mathbb{R}^{m_\ell \times n_\ell \times c_{\ell,\kappa+1}} \mapsto \mathbb{R}^{m_\ell \times n_\ell \times c_{\ell,\kappa}} \tag{4.4}$$

is defined to refine the number of channels. This prolongation map interpolates features $\widehat{u}_{\ell,\kappa+1}$ from a coarse in-channel level to a fine in-channel level, starting from the coarsest in-channel level, which uses a dense CNN-block, e.g., a ResNet-block [32]. As in multigrid, such a coarse-level update on κ is given by

$$\widehat{u}_{\ell,\kappa} = \widehat{u}_{\ell,\kappa} + \widehat{P}^{\kappa}_{\ell,\kappa+1}(\widehat{u}_{\ell,\kappa+1}).$$

Using this multigrid strategy for the channel dimension allows for a significant reduction in the number of weights without sacrificing much accuracy [32].

4.4.2 Multigrid in Resolution and Channel Dimensions

To obtain an architecture that ultimately scales linearly in the weight count with respect to all problem dimensions (resolution and channels), we connect the ideas of MgNet and FAS and introduce an multigrid smoothing-in-channels (`SiC`) block. Our `SiC`-block, depicted in Figure 4.6, uses Equation (4.3) and Equation (4.4) to build an

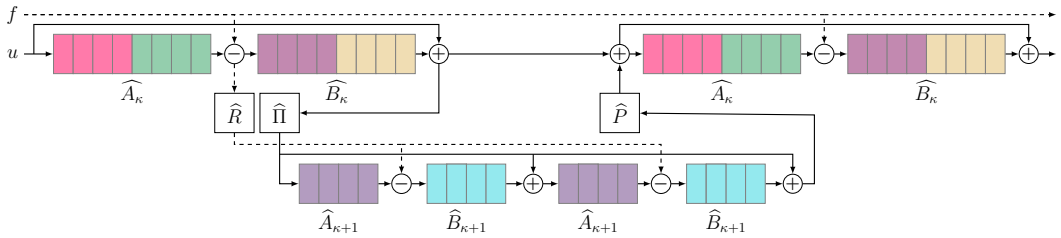


Figure 4.6: `SiC`-block on one resolution level with two in-channel levels. The convolutions $\widehat{A}$ and $\widehat{B}$ are grouped with group size $\mathtt{g}_s = 4$, as indicated by the colors. On each in-channel level, $\widehat{A}$ and $\widehat{B}$ are shared between pre-smoothing (left) and post-smoothing (right). Operators on the second, coarser level are different from those on the first level, but also shared. The transfer operators $\widehat{\Pi}$ and $\widehat{R}$ halve the number of channels while the prolongation mapping $\widehat{P}$ doubles the number of channels. Indices are omitted to simplify notation.

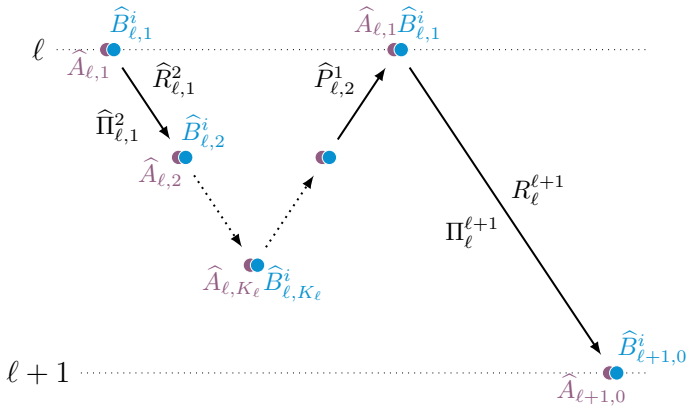


Figure 4.7: Structure of `SiC`-blocks on resolution levels ℓ and $\ell+1$.

in-channel hierarchy, incorporating the smoothing iteration from Equation (4.1) with weights shared within the in-channel level κ. To be more precise, we replace the maps A_ℓ and $B_\ell^{(\tau)}$ in Algorithm 5 by an in-channel V-cycle as follows. The convolutions A_ℓ and $B_\ell^{(\tau)}$, which are fully connected with respect to the channel dimension, are replaced by grouped convolutions $\widehat{A}_{\ell,1}$ and $\widehat{B}_{\ell,1}^{(\tau)}$, respectively. In addition, we introduce grouped convolutions $\widehat{A}_{\ell,\kappa}$ and $\widehat{B}_{\ell,\kappa}^{(\tau)}$ for $\kappa = 1, 2, \ldots, K_\ell$, as well as restrictions $\widehat{R}_{\ell,\kappa}^{\kappa+1}$ and interpolations $\widehat{P}_{\ell,\kappa+1}^{\kappa}$ for $\kappa = 1, 2, \ldots, K_\ell - 1$. Here, the number of levels K_ℓ is chosen such that $\widehat{A}_{\ell,K_\ell}$ and $\widehat{B}_{\ell,K_\ell}$ are fully connected with respect to the channel dimension. The overall structure of the resulting method is sketched in Figure 4.7. On each in-channel level κ, the convolutions $\widehat{A}_{\ell,\kappa}$ and $\widehat{B}_{\ell,\kappa}^{(\tau)}$ are arranged as in Equation (4.1), i.e., as a convolutional in-channel smoothing iteration. Analogous to an FAS-type projection $\Pi_\ell^{\ell+1}$ in MgNet, we introduce an in-channel FAS projection map

$$\widehat{\Pi}_{\ell,\kappa}^{\kappa+1} : \mathbb{R}^{m_\ell \times n_\ell \times c_{\ell,\kappa}} \mapsto \mathbb{R}^{m_\ell \times n_\ell \times c_{\ell,\kappa+1}}.$$

Clearly, both $\widehat{R}_{\ell,\kappa}^{\kappa+1}$ and $\widehat{\Pi}_{\ell,\kappa}^{\kappa+1}$ must also be grouped mappings to obtain linear scaling of the weight count with respect to the channel dimension. Our final multigrid smoothing-in-channels algorithm (`SiC`) is summarized in Algorithm 6.

Algorithm 6 (MG) Smoothing in channels $\texttt{SiC}(f_\kappa, u_\kappa)$

for $\tau = 1, \dots, \eta_{\text{pre}}$ **do**
  $u_\kappa = u_\kappa + \widehat{B}_\kappa^{(\tau)}(f_\kappa - \widehat{A}_\kappa(u_\kappa))$ ▷ pre-smoothing
end for
if $\kappa \neq K$ **then**
  $u_{\kappa+1} = \widehat{\Pi}_\kappa^{\kappa+1}(u_\kappa)$
  $f_{\kappa+1} = \widehat{R}_\kappa^{\kappa+1}(f_\kappa - \widehat{A}_\kappa(u_\kappa)) + \widehat{A}_{\kappa+1}(u_{\kappa+1})$
  $\widehat{u}_{\kappa+1} = \texttt{SiC}(f_{\kappa+1}, u_{\kappa+1})$
  $u_\kappa = u_\kappa + \widehat{P}_{\kappa+1}^\kappa(\widehat{u}_{\kappa+1})$
end if
for $\tau = 1, \dots, \eta_{\text{post}}$ **do**
  $u_\kappa = u_\kappa + \widehat{B}_\kappa^{(\tau)}(f_\kappa - \widehat{A}_\kappa(u_\kappa))$ ▷ post-smoothing
end for

The weight count thus scales linearly rather than quadratically with respect to the channel dimension, compared with the fully connected structure in ResNet or MgNet. Replacing the smoothing iteration in MgNet by `SiC`-blocks finally yields an MG-like architecture whose weight count scales linearly with respect to all dimensions. The resulting method, termed multigrid in all dimensions (MGiaD), is given in Algorithm 7.

Algorithm 7 MG in all dimensions $\texttt{MGiAD}(f_\ell)$

Initialization: $u_1 = 0$
for $\ell = 1, \dots, L$ **do**
  $u_\ell = \texttt{SiC}(f_\ell, u_\ell)$
  if $\ell < L$ **then**
    $u_{\ell+1} = \Pi_\ell^{\ell+1}(u_\ell)$
    $f_{\ell+1} = R_\ell^{\ell+1}(f_\ell - A_\ell(u_\ell)) + A_{\ell+1}(u_{\ell+1})$
  end if
end for

4.4.3 Experimental Setup and Evaluation

We evaluate our approach on improving the accuracy-weight trade-off for classification tasks on several popular datasets such as CIFAR-10, CIFAR-100 [70], Fashion-MNIST [121], Tiny ImageNet [72] and ImageNet [71]. We report the number of weights, as well as the train and test accuracies with standard deviation (std), obtained from three runs

with random seeds. Since the standard deviation of the train accuracy is less than 0.01, it is omitted in the following.

Training setup. Our models are implemented in PyTorch [88]. Unless otherwise stated, we train the models with a batch size of 128 for 400 epochs using an SGD-optimizer with a momentum of 0.9 and a weight decay of 10^{-4}. In accordance with ResNet [53], we use batch normalization followed by a ReLU activation function after every convolutional layer. The initial learning rate is set to 0.05 and we use a cosine-annealing learning rate schedule [80].

Evaluation of ResNet and MgNet on CIFAR-10. The CIFAR-10 dataset contains $60k$ color images of size 32×32 across 10 classes. We compare our MGiaD approach to ResNet18, ResNet20, and the corresponding MgNet architectures. The ResNet18 architecture is composed of four resolution levels with $[64, 128, 256, 512]$ channels and two ResNet-blocks on each resolution level. According to [50], we reduce the number of channels on the last resolution level from 512 to 256 in MgNet. ResNet20, a ResNet version specifically designed for CIFAR-10, is composed of three resolution levels with three ResNet-blocks on each level and $[16, 32, 64]$ channels. The parameters of the MgNet architectures are chosen accordingly in all cases. We include tests of MgNet with only A shared, referred to as MgNet^{A}, and with both A and B shared, referred to as $\text{MgNet}^{\text{A,B}}$. In our experiments, we observe that sharing convolutions within the ResNet-blocks significantly reduces the weight count of the models at the cost of a minor decrease in performance. For example, ResNet18 has $11{,}174k$ weights and achieves 96.26% accuracy, while the corresponding $\text{MgNet}^{\text{A,B}}$, i.e., with both A and B shared, achieves an accuracy of 96% with $2{,}751k$ weights. The smaller ResNet20 has $270k$ weights and achieves an accuracy of 92.44%, while the corresponding $\text{MgNet}^{\text{A,B}}$ has $101k$ weights and achieves an accuracy of 91.26%. Based on these findings, as well as those of He and Xu [51], we opt to share A and B by default when reporting results for MGiaD. In all tests, we choose learnable depthwise pooling operations Π and R in both MgNet and MGiaD.

Grouped convolutions in MgNet. To justify the use of a hierarchical structure in the channel dimension, we discuss replacing fully coupled convolutions by grouped convolutions in the ResNet-type architecture MgNet. The reduced number of connections between the grouped blocks limits the interaction between channels. While this achieves a significant reduction in the weight count, we observe in Figure 4.5 that the performance drops significantly. These findings can be attributed to the reduced exchange of information across channels.

Table 4.1 reports the influence of the group size g_s on the number of weights and the accuracy. Compared to ResNet18 and MgNet, both with fully coupled convolutions, smaller group sizes g_s in A and B result in a drastic reduction in the weight count,

Model	g_s	# weights (k)	accuracy (%) ± std	
			test	train
ResNet18	c	11,174	**96.26** (0.16)	98.14
MgNet^{A,B}	c	2,751	96.00 (0.27)	97.60
	64	831	93.88 (0.15)	95.50
	32	425	92.05 (0.28)	92.86
	16	223	89.74 (0.59)	88.08
	8	121	87.02 (0.76)	82.32

Table 4.1: Results of MgNet with four resolution levels and different group sizes g_s in convolutions A and B, with ResNet18 included as a reference on CIFAR-10. The group size for a fully coupled convolution equals the number of channels c. The model with the highest overall accuracy is highlighted with a purple background.

but also in the accuracy of MgNet. A model using convolutions with a rather large group size $\mathsf{g}_s = 32$ has $425k$ weights but achieves an accuracy of only 92%. The highest accuracy among the MgNet models with grouped A and B is achieved with $\mathsf{g}_s = 64$, resulting in $831k$ weights. These results indicate that a naive sparsification of the channel dimension without a hierarchical structure is insufficient and that a more sophisticated sparsification scheme in the channel dimension is required, as provided by our MGiaD architecture.

Evaluation of MGiaD on CIFAR-10. To identify the parameters that play a role in the accuracy-weight trade-off, we perform a large-scale parameter study on CIFAR-10. We start by studying the influence of the group size g_s and the size of the coarsest level c_K within the channel multigrid subcycle, followed by an analysis of channel scaling and post-smoothing.

Fully coupled channels and group sizes in MGiaD. In Table 4.2 we study the influence of the coarsest level c_K and g_s in combination. These results clearly indicate that the size of the fully connected coarsest level of the multigrid subcycle has a significant influence on the performance of the resulting network, while the group size of the grouped convolutions $\widehat{A}$ and $\widehat{B}$ has a much smaller impact. However, a small group size strongly affects the weight count.

With a group size of 8 and $c_K = 64$, the weight count of MGiaD is reduced by a factor of 24 with a small drop in accuracy of 1.7 pp, compared to ResNet18. Moreover, at $\mathsf{g}_s = 4$ and $c_K = 64$ we obtain an architecture with a weight count reduced by a factor of 30 compared to ResNet18, at a cost of only 1.6 pp in accuracy. Even compared to the weight-efficient MgNet^{A,B}, this architecture yields comparable performance at a 7× reduction in weight count. The more weight-efficient ResNet20 comes with only $270k$ weights. For MGiaD with $\mathsf{g}_s = 8$ and $c_K = 32$, we obtain a model with a similar weight count and substantially improved accuracy. For $g_s = 4$ and $c_K = 16$, we obtain roughly

Model	$\mathtt{g}_s$	λ	c_K	#**weights** (k)	**accuracy** (%) ± **std**	
					test	train
ResNet20	-	-	-	270	**92.44** (0.16)	93.54
$\text{MgNet}^{\text{A,B}}$	-	-	-	101	91.26 (0.09)	90.88
ResNet18	-	-	-	11,174	**96.26** (0.16)	98.14
$\text{MgNet}^{\text{A,B}}$	-	-	-	2,751	96.00 (0.27)	97.60
MGiaD	4	1	4	138	90.63 (0.16)	88.72
			8	139	90.82 (0.14)	88.93
			16	148	91.81 (0.35)	90.41
			32	193	93.01 (0.22)	93.13
			64	393	**94.62** (0.10)	95.83
	8	1	8	236	92.48 (0.37)	92.46
			16	240	92.62 (0.16)	92.69
			32	276	93.66 (0.25)	94.29
			64	458	**94.58** (0.25)	96.11
	16	1	16	424	94.13 (0.15)	95.29
			32	441	94.31 (0.14)	95.70
			64	586	**94.72** (0.35)	96.44
	32	1	32	773	94.97 (0.37)	96.65
			64	845	**95.20** (0.20)	96.81
	64	1	64	1,361	**95.47** (0.07)	97.23

Table 4.2: Influence of the number of channels on the coarsest level c_K and the group size $\mathtt{g}_s$ on accuracy and weight count for MGiaD on CIFAR-10. The model with the highest overall accuracy is highlighted with a purple background.

half the number of weights of ResNet20, while being on par accuracy-wise. All results are summarized in Figure 4.5. These findings show that our MGiaD is very efficient with fewer weights than its residual counterpart.

Channel scaling in MGiaD. In the previous experiments, we showed that a group size of 4 or 8 with $c_K = 64$ fully connected channels on the coarsest in-channel level reduces the weight count while maintaining accuracy. To utilize the freed-up capacity in terms of weight count, we introduce a channel scaling parameter λ, which is used to scale the initial number of channels at each resolution. In general, a higher overall number of channels leads to a deeper hierarchy, more parameters, and improved performance, as depicted in Figure 4.5.

We report results for $\lambda \in \{1, 2, 3\}$ in Table 4.3. Increasing the number of channels naturally increases the number of weights, which generally has a beneficial effect on accuracy. For a group size of 8 and $\lambda = 3$, the resulting model achieves almost 96% accuracy, with 1,269k weights, corresponding to approximately half the weight count of MgNet and a one eighth that of the ResNet18. This shows that, as long as the overall

Model	g_s	λ	**#weights** (k)	**accuracy** (%) ± **std**	
				test	train
ResNet18	c	-	11,174	**96.26** (0.16)	98.14
MgNet$^{\text{A,B}}$	c	-	2,751	96.00 (0.27)	97.60
MGiaD	4	1	393	94.62 (0.10)	95.83
		2	533	95.02 (0.21)	96.32
		3	1,020	**95.64** (0.09)	97.21
	8	1	458	94.58 (0.25)	96.11
		2	703	94.89 (0.06)	95.77
		3	1,269	**95.95** (0.12)	97.44

Table 4.3: Influence of the number of initial channels, scaled by $\lambda \in \{1, 2, 3\}$, on accuracy and weight count for MGiaD with $c_K = 64$ and $\mathsf{g}_s \in \{4, 8\}$. ResNet and MgNet results are included for comparison.

Model	g_s	η_{post}	**#weights** (k)	**accuracy** (%) ± **std**	
				test	train
MGiaD	4	1	1,020	95.64 (0.09)	97.21
	4	2	1,035	**95.83** (0.13)	97.25
	4	3	1,051	95.53 (0.17)	97.32
MGiaD	8	1	1,269	**95.95** (0.12)	97.44
	8	2	1,276	95.70 (0.26)	97.35
	8	3	1,300	95.90 (0.23)	97.31

Table 4.4: Influence of post-smoothing η_{post} in MGiaD with a channel scaling factor of $\lambda = 3$, $c_K = 64$ and $\mathsf{g}_s \in \{4, 8\}$ on CIFAR-10.

cost with respect to the channel dimension is linear, as in MGiaD, trading channel connectivity for channel dimension is beneficial.

Post-smoothing in MGiaD. Another way to improve performance is the reuse of $\widehat{A}$ and $\widehat{B}$ in the channel multigrid subcycle in a fashion akin to post-smoothing in multigrid [107], where such a process is known to reduce the time and computational work to solve the problem. Correspondingly, we add multiple post-smoothing steps η_{post} to the channel hierarchy. As the weight tensors are shared on within one in-channel level, only weights for new additional normalization layers are added to the total. We report results for slightly more complex MGiaD models with $\mathsf{g}_s = 8$ and $\mathsf{g}_s = 4$ and $\lambda = 3$ for varying numbers of post-smoothing iterations η_{post} in Table 4.4. Consistent with the observations made in Table 4.3, Table 4.2 and Figure 4.5, a larger group size improves the accuracy, while the effect of the number of post-smoothing iterations is ambiguous. We compile all results for CIFAR-10 in terms of weight count versus test accuracy in Figure 4.5, which also includes results for the MGIC method by Eliasof et al. [32]. We obtain favorable accuracy-weight trade offs using $\mathsf{g}_s = 4$ and varying values of c_K and λ. However, diminishing returns begin to set in at around $500k$ weights and

Model	λ	g_s	#weights (k)	GFLOPs	accuracy (%) ± std test	train
ResNet18	-	-	11,174	1.12	96.26 (0.16)	98.14
$\text{MgNet}^{\text{A,B}}$	-	-	2,751	1.40	96.00 (0.27)	97.60
MGiaD	3	4	1,020	1.14	95.64 (0.09)	97.21
	3	8	1,270	1.34	95.95 (0.12)	97.44

Table 4.5: Computational costs in terms of GFLOPs and weight counts for ResNet18, MgNet, and best-performing MGiaD models on CIFAR-10. We report MGiaD models with group sizes $\mathsf{g}_s = 4$ and 8, which are built with $c_K = 64$ fully coupled channels and $\lambda = 3$.

the best result we achieve uses $\mathsf{g}_s = 8$ combined with $\lambda = 3$. Due to the reduced number of channels through the multigrid hierarchy and shared weight tensors, the number of weights is drastically reduced. However, our MGiaD models require a similar number of FLOPs to achieve the same performance as ResNet and MgNet (cf. Table 4.5).

Evaluation on FashionMNIST. FashionMNIST contains $70k$ 32×32 grayscale images in 10 classes. The initial learning rate is set to 0.05 and multiplied by 0.1 every 25 epochs. In all our experiments, we observe strong overfitting, as reflected by the results reported in Table 4.6. However, MGiaD still improves the accuracy-weight trade-off. In particular, compared to ResNet18, MGiaD with $c_K = \mathsf{g}_s = 64$ achieves the same accuracy with a weight count reduced by a factor of 8. Additionally, the weight count for MGiaD can be cut by another factor of 3 while sacrificing only 0.1 pp in accuracy. To reduce overfitting, we decrease the number of resolution levels from 4 to 3 for experiments with $c_K = 16$. The resulting architectures are compared to ResNet20, which also consists of 3 resolution levels. We indeed observe a mild decrease in overfitting along with a reduction in weight count by a factor of 2.7.

Evaluation on CIFAR-100. CIFAR-100 contains 100 classes with 600 images each and otherwise has the same specifications as CIFAR-10. For CIFAR-10 and FashionMNIST, we observed in the previous experiments, that a large number of fully connected channels has a significant influence on the accuracy.Thus, we choose $c_K = 64$. In Table 4.7 we study the influence of the group size g_s and the number of channels, varied through the channel multiplier λ, on the accuracy. Similar to CIFAR-10, we observe that a higher weight count generally leads to a better accuracy. However, unlike for CIFAR-10, where the group size has only a minor impact, a large group size is essential for high accuracy on CIFAR-100. For example, a model with $\mathsf{g}_s = 8$ and $481k$ weights achieves an accuracy of almost 70%, whereas its counterpart with $\mathsf{g}_s = 64$ achieves over 72% with a substantially higher weight count of more than $1{,}300k$. Consistent with our CIFAR-10 results, channel scaling successfully utilizes the freed-up capacity. Multiplying the number of channels by $\lambda = 3$ results in the best overall accuracy. The resulting model has less than half the weight count of ResNet18 while improving its accuracy by 0.4 pp.

Model	c_K	$\mathtt{g}_s$	**#weights** (k)	**accuracy** (%) ± **std** test	train
ResNet18	-	-	11,003	**93.84** (0.17)	100
MgNet^{A,B}	-	-	2,747	**93.84** (0.16)	100
MGiaD	64	4	389	93.45 (0.12)	100
		64	1,357	**93.84** (0.10)	100
	32	4	189	93.32 (0.06)	100
		32	437	**93.71** (0.49)	100
	16	4	144	93.28 (0.08)	100
		16	420	**93.40** (0.21)	100
ResNet20	-	-	270	93.02 (0.31)	100
MgNet^{A,B}	-	-	101	**93.29** (0.15)	100
MGiaD	16	4	28	92.85 (0.06)	97.07
		8	37	**93.35** (0.09)	97.63
		16	55	93.29 (0.15)	98.48

Table 4.6: Influence of the number of fully coupled channels c_K and the group size $\mathtt{g}_s$ on accuracy and weight count for models trained on FashionMNIST. The highest accuracy in each comparison is marked in bold, and the best MGiAD model is highlighted in purple.

Evaluation on Tiny ImageNet. Tiny ImageNet consists of $100k$ 64×64 color images in 200 equally sized classes. Based on our previous findings, we do not tune c_K for this dataset specifically, i.e., we keep $c_K = 64$. In Table 4.7 we study accuracy and weight count as a function of both the group size $\mathtt{g}_s \in \{64, 64\}$ and the channel scaling λ. We compare the resulting models with ResNet18 and the corresponding MgNet. Consistent with our observations on CIFAR-100, a higher weight count generally leads to higher accuracy. With a group size of $\mathtt{g}_s = 64$, MGiaD reduces the weight count by a factor of 8 compared to ResNet and a factor of 2 compared to MgNet, while improving upon the accuracy of both architectures by at least 0.5 pp to 60.36%. Scaling the inital number of channels by $\lambda = 3$ leads to an accuracy gain of 3 pp over ResNet18 while requiring 2.3 times fewer weights.

Evaluation on ImageNet. The ImageNet database contains color images with a resolution of 224×224 across 1,000 semantic classes. As for Tiny ImageNet, we fix the number of fully coupled channels on the coarsest level to $c_K = 64$ and study the influence of the group size $\mathtt{g}_s \in \{8, 64\}$ in combination with $\lambda \in \{1, 2, 3\}$. We compare the resulting models with ResNet18 and MgNet. All models are trained with a batch size of 512, except for MGiaD with $\lambda = 3$, for which the batch size is reduced to 256 due to memory limitations. Note that this limitation could be remedied by an optimized implementation. Our results are reported in Table 4.7. In accordance with previous results on the other datasets, a higher weight count results in higher accuracy. Again, we find that channel scaling is a powerful tool to improve accuracy by utilizing the capacity freed up

Dataset			CIFAR-100			Tiny ImageNet			ImageNet		
Model	λ	g_s	#weights (k)	accuracy (%) (± std) test	train	#weights (k)	accuracy (%) (± std) test	train	#weights (k)	accuracy (%) (± std) test	train
ResNet18	-	-	11,220	**75.42** (0.13)	99.98	11,271	59.67 (0.66)	91.29	11,690	**71.89** (0.04)	75.26
MgNet^{A,B}	-	-	2,774	74.42 (0.28)	99.98	2,799	**60.12** (0.25)	87.77	3,013	67.83 (0.10)	64.87
MGiaD	1	8	481	69.91 (0.37)	99.25	508	56.23 (0.50)	85.37	721	59.61 (0.14)	58.12
		64	1,384	**72.53** (0.45)	99.97	1,411	**60.36** (0.54)	70.99	1,625	**66.38** (0.36)	63.35
	2	8	745	71.48 (0.48)	99.91	801	58.33 (0.09)	64.35	1,226	64.82 (0.20)	60.40
		64	3,067	**75.12** (0.71)	99.98	3,123	**62.24** (0.99)	81.17	3,549	**72.12** (0.12)	69.67
	3	8	1,338	72.75 (0.62)	99.97	1,422	59.87 (0.13)	70.49	2,060	68.23 (0.13)	64.09
		64	4,822	**75.85** (0.14)	99.98	4,906	**62.68** (0.75)	84.71	5,544	**74.39** (0.24)	72.50

Table 4.7: Influence of the overall number of channels, scaled by λ, on accuracy and weight count for CIFAR-100, Tiny ImageNet and ImageNet. For MGiaD, we set $c_K = 64$ and we study group sizes 64 and 8. For each dataset, the highest accuracy in each comparison is marked in bold, and the best MGiaD model is highlighted in purple.

by the multigrid hierarchy in the channel dimension. For example, an MGiaD model with $g_s = 64$ and $\lambda = 2$ improves the accuracy by 0.23 pp while reducing the weight count by a factor of 3 compared to ResNet18. with $g_s = 8$ and $\lambda = 3$, MGiaD reduces the weight count by approximately one third compared to MgNet while improving the accuracy by 0.4 pp. This demonstrates that our MGiaD architecture yields an improved accuracy-weight trade-off and reduces overparameterization on ImageNet.

4.4.4 Conclusion

In this work, we introduced a neural network architecture that utilizes the concept of multigrid in spatial and channel dimensions. Our experiments suggest that, although the reduction in weight count introduces an additional architectural bias, this bias does not seem to affect the overall performance of the network in most cases, in particular if the network is overparameterized. In problems requiring only limited capacity, e.g., CIFAR-10, the proposed architecture substantially reduces the weight count while almost maintaining performance in terms of accuracy. These observations generalize to larger problems such as Tiny ImageNet and ImageNet. For ImageNet, we even find architectures with an accuracy superior to ResNet18 and MgNet while requiring fewer weights. Our approach offers another way to address overparameterization in neural networks and achieves improved scaling behavior with respect to the depth and width hyperparameters. The follow-up study investigates how the smoother B can be replaced by a polynomial in A to additionally reduce the weight count.

4.5 Poly-MgNet: Polynomial Building Blocks in Multigrid-Inspired ResNets

In this section we develop polynomial building blocks inspired by multigrid smoothers. We explain the method in Section 4.5.1 and present an extensive parameter study in Section 4.5.2.

4.5.1 Polynomial (Smoothers) in Residual Blocks

Thus far the iterative scheme, Equation (4.5), was either considered to be stationary or non-stationary with matrices B or $B^{(\tau)}$, respectively. Even though B is a rough and computationally inexpensive approximation of A^{-1}, in order to achieve a reduction of the residual we can further reduce the number of learnable weights and increase the interpretability of B by replacing it by a polynomial (smoother) $p_d(A) \approx A^{-1}$ of degree $d \in \mathbb{N}_+$, i.e., the residual block then reads

$$u \leftarrow u + p_d(A)(f - A(u)). \tag{4.5}$$

Here, p_d is a polynomial

$$p_d(A) = \sum_{i=0}^{d} \alpha_i A^i, \tag{4.6}$$

with (learnable) coefficients α_i. In the case of polynomial smoothers ($B = p_d(A)$), the non-stationary case is considered, i.e., only the weight tensor A is shared. Furthermore, these models are denoted by MgNet^{p_d}.

In order to ease the following discussion note that under the assumption that A is diagonalizable, i.e., $A = X\Lambda X^{-1}$ with a matrix X containing the eigenvectors of A as its columns and a diagonal matrix Λ of eigenvalues, we find

$$p_d(A) = \sum_{i=0}^{d} \alpha_i A^i = X \left(\sum_{i=0}^{d} \alpha_i \Lambda^i \right) X^{-1}. \tag{4.7}$$

Thus, the action of the polynomial on a matrix A is determined solely by the evaluation of the scalar polynomial p_d, with coefficients α_i, on the eigenvalues of A. As this alleviates notation, we consider p_d both as a scalar polynomial and as a matrix-valued polynomial, using the same notation. Moreover, let $\Lambda = \mathrm{spec}(A)$ henceforth.

Residual blocks as polynomials. Recall that the residual propagates according to

$$r^{(\tau+1)} = f - Au^{(\tau+1)} = (I - AB^{(\tau)})r^{(\tau)}. \tag{4.8}$$

Let $B = p_d(A)$ be a polynomial feature extractor. Then

$$r^{(\tau+1)} = (I - Ap_d(A))r^{(\tau)} = q_{d+1}(A)r^{(\tau)},$$

where the factor $(I - Ap_d(A))$ itself is a polynomial $q_{d+1}(A)$ of degree $d + 1$. Hence, using Equation (4.6), we find

$$q_{d+1}(A) = I - Ap_d(A) = I - \sum_{i=0}^{d} \alpha_i A^{i+1},$$

and see that q_{d+1} is normalized with a constant coefficient equal to 1, i.e., $q_{d+1}(0) = 1$, which implies that residual components belonging to kernel modes of A are unaffected by such a polynomial correction approach.[1] Leveraging the normalization of q_{d+1}, we obtain its decomposition into a product of linear factors as

$$q_{d+1}(A) = \prod_{i=1}^{d+1}(I - \frac{1}{\zeta_i}A) \text{ with } q_{d+1}(\zeta_i) = 0 \text{ and } \zeta_i \in \mathbb{C}.$$

That is, instead of learning the coefficients α_i, we can equivalently learn the roots of the polynomial q_{d+1}. While it is not immediately clear how the α_i need to be chosen to ensure $p_d(A) \approx A^{-1}$ to obtain a reduction of the residual, the roots of the polynomial q_{d+1} have an immediate interpretation with respect to the reduction of the residual. Again assuming that A is diagonalizable and taking into account that polynomials tend to be small only close to their roots, we write $r^{(\tau)} = \sum_j \gamma_j x_j$ as a linear combination of the eigenvectors of A and find

$$q_{d+1}(A)r^{(\tau)} = \sum_j \prod_{i=1}^{d+1} \left(1 - \frac{\lambda_j}{\zeta_i}\right) \gamma_j x_j, \tag{4.9}$$

which is small if the roots ζ_i conjugated capture the distribution of the eigenvalues of A correctly. To be more specific it is clear that roots close to the boundaries of the spectrum of A, $\Lambda \subseteq \mathbb{C}$, are required in order to prevent catastrophic overcorrection of the respective eigencomponents of the residual [96] (cf.Figure 4.8).

Yet, the aim of minimizing the residual is consistent with keeping the polynomial small on the spectrum. To that end, eigenvalues located at the spectrum's boundary are chosen as roots for the polynomial. Obviously, $A^{\mathrm{T}} \neq A$, and since A is real, its complex eigenvalues occur in complex-conjugate pairs. This can be used to construct a polynomial in linear-factor representation, covering the extent of the real axis, as examplarily

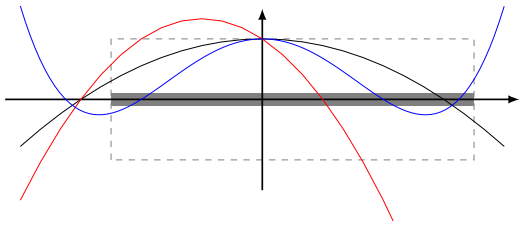


Figure 4.8: Schematic illustration of polynomials of type q_2 (two blocks) with different choices of roots from a representative real spectrum.

illustrated in Figure 4.8, and quadratic factors, covering the extent of the imaginary axis and avoiding complex arithmetic within the CNN at the same time.[2]

For a complex-conjugate pair z and $\bar{\mathrm{z}}$, the residual propagation in Equation (4.8) yields

$$r^{(\tau+1)} = (1 - \frac{1}{\mathrm{z}}A)(1 - \frac{1}{\bar{\mathrm{z}}}A)r^{(\tau)} = \tilde{q}_{d+1}r^{(\tau)},$$

where $\tilde{q}_{d+1}$ itself is a quadratic polynomial of degree $d+1$ with roots $\mathrm{z} = \mathrm{a} + \mathrm{ib}$ and its complex-conjugate counterpart $\bar{\mathrm{z}}$, where i denotes the imaginary unit. Thus, the resulting polynomial comprises m linear polynomials $\hat{q}_i$ and n quadratic polynomials $\tilde{q}_i$

$$q_{m+2n}(A) = \prod_{i=1}^{m} \hat{q}_i(A) \cdot \prod_{j=1}^{n} \tilde{q}_j(A), \quad \text{with} \quad m + 2n = d + 1.^3 \tag{4.10}$$

The second observation revolves around the fact that the decomposition of q_{d+1} into a product of linear factors can be viewed as a sequence of linear residual blocks, i.e., one residual block with $B = \hat{p}_d(A)$ is equivalent to $d+1$ residual blocks with $B^{(\tau)} = 1/\zeta_\tau$. The quadratic factors can also be viewed as a sequence of pairs of linear residual blocks, i.e., two residual blocks with $B = \hat{q}_{2d}$ are equivalent to $B^{(\tau)} = 1/z$ and $B^{(\tau+1)} = 1/\bar{z}$, respectively. The focus of our work is on the polynomial perspective of a residual block, described by the residual polynomial q. The iteration can be rewritten as

$$u^{(\tau+1)} = u^{(\tau)} + \left(\frac{2\mathrm{a}}{\mathrm{a}^2 + \mathrm{b}^2}I - \frac{1}{\mathrm{a}^2 + \mathrm{b}^2}A\right) r^{(\tau)}.$$

For ease of notation, from now on, the degree of a polynomial q_d is d. Consequently, the corresponding polynomial p has degree $d-1$. Furthermore, combining Equation (4.5) with Equation (4.10) results in a polynomial version of MgNet, referred to as Poly-MgNet, which is outlined in Algorithm 8.

[1] Clearly, such modes are also unaffected in a general residual block with a (full) parameter matrix B.

[2] For example, the ReLU activation function nn.ReLU from the PyTorch library does not support complex values, (cf. issues #47052, #46642).

[3] Remark: the symbols m and n, when used as polynomial degrees should not be confused with the input dimension $m \times n$. In particular, the polynomial degrees, are independent of the input dimension.

Algorithm 8 Poly-\-MgNet(f_ℓ, $\{\zeta_\ell^{(i)}\}_{i=1}^{m+2n}$)

Initialization: $u_1^{(0)} = 0$
for $\ell = 1, \dots, L$ **do**
  for $\tau = 0, \dots, m-1$ **do**
    $u_\ell^{(\tau+1)} = u_\ell^{(\tau)} + \frac{1}{\zeta_\ell^{(\tau+1)}}(f_\ell - A_\ell(u_\ell^{(\tau)}))$
  end for
  for $j = 1, \dots, n$ **do**
    $\mathrm{z} = \zeta_\ell^{(m+2j-1)} \in \mathbb{C}$
    $\mathrm{a} = \Re(\mathrm{z}), \mathrm{b} = \Im(\mathrm{z})$
    $r_\ell^{(m+j-1)} = f_\ell - A u_\ell^{(m+j-1)}$
    $u_\ell^{(m+j)} = u_\ell^{(m+j-1)} + \frac{1}{\mathrm{a}^2+\mathrm{b}^2}(2\mathrm{a} - A) r_\ell^{(m+j-1)}$
  end for
  if $\ell < L$ **then**
    $u_{\ell+1} = \Pi_\ell^{\ell+1} u_\ell^{(m+n)}$
    $f_{\ell+1} = R_\ell^{\ell+1}(f_\ell - A_\ell(u_\ell^{(m+n)})) + A_{\ell+1}(u_{\ell+1})$
  end if
end for

Based on observations from our experiments, we limit the polynomial consisting of linear factors, i.e., $\hat{q}_i$ in Equation (4.10), to $m = 2$. To cover the extent of the spectrum on the real axis, we choose the real part ($\Re$) of eigenvalues with smallest and largest real parts for the polynomial roots, i.e., $\zeta^{(1)} = \min \Re(\Lambda)$ and $\zeta^{(2)} = \max \Re(\Lambda)$ as polynomial coefficients $\alpha^{(\tau)} = 1/\zeta^{(\tau)}$. Models with polynomial building blocks with polynomial degree $d = m = 2$ are denoted by MgNet^{q_2}. For degrees $d \geq 4$, the roots for the quadratic polynomials $\tilde{q}_i$ are chosen as follows. For a single quadratic factor $\tilde{q}_1$, $\mathrm{z} = \mathrm{argmax}(\Im(\Lambda))$ is the eigenvalue with largest imaginary part ($\Im$), and $\bar{\mathrm{z}}$ is its complex-conjugate counterpart. For polynomials of higher degree we continue to choose roots located at the boundary of the spectrum to obtain small polynomial values. The roots calculated for a polynomial of degree $d = 4$ are deployed in the resulting polynomial $q_4(A)$, which is subsequently evaluated on the spectrum. Consequently, the spectrum is a plane spanned by the roots, as exemplified shown in Figure 4.9. The eigenvalue with the largest imaginary part and its complex-conjugate counterpart are then chosen as additional roots for $q_6(A)$. This recursive procedure of spanning the spectrum on the roots to determine new maxima can be repeated as needed, as summarized in Algorithm 9. The resulting models are denoted by Poly-MgNet^{q_d}.

Another observation is that enhancing the impact of the real-valued coefficients by constructing a quadratic version of $\hat{q}$ in Equation (4.10), such that

$$r^{(\tau+1)} = (I - \alpha A^2) r^{(\tau)} = \hat{g}_d(A^2) r^{(\tau)}$$

has a beneficial effect on accuracy. Omitting iteration indices, a polynomial $\hat{g}(A^2)$ with coefficient α has roots at $\pm 1/\sqrt{\alpha}$. Following the idea of keeping the polynomial

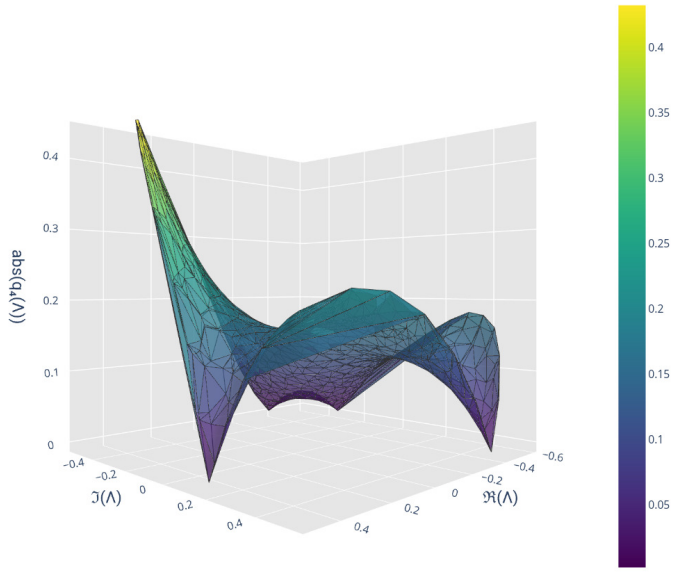


Figure 4.9: Surface representation of the spectrum Λ for the corresponding matrix $A \in \mathbb{R}^{64\times64\times3\times3}$. The x-axis represents the real part, while the y-axis represents the imaginary part. The z-axis illustrates the magnitude of the polynomial $|q_4(\Lambda)|$, which has roots at the eigenvalues with minimum and maximum real parts, as well as the complex-conjugate pair containing the eigenvalue with the maximal imaginary part. This visualization allows for an intuitive identification of the maximum $|q_4(\Lambda)|$ over the spectrum.

Algorithm 9 Roots $\zeta^{(d)}$ for polynomials $d \geq 6$

Initialization:
$\zeta^{(1)} = \min(\Re(\Lambda))$
$\zeta^{(2)} = \max(\Re(\Lambda))$
$\zeta^{(3)} = \operatorname{argmax}(\Im(\Lambda))$
$\zeta^{(4)} = \overline{\zeta^{(3)}}$
for $d = 6, 8, \ldots$ **do**
    $q_{d-2} = \prod_{\tau=1}^{d-2}(I - \frac{1}{\zeta^{(\tau)}}A)$
    $\zeta^{(d-1)} = \operatorname{argmax}(|q_{d-2}(\Lambda)|)$
    $\zeta^{(d)} = \overline{\zeta^{(d-1)}}$
end for

small on the spectrum, we choose the coefficient $\alpha = 1/\zeta^2$ with $\zeta_\tau = \max(|\max(\Re(\Lambda))|, |\min(\Re(\Lambda))|)$ to ensure that the root is at or beyond the boundary of the spectrum. Note that the degree of the overall polynomial g_d is now $d = 2m + 2n$. Consequently, Equation (4.10) is rewritten as

$$g_{2m+2n}(A) = \prod_{i=1}^{m} \hat{g}_i(A^2) \cdot \prod_{j=1}^{n} \tilde{q}_j(A), \tag{4.11}$$

and models that include the reinforced real part polynomial g are denoted by Poly-MgNet^{g_d}. The following discussion refers to $\hat{q}_d$, although it can be applied without restriction to $\hat{g}_d$.

To ReLU or not to ReLU. Thus far, we have ignored the fact that standard ResNet-blocks are typically built with rectified linear unit (ReLU) activation functions $\phi = \max(0, z)$ (cf. Section 3.2.1). In ResNet, as depicted in Figure 4.2, every convolution A and B is followed by ReLU, which also applies to MgNet, independent of shared operations. Furthermore, this setting is applied in the special case $B = p_{d-1}(A)$, especially for $d = 1$. The iterative scheme with ReLU functions is given by

$$u^{(\tau+1)} = u^{(\tau)} + \phi\left(p_{d-1}(A)\,\phi(f - Au^{(\tau)})\right). \tag{4.12}$$

However, our polynomial view of residual blocks conflicts with the fact that the max-function typically cannot be a term of a polynomial. Nevertheless, ReLU has a major influence on the expressiveness of CNNs which prompts us to review ReLU placements in our polynomial residual blocks. To pursue the original idea of ReLU-free polynomials, a single ReLU is applied on every resolution level, namely before resolution coarsening, such that the initial solution is given by

$$u_{\ell+1}^{(0)} = \Pi_\ell^{\ell+1} \phi\,(u_\ell^{(\nu)}), \tag{4.13}$$

where $\nu = m + n$ blocks denotes a polynomial degree $d = m + 2n$. In our experiments, we found that in this setting our models are not able to capture sufficient non-linearity in the data, resulting in a loss in accuracy. To regain expressive capacity of our polynomial models to represent non-linear relationships in the data, we placed ReLU after calculating the residual

$$u^{(\tau+1)} = u^{(\tau)} + p_{d-1}(A)\,\phi\,(f - Au^{(\tau)}), \tag{4.14}$$

which proved to be beneficial to the model performance. Furthermore, the combination of Equation (4.13) and Equation (4.14) proved the most beneficial for accuracy. The introduced options are summarized in Table 4.8.

Another question of interest is the placement of batch normalization (bn) operations within the polynomials. While their role is not fully understood, we found that different placements affect the classification accuracy of the model. Although bn is typically applied before ReLU, this ordering is not required. In the following section, we analyze various bn and ReLU configurations in MgNet^{q_d} to identify the most effective setup.

	$\phi\, u_\ell^{(\nu)}$	$\phi\, p_{d-1}(A)$	$\phi\, r$
Equation (4.12)		×	×
Equation (4.13)	×		
Equation (4.14)			×
Equation (4.13) + Equation (4.14)	×		×

Table 4.8: Overview of possible placements of the ReLU function ϕ in a polynomial block.

4.5.2 Experimental Setup and Evaluation

We study the effect of our polynomial building blocks $q_d(A)$ on the accuracy-weight trade-off by evaluating the corresponding models on CIFAR-10. We report the weight count and mean train and test accuracy with standard deviation (std.) over three runs. Although Poly-MgNet is robust to different coefficient initializations (cf. Table 4.9), we initialize the coefficients of our models based on the spectra of the convolution operators to align with our underlying intuition.

Experimental setup. Our models are implemented in PyTorch [88]. We train our models in batches of 128 for 400 epochs and use stochastic gradient descent with a momentum of 0.9 and a weight decay of 10^{-4}. The initial learning rate is set to 0.05, which is adapted by a cosine-annealing scheduler [80].

Weight count of residual blocks. As the weight tensor A is shared across the polynomial blocks in our approach, the only learnable parameter is the single polynomial coefficient $\alpha_\ell^{(\tau)} = 1/\varsigma_\ell^{(\tau)}$, associated with each block. A polynomial of degree d only incurs d weights, and thus, it is essentially free, with respect to the weight count, to increase the degree. More specifically, let ν be the number of polynomial building blocks on one resolution level ℓ and let the size of the weight tensor be given by $s^2 \times c^2$, then the weight count of different types of residual blocks and polynomial settings is given as follows:

- ResNet18: $(s^2 \times c^2) \cdot 2 \cdot \nu$,
- MgNet$^{\mathrm{A}}$: $(s^2 \times c^2) \cdot (1 + \nu)$,
- MgNet$^{\mathrm{AB}}$: $(s^2 \times c^2) \cdot 2$,
- MgNet^{q_d}: $(s^2 \times c^2) + \nu$.

A regular ResNet18 block is built from two residual blocks on four resolution levels with $[64, 128, 256, 512]$ channels. The overall weight count of ResNet18 is 11.2 million (M). The corresponding MgNet$^{\mathrm{A,B}}$ has the same number of resolution levels, but according to [50], the number of channels on the fourth resolution layer is reduced to 256. This setting results in 2.7M weights overall. Poly-MgNet^{q_d} with the same number of resolution levels has 1.3M weights.

Dataset	polynomial	initialization	accuracy (%) ± std
CIFAR-10	q_2	$\mathcal{U}(\lambda_{\min}, \lambda_{\max})$	94.89 (0.16)
		Xavier	94.76 (0.16)
	g_4	$\mathcal{U}(\lambda_{\min}, \lambda_{\max})$	95.28 (0.19)
		Xavier	95.32 (0.13)

Table 4.9: Influence of coefficient initialization for the models Poly-MgNet^{q_2} and Poly-MgNet^{g_4}. The coefficients are drawn either uniformly from the interval spanned by the spectrum Λ, i.e. from $\mathcal{U}(\lambda_{\min}, \lambda_{\max})$, or from $\mathcal{U}(-t, t)$ with $t = \sqrt{\frac{6}{c_{\text{in}}+c_{\text{out}}}}$. The latter corresponds to Xavier uniform initialization, a standard approach for initializing convolutional weight tensors [38].

ReLU placement. We examine different placements of activation functions and batch normalization for the polynomials q_2 (Equation (4.10)), g_4, g_6 and g_8 (Equation (4.11)) on CIFAR-10 to determine suitable configurations. Table 4.8 summarizes the ReLU combinations we determined as the most relevant.

The impact of these combinations on accuracy of Poly-MgNet^{q_2} is shown in Table 4.10. Recall that q_2 corresponds to $\hat{q}_2$ in Equation (4.10), comprising two linear factors with real-valued roots. We observe that placing ReLU and bn at the same positions as in MgNet$^{\text{A,B}}$, where both A and B are followed by ReLU and bn, results in lower accuracy than MgNet$^{\text{A,B}}$. Furthermore, the naive approach with a single ReLU and no additional bn (Equation (4.13)), on one resolution level, is not robust (cf. Table 4.10). One of the three experimental runs fails to learn, resulting in a low accuracy with a high standard deviation. It is well known that polynomials suffer from instabilities with increasing degree. In our experiments, we observed that higher polynomial degrees (greater than eight) often led to instabilities. Polynomials of moderate degree mostly achieved superior accuracy. In contrast, the same ReLU placement paired with bn applied after the calculation of the residual achieves the highest accuracy (cf. Table 4.10). Furthermore, we study the ReLU placement in Poly-MgNet^{g_4}, which is based on polynomials that are quadratic in the real-valued term. Note that $g_4 = \hat{g}_4$ (Equation (4.11)). This emphasis on the real-valued part increases the accuracy by 0.4 pp.

Selected results for the ReLU placement in models with polynomials g_6 and g_8 are summarized in Table 4.11. Recall that polynomials q_d for $d > 2$ and g_d for $d > 4$ are composed of a real-valued part and an imaginary-valued part with respect to their roots (cf. Equation (4.10)). Specifically, the composition of g_6 is given by $g_6 = \hat{g}_1(A^2) \cdot \hat{g}_2(A^2) \cdot \tilde{q}_1(A)$. For the $\hat{g}$-terms, the placement of ReLU and bn follows the optimal configuration of g_4 shown in Table 4.10. We observe for both Poly-MgNet^{g_6} and Poly-MgNet^{g_8} that applying ReLU and bn after the residual update is essential for high accuracy.

Channel scaling. Having identified feasible placements for ReLU and bn, we investigate the influence of the capacity in terms of weights on accuracy of our Poly-MgNet models. To this end, we introduce a channel scaling parameter λ to scale the number

Model	bn	$\phi\, u_\ell^{(\nu)}$	bn	$\phi p_{d-1}(A)$	bn	ϕr	accuracy (%) test (±std)	train
MgNet$^{\mathrm{A,B}}$							95.94 (0.27)	97.60
Poly-MgNet^{q_2}	×	×					65.27 (0.40)	64.61
			×	×	×	×	93.04 (0.19)	94.00
	×	×			×	×	94.38 (0.35)	96.54
	×	×			×		**94.89** (0.16)	97.03
		×	×		×		94.71 (0.21)	94.71
Poly-MgNet^{g_4}	×	×					94.65 (0.33)	96.70
			×	×	×	×	**95.28** (0.19)	97.23
	×	×			×	×	95.02 (0.18)	97.13
		×	×		×	×	95.04 (0.09)	97.11
		×	×		×		95.00 (0.18)	97.11

Table 4.10: Influence of ReLU and bn placement (cf. Table 4.8) on accuracy for Poly-MgNet^{q_2} and Poly-MgNet^{g_4} trained on CIFAR-10. The best accuracy for each model is highlighted in bold. In the blocks corresponding to Equation (4.12) each ReLU is followed by bn (first row). For Poly-MgNet^{q_2}, we consider the naive placement of one ReLU before resolution coarsening, i.e. after the polynomial q_2 (cf. Equation (4.13)). We contrast this with an additional bn after ReLU with bn after the residual r (rows 2, 3). For Poly-MgNet^{g_4}, we limit the shown results to the configurations with the highest accuracy.

of weights in Poly-MgNet according to the weight count of ResNet and MgNet. Conversely, we scale the channels of ResNet and MgNet to meet the capacity of Poly-MgNet. Specifically, the initial number of channels[4] is multiplied by a scaling parameter $\lambda < 1$ to reduce the overall weight count to a target of 1.3M weights, corresponding to the average weight count of polynomial MgNet models. The results are shown in Figure 4.10. A regular ResNet18 has over 11M weights and achieves an accuracy of over 96%. Limiting the total weight count to 1.3M, representing a reduction by a factor of more than 8, results in an accuracy drop of about 2 pp. In contrast, the standard MgNet$^{\mathrm{A,B}}$ with 2.7M weights already achieves a reduction by a factor of 4 compared to ResNet18 while maintaining competitive accuracy. Despite the lower weight count, MgNet$^{\mathrm{A,B}}$ achieves an accuracy of 96%. Reducing the weight count of MgNet to 1.3M results in a loss in accuracy of 0.6 pp. These findings highlight that, compared to ResNet, the weight count can be drastically reduced without significantly affecting accuracy.

[4]The initial numbers of channels at the four resolution levels are $[64, 128, 256, 512]$ for ResNet18 and $[64, 128, 256, 256]$ for MgNet.

Model	bn	$\phi\, u_\ell^{(\nu)}$	bn	$\phi\, p_{d-1}(A)$	bn	ϕr	accuracy (%) test (± std)	train
Poly-MgNet^{g_6}			×	×	×	×	95.25 (0.41)	97.29
	×	×			×	×	**95.55** (0.07)	97.34
		×	×		×	×	95.23 (0.13)	97.17
Poly-MgNet^{g_8}		×	×		×	×	95.33 (0.33)	97.35
	×	×			×	×	**95.60** (0.34)	97.40

Table 4.11: ReLU and bn placement in Poly-MgNet^{g_6} and Poly-MgNet^{g_8}.

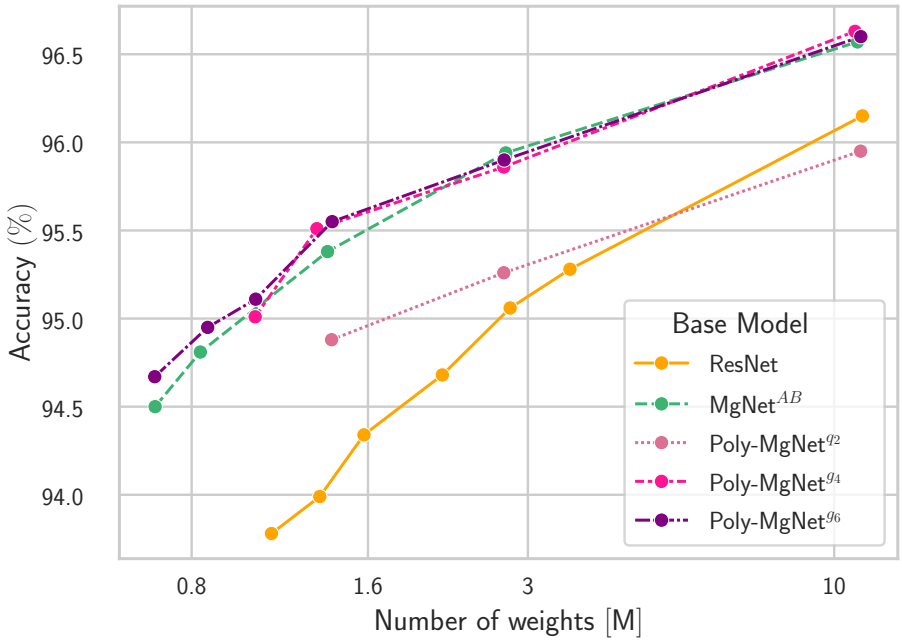


Figure 4.10: Accuracy-weight trade-off of ResNet and MgNet models on CIFAR-10: influence of the overall capacity of residual networks on classification accuracy. The number of channels in the residual blocks of ResNet and MgNet^{A,B} is scaled by factors of $1/\sqrt{2}$ and $1/\sqrt{8}$. In contrast, the number of channels of Poly-MgNet^{q_d} are scaled by multiplying by $\sqrt{2}$ and $\sqrt{8}$.

Our initial Poly-MgNet uses around 1.3M weights. With this capacity, Poly-MgNet^{q_2} achieves an accuracy of 94.89%. Compared to ResNet18, with over 11M weights, this represents a reduction in weight count by a factor of over 8.5 while sacrificing less than 1.5 pp of accuracy. Poly-MgNet^{g_4} and Poly-MgNet^{g_6} achieve accuracies of over 95.5% with an initial capacity of approximately 1.4M weights, slightly improving on MgNet^{A,B} by 0.13 pp at the same capacity. By increasing the number of channels for MgNet^{A,B} and Poly-MgNet with building blocks g_4 and g_6, all models achieve approximately 96.60%, thereby drastically improving the accuracy-weight trade-off compared to ResNet18.

4.5.3 Conclusion

In this work, we introduced Poly-MgNet, which complements MgNet with a new neural network building block inspired by polynomial multigrid smoothers. We conducted comprehensive numerical studies, including baseline comparisons and several ablations. Compared to MgNet, we found more favorable trade-offs between the number of weights in Poly-MgNet and the achieved accuracy. Our ablation studies show that the incorporation of multiple batch normalizations and ReLU activation functions within a single building block is crucial for achieving high accuracy, despite the counterintuitive use of non-linearities in polynomials. Additionally, we observed that quadratic polynomials provide notable benefits in terms of accuracy. We make our code for reproducing all experiments publicly available under `https://github.com/vanbetteray/Poly-MgNet`.

5 Efficient SVD by Local Fourier Analysis

This chapter is mostly taken word by word from our publication [110].

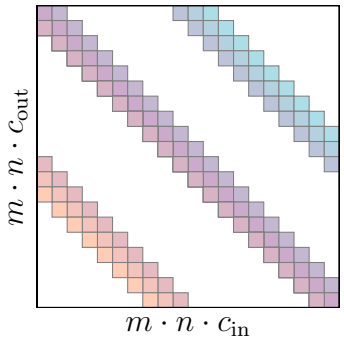
5.1 Introduction

Deep convolutional neural networks are powerful methods for image recognition tasks [53, 71, 73, 94]. Many recent deep learning architectures continue to employ convolutional mappings as fundamental components [28, 85, 114, 119, 120], which perform linear transformations of the input to extract relevant features [73]. The spectral properties of convolutional mappings have diverse applications. For instance, the spectral norm is used for regularization to improve generalizability [42, 99, 124] or robustness to adversarial attacks [19, 24]. Furthermore, the singular value decomposition can be used for low-rank approximation to enable model compression [1, 4, 18, 60, 63, 127] or to study interpretability in CNNs [89]. Pseudoinvertible neural networks are designed to learn both a task and its inverse [60]. Notably, the pseudoinverse can be computed directly via singular value decomposition.

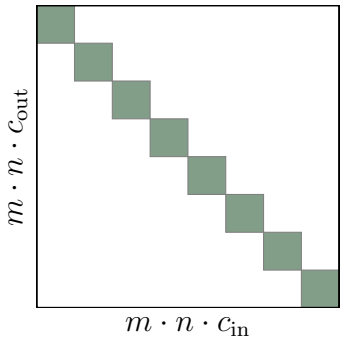
As seen in Section 3.2.4, the weight tensor corresponding to a convolutional mapping is a four-dimensional tensor. Its dimensions correspond to the numbers of input and output channels and the filter height and width. During the convolution, each two-dimensional filter slides across the input feature map. At each position, the operation computes the sum of element-wise multiplications between the filter and the corresponding overlapping region of the input, producing the output feature map.

Considering an input with spatial dimensions $m \times n$ and c_{in}, c_{out} denoting the number of input and output channels, respectively, the convolution mapping is given by

$$A : \mathbb{R}^{m \times n \times c_{\text{in}}} \to \mathbb{R}^{m \times n \times c_{\text{out}}} . \tag{5.1}$$



(a) Matrix form of an $m \times n$ input convolved with a 3×3 kernel



(b) Matrix composed of $m \cdot n$ blocks of size $c_{\text{out}} \times c_{\text{in}}$

Figure 5.1: Sparsity patterns of matrices corresponding to a convolutional mapping (a) in matrix-vector multiplication form; and (b) after the Fourier transform.

The corresponding two-dimensional matrix is sparse with dimension $(m \cdot n \cdot c_{\text{in}}) \times (m \cdot n \cdot c_{\text{out}})$ with sparsity pattern according to Figure 5.1a. It is clear that the size of this matrix grows rapidly as the number of channels and/or the input size increases, whereas applying the singular value decomposition to large matrices is time and memory consuming. Assuming a square input size and equal number of channels, i.e., $c := c_{\text{in}} = c_{\text{out}}$ and $n = m$, the computational complexity for the brute-force approach is $\mathcal{O}(n^6 c^3)$ [99]. The focus of this work is on an efficient approach to reduce the computational complexity of computing the singular value decomposition of large matrices corresponding to convolutional weight tensors. The view of a convolution as a doubly circulant matrix allows for the use of the (discrete) Fourier transform. By applying the 2D Fast Fourier Transform (FFT), studied in Section 2.3.1, the computational complexity can be reduced to $\mathcal{O}(n^2 c^2 (c + \log n))$ [99]. In contrast, our approach takes into account that convolutional mappings are translation invariant. This allows us to apply local Fourier analysis (LFA) [66]. LFA, see Section 2.3.2, is primarily used in the study of iterative methods for solving PDEs, especially multigrid methods. Structural analogies between CNNs and multigrid have been analyzed by [32, 50, 53, 109, 111]. We utilize the convolution theorem, which allows convolutional operators to be block-diagonalized by the Fourier transform, see Section 2.3.2. In our approach, we employ complex Fourier exponentials as ansatz functions for the basis change in the frequency domain. This allows us to carry out the analysis in the frequency domain, where the action of convolutional operators on different frequency components, i.e., Fourier modes, can be analyzed independently. In particular, the orthogonality of the Fourier basis functions ensures that a convolution in the spatial domain corresponds to a pointwise multiplication in the frequency domain. Consequently, the convolutional mapping based on a four-dimensional weight tensor is transformed into a collection of matrix-vector multiplications, one for each frequency, as discussed in Figure 5.1b. Each block corresponds to a specific frequency component and has dimensions $c_{\text{out}} \times c_{\text{in}}$, coupling only across channels. The operator remains block diagonal along the spatial dimensions n and m. This block structure allows for indepen-

dent computation of the singular value decomposition of smaller matrices, which reduces the computational complexity to $\mathcal{O}(n^2c^3)$, compared to the FFT-based approach which requires $\mathcal{O}(n^2c^2(c + \log n))$.

To preserve the spatial dimension of the input after the application of the convolution and effectively process the edge pixels of the input image, the input is typically padded with zeros around the border. In the context of PDEs, zero padding corresponds to Dirichlet boundary conditions (cf. Section 2.4). While zero padding is the standard for image recognition tasks, the application of LFA as well as the FFT assumes periodic boundary conditions for the convolution. This raises the question of how boundary conditions influence the accuracy of the computed spectrum of the convolutional mapping.

The contribution of this chapter can be summarized as follows.

- We propose an algorithm for efficient computation of singular values of convolutional mappings by exploiting the translation invariance of convolutional operators, which allows for the application of LFA.
- We provide a theoretical analysis of our algorithm's runtime and validate its efficiency through numerical experiments.
- We study the influence of Dirichlet and periodic boundary conditions and analyze and compare the resulting spectra.

Our method can improve the efficiency and scalability whenever a given task requires the computation of a singular value decomposition (SVD) of convolutional mappings.

The remainder of this chapter is structured as follows. Section 5.2 reviews related work, including methods for determining spectra, approximating the largest singular values and selected applications. Building on foundations introduced in Section 2.3 and Section 3.2.4, in Section 5.3 we briefly review the convolution theorem and the singular value decomposition. Subsequently, we present our algorithm and analyze its computational complexity. In Section 5.4, we validate and study our proposed method through numerical experiments and discuss the results. Finally, we conclude this chapter by summarizing our contributions and findings in Section 5.5.

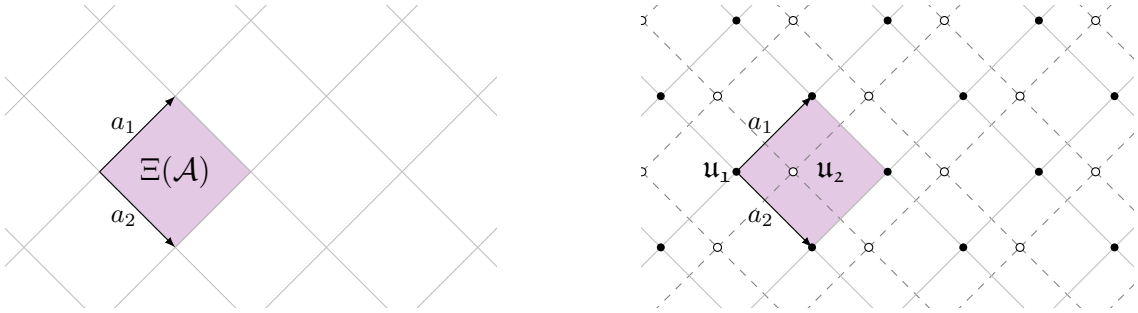
5.2 Related Work

In this section, we review related work that analyze the spectral properties of convolutions. These works can be broadly categorized into 1) exact computation of the full spectrum, 2) approximation of the (largest) singular values, and 3) applications. For the sake of readability, let $m = n$ and $c := c_{\text{in}} = c_{\text{out}}$ throughout the section. Furthermore, we assume square kernels of extent $s \times s$.

Exact computation of the full SVD. An approach to calculate the complete spectrum is introduced in [99]. They characterize the singular values of a convolutional layer using c^2 FFTs followed by n^2 singular value decompositions. The FFTs require $\mathcal{O}(n^2 \log n)$ FLOPs, while the singular value decomposition requires $\mathcal{O}(c^3)$, resulting in an overall computational complexity of $\mathcal{O}(n^2c^2(c+\log n))$. This approach is closely related to ours, which is based on the calculation of the so-called symbol, requiring $n^2 \cdot \mathcal{O}(1)$. Compared to [99], our approach reduces the computational complexity of calculating the singular values by a factor $\mathcal{O}(\log n)$ to $\mathcal{O}(n^2c^3)$.

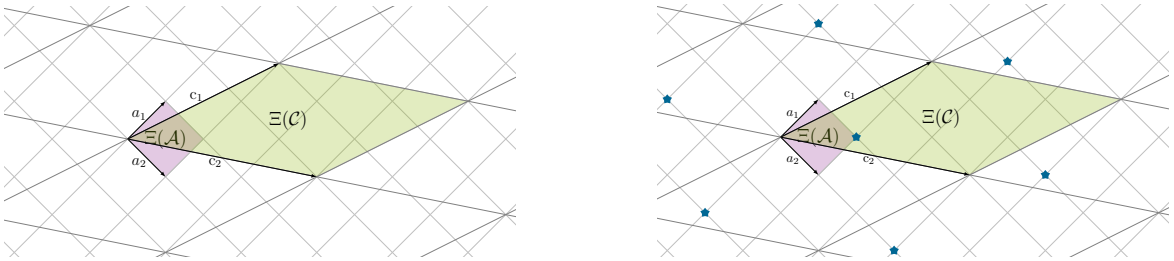
Approximation of the largest singular value. Yoshida and Miyato [124] reshape the weight tensor into a dense $c \times c \cdot s^2$ matrix and use a power iteration method to approximate the spectral norm, i.e., the largest singular value. However, the reshaped matrix yields only $c \cdot s^2$ singular values, whereas the corresponding linear transformation of the convolution has $c \cdot n^2$ singular values. Nevertheless, they utilize the operator norm as a regularizer, demonstrating that the approximation obtained by their representation is sufficient for the given purpose, despite being a loose upper bound, see also [24,108]. An efficient method to compute the L^1 and L^∞ norms of convolutional layers was proposed in [42] to approximate their spectral norm. In case of convergence, their method computes the exact spectral norm of a convolutional layer. However, this does not provide insight into the full spectrum of the layer.

Application domains. The ability to compute the singular value decomposition of convolutions paves the way for multiple applications. For instance, singular values can be utilized for the regularization of CNNs to improve generalizability. Regularizers based on the spectral norm, which equals the largest singular value, are studied in [42, 84,99,124] and more recently in [95,102]. The spectral norm was also utilized in [19,24] to improve robustness to adversarial attacks. Instead of focusing only on the largest singular value, other domains require access to a broader portion of the spectrum. The singular value decomposition can be used for low-rank approximations to enable model compression. Extensive studies on low-rank approximation in neural networks have been conducted by [4, 27, 63, 127]. These methods have been further developed to compress word embedding layers [1, 18, 60]. The application of the singular value decomposition in CNN interpretability is studied in [89]. Furthermore, in [111], the minimal and maximal singular values are utilized for the initialization of polynomial CNN blocks. Recently, a class of pseudoinvertible neural networks designed to learn both a task and its inverse was introduced in [8]. They leverage the inverse operation as a method for image generation, formulating the pseudoinverse $B = A^{\mathrm{T}}(AA^T)^{-1}$ of a convolution A as a transposed convolution. Rather than computing the exact inverse, they restructure CNN layers in order to approximate it. Alternatively, the exact pseudoinverse can be computed using an efficient singular value decomposition.



(a) Unit cell $\Xi(\mathcal{A})$ corresponding to $\mathbb{L}(\mathcal{A})$ (b) Crystal $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$

Figure 5.2: (a) Lattice $\mathbb{L}(\mathcal{A})$ with basis $\mathcal{A} = [\mathfrak{a}_1\ \mathfrak{a}_2]$ and (b) a crystal $\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$ consisting of copies of the lattice $\mathbb{L}(\mathcal{A})$, shifted by $\mathfrak{u}_1, \ldots, \mathfrak{u}_\nu$.



(a) Sublattice with primitive vectors c_1 and c_2 (b) Crystal torus

Figure 5.3: (a) Sublattice $\mathbb{L}(\mathcal{C}) \subseteq \mathbb{L}(\mathcal{A})$ with $\mathcal{C} = \mathcal{A} \cdot \begin{bmatrix} 3 & 2 \\ 1 & 3 \end{bmatrix}$ and (b) periodicity of the corresponding crystal torus $\mathbb{T}^{\mathfrak{u}}_{\mathcal{A},\mathcal{C}}$.

5.3 SVD of Convolutional Mappings by LFA

In this section, we present our approach to efficiently compute the singular value decomposition using LFA. Our method is based on crystalline structures introduced in Section 2.2. An (ideal) crystal is characterized by the infinite repetition of a basic structural unit, the unit cell, arranged in a regular pattern defined by a lattice (cf. Figure 5.2). In the presence of a structure element $\mathfrak{u} \in \Xi(\mathcal{A})^{|\mathfrak{u}|}$ we consider a crystal. The concepts of the sublattice and the crystal torus are depicted in Figure 5.3.

Although our experiments in this work focus on rectangular grids defined by images, crystalline structures can come in other shapes, e.g., octagonal, which allows for a generalization of our method.

In particular, in Section 2.2 we reviewed the concepts of lattice and crystal structures and introduced the LFA framework in Section 2.3. In Section 3.2.4 we explored convolutional mappings in the context of crystalline structures. Based on these foundations, we present our algorithm and establish its asymptotic computational complexity, showing that it is optimal with respect to the input dimension. To provide some context, we begin

with a short review of the convolution theorem and the singular value decomposition of multiplication operators defined on crystal tori.

Spectral decomposition of multiplication operators. Recall from Section 2.3.2 that crystal structures allow us to define wave functions $\varphi_k = \mathrm{e}^{2\pi\mathrm{i}\langle k,x\rangle}$ with $x \in \mathbb{T}_{n,m}$ and frequencies $k \in \mathbb{T}^*_{n,m}$ as elements of the dual lattice. The extension to the crystal case is straightforward and a canonical basis of wave functions can be chosen. Applying a convolution A to these wave functions immediately yields

$$(A * \varphi_k)(x) = \Big(\sum_{y \in \mathcal{N}} M_y \cdot \mathrm{e}^{2\pi\mathrm{i}\langle k,y\rangle} \Big)\, \mathrm{e}^{2\pi\mathrm{i}\langle k,x\rangle}\,.$$

In particular, we find a set of c_{in} or c_{out} Fourier basis functions for each frequency $k \in \mathbb{T}^\star_{n,m}$, collected as columns of matrices $\mathbf{F}_k^{\mathfrak{c}_{\mathrm{in}}}$ and $\mathbf{F}_k^{\mathfrak{c}_{\mathrm{out}}}$, respectively (cf. Section 2.3.2). Equivalently, the above statement can be written as

$$A\mathbf{F}_k^{\mathfrak{c}_{\mathrm{in}}} = \mathbf{F}_k^{\mathfrak{c}_{\mathrm{out}}} A_k.$$

Since $\mathbf{F}_k^{\mathfrak{c}_{\mathrm{in}}}$ and $\mathbf{F}_k^{\mathfrak{c}_{\mathrm{out}}}$ have orthonormal columns, we can calculate the singular value decomposition of A by first computing the decomposition

$$A_k = U_k \Sigma_k V_k^\star$$

for each $k \in \mathbb{T}^\star_{n,m}$. Then,

$$\widehat{U}_k = \mathbf{F}_k^{\mathfrak{c}_{\mathrm{out}}} U_k \quad \text{and} \quad \widehat{V}_k = \mathbf{F}_k^{\mathfrak{c}_{\mathrm{in}}} V_k$$

have orthonormal columns and serve as the left and right singular vectors of A, with corresponding singular values contained in Σ_k. Collecting the contributions from all frequencies $k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}}$, we find

$$U = \left(\bigoplus_{k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}}} \widehat{U}_k \right), \quad \Sigma = \left(\bigoplus_{k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}}} \Sigma_k \right) \quad \text{and} \quad V^* = \left(\bigoplus_{k \in \mathbb{T}^*_{\mathcal{A},\mathcal{Z}}} \widehat{V}_k^* \right),$$

yielding the full singular value decomposition of the crystal operator $A = U\Sigma V^*$. Most importantly, the set of all singular values of the convolution can be computed without forming the global singular vectors $\widehat{U}_k$ and $\widehat{V}_k$.

The singular value decomposition of multiplication operators can be interpreted as a two-step diagonalization process:

1. *Block-diagonalization by LFA*
2. *Diagonalization by SVD*

Computing the singular values in this fashion alleviates the dependence of this computation on the spatial dimensions n and m, thus drastically reducing the computational complexity. This process is summarized in Algorithm 10.

Algorithm 10 SVD with LFA(A, m, n)

1: **Init** $X = \{0, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}\}$, $Y = \{0, \frac{1}{m}, \frac{2}{m}, \dots, \frac{m-1}{m}\}$, $K = X \times Y$
2: **for** $i = 1, \dots, n$ **do**
3: **for** $j = 1, \dots, m$ **do**
4: $k = K_{i,j}$
5: $B_{i,j} = \sum_{y \in \mathcal{N}} M_y \cdot e^{2\pi i \langle k, y \rangle}$
6: $U_{i,j}, \Sigma_{i,j}, V^*_{i,j} = \mathrm{SVD}(B_{i,j})$
7: **end for**
8: **end for**

Method	spatial input dimension	channel dimension	time complexity
explicit	$n = m$	$c = c_{\text{in}} = c_{\text{out}}$	$\mathcal{O}(n^6 c^3)$
FFT	$n = m$	$c = c_{\text{in}} = c_{\text{out}}$	$\mathcal{O}(n^2 c^2 (c + \log n))$
LFA (ours)	$n = m$	$c = c_{\text{in}} = c_{\text{out}}$	$\mathcal{O}(n^2 c^3)$
	$n \neq m$	$c = c_{\text{in}} = c_{\text{out}}$	$\mathcal{O}(nmc^3)$
		$c_{\text{in}} \geq c_{\text{out}}$	$\mathcal{O}(nmc_{\text{in}}^2 c_{\text{out}})$
		$c_{\text{in}} \leq c_{\text{out}}$	$\mathcal{O}(nmc_{\text{in}} c_{\text{out}}^2)$

Table 5.1: Time complexity of Algorithm 10 for computing the SVD using our proposed LFA-based approach, compared with the FFT-based approach [99] and the explicit matrix representation of the convolution mapping.

Computational complexity. The time complexity of our singular value decomposition with LFA, as described in Algorithm 10, will be analyzed in the following. Algorithm 10 comprises two nested **for**-loops with $\mathcal{O}(n)$ and $\mathcal{O}(m)$ iterations, respectively, see lines 2 and 3, followed by the computation of the Fourier symbol in line 5 with time complexity $\mathcal{O}(1)$. The time complexity of the SVD in line 6 depends on the channel dimensions, since $B_{i,j} \in \mathbb{C}^{c_{\text{in}} \times c_{\text{out}}}$. For an equal number of input and output channels c, the time complexity of the singular value decomposition for dense matrices is $\mathcal{O}(c^3)$ [105]. In summary, the overall computational complexity is $\mathcal{O}(n^2 c^3)$ if $n = m$. The time complexities for unequal numbers of input and output channels are specified in Table 5.1.

Boundary conditions. Recall from Section 2.3 that LFA used to compute the singular values of the convolution operators implicitly assumes periodic boundary conditions (cf. Figure 5.4). In many applications, however, these convolutions are formulated with Dirichlet boundary conditions, corresponding to zero padding. In Section 5.4, we take a closer look at the distribution of singular values for both types of boundary conditions to gauge their impact.

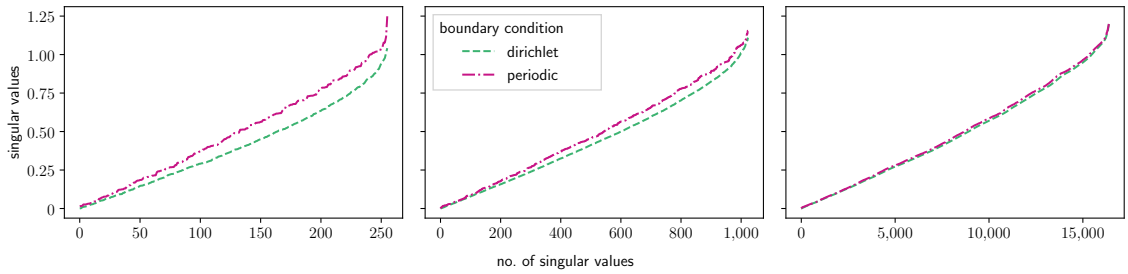


Figure 5.4: Effect of boundary conditions for increasing input sizes ($n = 4, 8, 32$; left to right). The number of input and output channels is fixed to 16.

Strided and dilated convolutions. Thus far, we have assumed dense convolutional kernels with a stride of one. However, CNNs also employ dilated or strided convolutions, both of which can be dealt within our framework. As introduced in Section 3.2.4, in dilated convolutions, the kernel is expanded by inserting many zeros between elements. Since the stride remains one, the underlying shift-invariance is preserved and the derivation remains unchanged. Strided convolutions, on the other hand, keep the kernels unchanged but increase the stride, thereby modifying the shift-invariance of the convolution operator. This difficulty can be resolved by rewriting the kernel matrices with respect to this shift-invariance, which implicitly increases the number of channels, along the lines of [66]. In this way, our analysis extends naturally to both dilated and strided convolutions.

5.4 Numerical Results

In this section, we present and discuss numerical results. We start by analyzing the dependence of our approach on the given boundary conditions in order to study the applicability of our approach to CNNs, which are typically implemented with zero padding, i.e., Dirichlet boundary conditions. The computational complexity of our algorithm was already studied in Section 5.3. In this section, we complement our theoretical analysis with experiments evaluating the computational efficiency of our method in terms of wall-clock runtime for computing the singular value decomposition. Unless stated otherwise, we focus on the case of square matrices. Note that the corresponding (unrolled) matrices have full rank, so that the number of non-zero singular values equals the number of rows and columns. Finally, we discuss how the memory layout affects computational runtime.

Boundary conditions. LFA requires periodic boundary conditions. However, in practice convolutions are typically padded with zeros, which, from a PDE perspective, corresponds to Dirichlet boundary conditions. In the following, we relax the assumption of

n	4	8	16	32
W	0.0874	0.0440	0.0238	0.0117

Table 5.2: Wasserstein distances (W) between spectra computed with Dirichlet and periodic boundary conditions.

periodic boundary conditions and study how boundary conditions affect the spectrum. We compare the singular values computed by our LFA-based method, which assumes periodic boundary conditions, with those obtained from a baseline approach that explicitly constructs the sparse matrix corresponding to the convolution with zero padding and computes its singular values using NumPy [48].

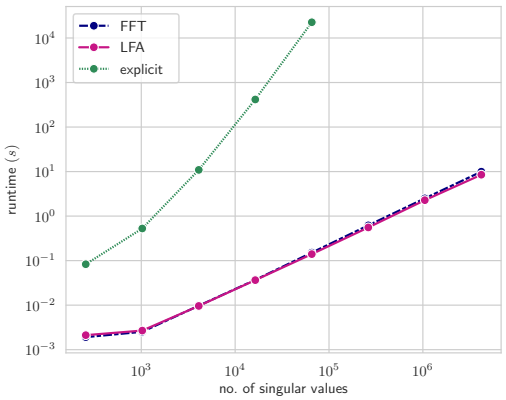
Figure 5.4 depicts the singular values for three weight tensors, each with 16 input and output channels, over a range of input sizes n. As expected, the number of singular values increases quadratically with the input size. We observe that for small input sizes n, the choice of boundary conditions noticeably affects the approximation quality of our method. However, this effect diminishes as the number of singular values increases, which is expected since the boundary has less influence in larger lattices. Consequently, for larger lattice sizes, our method provides accurate approximations of the singular values for convolutions with zero padding. This observation is supported by computing the Wasserstein distance between singular values obtained with zero padding ($\sigma^{\text{Dirichlet}}$) and periodic padding ($\sigma^{\text{periodic}}$).

The Wasserstein distance provides a measure of dissimilarity between two probability distributions (or finite sets of values). Intuitively, it quantifies the minimal work required to transform one set into the other. For sorted sets, the Wasserstein distance is

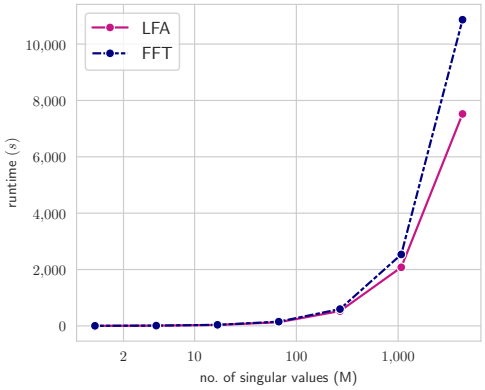
$$W = \frac{1}{n^2 \cdot c} \sum_{i=1}^{n^2 \cdot c} |\sigma_i^{\text{periodic}} - \sigma_i^{\text{Dirichlet}}|.$$

As shown in Table 5.2, the influence of boundary conditions on the spectrum decreases in terms of the Wasserstein distance as the dimension of the problem grows. Thus, for higher dimensions, the contribution of boundary effects becomes negligible relative to the bulk of the system, and Fourier-based approaches for computing the singular value decomposition become increasingly accurate.

Runtime analysis. To demonstrate the computational efficiency of our algorithm, we compare its execution runtime, in seconds (s), for computing singular values using our algorithm with both an explicit approach and the FFT-based method proposed in [99]. We use an Intel(R) Xeon(R) Gold 6242 CPU @2.80GHz, with all 16 cores available during the calculation. We start the timing before the transformation of the weight tensor with the FFT and LFA, respectively, and stop after the calculation of the singular values. Our implementation operates on PyTorch convolutional weight tensors, which are



(a) Comparison of runtimes for computing singular values using a naive explicit matrix approach and Fourier-based methods (FFT and LFA). The input feature map size varies as $n \in \{4, 8, 16, 32, 64, 256, 512\}$. Both axes are shown on a logarithmic scale.



(b) Runtime for computing the SVD with our LFA-based approach compared to the FFT-based method. The input feature map size varies as $n \in \{2^i : i = 8, \ldots, 14\}$. The x-axis shows the total number of singular values in millions (M) on a logarithmic scale.

Figure 5.5: Runtimes in seconds (s) for computing singular values with varying input sizes. The evaluated matrices are square, with both input and output channel dimensions fixed at 16. (a) includes explicit SVD computation; (b) shows Fourier-based approaches for large inputs.

stored in a channel-first format. To ensure correct alignment of the spatial dimensions, the weight tensors must therefore be transposed prior to applying the FFT. The computational cost of this transposition, in terms of runtime, is independent of the input dimension and is negligible at less than 8×10^{-6} seconds. For both approaches, we utilize the svd function from NumPy's linalg module with the option compute_uv = False.

In Figure 5.5a, we first analyze the execution time required for calculating all singular values using the Fourier-based approaches, and compare these results with the naive explicit method, which constructs the convolutional operator as a sparse matrix and computes its singular values directly. The largest matrix we decomposed by explicitly unfolding the convolutional mapping was 65,536 × 65,536, corresponding to a convolutional operator with 16 input and output channels applied to an input of dimension $n = 64$. Beyond that, memory capacity quickly becomes a limiting factor. Figure 5.5a shows that the runtime for the explicit approach grows rapidly with increasing n. For example, the computation of the singular values of a 1,024 × 1,024 matrix takes 0.30 seconds, while it takes over 400 seconds to compute the singular values for a matrix of size 16,384 × 16,384. For very small values of n, i.e., $n = 4$ and $n = 8$, the FFT-based approach is the fastest. However, as n increases, its runtime grows more quickly than that of our LFA-based approach. Consequently, for $n \geq 16$, the computation of the singular values is faster with our LFA-based approach than with the FFT. With $n = 256$, we observe that the FFT-based approach requires 2.51 s, while our LFA-based approach requires 2.30 s, resulting in a speed-up factor of 1.09 (cf. Table 5.3).

n	no. of singular values	method	runtime (s)	$s_{\text{FFT}}/s_{\text{LFA}}$
256	1,048,576	FFT	2.51	
		LFA	2.30	1.09
512	4,194,304	FFT	9.96	
		LFA	8.49	1.17
1,024	16,777,216	FFT	38.00	
		LFA	33.01	1.15
2,048	67,108,864	FFT	149.96	
		LFA	130.20	1.15
4,096	268,435,456	FFT	596.44	
		LFA	520.21	1.15
8,192	1,073,741,824	FFT	2535.28	
		LFA	2077.44	1.22
16,384	4,294,967,296	FFT	10,864.97	
		LFA	7,521.93	1.44

Table 5.3: Ratio $s_{\text{FFT}}/s_{\text{LFA}}$ of runtimes for computing the singular value decomposition using FFT- and LFA-based approaches. s_{FFT} denotes the overall runtime for the FFT-based approach and s_{LFA} that of our LFA-based approach.

This trend continues as the input size n increases further, which is also shown in Figure 5.5b. Here we compare the runtime of our LFA-based approach to the runtime of the FFT-based approach for $n \geq 256$. Especially for very large n, the gap in runtime between our LFA-based approach and the FFT-based approach becomes more pronounced. In the case of $n = 16{,}384$, the computation of the singular value decomposition involves over 4 billion singular values. Their computation requires approximately 181 minutes with the FFT-based approach, whereas our LFA-based method completes in about 125 minutes, yielding a speed-up factor of 1.44 (cf. Table 5.3). This corresponds to a 31% reduction in runtime compared to the FFT-based approach. This improvement is visualized in Figure 5.5b. Overall, as expected from our theoretical runtime analysis, the speed-up factor increases with n.

Memory layout effects. Our proposed method to efficiently compute the singular values of a weight tensor convolved with some input feature map consists of two steps:

1. Transformation by LFA (or FFT)
2. Computation of the singular values

Both Fourier-based approaches, the FFT and LFA, produce n^2 block matrices of dimension $c \times c$, each treated separately by the singular value decomposition routine. Given the identical dimensions and data types of the transformed tensors, one would intuitively expect the singular value decomposition to require roughly the same overall computational effort for both LFA and FFT. However, in practice, we observed that

n	no. of singular values	method ($\mathcal{F}$)	$s_{\mathcal{F}}$	s_{SVD}	s_{total}
16,384	4,294,967,296	FFT	1,966.89	8,898.08	10,864.97
		LFA	1,595.02	5,926.91	7,521.93
8,192	1,073,741,824	FFT	317.95	2,217.33	2,535.28
		LFA	82.48	1,994.97	2,077.45
4,096	268,435,456	FFT	69.11	527.33	596.44
		LFA	20.57	499.64	520.21
2,048	67,108,864	FFT	18.19	131.77	149.96
		LFA	4.97	125.23	130.20
1,024	16,777,216	FFT	4.86	33.14	38.00
		LFA	1.28	31.74	33.02

Table 5.4: Runtime for computing the singular values for different values of n. The total runtime (s_{total}) consists of the transformation time ($s_{\mathcal{F}}$) and the time required to compute the singular value decomposition (s_{SVD}).

the computation time required for the singular value decomposition differs between the two methods, as shown in Table 5.4. We found that the memory layouts of the tensors obtained after transformation with FFT and LFA differ. In general, Python stores data in row-major format.[1] In earlier experiments, we found that ensuring a row-major memory layout prior to the FFT yields a faster total runtime for $n < 8{,}192$, while doing so prior to LFA yields a faster total runtime for $n \leq 8{,}192$. This memory layout conversion is independent of the input dimension and is negligible for the total runtime.[2] Additionally, the memory layout is maintained during the transformation part of the LFA in our implementation. However, this does not hold for NumPy's FFT routine, which does not return a row-major memory layout, even if the input is ensured to be row-major. Table 5.4 reports an overview of the runtimes for the two steps involved in efficiently computing the singular value decomposition: the transformation time ($s_{\mathcal{F}}$) and the computation time of the singular value decomposition (s_{SVD}), which together sum up to the total computation time (s_{total}).

As already pointed out, the runtimes for computing the singular value decomposition differ for the same input dimensions for both approaches. Furthermore, Table 5.5 shows that the row-major memory layout in our LFA-based approach not only improves the runtime for the transformation by LFA but also reduces the computation time of the subsequent singular value decomposition. To ensure that the computation of the singular value decomposition is equally fast for both approaches, we converted the tensors obtained after the FFT into row-major memory layout. For comparison, we include experiments for our LFA-based approach where the data is only converted to row-major

[1] Other orders, such as column-major format, are also possible but are not relevant in this work.

[2] The results reported in this work correspond to the fastest runtimes achieved for both methods (LFA and FFT), regardless of the memory layout.

n	$\mathcal{F}_{\text{method}}$	row-major	$s_{\mathcal{F}}$	s_{copy}	s_{SVD}	s_{total}
8,192	FFT	✓	317.95	–	2,217.33	2,535.28
	FFT	✓	333.18	558.88	1,986.25	2,878.31
	LFA	✓	82.48	–	1,994.97	2,077.44
	LFA	×	87.37	1,021.24	1,987.23	3,095.83
4,096	FFT	✓	68.96	–	527.62	596.58
	FFT	✓	67.44	63.48	497.35	628.28
	LFA	✓	20.57	–	499.64	520.21
	LFA	×	22.59	148.56	497.01	668.16
2,048	FFT	✓	17.88	–	131.73	149.61
	FFT	✓	18.16	13.81	125.61	157.58
	LFA	✓	4.97	–	125.23	130.20
	LFA	×	5.73	30.89	125.47	162.10
1,024	FFT	✓	4.87	–	33.07	37.94
	FFT	✓	4.87	3.07	31.89	39.84
	LFA	✓	1.28	–	31.74	33.01
	LFA	×	1.43	7.00	31.76	40.19

Table 5.5: Effect of row-major layout on the runtimes $s_{\mathcal{F}}$, s_{SVD} and s_{total}. Inputs to the transformation $\mathcal{F}$ stored in row-major layout are marked with ✓, and those stored in other layouts with ×. Note that the FFT does not preserve row-major layout. The time required to convert to row-major layout is reported under s_{copy}. A dash (–) indicates that no conversion was performed.

after the LFA, rather than beforehand. The results are presented in Table 5.5. Consistent with the observations made in Table 5.4, a row-major memory layout enables more efficient computation of the singular value decomposition. According to Table 5.5, for $n \leq 8{,}192$, it is possible to convert the memory layout of the transformed tensors such that after the FFT and the LFA, the respective subsequent singular value decompositions require approximately the same runtime. Although computing the singular value decomposition in row-major memory layout is faster than in the layout produced by the FFT, the overhead of converting to row-major layout outweighs the benefit, making it more efficient to compute the singular value decomposition directly on the layout obtained by the FFT. This effect is amplified for large n. Consequently, our approach not only enables faster transformations by LFA but also preserves a memory layout that facilitates efficient computation of the singular value decomposition.

5.5 Conclusion

In this work, we presented an efficient algorithm for computing the singular values of convolutional operators corresponding to large sparse matrices. Unlike previous approaches, our LFA-based method exploits the translation invariance of convolutional operators to achieve asymptotically optimal scaling of the computational effort with

the spatial resolution n. We provided a theoretical analysis supporting this result. Compared to a previous FFT-based approach, our method improves the computational complexity by a factor of $\log(n)$, which we corroborate with runtime studies. We verified that computation time for the singular value decomposition increases sublinearly as the input size n grows. We also demonstrated that our LFA-based implementation yields a memory layout that ia advantageous for the subsequent singular value decomposition, further reducing runtime. We make our code publicly available at `https://github.com/vanbetteray/conv_svd_by_lfa`. It is worth noting that, unlike the FFT, LFA is embarrassingly parallel.

6 Discussion & Outlook

In this thesis, we investigated how multigrid methods can inform the design and analysis of deep CNNs for image classification. By transferring concepts originally developed for numerical solvers into the setting of residual architectures, we demonstrated how such perspectives can improve efficiency, yield structural insights, and provide a foundation for the systematic development of CNN components. In the following, we summarize our contributions presented in Chapter 4 and Chapter 5 and outline directions for future research and potential applications, complementing the chapter-specific conclusions.

Contributions and conclusion. In this thesis, we have seen that discretizing PDEs yields operators that act as convolution-like multiplication operators, the same structures that form the core building blocks in CNNs. Solving the resulting system of (non-) linear equations typically involves iterative schemes, where the iteration matrix depends on the fixed system matrix. The corresponding error propagator is structurally equivalent to a ResNet-block and repeated iterations naturally parallel the stacking of multiple ResNet-blocks. This analogy has motivated multigrid-inspired CNN designs, for example by exploiting the fixed system matrix to justify weight sharing across blocks or by transferring multigrid structures into the channel domain. Although such approaches already achieve linear scaling with respect to the channel dimension, the full multigrid perspective has not yet been exploited.

To address this gap, we introduced a hierarchical building block that mirrors a multigrid V-cycle across channels. To mitigate the quadratic growth of parameters with respect to the number of channels, we successively reduce the channel dimension and apply ResNet-like blocks with shared weights and grouped convolutions on each in-channel level. On the coarsest in-channel level, fully coupled convolutions act as direct solvers, after which the learned information is prolongated back to the original channel dimension. At each level, the same blocks serve as pre-smoothers in the restriction phase and as

post-smoothers in the prolongation phase, yielding a weight-efficient multigrid-inspired mechanism for structured learning across channels (Section 4.4).

The analogy between residual blocks and iterative schemes also allows for more refined interpretations of ResNet-blocks. In particular, a ResNet block can be viewed as a step of a Richardson iteration, which reveals that only a single convolutional operator, corresponding to the system matrix, is required. From this perspective, the second convolution does not contribute additional expressive power. By replacing it by a scalar parameter, the computational cost is significantly reduced while preserving the iterative structure. Moreover, the interpretation of residual blocks as residual propagator polynomials provide new insights into initialization strategies. In iterative methods, the convergence behavior is determined by the spectral properties of the iteration matrix, with singular values near the spectral boundary playing a decisive role. Translating this principle to CNNs, where initialization is key for convergence during training, we proposed initializing the polynomial coefficients in accordance with singular values close to the boundary. This ensures that the residual propagation dynamics reflect favorable convergence properties from the beginning, leading to more stable training compared with standard heuristic initialization (Section 4.5).

However, the explicit calculation of the spectrum of convolutional mappings is typically unfeasible. To address this issue, we developed an efficient algorithm for computing the singular value decomposition of convolutional mappings. We employed LFA, a framework originally developed for multigrid analysis. Under periodic boundary conditions, image data can be viewed as a function defined on the lattice points of a crystal torus, where Fourier functions diagonalize convolution operators. This yields a block-diagonal structure from which the singular value decomposition can be computed independently on smaller blocks. Existing approaches obtain this diagonalization and subsequent singular value decomposition in $\mathcal{O}(N \log N)$ by employing FFT, where N denotes the number of lattice points for fixed channel dimensions. FFT-based methods numerically evaluate the full discrete Fourier transform using divide-and-conquer strategies. Note that this complexity additionally includes the computation required for the singular value decomposition. In contrast, we reduce the computational complexity of computing the full spectrum by a factor of $\log N$. In particular, LFA exploits the fact that convolutions on the lattice tori are shift-invariant and become multiplication operators in the Fourier basis. The block-diagonal structure emerges naturally in the form of the operator's Fourier symbols, allowing the singular value decomposition to be computed in $\mathcal{O}(N)$. While periodic boundary conditions are not standard in deep learning applications, we showed that their influence diminishes rapidly as the domain size increases (Chapter 5).

Future research. In this thesis, our focus was on the design of weight-efficient ResNet-like architectures for image classification tasks. Both proposed methods follow the structure of a ResNet-18 with four different resolution levels, corresponding to the left leg of a multigrid V-cycle. A natural next step is to extend this design into a full V-cycle,

possibly followed by another left leg, thereby increasing representational capacity without substantially increasing complexity. Such an extension may lead to more robust classifiers and the deployment of memory-light models on leaner hardware.

Beyond classification, ResNets are commonly adopted as backbones for tasks such as semantic segmentation [17] or object detection [7, 91]. Our architectures, developed in Chapter 4, are well positioned to serve as efficient backbones in these contexts. In particular, the hierarchical channel structure of our in-channel building block resembles encoder-decoder designs, which are commonly used in segmentation and detection, where multi-scale representations are crucial. In addition, the polynomial-based building blocks can reduce computational cost for deeper backbones, thereby improving the efficiency of models that are otherwise expensive to train.

The improved accuracy–efficiency trade-off also suggests that our architectures are well suited for semi-supervised learning [3, 14], where only a small set of labeled examples is available. In such settings, models must extract reliable features from limited supervision while scaling efficiently to large amounts of unlabeled data. The weight sharing and hierarchical structure of our designs naturally support this by reducing overfitting risk and lower training cost and encourage consistent feature representations across scales.

Recent advances in vision increasingly employ attention-based architectures such as *Transformers* [29, 112], which capture both local and global dependencies through self-attention but at the cost of substantial training overhead. Multigrid-inspired approaches have also been used to provide insights by operating across scales [15, 76, 78], combining operations on fine scales with global updates. On the other hand, empirical studies have shown that many attention heads in Transformers exhibit substantial redundancy, suggesting that the architecture is overparameterized [81,113]. Sharing weights across heads therefore could reduce the weight count and memory footprint, potentially improving inference efficiency without significantly decreasing representational power.

Finally, our algorithm for computing the spectrum of convolutional mappings presented in Chapter 5 could be explored for pseudoinverse-based reconstruction method as a lightweight, training-free approach to image reconstruction or generation.

Our code is publicly available to enable reproducible benchmarking, facilitate further research and support adoption by the research community.

List of Figures

1.1 Fine and coarse grid scaling . . . 3
1.2 Separability of error components . . . 4
1.3 Sparsity patterns of neural network mappings in matrix form . . . 6

2.1 Singular value decomposition of a 2×2 matrix . . . 13
2.2 Full and reduced singular value decomposition . . . 14
2.3 Two-dimensional lattice and crystal . . . 15
2.4 Sublattice and crystal torus . . . 16
2.5 Aliasing of Fourier modes in $\mathbb{R}$ . . . 21
2.6 Periodic structure of Fourier functions under restriction . . . 22
2.7 Uniform discretization of the unit square . . . 30
2.8 Error polynomials . . . 34
2.9 Functions of the relaxation parameter . . . 36
2.10 Scale separation of error components . . . 38
2.11 Weighted restriction . . . 40
2.12 Prolongation . . . 40
2.13 Two- and three-level multigrid V-cycles . . . 43

3.1 Underfitting, overfitting, and well-fitted hypotheses on a toy dataset . . . 52
3.2 Single-layer perceptron . . . 54
3.3 Multilayer perceptron . . . 56
3.4 Activation functions . . . 57
3.5 Edge detection using a Laplacian filter . . . 66
3.6 Two-dimensional convolution . . . 67
3.7 Neighborhood of a 3×3 convolution operator . . . 68
3.8 Padding modes . . . 70
3.9 Matrix representation of a convolution . . . 71

3.10 Connectivity patterns of convolution operations 72
3.11 Average pooling on a crystal torus 73
3.12 Pooling operations 74
3.13 ResNet-block with two convolutional layers 76
3.14 ResNet18 architecture 77

4.1 Sparsity patterns of NN layers in matrix representation 80
4.2 Weight sharing in ResNet- and MgNet-blocks 81
4.3 Data-feature relations at resolution level ℓ 88
4.4 Connectivity patterns of convolution operations 89
4.5 Accuracy-weight trade-off of MGiaD on CIFAR-10 90
4.6 Smoothing-in-channels block with two in-channel levels 91
4.7 Structure of `SiC`-blocks on two resolution levels 91
4.8 Polynomials of type q_2 with different root choices 102
4.9 Residual polynomial evaluated on the spectrum 104
4.10 Accuracy-weight trade-off of Poly-MgNet on CIFAR-10 109

5.1 Sparsity patterns of convolutional mappings 112
5.2 Lattice and crystal 115
5.3 Sublattice and crystal torus 115
5.4 Spectra computed with different boundary conditions 118
5.5 Runtimes for computing singular values 120

List of Tables

4.1 Different group sizes in MgNet 94
4.2 MGiaD accuracy on CIFAR-10 95
4.3 Channel scaling in MGiaD on CIFAR-10 96
4.4 Post-smoothing in MGiaD 96
4.5 GFLOPs of MGiaD 97
4.6 MGiaD performance on FashionMNIST 98
4.7 Channel scaling in MGiaD on larger datasets 99
4.8 ReLU placement in Poly-MgNet 106
4.9 Polynomial initialization 107
4.10 ReLU and bn placement in q_2 and g_4 108
4.11 ReLU and bn placement in g_6 and g_8 109

5.1 Time complexity of SVD computation 117
5.2 Wasserstein distance between spectra 119
5.3 Runtime ratio of FFT- and LFA-based approaches 121
5.4 Runtimes for varying input sizes 122
5.5 Runtimes with different memory layouts 123

List of Notations

The following abbreviations and notations are used across all chapters:

Abbreviations

CE	cross-entropy
CNN	convolutional neural network
DFT	discrete Fourier transform
ERM	empirical risk minimization
EVD	eigenvalue decomposition
FAS	full approximation scheme
FFT	fast Fourier transform
FLOPs	floating point operations
i.i.d.	independently and identically distributed
KL divergence	Kullback-Leibler divergence
LFA	local Fourier analysis
MG	multigrid
MGiaD	Multigrid in all Dimensions
MGIC	Multigrid in Channels
ML	machine learning
NN	neural network
PAC learnability	probably approximately correct learnability
PDE	partial differential equation
pp	percentage points
ReLU	rectified linear unit
SGD	stochastic gradient descent
SiC	Smoothing in Channels
SVD	singular value decomposition

Symbols

$\mathcal{O}$	Landau symbol
$\Re$	Real part of a complex value
$\Im$	Imaginary part of a complex value
A^{T}	Transpose of A
A^*	Adjoint of A
$\det(A)$	Determinant of A
$\operatorname{spec}(A)$	Spectrum of A
$\rho(A)$	Spectral radius of A
$\kappa(A)$	Condition number of A
λ_i	Eigenvalue of A
σ_i	Singular value of A
$\mathcal{D}$	Data-generating distribution
$\mathcal{G}$	Gaussian distribution
$\mathcal{U}$	Uniform distribution
S	Training dataset
$z \in \mathcal{X}$	Input data space, i.e., an image
$y \in \mathcal{Y}$	Label
$\varrho(x) : \mathcal{X} \to \mathcal{Y}$	True labeling function
$\mathcal{L}$	Loss
$\psi(x) : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$	Parameterized function of a linear mapping
ϕ_{ReLU}	ReLU activation function
$\varphi_k(x)$	Wave function $e^{2\pi \mathrm{i}\langle x,k\rangle}$
$\mathbb{L}(\mathcal{A})$	n-dimensional lattice
$\Xi(\mathcal{A})$	Primitive cell of $\mathbb{L}(\mathcal{A})$
$\mathbb{L}^{\mathfrak{u}}(\mathcal{A})$	Crystal with structure element $\mathfrak{u} \in \Xi(\mathcal{A})$
$\mathbb{L}(\mathcal{A})^*$	Dual lattice of $\mathbb{L}(\mathcal{A})$ with $\mathbb{L}(\mathcal{A}^{-\mathrm{T}})$
$\mathfrak{L}(\mathbb{T}_{\mathcal{A},\mathcal{B}})$	Set of functions on the lattice torus $\mathbb{T}_{\mathcal{A},\mathcal{B}}$
$A : \mathfrak{L}(\mathbb{T}^{\mathfrak{d}}_{\mathcal{A},\mathcal{Z}}) \to \mathfrak{L}(\mathbb{T}^{\mathfrak{c}}_{\mathcal{A},\mathcal{Z}})$	Multiplication operator on crystal torus
$R : \mathfrak{L}(\mathbb{T}^{\mathfrak{d}+\mathfrak{a}}_{\mathcal{B},\mathcal{Z}}) \to \mathfrak{L}(\mathbb{T}^{\mathfrak{c}}_{\mathcal{B},\mathcal{Z}})$	Restriction operator on crystal torus

Bibliography

[1] A. Acharya, R. Goel, A. Metallinou, and I. Dhillon, *Online embedding compression for text classification using low rank matrix factorization*, Proceedings of the AAAI Conference on Artificial Intelligence, 33 (2019), pp. 6196–6203.

[2] J. M. Alvarez and M. Salzmann, *Learning the number of neurons in deep networks*, in Proceedings of the 30th International Conference on Neural Information Processing Systems, Curran Associates Inc., 2016, pp. 2270–2278.

[3] S. Basu, *Semi-supervised learning*, in Encyclopedia of Database Systems, Springer US, 2009, pp. 2613–2615.

[4] M. Ben Noach and Y. Goldberg, *Compressing pre-trained language models by matrix decomposition*, in Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, Association for Computational Linguistics, 2020, pp. 884–889.

[5] C. M. Bishop, *Pattern recognition and machine learning*, Information science and statistics, Springer, New York, 2006.

[6] A. L. Blum and R. L. Rivest, *Training a 3-node neural network is NP-complete*, Neural Networks, 5 (1992), pp. 117–127.

[7] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *YOLOv4: Optimal speed and accuracy of object detection*, 2020. arXiv:2004.10934.

[8] E. D. Bolluyt and C. Comaniciu, *Pseudoinvertible neural networks*, IEEE Transactions on Artificial Intelligence, 5 (2024), pp. 602–612.

[9] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Mathematics of Computation, 31 (1977), pp. 333–390.

[10] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, 2nd ed., 2000.

[11] R. Caruana, *Multitask learning*, Machine Learning, 28 (1997), pp. 41–75.

[12] S. Cha, T. Kim, H. Lee, and S.-Y. Yun, *A survey of supernet optimization and its applications: Spatial and temporal optimization for neural architecture search*, 2023. arXiv:2204.03916.

[13] S. Changpinyo, M. Sandler, and A. Zhmoginov, *The power of sparsity in convolutional neural networks*, CoRR, arXiv:1702.06257 (2017).

[14] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-supervised learning*, MIT Press, Cambridge, Massachusetts, 2006.

[15] C.-F. R. Chen, Q. Fan, and R. Panda, *CrossViT: Cross-attention multi-scale vision transformer for image classification*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 347–356.

[16] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 40 (2018), pp. 834–848.

[17] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-decoder with atrous separable convolution for semantic image segmentation*, in Computer Vision – ECCV, vol. 11211, Springer International Publishing, 2018, pp. 833–851.

[18] P. Chen, S. Si, Y. Li, C. Chelba, and C.-J. Hsieh, *GroupReduce: Block-wise low-rank approximation for neural language model shrinking*, in Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018.

[19] W. Chen, X. Xu, X. Wang, Z. Li, and Y. Chen, *Invisible backdoor attack through singular value decomposition*, in Pattern Recognition and Computer Vision, vol. 15032, Springer Nature Singapore, 2025, pp. 174–188.

[20] K. J. Cheoi, H. Choi, and J. Ko, *Empirical remarks on the translational equivariance of convolutional layers*, Applied Sciences, 10 (2020), p. 3161.

[21] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1800–1807.

[22] G. G. Chrysos, S. Moschoglou, G. Bouritsas, J. Deng, Y. Panagakis, and S. Zafeiriou, *Deep polynomial neural networks*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 44 (2022), pp. 4021–4034.

[23] G. G. Chrysos, S. Moschoglou, G. Bouritsas, Y. Panagakis, J. Deng, and S. Zafeiriou, *P–Nets: Deep polynomial neural networks*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 7323–7333.

[24] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, *Parseval networks: Improving robustness to adversarial examples*, in Proceedings of the 34th International Conference on Machine Learning, vol. 70, PMLR, 2017, pp. 854–863.

[25] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals, and Systems, 2 (1989), pp. 303–314.

[26] C. Darken, J. Chang, and J. Moody, *Learning rate schedules for faster stochastic gradient search*, in Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop, 1992, pp. 3–12.

[27] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, *Exploiting linear structure within convolutional networks for efficient evaluation*, in Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 1 of NIPS'14, MIT Press, 2014, pp. 1269–1277.

[28] K. Djoumessi, S. O. Mensah, and P. Berens, *A hybrid fully convolutional CNN-Transformer model for inherently interpretable medical image classification*, 2025. arXiv:2504.08481.

[29] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16×16 words: Transformers for image recognition at scale*, in International Conference on Learning Representations (ICLR), 2021.

[30] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, 2016. arXiv:1603.07285.

[31] R. Eldan and O. Shamir, *The power of depth for feedforward neural networks*, in Conference on Learning Theory, PMLR, 2016, pp. 907–940.

[32] M. Eliasof, J. Ephrath, L. Ruthotto, and E. Treister, *MGIC: Multigrid-in-channels neural network architectures*, SIAM Journal on Scientific Computing, 45 (2023), pp. S307–S328.

[33] T. Elsken, J. H. Metzen, and F. Hutter, *Neural architecture search*, in Automated Machine Learning, Springer International Publishing, 2019, pp. 63–77.

[34] S. J. Farlow, *The GMDH algorithm of Ivakhnenko*, The American Statistician, 35 (1981), pp. 210–215.

[35] K.-I. Funahashi, *On the approximate realization of continuous mappings by neural networks*, Neural Networks, 2 (1989), pp. 183–192.

[36] T. Gale, E. Elsen, and S. Hooker, *The state of sparsity in deep neural networks*, 2019. arXiv:1902.09574.

[37] W. M. Gentleman and G. Sande, *Fast Fourier transforms: for fun and profit*, in Proceedings of the fall joint computer conference, ACM Press, 1966, pp. 563–578.

[38] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[39] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins studies in the mathematical sciences, The Johns Hopkins University Press, Baltimore, 4th ed., 2013.

[40] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, Adaptive computation and machine learning, The MIT Press, Cambridge, Massachusetts, 2016.

[41] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, *MorphNet: Fast & simple resource-constrained structure learning of deep networks*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 1586–1595.

[42] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree, *Regularisation of neural networks by enforcing Lipschitz continuity*, Machine Learning, 110 (2021), pp. 393–416.

[43] H. Guo, G. Sitton, and C. Burrus, *The quick discrete Fourier transform*, in Proceedings of ICASSP. IEEE International Conference on Acoustics, Speech and Signal Processing, vol. iii, 1994, pp. III/445–III/448.

[44] M. Götz and H. Anzt, *Machine learning-aided numerical linear algebra: Convolutional neural network for the efficient preconditioner generation.* Karlsruher Institut für Technologie (KIT), 2018.

[45] W. Hackbusch, *Multi-Grid Methods and Applications*, vol. 4 of Springer Series in Computational Mathematics Ser, Springer, Berlin, Heidelberg, 2003.

[46] S. Han, J. Pool, J. Tran, and W. Dally, *Learning both weights and connections for efficient neural network*, in Advances in Neural Information Processing Systems, vol. 28, Curran Associates, Inc., 2015.

[47] S. Han, J. Pool, J. Tran, and W. J. Dally, *DSD: Dense-sparse-dense training for deep neural networks*, in International Conference on Learning Representations (ICLR), 2017.

[48] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Rio, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, *Array programming with NumPy*, Nature, 585 (2020), pp. 357–362.

[49] B. Hassibi and D. Stork, *Second order derivatives for network pruning: Optimal brain surgeon*, in Advances in Neural Information Processing Systems, vol. 5, Morgan-Kaufmann, 1992.

[50] J. He and J. Xu, *MgNet: A unified framework of multigrid and convolutional neural network*, Science China Mathematics, 62 (2019), pp. 1331–1354.

[51] J. He, J. Xu, L. Zhang, and J. Zhu, *An interpretive constrained linear model for ResNet and MgNet*, Neural Networks, 162 (2023), pp. 384–392.

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

[53] ——, *Deep residual learning for image recognition*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[54] ——, *Identity mappings in deep residual networks*, in Computer Vision – ECCV 2016, vol. 9908, Springer International Publishing, 2016, pp. 630–645.

[55] Y. He, X. Zhang, and J. Sun, *Channel pruning for accelerating very deep neural networks*, in IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1398–1406.

[56] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, *Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks*, Journal of Machine Learning Research, 22 (2021), pp. 1–124.

[57] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks, 2 (1989), pp. 359–366.

[58] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, *Searching for MobileNetV3*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 1314–1324.

[59] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *MobileNets: Efficient convolutional neural networks for mobile vision applications*, 2017. arXiv:1704.04861.

[60] Y.-C. Hsu, T. Hua, S. Chang, Q. Lou, Y. Shen, and H. Jin, *Language model compression with weighted low-rank factorization*, 2022. arXiv:2207.00112.

[61] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in Proceedings of the 32nd International Conference on Machine Learning, vol. 37 of PMLR, 2015, pp. 448–456.

[62] A. G. Ivakhnenko, *The group method of data handling-A rival of the method of stochastic approximation*, Soviet Automatic Control, 1 (1968), pp. 43–55.

[63] M. Jaderberg, A. Vedaldi, and A. Zisserman, *Speeding up convolutional neural networks with low rank expansions*, in Proceedings of the British Machine Vision Conference 2014, British Machine Vision Association, 2014, pp. 88.1–88.13.

[64] K. Kahl, *Adaptive algebraic multigrid for lattice QCD computations*, dissertation, Bergische Universität Wuppertal, 2009.

[65] ——, *Discrete Fourier transforms - A geometric approach*, 2025. private communication.

[66] K. Kahl and N. Kintscher, *Automated local Fourier analysis (aLFA)*, BIT Numerical Mathematics, 60 (2020), pp. 651–686.

[67] A. Katrutsa, T. Daulbaev, and I. Oseledets, *Deep multigrid: learning prolongation and restriction matrices*, 2017. arXiv:1711.03825.

[68] T.-W. Ke, M. Maire, and S. X. Yu, *Multigrid neural architectures*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4067–4075.

[69] N. Kintscher, *Automated local Fourier analysis (aLFA) and geometric multigrid for graphene*, dissertation, Bergische Universität Wuppertal, 2019.

[70] A. Krizhevsky, *Learning multiple layers of features from tiny images*, tech. rep., University of Toronto, 2009.

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012.

[72] Y. Le and X. S. Yang, *Tiny imagenet visual recognition challenge*, 2015.

[73] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[74] N. Lee, T. Ajanthan, and P. H. S. Torr, *SNIP: Single-shot network pruning based on connection sensitivity*, in International Conference on Learning Representations (ICLR), 2019.

[75] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, *Pruning filters for efficient convnets*, in International Conference on Learning Representations (ICLR), 2017.

[76] Y. Li, C.-Y. Wu, H. Fan, K. Mangalam, B. Xiong, J. Malik, and C. Feichtenhofer, *MViTv2: Improved multiscale Vision Transformers for classification and detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 4804–4814.

[77] S. Liu, H. Zhang, and Y. Jin, *A survey on computationally efficient neural architecture search*, Journal of Automation and Intelligence, 1 (2022), p. 100002.

[78] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, *Swin Transformer V2: Scaling up capacity and resolution*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 11999–12009.

[79] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, *A convnet for the 2020s*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 11966–11976.

[80] I. Loshchilov and F. Hutter, *SGDR: Stochastic gradient descent with warm restarts*, in International Conference on Learning Representations, ICLR, 2017.

[81] P. Michel, O. Levy, and G. Neubig, *Are sixteen heads really better than one?*, in Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.

[82] M. Minsky and S. Papert, *Perceptrons: an introduction to computational geometry*, MIT Press, expanded ed., 1988.

[83] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, *Pruning convolutional neural networks for resource efficient inference*, in International Conference on Learning Representations (ICLR), 2017.

[84] B. Neyshabur, S. Bhojanapalli, and N. Srebro, *A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks*, in International Conference on Learning Representations (ICLR), 2018.

[85] B. H. Ngo, N.-T. Do-Tran, T.-N. Nguyen, H.-G. Jeon, and T. J. Choi, *Learning CNN on ViT: A hybrid model to explicitly class-specific boundaries for domain adaptation*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 28545–28554.

[86] A. B. J. Novikoff, *On convergence proofs for perceptrons*, in Proceedings of the Symposium on the Mathematical Theory of Automata, 1963, pp. 615–622.

[87] S.-K. Oh, W. Pedrycz, and B.-J. Park, *Polynomial neural networks architecture: analysis and design*, Computers & Electrical Engineering, 29 (2003), pp. 703–725.

[88] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An imperative style, high-performance deep learning library*, in Proceedings of the 33rd International Conference on Neural Information Processing Systems, Curran Associates Inc., 2019, pp. 8026–8037.

[89] B. Praggastis, D. Brown, C. O. Marrero, E. Purvine, M. Shapiro, and B. Wang, *The SVD of Convolutional Weights: A CNN Interpretability Framework*, 2022. arXiv:2208.06894.

[90] N. Qian, *On the momentum term in gradient descent learning algorithms*, Neural Networks, 12 (1999), pp. 145–151.

[91] J. Redmon and A. Farhadi, *YOLOv3: An incremental improvement*, 2018. arXiv:1804.02767.

[92] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, 65 (1958), pp. 386–408.

[93] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by backpropagating errors*, Nature, 323 (1986), pp. 533–536.

[94] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet large scale visual recognition challenge*, International Journal of Computer Vision, 115 (2015), pp. 211–252.

[95] E. Ryu, J. Liu, S. Wang, X. Chen, Z. Wang, and W. Yin, *Plug-and-play methods provably converge with properly trained denoisers*, in Proceedings of the 36th International Conference on Machine Learning, K. Chaudhuri and R. Salakhutdinov, eds., vol. 97 of Proceedings of Machine Learning Research, PMLR, 2019, pp. 5546–5557.

[96] Y. Saad, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2nd ed., 2007. Reprint.

[97] S. Salmani Pour Avval, N. D. Eskue, R. M. Groves, and V. Yaghoubi, *Systematic review on neural architecture search*, Artificial Intelligence Review, 58 (2025), p. 73.

[98] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *MobileNetV2: Inverted residuals and linear bottlenecks*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510–4520.

[99] H. Sedghi, V. Gupta, and P. M. Long, *The singular values of convolutional layers*, in International Conference on Learning Representations (ICLR), 2019.

[100] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: from theory to algorithms*, Cambridge University Press, New York, 2014.

[101] Y. Shin and J. Ghosh, *The pi-sigma network: an efficient higher-order neural network for pattern classification and function approximation*, in International Joint Conference on Neural Networks (IJCNN), vol. 1, IEEE, 1991, pp. 13–18.

[102] S. Singla and S. Feizi, *Fantastic Four: Differentiable and efficient bounds on singular values of convolution layers*, in International Conference on Learning Representations (ICLR), 2021.

[103] M. Telgarsky, *Benefits of depth in neural networks*, in Conference on Learning Theory, PMLR, 2016, pp. 1517–1539.

[104] C. Tomasi and R. Krause, *Construction of grid operators for multilevel solvers: a neural network approach*, in Domain Decomposition Methods in Science and Engineering XXVI, vol. 145, Springer International Publishing, 2022, pp. 579–587.

[105] L. N. Trefethen and D. Bau, *Numerical linear algebra*, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[106] E. Treister and I. Yavneh, *On-the-fly adaptive smoothed aggregation multigrid for Markov chains*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2927–2949.

[107] U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, San Diego, 2001.

[108] Y. Tsuzuku, I. Sato, and M. Sugiyama, *Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks*, in Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, p. 6542–6551.

[109] A. Van Betteray, M. Rottmann, and K. Kahl, *MGiaD: Multigrid in all dimensions. Efficiency and robustness by weight sharing and coarsening in resolution and channel dimensions*, in IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2023, pp. 1284–1293.

[110] ——, *LFA applied to CNNs: Efficient singular value decomposition of convolutional mappings by local Fourier analysis*, in Proceedings of the European Conference on Artificial Intelligence (ECAI), vol. 413 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2025, pp. 90–97.

[111] ——, *Poly-MgNet: Polynomial building blocks in multigrid-inspired ResNets*, in Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods (ICPRAM), SCITEPRESS - Science and Technology Publications, 2025, pp. 181–191.

[112] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.

[113] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, *Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned*, in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2019, pp. 5797–5808.

[114] C.-Y. Wang, I.-H. Yeh, and H.-Y. Mark Liao, *YOLOv9: Learning what you want to learn using programmable gradient information*, in Computer Vision – ECCV 2024, vol. 15089, Springer Nature Switzerland, 2025, pp. 1–21.

[115] L. Wei, Z. Ma, C. Yang, and Q. Yao, *Advances in the neural network quantization: A comprehensive review*, Applied Sciences, 14 (2024), p. 7445.

[116] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, *Learning structured sparsity in deep neural networks*, in Advances in Neural Information Processing Systems, vol. 29, Curran Associates, Inc., 2016.

[117] P. J. Werbos, *Generalization of backpropagation with application to a recurrent gas market model*, Neural Networks, 1 (1988), pp. 339–356.

[118] D. H. Wolpert, *The lack of a priori distinctions between learning algorithms*, Neural Computation, 8 (1996), pp. 1341–1390.

[119] S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon, and S. Xie, *ConvNeXt V2: Co-designing and scaling ConvNets with masked autoencoders*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023, pp. 16133–16142.

[120] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, *CvT: Introducing convolutions to Vision Transformers*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 22–31.

[121] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms*, 2017. arXiv:1708.07747.

[122] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5987–5995.

[123] H. Yang and S.-i. Amari, *The efficiency and the robustness of natural gradient descent learning rule*, in Advances in Neural Information Processing Systems, vol. 10, MIT Press, 1997.

[124] Y. Yoshida and T. Miyato, *Spectral norm regularization for improving the generalizability of deep learning*, 2017. arXiv:1705.10941.

[125] F. Yu and V. Koltun, *Multi-scale context aggregation by dilated convolutions*, in International Conference on Learning Representations (ICLR), 2016.

[126] X. Zhang, X. Zhou, M. Lin, and J. Sun, *ShuffleNet: An extremely efficient convolutional neural network for mobile devices*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 6848–6856.

[127] X. Zhang, J. Zou, K. He, and J. Sun, *Accelerating very deep convolutional networks for classification and detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 38 (2016), pp. 1943–1955.

[128] D.-X. Zhou, *Universality of deep convolutional neural networks*, Applied and Computational Harmonic Analysis, 48 (2020), pp. 787–794.

[129] Y. T. Zhou and R. Chellappa, *Computation of optical flow using a neural network*, in IEEE International Conference on Neural Networks, vol. 2, 1988, pp. 71–78.

Appendix

AI based tool	usage	affected parts of the thesis	comments
DeepLTranslate[1]	translation of sentences and words	throughout the thesis	mainly used to find phrasings ahd synonyms in the appropriate contex
DeepLWrite[2]	language improvement of sentences	throughout the thesis	
ChatGPT5[3]	writing improvement of sentences and paragraphs	throughout the thesis	usage as review tool to refine passages written by myself
	generation of tikz code for figures or figure modules	Figure 1.2 Figure 2.7 Figure 2.8 Figure 2.6 Figure 3.2 Figure 3.3	generated code served as a basis, all figures were modified and adjusted by hand
	literature review	Chapter 2 Chapter 3 Chapter 6 bibliography	usage as search engine to request addition literature to specific topics to ensure completeness searching for peer-reviewed sources and identifiers, e.g., doi
ChatGPT-5.6 sol[3]	writing improvement of sentences and paragraphs	throughout the thesis	usage as review tool to refine passages written by myself

[1] `https://www.deepl.com/de/translator`
[2] `https://www.deepl.com/de/write`
[3] `https://chatgpt.com/`