# Poly-MgNet: Polynomial Building Blocks in Multigrid-Inspired ResNets

Antonia van Betteray[a], Matthias Rottmann[b] and Karsten Kahl[c]
*IZMD, University of Wuppertal, Germany*
{*vanbetteray, rottmann, kkahl*}*@uni-wuppertal.de*

Keywords: ResNets, Multigrid Methods, Polynomial Smoother, Accuracy-Weight Trade-Off.

Abstract: The structural analogies of ResNets and Multigrid (MG) methods such as common building blocks like convolutions and poolings where already pointed out by He et al. in 2016. Multigrid methods are used in the context of scientific computing for solving large sparse linear systems arising from partial differential equations. MG methods particularly rely on two main concepts: smoothing and residual restriction / coarsening. Exploiting these analogies, He and Xu developed the MgNet framework, which integrates MG schemes into the design of ResNets. In this work, we introduce a novel neural network building block inspired by polynomial smoothers from MG theory. Our polynomial block from an MG perspective naturally extends the MgNet framework to Poly-Mgnet and at the same time reduces the number of weights in MgNet. We present a comprehensive study of our polynomial block, analyzing the choice of initial coefficients, the polynomial degree, the placement of activation functions, as well as of batch normalizations. Our results demonstrate that constructing (quadratic) polynomial building blocks based on real and imaginary polynomial roots enhances Poly-MgNet's capacity in terms of accuracy. Furthermore, our approach achieves an improved trade-off of model accuracy and number of weights compared to ResNet as well as compared to specific configurations of MgNet.

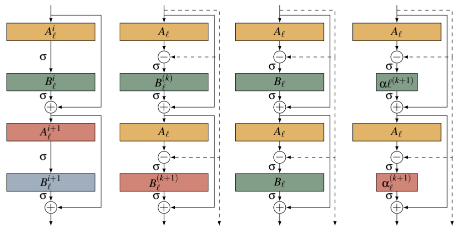
## 1 INTRODUCTION

Deep convolutional neural networks (CNNs) are state-of-the-art methods for image classification tasks (Krizhevsky et al., 2012; Russakovsky et al., 2015; He et al., 2015; Dosovitskiy et al., 2021). Especially ResNets (He et al., 2016a; He et al., 2016b; Liu et al., 2022) have become increasingly popular, as they successfully overcome the vanishing gradient problem (Glorot and Bengio, 2010a).

Nevertheless these networks contain $O(10^7)$ – $O(10^{10})$ weights, thus being heavily over parameterized. A reduction of the weight count is clearly desirable, which, however, can result in an undesired bias. This trade-off is referred to as "bias-complexity trade-off" which constitutes a fundamental problem of machine learning, see e.g. (Shalev-Shwartz and Ben-David, 2014). In this work, we address this problem from a multigrid (MG) perspective. MG methods are hierarchical solvers for large sparse systems of linear equations that arise from discretizations of partial differential equations (Trottenberg et al., 2001). The main idea of MG consists of two components, namely a local relaxation scheme, which is cheap to apply, but slow to converge as it lacks the possibility to address global features. Thus it is complemented with a coarse grid correction, that exploits a representation of the problem formulation on a coarser scale thus making long range information exchange easier. In classical MG theory this complementarity can be associated with the split of the error into geometrically oscillatory and smooth functions. Where the oscillatory part is quickly dampened by the local relaxation scheme and the smooth part accurately described and dealt with on coarser scales (Trottenberg et al., 2001). The authors of ResNet (He et al., 2016a) already mentioned the inherent similarities between MG and residual layers. This structural connection was further elaborated in (He and Xu, 2019), where ResNets, composed of residual layers (representing the smoothers) and pooling operations (representing the coarse grid restrictions), are cast into a full approximation scheme (FAS). The resulting framework, termed MgNet, further exploits the similarity to MG methods, in which the discretized operators stemming from PDEs remain fixed between consecutive coarsenings/poolings. This yields a justification for sharing weight tensors across multiple residual layers, e.g. fig. 1(b) and fig. 1(c), reducing the

[a] https://orcid.org/0000-0002-2338-1753
[b] https://orcid.org/0000-0003-3840-0184
[c] https://orcid.org/0000-0002-3510-3320

van Betteray, A., Rottmann, M. and Kahl, K.
Poly-MgNet: Polynomial Building Blocks in Multigrid-Inspired ResNets.
DOI: 10.5220/0013382800003905
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2025)*, pages 181-191
ISBN: 978-989-758-730-6; ISSN: 2184-4313
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.



(a) residual (b) non-stationary (c) stationary (d) polynomial

Figure 1: Weight sharing in ResNet and MgNet; (1a) ResNet-blocks, no weight sharing; (1b) MgNet-blocks, shared $A_\ell$; (1c) MgNet-blocks, shared layers $A_\ell$ and $B_\ell$ and Poly-MgNet (fig. 1d).

overall weight count of the network. A joint perspective of multigrid structures in all dimensions was taken by (van Betteray et al., 2023), resulting in an improved weight-accuracy trade-off and thus demonstrates that ResNets are overparameterized.

In this work, we exploit additional structural similarities between MG and residual networks to further reduce the weight count of MgNet. A typical ResNet block is built from two convolutions $A$ and $B$ gated by a residual connection. Considering a given system of linear equations $Au = f$, so-called *MG smoothing* applies an operation $B$ to the residual $f - Au$, i.e.,

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

in order to improve $u$, i.e., reduce the residual, iteratively. In a nutshell, this operation has the property of smoothing the error associated to 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 linear-algebra analogy, we consider polynomials $p(A)$ for $B$ with only a handful of learnable parameters, thus reducing the number of parameters by almost 50%. A special case and the simplest instance of this idea is the Richardson iteration (Trottenberg et al., 2001) where $B = \omega I$, with $I$ the identity matrix and $\omega$ a scalar. Clearly, we do not want to limit ourselves to such trivial polynomials and explore in this work to which extent smoothing iterations using polynomial approximations are suitable within MgNet, e.g. fig. 1(c), to further reduce the number of weights.

We summarize our contribution as follows:

1. We introduce a new building block, to further reduce the number of weights in multigrid-inspired CNNs, such as MgNet. These methods are inspired by MG smoothers, namely block smoothers, polynomial smoothers and Richardson smoothers, yielding layer modules of reduced weight counts.
2. We implement this building block into MgNet and show significant reductions of weight counts while almost maintaining classification accuracy.
3. Our approach yields further insights into the inherent connection of MG and ResNets and demonstrates that MG methodology is useful for the construction of weight-count-efficient CNNs.

The remainder of this article is organized as follows: Section 2 discusses related works. In section 3 we elaborate on the similarities of residual networks and MG, followed by the construction of our layers inspired by MG smoothers. Ultimately numerical results are presented in section 4.

## 2 RELATED WORKS

In this section, we provide an overview of related works, classified into four categories of approaches, sorted from remotely to closely related.

**Channel Reduction.** The set of existing methods for the reduction of weight counts and the set of existing methods to reduce the computational complexity of CNNs have a significant intersection. As one of many possible approaches, a reduction of computational complexity can be achieved by a reduction of the number of channels in convolutional layers. Considering the simplified case where the number of input channels $c$ equals the number of output channels of a given convolutional layer and the filter extent $s$ being equal in both directions, the number of weights of such a layer is given by $s^2 \cdot c^2$. Thus, a reduction of $c$ results in a clear reduction of the number of weights. In (Molchanov et al., 2016) it was shown experimentally, that there is redundancy in CNNs, which allows for cutting connections between channels after the CNN has been trained. This process is known as CNN pruning (Hassibi and Stork, 1992; Han et al., 2015; Li et al., 2017). At the same time pruning and other sparsity enhancing methods (Changpinyo et al., 2017; Han et al., 2017) reduce the model complexity in terms of weights. While these approaches first train a CNN to convergence, in (Gale et al., 2019) the CNN is pruned periodically during training. In (Lee et al., 2019) a saliency criterion to identify structurally important connections is proposed, which allows for pruning before training. We also reduce the weight count before training, however, our approach is based on the inherent similarity of MG and ResNets, utilizing MG methodology to find an explanation for more weight count efficient layer modules.

A related research area is the field of neural architecture search (Elsken et al., 2019; Cha et al., 2022), where the goal is to search the space of architectures close to optimal ones w.r.t. chosen optimization criteria. If the focus is on computational efficiency, then tuning the channel hyper-parameters is one of many possible approaches (Gordon et al., 2017), resulting in a reduction of weight count. While the aforementioned approaches use optimization procedures to reduce the number of weights, we utilize MG methodology that yields an explanation for the achieved efficiency.

**Modified Layers.** Another line of research addresses the specific construction of convolutional layers. Compared to the aforementioned approaches, the constructions outlined in this section are based on human intuition and conventional methods to improve computational efficiency. Also in this line of research, there is a close connection between computational efficiency and the reduction of the number of weights. One approach to reduce the number of weights is the use of grouped convolutions (Krizhevsky et al., 2012; Xie et al., 2017), where the channels are grouped into $g$ decoupled subsets, each convoluted with its own set of filters, thus decreasing the weight count to $s^2 \cdot \left(\frac{c}{g}\right)^2$. This decoupled structure impedes the exchange of information across the channels. To overcome this issue, e.g. in (Zhang et al., 2018) a combination of groupings and channel shuffling, termed ShuffleNet, has been proposed.

A combination of layers, which also allows for a reduction in floating point operations, are so-called depth-wise separable convolutions, introduced in (Howard et al., 2017; Sandler et al., 2018; Howard et al., 2019) as key feature for the MobileNet architecture: a depth-wise convolution, i.e., grouped convolution with $g = c$ is followed by a point-wise $1 \times 1$ convolution, to create a linear combination of the output of the former. This results in a reduced weight count of $s^2 + 2c$, i.e., each kernel slice of extent $c^2$ is replaced by a rank - one approximation.

**Polynomial Neural Networks.** Polynomial neural networks are NNs that produce an output being a polynomial of the input. This approach was first pursued via the group method of data handling (GMDH) (Oh et al., 2003). It determines the structure of a polynomial model in a combinatorial fashion 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 considers the output of the network as a high-order polynomial of the input (Shin and Ghosh, 1991; Chrysos et al., 2020; Chrysos et al., 2022). Our goal is to utilize polynomials as a building block in CNNs without reframing their purpose to polynomial approximation.

**Multigrid Inspired Architectures.** In scientific computing, MG methods are algorithms based on hierarchical discretizations, to efficiently solve systems of (non-)linear differential equations (PDEs) (Trottenberg et al., 2001; Treister and Yavneh, 2011; Kahl, 2009). Some works utilizes CNNs to solve PDEs (Tomasi and Krause, 2021), e.g. by estimating optimal preconditioners (Götz and Anzt, 2018) or prolongation and restriction operators (Katrutsa et al., 2017). Another line of research focuses on architectures based on common computational components as well as similarities of CNNs and MG (He et al., 2016a). In (Ke et al., 2017) an architecture of pyramid layers of differently scaled convolutional layers, with each pyramid processing coarse and fine grid representations in parallel, is proposed. This MG architecture improves accuracy, while being weight-count efficient.

The close similarities between CNNs and MG are further exploited in (He and Xu, 2019), where a framework termed MgNet is introduced. It yields a justification for sharing weight tensors within convolutions in subsequent ResNet blocks with the same spatial extent. Utilizing MG in the spatial dimensions, MgNet models have, compared to ResNets with the same number of layers, fewer weights, while maintaining classification accuracy. An alternating stack of MgNet blocks and poolings can be viewed as the left leg of an MG $V$-cycle. Another hierarchical structure, however in the channel dimension, is proposed in (Eliasof et al., 2020). Their building block, termed multigrid-in-channels (MGIC), is built on grouped convolutions and coarsening via channel pooling. Utilizing MG in the channel dimensions, this approach improves the scaling of the number of weights with the number of channels from quadratic to linear. A unified MG perspective is taken on both the spatial and channel dimensions in (van Betteray et al., 2023). 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. Similarly to these approaches, we exploit the inherent similarities between CNNs and MG to further reduce the weight count. Our focus in this work is to cast MG smoothers into CNN layer modules, which has not been studied in related work.

## 3 RESIDUAL NETWORKS AND MULTIGRID METHODS

MG consists of two complementary components, the smoother and the coarse grid correction. In a recursive fashion, the coarse grid correction is typically treated by restricting/pooling in spatial dimensions, then smoothing on the next coarser scale and then applying another even coarser coarse grid correction, etc. This structure is already resembled by typical CNNs which alternate between convolutions and poolings. In MG and in CNNs, the smoother is the main work step with the highest computational effort. A suitable choice of it is vital for the success of an MG method and we will see that these findings are beneficial for CNNs as well. In this section we introduce the smoother, the coarsening and unify the MG and CNN perspective.

**Revisiting ResNet and MgNet.** 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 (He and Xu, 2019). 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

$$A : \mathbb{R}^{m\times n\times h} \mapsto \mathbb{R}^{m\times n\times c}, \quad \text{s.t. } A(u) = f \tag{1}$$

$$B : \mathbb{R}^{m\times n\times c} \mapsto \mathbb{R}^{m\times n\times h}, \quad \text{s.t. } u \approx B(f), \tag{2}$$

where $m$ and $n$ characterize the spatial resolution dimensions of the input and $c$ and $h$ determine the dimension of the input and output channel respectively, i.e. number of input and output channels can differ. $A$ can be considered as a *feature-to-data map*, while $B$ is applied to elements of the data space, is also referred to as *feature extractor*. In MG the property $u \approx B(f)$ is beneficial for the method's convergence, which will be explained in the following. In the context of CNNs, convolutional mappings usually are combined with non-linear activation functions. For a clear presentation of the similarities between CNNs and MG, the non-linearities are omitted for now. Considering a large sparse system of linear equations $A(u) = f$, obtaining the direct solution $u$ is not feasible. Therefore, the true solution is iteratively approximated by $\widetilde{u}$. The resulting error $e = u - \widetilde{u}$ fulfills the residual equation

$$r = f - A(\widetilde{u}) = A(u - \widetilde{u}) = Ae, \tag{3}$$

where $r$ is referred to as residual. Now, given an appropriate feature extractor $B$, the approximated error $\widetilde{e} = Br$ can be used to update the approximated solution $\widetilde{u} \leftarrow \widetilde{u} + \widetilde{e}$. Repeating this scheme with eq. (3) yields a non-stationary iteration

$$u \leftarrow u + B^{(k)}(f - A(u)) \quad \text{for } k = 1, 2, \ldots \tag{4}$$

to solve $A(u) = f$ approximately. Hence, that feature extractors $B^{(k)}$ depend on iteration $k$, yet eq. (4) 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 examined in (He and Xu, 2019) the structure of eq. (4), with activation functions added, resembles a ResNet-block. Combined with eq. (3) yields a non-stationary iterative scheme. Explicitly, given the solution $u^{(k)}$ of iteration $k$ the residual

$$r^{(k+1)} = f - Au^{(k+1)} = (I - AB^{(k)})r^{(k)} \tag{5}$$

is propagated to $k+1$-th iteration and the coefficient $(I - BA)$ equals a ResNet-block $r^{(k+1)} = r^{(k)} + BAr^{(k)}$ (cf. fig. 1a). The main difference between MgNet and ResNet is sharing one or more weight tensors. Two ResNet-blocks at one resolution level do not share any weight tensors (cf. fig. 1a). The weight count of such two distinct blocks is given by $4 \cdot (s^2 \times c \times h)$. In MgNet, for the non-stationary case, i.e. only $A$ is shared, the weight count is reduced to $3 \cdot (s^2 \times c \times h)$. For the stationary case, i.e. sharing both $A$ and $B$ the weight count for two blocks is only $2 \cdot (s^2 \times c \times h)$. The pseudo-inverse of $A$ is generally a dense matrix, which requires the representation as a fully connected weight layer, but this conflicts with the required convolutional structure of the feature extractors $B^{(k)}$ (respectively $B$). Even with an optimal choice of $B^{(k)}$ the convergence of the iteration eq. (4) is inevitably slow, due to acting locally. However, convolutional operators $B^{(k)}$ are computationally light. A few applications of the iteration have a smoothing effect on features, and the resulting error can be accurately represented at a coarser resolution.

**Resolution Coarsening.** The restriction of the (residual) data to coarser scales, facilitated by mappings

$$R_\ell^{\ell+1} : \mathbb{R}^{m_\ell\times n_\ell\times c_\ell} \mapsto \mathbb{R}^{m_{\ell+1}\times n_{\ell+1}\times c_{\ell+1}}, \tag{6}$$

yields a hierarchy of resolution levels $\ell = 1, \ldots, L$. On each level $\ell$ the smoothing iteration (4) can be applied by resolution-wise mappings $A_\ell$ and $B_\ell^{(k)}$, where on each level $\ell$ initially $u_\ell^{(0)} = 0$. Note that the feature extractors can also be stationary. Equivalent to restrictions in MG, in CNNs the resolution dimension is reduced by pooling operations with stride greater than 1. In CNNs usually the channel dimension is increased in the process. Corresponding to the coarsening leg of a standard MG $V$-cycle (Trottenberg et al., 2001), the combination of $\nu$ smoothing iterations on each resolution level $\ell$ and $L-1$ restrictions yields algorithm 1.

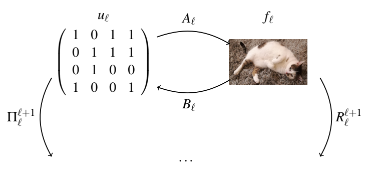


Figure 2: Data-feature relations $A_\ell$ and $B_\ell$ on resolution level $\ell$ followed by transfer to coarser resolution $\ell+1$. $A_\ell$ applied to the features $u_\ell$, calculation of the residual $r_\ell = f_\ell - A_\ell u_\ell$, on which the feature extractor $B_\ell$ is applied.

Algorithm 1: \-MgNet($f_\ell$).

```
  Initialization: $u_\ell = 0$
1 for $\ell = 1,\dots,L$ do
2     for $k = 1,\dots,\nu$ do
3         $u_\ell^{(k+1)} = u_\ell^{(k)} + B_\ell^{(k)}(f_\ell - A_\ell(u_\ell^{(k)}))$
4     end
5     $u_{\ell+1}^{(0)} = 0$
6     $f_{\ell+1} = R_\ell^{\ell+1}(f_\ell - A_\ell(u_\ell^{(\nu)}))$
7 end
```

**Full Approximation Scheme (FAS).** So far activation functions, potential non-linear poolings and normalization operations, which are characteristic of CNNs, have been disregarded. However, if we now take these into account, we obtain a non-linear overall structure for CNNs. The non-linearity in MG problems yields multiple possible minima, so that the initial solution not only determines the solution, but also has a crucial influence on the convergence rate. Consequently, an initial guess $u_{\ell+1}^{(0)}$ at the coarser scale determined by the current feature approximation $u_\ell$, is likely to dominate $u_{\ell+1}^{(0)} = 0$. Therefore, the feature approximations of non-linear problems are also projected to the coarser scale by a mapping

$$\Pi_\ell^{\ell+1} : \mathbb{R}^{n_\ell \times m_\ell \times c_\ell} \mapsto \mathbb{R}^{n_{\ell+1} \times m_{\ell+1} \times c_{\ell+1}}, \quad (7)$$

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 $A_{\ell+1}(u_{\ell+1})$.

Accordingly, in algorithm 1 lines 5 and 6 are changed to

$$u_{\ell+1}^{(0)} = \Pi_\ell^{\ell+1} u_\ell^{(\nu)} \quad (8)$$

$$f_{\ell+1} = R_\ell^{\ell+1}(f_\ell - A_\ell(u^{(\nu)}{}_\ell)) + A_{\ell+1}(u^{(0)}{}_{\ell+1}). \quad (9)$$

In the CNN context, the projection operation (7) corresponds to another pooling operation, but it has no exact counterpart in the general ResNet architecture. Figure 2 summarizes relevant mappings and schemes the role of $B_\ell$ as feature extractor, $A_\ell$ as data-feature mapping, followed by restriction and projection, respectively.

## 3.1 Polynomial (Smoother) in Residual Blocks

Thus far the iterative scheme, eq. (4), was either considered to be stationary or non-stationary with matrices $B^{(k)}$ or $B$, respectively. Even though $B$ is a rough and computationally inexpensive approximation for $A^{-1}$, 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

$$u \leftarrow u + p_d(A)(f - A(u)). \quad (10)$$

In here $p_d$ is a polynomial

$$p_d(A) = \sum_{i=0}^{d} \alpha_i A^i. \quad (11)$$

with (learnable) coefficients $\alpha_i$. In the case of polynomial smoothers ($B = p_d(A)$), the non-stationary case is considered, e.g. only 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 $\Lambda$ of eigenvalues, we find

$$p_d(A) = \sum_{i=0}^{d} \alpha_i A^i = X\left(\sum_{i=0}^{d} \alpha_i \Lambda^i\right) X^{-1}. \quad (12)$$

Thus the action of the polynomial on a matrix $A$ is determined solely by the evaluation of the scalar polynomial $p_d$ with coefficients $\alpha_i$ on the eigenvalues of $A$. As it alleviates notation we are thus considering the polynomial $p_d$ both as a scalar polynomial and a matrix valued polynomial using the same notation. Moreover, from now on for clarity in notation, let $\Lambda$ denote the spectrum of $A$.

**Residual Blocks as Polynomials.** Let $B = p_d(A)$ be a polynomial feature extractor, than the residual equation 3 yields

$$r^{(k+1)} = (I - Ap_d(A))r^{(k)} = q_{d+1}(A)r^{(0)}, \quad (13)$$

where the factor $(I - Ap_d(A))$ itself is a polynomial $q_{d+1}(A)$ of degree $d+1$. Hence, using eq. (11) we find

$$q_{d+1}(A) = I - Ap_d(A) = I - \sum_{i=0}^{d} \alpha_i A^{i+1}. \quad (14)$$

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

$$q_{d+1}(A) = \prod_{i=1}^{d+1} (I - \frac{1}{\zeta_i} A) \text{ with } q_{d+1}(\zeta_i) = 0 \text{ and } \zeta_i \in \mathbb{C}. \tag{15}$$

That is, instead of learning the coefficients $\alpha_i$ we can equivalently learn the roots of the polynomial $q_{d+1}$. While it is not immediately clear how $\alpha_i$ 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. That is, writing $r^{(k)} = \sum_j \gamma_j x_j$ as a linear combination of the eigenvectors of $A$ we find

$$q_{d+1}(A) r^{(k)} = \sum_j \prod_{i=1}^{d+1} \left(1 - \frac{\lambda_j}{\zeta_i}\right) \gamma_j x_j, \tag{16}$$

which is small, if the roots $\zeta_i$ 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 over correction of the respective eigencomponents of the residual (Saad, 2003)(cf. fig. 4). Yet, the aim of minimizing the residual is consistent with keeping the polynomial within the spectrum small. To that end eigenvalues located at the spectrum's boundary are chosen as roots for the polynomial. Obviously $A^T \neq A$ holds, which yields guaranteed complex conjugated pairs of eigenvalues. This can be used to construct a polynomial in linear factor representation, covering the extend of the real axis, e.g. fig. 4, and quadratic terms, covering the extend of the imaginary area and avoiding complex arithmetic within the CNN at the same time[2].

The residual propagation eq. (3) for a pair of complex conjugated eigenvalues $z$ and $\bar{z}$ yields

$$r^{(k+1)} = (1 - \frac{1}{z} A)(1 - \frac{1}{\bar{z}} A) r^{(k)} = \tilde{q}_{d+1} r^{(k)} \tag{17}$$

where $\tilde{q}_{d+1}$ itself is a quadratic polynomial of degree $d+1$ with roots $z = a + ib$ and its complex conjugated counterpart $\bar{z}$ with i the imaginary unit. Thus, the resulting polynomial is the product of $m$ linear polynomials $\hat{q}_i$ and $n$ quadratic polynomials $\tilde{q}_i$

[1]Clearly such modes are not affected in a general residual block with a (full) parameter matrix $B$ either.

[2]E.g. nn.ReLU does not support complex values, c.f. issues #47052, #46642.

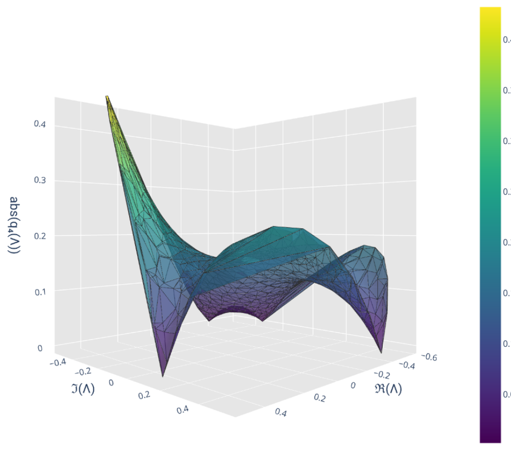


Figure 3: Surface representation of the spectrum $\Lambda$ for the corresponding matrix $A \in \mathbb{R}^{64 \times 64 \times 3 \times 3}$. The $x$-axis represents the real parts, while $y$-axis corresponds to the imaginary part. The $z$-axis illustrates the amplitude of the polynomial function of abs($q_4(\Lambda)$), which has roots at the eigenvalues with minimal and maximal real parts, as well as the and complex conjugated pair of eigenvalues with the maximal imaginary part. This visualization allows for an intuitive identification of the spectrum's maximal amplitude.

$$q_{m+2n}(A) = \prod_{i=1}^{m} \hat{q}_i(A) \cdot \prod_{j=1}^{2n} \tilde{q}_i(A), \tag{18}$$

s.t. $m + 2n$.

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 $B^{(k)} = 1/\zeta_k$. The quadratic terms also can be viewed as a sequence of pairs of linear residual blocks, i.e. two residual blocks with $B = \hat{q}_{2d}$ are equivalent to $B^{(k)} = 1/z$ and $B^{(k+1)} = 1/\bar{z}$ respectively. The focus of our work is on the polynomial perspective of a residual block, denoted by $q$. The iteration can be rewritten as

$$u^{(k+1)} = u^{(k)} + (\frac{2a}{a^2 + b^2} - \frac{1}{a^2 + b^2}) A r^{(k)}. \tag{19}$$

For the ease of notation from now on the degree of a polynomial $q_d$ is $d$. Consequently the degree of a polynomial $p$ is $d-1$. Furthermore combining eq. (10) with eq. (18) results in a polynomial version of MgNet, referred to as Poly-MgNet, which is outlined in algorithm 2.

Due to observations made in our experiments we limit the polynomial with linear factors, i.e. $\hat{q}_i$ in eq. (18) 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 biggest real part for the polynomial roots, i.e. $\zeta^{(1)} = \min \Re(\Lambda)$ and $\zeta^{(2)} =$

Algorithm 2: Poly-\-MgNet($f_\ell$).

```
   Initialization: $u_1^{(0)} = 0$
1  for $\ell = 1,\dots,L$ do
2      for $k = 1,\dots,m$ do
3          $u_\ell^{(k+1)} = u_\ell^{(k)} + \frac{1}{\zeta_\ell^{(k)}}(f_\ell - A_\ell(u_\ell^{(k)}))$
4      end
5      $u_\ell = u^{(m)}$
6      for $k = m+1,\dots,n$ do
7          $z = \zeta_\ell^{(k)} \in \mathbb{C}$
8          $a = \Re(\zeta^{(k)}), b = \Im(\zeta^{(k)})$
9          $r^{(k)} = f_\ell - Au_\ell^{(k)}$
10         $u_\ell^{(k+1)} = u_\ell^{(k)} + \frac{1}{a^2+b^2}(2a - A)r^{(k)}$
11     end
12     $u_{\ell+1} = \Pi_\ell^{\ell+1} u_\ell^{(m+n)}$
13     $f_{\ell+1} = R_\ell^{\ell+1}(f_\ell - A_\ell(u_\ell)) + A_{\ell+1}(u_{\ell+1})$
14 end
```

$\max \Re(\Lambda)$ as polynomial coefficients $\alpha^{(k)} = 1/\zeta^{(k)}$. Models with polynomial building blocks with a 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 term $\tilde{q}_1$, $z = \text{argmax}(\Im(\Lambda))$ is the eigenvalue with biggest imaginary part ($\Im$), and $\bar{z}$ its complex counterpart. For polynomials of higher degrees we continue to choose roots, that are located at the border of the spectrum to satisfy the requirement of small polynomial values. The roots calculated for a polynomial with degree $d = 4$ are deployed in the resulting polynomial $q_4(A)$ followed by its evaluation on $\Lambda$. Consequently, the spectrum is a plane spanned over the roots, e.g. shown in fig. 3. The biggest eigenvalue w.r.t. the imaginary part, and its complex counterpart are chosen as roots for $q_6(A)$. This recursion of spanning the spectrum on the roots to determine new maxima can be repeated as often as required, as summarized in algorithm 3. 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 eq. (18), s.t.

$$r^{(k+1)} = (I - \alpha A^2)r^{(k)} = \hat{g}_d(A^2)r^{(k)} \tag{20}$$

has a beneficial impact on accuracy. Omitting iteration indices, a polynomial $\hat{g}(A^2)$ with coefficient $\alpha$ has roots at $\pm\frac{1}{\sqrt{\alpha}}$. Corresponding to the idea to keep the polynomial small within the spectrum we chose the coefficient $\alpha = \frac{1}{\zeta^2}$ with $\zeta_k = \max(|\max(\Re(\Lambda))|, |\min(\Re(\Lambda))|)$ to ensure that the root is at least at the border (or beyond) of the spectrum. Note that the degree of the overall polynomial $g_d$ is now $d = 2m + 2n$. Consequently eq. (18) is rewritten as

$$g_{2m+2n}(A) = \prod_{i=1}^{2m} \hat{g}_i(A^2) \cdot \prod_{j=1}^{2n} \tilde{q}_i(A), \tag{21}$$

and models that include the reinforced real part polynomial $g$ are denote by Poly-MgNet$^{g_d}$. The following discussions refer to $\hat{q}_d$, although they can be applied without restriction fo $\hat{g}_d$.

Algorithm 3: Roots $\zeta^d$ for polynomials $d \geq 6$.

```
   Initialization: $\zeta^{(1)} = \max(\Re(\Lambda))$
                   $\zeta^{(2)} = \min(\Re(\Lambda))$
                   $\zeta^{(3)} = \text{argmax}(\Im(\Lambda)), \quad \zeta^4 = \overline{\zeta^{(3)}}$
1  for $d = 6, 8, \dots$ do
2      $q_d = \prod_{k=1}^{d-2}(1 - \frac{1}{\zeta^{(k)}}\Lambda)$
3      $\zeta^{(d-1)} = \text{argmax}(|q_d(\Lambda)|)$
4      $\zeta^{(d)} = \overline{\zeta^{(d-1)}}$
5  end
```

**To** ReLU **or not to** ReLU Thus far we ignored the fact, that typically regular ResNet-blocks are build with rectified linear unit (ReLU) activation functions $\sigma = \max(0, x)$. In ResNet, as depicted in fig. 1, every convolution $A$, $B$ is followed by ReLU, which also applies for 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 $\sigma$ are given by

$$u^{(k+1)} = u^{(k)} + \sigma p_{d-1}(A)\sigma(f - Au^{(k)}) \tag{22}$$

However, our polynomial view on residual blocks collides with the fact that the max-function typically cannot be a term of a polynomial. Nevertheless ReLU has a major influence on a high expressiveness of CNNs which prompts us to review ReLU placements in our polynomial residual blocks. To peruse the actual idea of ReLU-free polynomials a single ReLU is applied on every resolution, namely before resolution coarsening level, s.t. the initial solution is given by

$$u_{\ell+1}^{(0)} = \Pi_\ell^{\ell+1}\sigma u_\ell^{(\nu)} \tag{23}$$

where $\nu = m + n$ blocks denotes a polynomial degree $d = m + 2n$. In our experiments we found, that given this setting our models are not able to explain enough non-linearity from the data resulting in a loss in accuracy. To regain expressive capacity of our polynomial models towards data non-linearity, we placed ReLU after calculating the residual

$$u^{(k+1)} = u^{(k)} + p_{d-1}(A)\sigma(f - Au^{(k)}), \tag{24}$$

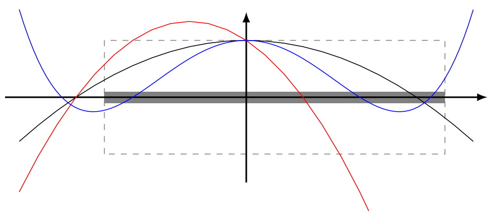


Figure 4: Schematic illustration of polynomials $q_2$, (two blocks), with different choices of roots from an exemplary (real) spectrum.

which showed to be beneficial to the models performances. Furthermore the combination of eq. (23) and eq. (24) showed to be the most beneficial for the accuracy. The introduced options are summarized in table 1. Another question of interest is the placement of batch normalization operations (bn) within the polynomials. While their role is not completely clear, we found, that different placements have influence on the classification accuracy of the corresponding model. Although bn is typically applied before ReLU, it is not required. In the following section, we analyze and discuss various MgNet$^{q_d}$ bn and ReLU configurations to identify the most effective setup.

Table 1: Overview over possible placements of ReLU functions $\sigma$ in a polynomial block $u \leftarrow u + p_{d-1}(A)r$.

| | $\sigma u_\ell^{(\nu)}$ | $\sigma p_{d-1}(A)$ | $\sigma r$ |
|---|---|---|---|
| eq. (22) | | × | × |
| eq. (23) | × | | |
| eq. (24) | | | × |
| eq. (23) + eq. (24) | × | | × |

## 4 EXPERIMENTAL RESULTS

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.) of three runs. Although Poly-MgNet is robust to different coefficient initializations, cf. table 2, we initialize the coefficients of our models based on the spectrum of the convolutions to align with our underlying intuition.

**Experimental Setup.** Our models are implemented in Pytorch (Paszke et al., 2019). We train our models in batches of 128 for 400 epochs and use an optimizer based on 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 (Loshchilov and Hutter, 2017).

Table 2: Influence of coefficient initialization for models Poly-MgNet$^{q_2}$ and Poly-MgNet$^{g_4}$. The coefficients are drawn either uniformly from the spectrum $\Lambda$. s.t. $\mathcal{U}(\lambda_{\min}, \lambda_{\max})$ or from $\mathcal{U}(-t,t)$ with $t = \sqrt{\frac{6}{c_{in}+c_{out}}}$. The latter follows the Xavier uniform distribution, which is the standard approach for initializing convolutional weights (Glorot and Bengio, 2010b).

| Dataset | polynomial | initialization | acc ($\pm$ 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) |

**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 $\alpha_\ell^{(k)} = 1/\zeta_\ell^{(k)}$, which is associated to each block. A polynomial of degree $d$ only incurs $d$ parameters and it is thus essentially for free to increase the polynomial degree w.r.t. the number of parameters.

More specifically, let $\nu$ be the number of polynomial building blocks on one resolution level $\ell$ 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 setting 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 2 residual blocks on 4 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 (He and Xu, 2019), the number of channels on the fourth layer is reduced to 256. This setting results in an overall weight count of 2.7M weights. Poly-MgNet$^{q_d}$ with the same number of resolution levels has a weight count of 1.3M.

**ReLU Placement.** In this section we examine the different options for the placement of activation functions and batch normalizations for polynomials $q_2$, corresponding to eq. (18) and $g_4$, $g_6$ and $g_8$, corresponding to eq. (21) on CIFAR-10 to determine suitable settings. In table 1 the ReLU combinations we determined as the most relevant are summarized. The impact of these combinations on the accuracy of Poly-MgNet$^{q_2}$ are compared in table 3. Recall that $q_2$ corresponds to $\hat{q}_2$ in eq. (18) and is linear with real-valued roots. We observe that placing ReLU and bn at the same places as in MgNet$^{\mathrm{A,B}}$, i.e. both $A$ and $B$ are followed by ReLU and bn, leads to a lower accuracy

Table 3: Influence of ReLU and bn placement (cf.table 1) on the accuracy of 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 eq. (22) 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. eq. (23), we contrast this to an additional bn after ReLU with bn after the residual $r$ (row 2, 3). For Poly-MgNet$^{g_4}$ we limit the shown results to the one with highest accuracy.

| Model | bn | $\sigma u_\ell^{(\nu)}$ | bn | $\sigma p_{d-1}(A)$ | bn | $\sigma r$ | accuracy test ($\pm$std) | train |
|---|---|---|---|---|---|---|---|---|
| MgNet$^{A,B}$ | | | | | | | 95.94 (0.27) | 97.60 |
| Poly-MgNet$^{q_2}$ | × | × | | | | | 65.27 (63.4) | 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 |

than MgNet$^{A,B}$. Furthermore the naive approach with a single ReLU and no additional bn (eq. (23)) (on one resolution level) is not robust, cf. table 3. One out of three experimental runs fails to learn, resulting in a low average accuracy with high standard deviation. It is well known that polynomials suffer from instabilities with increasing degree. In our experiments, we observed in general that higher polynomial degrees (greater than eight) often led to instabilities. Polynomials of moderate degree mostly achieved superior accuracy.

In contrast to that, the same ReLU placement paired with bn applied after the calculation of the residual achieves the highest mean accuracy, cf. table 3. 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 $g_4 = \hat{g}_4$ according to eq. (21). This emphasize on the real-valued part increases the accuracy by 0.4 percentage points (pp), cf. table 3.

Selected results for the ReLU placement in models with polynomials $g_6$ and $g_8$ are summarized in table 4. Recall, that polynomials with $d > 2$, and respectively $d > 4$ are composed of a real valued part and an imaginary valued part w.r.t. their roots; cf. eq. (18). 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 $g_4$ determined in table 3.

We observe that for both Poly-MgNet$^{g_6}$ and Poly-MgNet$^{g_8}$ that the application of ReLU and bn after the residual update are essential for high accuracy.

**Channel Scaling.** After determining feasible settings for the placements of ReLU and bn, we study the influence of the capacity in terms of weights on

Table 4: Influence of ReLU and bn placement in Poly-MgNet$^{g_6}$ and Poly-MgNet$^{g_8}$ on the accuracy.

| Model | bn | $\sigma u_\ell^{(\nu)}$ | bn | $\sigma p_{d-1}(A)$ | bn | $\sigma r$ | accuracy test ($\pm$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) | 97.4 |

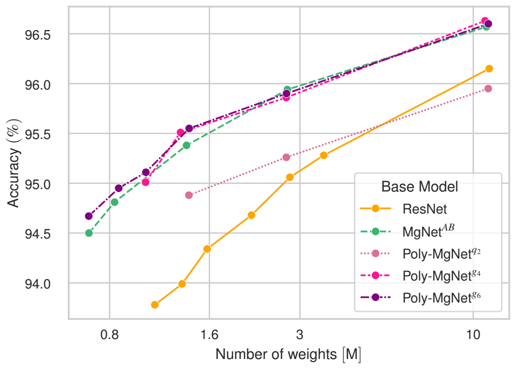


Figure 5: Accuracy-weight trade-off of ResNet and MgNet models: influence of overall capacity of residual networks on classification accuracy. The number of channels in the residual blocks of ResNet and MgNet$^{A,B}$ are scaled by $1/\sqrt{2}$ and $1/\sqrt{8}$. In contrast, the channels of Poly-MgNet$^{q_d}$ are rescaled by multiplying by $\sqrt{2}$ and $\sqrt{8}$.

the accuracy of our Poly-MgNet models. To this end, we introduce a channel scaling parameter to increase the number of weights in Poly-MgNet according to the weight count of ResNet and MgNet. Respectively, we scale the channels of ResNet and MgNet to meet the capacity of Poly-MgNet. Specifically, the overall initial number of channels[3] is multiplied by a channel scaling parameter $\lambda < 1$, to reduce the overall weight count to a target limit of 1.3M weights, which corresponds to the average weight count of polynomial MgNet models. The results are depicted in fig. 5. A regular ResNet18 has over 11M weights and achieves an accuracy over 96%. Limiting the total weight count to 1.3M – representing a reduction of more than a factor 8 – results in an accuracy drop about 2 pp. In contrast, the standard MgNet$^{A,B}$ 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$^{A,B}$ still achieves an accuracy of 96%. When the weight count of MgNet is reduced to 1.3M, we observe 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 the per-

[3]Initial number of channels in the residual blocks for ResNet18 is $[64, 128, 256, 512]$ and $[64, 128, 256, 256]$ for MgNet

formance in terms of accuracy.

Our initial Poly-MgNet utilizes around 1.3M weights. With this capacity Poly-MgNet$^{q_2}$ achieves an accuracy of 94.89%. Compared to ResNet18, which has over 11M weights, this represents a reduction in weight count by a factor of more than 8.5, while sacrificing less than 1.5 pp of accuracy. Poly-MgNet$^{g_4}$ and Poly-MgNet$^{g_6}$ achieve accuracies over 95.5% with an initial capacity of approximately 1.4M weights, slightly improving the accuracy of MgNet$^{\mathrm{A,B}}$ by 0.13 pp while requiring identical capacity. By increasing the number of channels for both MgNet$^{\mathrm{A,B}}$ and Poly-MgNet with building blocks $g_4$ and $g_6$, all models achieve an accuracy of approximately 96.60%, thereby drastically improving the accuracy-weight trade-off compared to the standard ResNet18.

## 5 CONCLUSION

In this work, we introduced Poly-MgNet, which complements MgNet by a new network building block inspired by polynomial MG smoothers. We conducted comprehensive numerical studies on our building block, including baseline comparisons and several ablations. More specifically, in comparison to MgNet we found more favorable trade-offs between the number of weights in Poly-MgNet and the achieved accuracy. In our ablation studies, we found 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 building blocks utilizing quadratic polynomials provide notable benefits in terms of accuracy. We make our code for reproduction of all experiments publicly available under https://github.com/vanbetteray/Poly-MgNet.

## ACKNOWLEDGEMENTS

This work is partially supported by the German Federal Ministry for Economic Affairs and Climate Action, within the project “KI Delta Learning” (grant no. 19A19013Q). M.R. acknowledges support by the German Federal Ministry of Education and Research within the junior research group project “UnrEAL” (grant no. 01IS22069). The contribution of K.K. is partially funded by the European Union’s HORIZON MSCA Doctoral Networks programme project AQTIVATE (grant no. 101072344).

## REFERENCES

Cha, S., Kim, T., Lee, H., and Yun, S.-Y. (2022). Supernet in neural architecture search: A taxonomic survey. *ArXiv*, abs/2204.03916.

Changpinyo, S., Sandler, M., and Zhmoginov, A. (2017). The Power of Sparsity in Convolutional Neural Networks. *ArXiv*.

Chrysos, G. G., Moschoglou, S., Bouritsas, G., Deng, J., Panagakis, Y., and Zafeiriou, S. (2022). Deep polynomial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4021–4034.

Chrysos, G. G., Moschoglou, S., Bouritsas, G., Panagakis, Y., Deng, J., and Zafeiriou, S. (2020). P-nets: Deep polynomial neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Eliasof, M., Ephrath, J., Ruthotto, L., and Treister, E. (2020). Mgic: Multigrid-in-channels neural network architectures. *SIAM J. Sci. Comput.*, 45:S307–S328.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(1):1997–2017.

Gale, T., Elsen, E., and Hooker, S. (2019). The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574.

Glorot, X. and Bengio, Y. (2010a). Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*.

Glorot, X. and Bengio, Y. (2010b). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Gordon, A., Eban, E., Nachum, O., Chen, B., Yang, T.-J., and Choi, E. (2017). Morphnet: Fast & simple resource-constrained structure learning of deep networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1586–1595.

Götz, M. and Anzt, H. (2018). Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, pages 49–56.

Han, S., Pool, J., Narang, S., Mao, H., Gong, E., Tang, S., Elsen, E., Vajda, P., Paluri, M., Tran, J., Catanzaro, B., and Dally, W. (2017). DSD: Dense-Sparse-Dense Training for Deep Neural Networks. *ICLR*.

Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both Weights and Connections for Efficient Neural Network. *NIPS*.

Hassibi, B. and Stork, D. (1992). Second order derivatives for network pruning: Optimal brain surgeon. In Hanson, S., Cowan, J., and Giles, C., editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann.

He, J. and Xu, J. (2019). MgNet: A unified framework of multigrid and convolutional neural network. *Science China Mathematics*, 62(7):1331–1354.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Santiago, Chile. IEEE.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. ISSN: 1063-6919.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, pages 630–645, Cham. Springer International Publishing.

Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., and Le, Q. (2019). Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, Seoul, Korea (South). IEEE.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv*.

Kahl, K. (2009). *Adaptive Algebraic Multigrid for Lattice QCD Computations*. PhD thesis, Wuppertal U.

Katrutsa, A., Daulbaev, T., and Oseledets, I. (2017). Deep multigrid: learning prolongation and restriction matrices. *arXiv: Numerical Analysis*.

Ke, T.-W., Maire, M., and Yu, S. X. (2017). Multigrid neural architectures. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4067–4075.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Lee, N., Ajanthan, T., and Torr, P. (2019). SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. In *International Conference on Learning Representations*.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. (2022). A convnet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11966–11976.

Loshchilov, I. and Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv:1608.03983 [cs, math].

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning convolutional neural networks for resource efficient transfer learning.

Oh, S.-K., Pedrycz, W., and Park, B.-J. (2003). Polynomial neural networks architecture: analysis and design. *Computers & Electrical Engineering*, 29(6):703–725.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks . In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, Los Alamitos, CA, USA. IEEE Computer Society.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: from theory to algorithms*. Cambridge University Press, New York, NY, USA.

Shin, Y. and Ghosh, J. (1991). The pi-sigma network: an efficient higher-order neural network for pattern classification and function approximation. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume i, pages 13–18 vol.1.

Tomasi, C. and Krause, R. (2021). Construction of grid operators for multilevel solvers: a neural network approach. *arXiv preprint arXiv:2109.05873*.

Treister, E. and Yavneh, I. (2011). On-the-Fly Adaptive Smoothed Aggregation Multigrid for Markov Chains. *SIAM J. Scientific Computing*, 33:2927–2949.

Trottenberg, U., Oosterlee, C. W., and Schüller, A. (2001). *Multigrid*. Academic Press, San Diego.

van Betteray, A., Rottmann, M., and Kahl, K. (2023). Mgiad: Multigrid in all dimensions. efficiency and robustness by weight sharing and coarsening in resolution and channel dimensions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 1292–1301.

Xie, S., Girshick, R., Dollar, P., Tu, Z., and He, K. (2017). Aggregated Residual Transformations for Deep Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, Honolulu, HI. IEEE.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856. ISSN: 2575-7075.