Multigrid Methods in Neuronal Networks

MASTER THESIS

to obtain the academic degree of
Master of Science

submitted to the Department IV - Data Science
of the University of Trier

by

Sean, Kallenberg,
Zurmaienerstraße 166
54292 Trier

Supervisor: Prof. Dr. Volker Schulz

Trier, December 8, 2020

Statutory Declaration

I hereby declare that I wrote this thesis independently and have not used any other source or tool than the ones indicated. Any thought taken directly or indirectly from external sources are marked as such. This master thesis was not presented to any other examination office in the same or similar form. It was not yet published.

December 8, 2020

(Date) (Signature)

Contents

List of Figures

1 Introduction

The task of image analysis has a steadily growing importance in everyday situations. Whether it is in form of classification, segmentation or object detection - many parts of modern applications rely on the technology and the results reachable today. This is mostly thanks due to the astounding precision the research community was able to produce using artificial neuronal networks (ANN).
An ANN aims to approximate complex or simply unknown relations of incoming data to a target space. For this task it uses blocks of linear transformations and links them together through non-linear functions, so-called activation functions. The combination of a linear block and non-linear activation is then called a neuron. A network consists of many layers consisting of differing amounts of neurons. In general each neuron in a layer will propagate its output to all neurons in the next layer. The result of this chain is then fed into a transformation that maps onto the target space. Networks that follow this structure are accordingly named feed-forward networks, and the process the input undergoes is called forward propagation. Before producing sensible results the network has to be repeatedly trained on a correctly labeled data set where in each iteration a loss metric is computed to correct the weights in the linear transformation until they are able to correctly reproduce the data-target relationship. This is then correspondingly called backpropagation.
The term neuronal network stems from the loose resemblance of the structure of a feed-forward network to a neuronal system as each artificial neuron signals many other neurons, and if their connection produces sensible results in terms of the loss metric the weights inside the neuron should adapt to strengthen this connection.
The idea to base computational models on the neuronal system of the human brain was first introduced in the 1940s by Warren McCulloch and Warren Pitts [20] and quickly further build upon by Donald Hebb [13]. But soon research reached its first bottleneck by the limited computational resources at the time as stated by Marvin Minsky and Seymour Papert [21]. It was not possible to build and train large networks that would produce results that would justify the computational effort. Thus the idea of a learned model to classify input was replaced through systems based on if-then decisions. Nonetheless Ivakhnenko and Lapa published the first functional multi-layer networks in the mid 60s [16].
Paul John Werbos introduced the backprogagation algorithm, which is used to this day, in 1975 [29]. Even though it allowed for efficient training of large neuronal networks, in most use cases they were still overtook by classifiers with a mathematical

exact solution like support vector machines or linear regressions. In this time the fields neural networks were able to dominate in were medical applications such as predicting in the structure of proteins in the 1980s [22]. It took until the new millennium and the invention of high performing GPUs to make the training of very large networks feasible and thus superhuman precision possible. With the development of stronger GPUs the interest in ANNs steadily rose. New types of networks appeared, like convolutional and recurrent neuronal networks and with them new problems to overcome. The research group around Jürgen Schmidhuber was able to steadily improve the architecture of neuronal networks for pattern recognition and machine learning, e.g. in the task of handwriting recognition [10].
Dan Ciresan and colleagues, including aforementioned Schmidhuber, were the first to achieve human-competitive performance on benchmark tasks like traffic sign recognition [5]. In 2011 the fist convolutional neuronal networks (short CNN) were published [4] and achieved superhuman precision in image classification [6].
Since then there were many architectural breakthroughs like residual and dense networks that further upped the limit to what is possible. But one connection applies to all kinds of network: the more layers and therefore more parameters a network has the higher is the possible precision achievable. This simple comes from the increasing non-linearity which allows for a better approximation and can be also be observed in other predictive models. But with the number of parameters already being extremely high, reaching into the tens of millions, it quickly becomes difficult to train those models in a timely manner. This creates the need to research network architectures that are able to maintain high precision but reduce the computational overhead.
In this work we want to study one of the possible solutions to this problem by utilizing a well known and thoroughly studied method used in mathematical optimization: Multigrid algorithms. These originally aim to solve equation systems in a efficient manner by approximating the solution of the underlying problem on a coarser grid and then use this coarse result to improve the initial guess for the search of the wanted solution. This has been transferred to the architecture of CNNs by performing multigrid cycles through the network layers. By using shared operators for grids of the same size we reduce the number of to be trained parameters by a large amount. We want to study the possibility of multigrid methods incorporated in CNNs to reduce computational overhead while keeping the precision at a competitive level in comparison to their regular counterparts. This work is structured as such: First we are going to introduce the standalone concept of multigrid methods in a mathematical context. There the core idea behind those methods as well as the most common algorithms are going to be explored and the convergence theory behind them elaborated.
Secondly we need to introduce a precise definition of ANNs and CNNs to be able to integrate multigrid methods. There we will also introduce sophisticated architectures for CNNs which we will later use to examine the results of the algorithms.
Afterwards the actual combination of multigrid and CNN is defined and illustrated before we lastly explain our implementation of the networks and analyze the results of

the chosen architectures in a direct comparison. As conclusion we aim to answer the following question:
Is the incorporation of multigrid methods in a convolutional neuronal network able to either increase performance or at least reduce computational overhead on the benchmark task of image classification?

2 Related Work

Convolutional neuronal networks have been the subject to a wide field of previous works. Ian Goodfellow et al published a comprehensive guide to the fundamentals of deep learning and CNNs [9], to which we will refer to when covering the theory of ANNs.
Since the introduction of CNNs there where many papers that proposed architectures with groundbreaking results. The most commonly cited are LeNet-5 (1998) [19], AlexNet (2012) [17], ZFNet (2013) [31], GoogleNet (2014) [27] and VGG-16 (2015) [23]. Each surpassed the previous one and improved the foundation for future works. Two notable architectural innovations of the recent years are ResNet [12] and DenseNet [14], which can be seen as successor to AlexNet and VGG-16. Many improvements proposed today are based on those networks in connection with new found regularization or data augmentation techniques, like EfficientNet [28]. Other networks follow the path of GoogleNet, like PolyNet [32] and Inception-v4 [26], which is a direct improvement of GoogleNet by the same research team. These architectures work with parallel but segregated paths in each transformation block that are concatenated between layers in contrast to ResNet that operates with a more classic feed-forward approach. DenseNet can be seen as middle-ground between Inception-like networks and ResNet-like networks as it utilizes parallel paths in each block but they share the same applied operations. A visualization of these types can be found in FigureXYZ.
Closely related to parameter reduction is the handling of overfitting and regularization. The first method that tackles this problem that should be listed is the Dropout-layer ,which works through deactivating neurons in the forward propagation with a certain probability and thus randomizing which weights are applied to the input [24].
Secondly we want to name the stochastic depth approach build directly upon the ResNet [15]. Each residual block of the network has a given probability to be replaced by a identity projection to the next block which can drastically reduce the number of layers in the model. The replacement probability can be set constant or proportional to the position of the block. This randomizes the depth of the network in each iteration. A similar method is applied in the FractalNet [18]. Each block of the FractalNet consists of multiple parallel paths which are partially concatenated throughout the block. The DropPath method works by either randomly deactivating single operations in each block or deactivating all but one path in a block. This creates a high degree of randomization and simultaneously reduces the amount of parameters by a large

factor.
As all of these approaches deactivate parts of the network they can also be seen as method for parameter reduction.

Contrary to CNNs the theory around multigrid methods is thoroughly studied and concluded. Multigrid methods where first proposed in the early 1960s by Radii Petrovich Fedorenko to solve the Poisson equation in a unit square [7, 8] and were quickly build upon by Nikolai Sergeevich Bakhvalov, who extended the applications to more general elliptic boundary value problems [1]. Wolfang Hackbusch's released work in the late 70s further studied the efficiency of multigrid methods and introduced important convergence results for the field [11]. He first proposed the full multigrid scheme (FMG). Next to Hackbusch Achi Brandt proposed the Full Approximation Scheme (FAS) for multigrid methods which extends their applications to non-linear problems [2, 3]. In the late 1990s algebraic multigrid methods were introduced which apply the multigrid techniques to problems unrelated to differential equations [25]. This approach was further generalized in the recent years and made applicable to abstract problems [30].

The combination of multigrid methods and neuronal networks was only explored in recent years and is open to further studying.
In 2015 the UNet was proposed for the task of image segmentation by Ronneberger et al. [**?**]. The UNet architecture can be compared to a multigrid V-cycle as it scales the input down to a low resolution and then back up to its original form. Additionally it adds the result of the downscaling process to the input of the same size in the upscaling through concatenation.
Regarding the task of image classification there where two separate approaches to incorporate multigrid methods. The first was proposed by Ke et al. in 2016 and implements the idea of working on different grid sizes by building three parallel network paths, each starting with a different input resolution. Before a convolutional layer is applied, the outputs of the paths are up or downscaled to the resolution of the adjacent path and then both concatenated to form the input for the convolution. Even though this architecture can produce competitive resutls to regular networks, it created massive computational overhead.
The approach which will be the focus of this work was first proposed in 2017 be Lu et al. [**?**] and further generalized by He at al. in 2019 [**?**]. The work of Lu et al. can be seen as special case of the architecture by He et al. which is why we will cover only the second work. He redefines the forward propagation through equations derived from classic multigrid methods. The Input is cycled down to a coarser resolution and then upscaled again, while in each step the current output is corrected by a residual. The computational overhead is kept small by applying the same weights for down- and upscaling in each cycle. By using residual correction the MGNet be seen as variation of the previously mentioned ResNet.

After having reviewed the most relevant literature we can continue to define the mathematical theory around multigrid methods.

3 Multigrid Methods

3.1 Fundamentals

3.2 Iterative Algorithms

3.3 Convergence Theory

4 Artificial Neuronal Networks

To define neuronal networks we will first outline the core concept and workings inside a generic network. Afterwards we are going to define commonly used layers inside convolutional networks and elaborate sophisticated network architectures. Finally the combination of multigrid methods and convolutional networks is going to be defined.

4.1 Base Concepts

4.1.1 Core Definition

Fundamentally an ANN works as approximation technique that is typically used for complex data relationships that can not be thoroughly defined. The most common example is the previously mentioned classification of images, which will later be used as benchmark task to test network proficiency.
A neuronal network can be defined as mapping $N : I \to T$ where I is the n-dimensional input space and T is the target space. Exact definitions for input and targets depend on the task at hand, e.g. a common example for image classification would be $I \in \mathbb{R}^{C\times H\times W}$ and $T := \{t_i\}_n^{i=1}$. Here $H \times W$ is the pixel resolution of the image and $C \in \{1, 3\}$ is the amount of color channels, i.e. 1 for grayscale and 3 for full color. Each t_i represents one class the input is mapped to.
In order to approximate arbitrary data mappings a neuronal network is build out of layers consisting of affine mappings and non-linear activation functions between them. A singular affine mapping can be given by the following definition.

Definition 4.1. *For input matrix $\boldsymbol{X} \in \mathbb{R}^{H\times W}$, weight vector $w \in \mathbb{R}^{W\times 1}$, bias vector $b \in \mathbb{R}^{H\times 1}$ and activation function σ a affine neuronal mapping is defined as*

$$\sigma(\boldsymbol{X}w + b) \tag{4.1}$$

For activation function the most common is the rectified linear unit, abbreviated ReLU.

Definition 4.2. *The function of the form*

$$f(x) = \max 0, x \tag{4.2}$$

is called rectified linear unit (ReLU).

As this equation partially non-differential and suffers from the vanishing gradient problem we want to introduce a modification derived from the exponential linear unit function ELU

Definition 4.3. *The function of the form*

$$f_\alpha(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^{\frac{x}{\alpha}} - 1), & x < 0 \end{cases} \tag{4.3}$$

where $\alpha \in (0, \inf)$, is called continuously differentiable exponential linear unit (CELU).

For $\alpha = 1$ it is equal to the ELU function.
In general the combination of two or more layers with activation functions is called a block. The base structure of such a network can be seen in Figure XYZ.
In depth a generic layer consists of multiple neurons which represent a mapping defined by multiple weights applied on the input. Each neuron then passes its output to each neuron of the next layer. This is then called a fully-connected layer. With the definition of a mapping above set as neuron we can detail the definition of a layer.

Definition 4.4. *For layer input $\boldsymbol{X} \in \mathbb{R}^{H \times W}$ the i-th affine layer $Y^{(i)}$ consisting of M_i neuronal mappings with weight matrix $\boldsymbol{W_i} \in \mathbb{R}^{W \times M_i}$, $M \in \boldsymbol{N}$, bias matrix $\boldsymbol{b_i} \in \mathbb{R}^{H \times M_i}$ and activation function σ is defined as*

$$Y^{(i)}(\boldsymbol{X}) := \sigma(\boldsymbol{X}\boldsymbol{W_i} + \boldsymbol{b_i}) \tag{4.4}$$

With its components in place we can now define our network as sequence of layers applied to the input:

Definition 4.5. *A artificial neuronal network with layers $Y_1, \ldots, Y_N$ and input $\boldsymbol{X}$ is defined as*

$$ANN_{Y_1,\ldots,Y_N}(\boldsymbol{X}) := (Y_N \circ Y_{N-1} \circ \ldots \circ Y_2 \circ Y_1)(\boldsymbol{X}) \tag{4.5}$$

As abbreviation a network is often defined through its forward propagation equation

$$\boldsymbol{Y}^{(j)} := \sigma(\boldsymbol{Y}^{(j-1)}\boldsymbol{W}_j + \boldsymbol{b}_j),\ j = 1, \ldots, N \tag{4.6}$$

where $\boldsymbol{Y}^0$ is the initial input $\boldsymbol{X}$.
The output of the last layer can either already be in the target space, in case of a regression for example, or is fed into a separate function to pass it onto the target space, which is commonly found in classifications. One such function would be the Softmax-function which is defined as such:

Definition 4.6. *A function of the form*

$$f(x) = (f(x)_i, \ldots, f(x)_n)^T = \frac{1}{\sum_{j=1}^{n} e^{x_j}} * e^{(\boldsymbol{x})} \tag{4.7}$$

where $\sum_{i=1}^{n} f(x)_i = 1, f(x)_i > 0$ *is called Softmax.*

where n is the output dimension and simultaneously the number of classes. The resulting vector entries are between
As this network is still only general, we have to fit it onto our task with the available data. To achieve this the network is applied on a already correctly labeled subset of the given data, called the training set. The labels can either be class labels for classifications or correct values for regressions. In each iteration we calculate a metric which indicates if the defined network approximates the task at hand properly and is used to correct the weights of the network until it produces a sensible result. To create such a metric the output of the network and the known targets are put into a so called loss function. As initial definition for arbitrary loss functions we can set the following requirements

$$1.\ S(ANN_{\ell_1,\ldots,\ell_N}(\boldsymbol{X}), \boldsymbol{T}) = \frac{1}{s}\sum_{i=1}^{s} S(ANN_{\ell_1,\ldots,\ell_N}(x_i), t_i) \tag{4.8}$$

$$2.\ S(ANN_{\ell_1,\ldots,\ell_N}(\boldsymbol{X}), T) = S(\boldsymbol{Y}^{(N)}, \boldsymbol{T}) \tag{4.9}$$

Meaning:
1. The total error equals the arithmetic mean of individual errors
2. The loss can be written as function of the final output
For a regression this could be a mean squared error metric or for classifications a cross.entropy loss. The definitions for these are as follows:

Definition 4.7. *For* $y \in \mathbb{R}^n$ *and* $t \in \mathbb{R}^n$ *the function*

$$S(\boldsymbol{Y}^{(N)}, \boldsymbol{T}) = \frac{1}{2s}\sum_{i=1}^{s} \|y_i^{(N)} - t_i\|^2 \tag{4.10}$$

is called mean square error, abbreviated MSE.

Definition 4.8. *For* $y \in \mathbb{R}^{n\times k}$ *as output of a Softmax function and* $t \in \mathbb{N}^k$ *as the respective classes the output is mapped to, the function*

$$CE(y, t) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}[\mathbf{1}_{t_j}(y_i) * \ln(y_{i,j}) + (1 - \mathbf{1}_{t_j}(y_i)))ln(1 - y_{i,j}) \tag{4.11}$$

where $\mathbf{1}_{t_j}(y_i) = \begin{cases} 1, \textit{if } y_i \textit{ belongs to class } t_j \\ 0, \textit{otherwise} \end{cases}$ *is called cross-entropy loss.*

With the definition of a loss function we can formulate a minimization problem for the approximation task.

Theorem 4.9. *Let $\boldsymbol{X} = (x_1, \ldots, x_s)^T \in \mathbb{R}^{s \times n}$ and $ANN_{y_1,\ldots,y_N}$ given. Let S be the loss function. With the previous definitions we can state the minimization problem*

$$\min_{\boldsymbol{\theta}} S(\boldsymbol{Y}^{(N)}(\boldsymbol{\theta})) + \alpha R(\boldsymbol{\theta}) \tag{4.12}$$

α and $R(\boldsymbol{\theta})$ are regularization parameters used to condition the optimization problem, e.g. $L2-$ or tikhonov regularization.

Definition 4.10. *A regularization parameter $\alpha R(\boldsymbol{\theta})$ for the optimization problem 4.9 is called $L2$ regularization, if*

$$\alpha R(\boldsymbol{\theta}) = \lambda \parallel \boldsymbol{\theta} \parallel_2^2,\ \lambda \in \mathbb{R} \tag{4.13}$$

To now shift our weights in a way that produces a lower loss metric we calculate the gradient of the loss respective to all applied weights and correct them through a optimization function. In the simplest form this can be done through batch gradient descent function which is defined here

Definition 4.11. *Das Optimierungsverfahren für das Minimierungsproblem 4.9 mit der Iterationsvorschrift*

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \frac{\eta}{k} \sum_{i=1}^{k} \nabla_{\boldsymbol{\theta}} S(\boldsymbol{Y}_i^{(N)}(\boldsymbol{\theta}_j)) \tag{4.14}$$

heißt

- *batch Gradientenverfahren, falls $k = s$*
- *stochastisches Gradientenverfahren, falls $k = 1$*
- *mini-batch Gradientenverfahren, falls $1 < k < s$*

A batch is the subset of the labeled data that is processed at once.
The process of correcting the weights is called backpropagation and is repeated multiple times on the whole available data until the resulting loss is deemed low enough or until no further reduction can be achieved. With our previous definition we can define the backpropagation algorithm as such:

Algorithm 1: Backpropagation

Data: Output $\boldsymbol{Y}^{(N)}$ und Ableitungen $\sigma'(\boldsymbol{Z}^{(j)})$
$\delta^{(N)} = \nabla S \circ \sigma'(\boldsymbol{Z}^{(N)})$
for $j = N-1, \ldots, 1$ **do**
  $\delta^{(j)} = (\delta^{(k+1)}\boldsymbol{W}^{(k+1)}) \circ \sigma'(Z^{(k)})$
for $j = 1, \ldots, N$ **do**
  for $k = 1, \ldots, m_j$ **do**
    $\frac{\partial S}{\partial w_{k,i}^{(j)}} = \delta_k^{(j)} y_i^{(j-1)}$
    $\frac{\partial S}{\partial b_{k,i}^{(j)}} = \delta_k^{(j)}$
Result: Gradienten $\frac{\partial S}{\partial w_{k,i}^{(j)}}, \frac{\partial S}{\partial b_{k,i}^{(j)}}$

The final performance of the network is tested on a not previously used set of the data, called validation set.

4.1.2 Convolutional Networks

With the now defined networks we can theoretically process all types of input. But it shows that for more complex input like images we need a more sophisticated approach than affine mappings. For this task the convolution layers established themselves as main building block and gave the commonly known convolutional neuronal network its name.
Convolution layers take in a three dimensional input consisting of C_0 $H \times W$ matrices and are defined by C_1 pairs of C_0 $k \times l$-dimensional kernels, whose entries are trained through backpropagation. C_0 and C_1 are often called the number of in- and output-channels. For each matrix in a channel of the input there exists at least one kernel that is applied on it that produces a new $n \times m$ matrix, called feature map. The entries of the feature map are calculated through the inner product of the kernel with an excerpt of the input matrix of the same size as the kernel. The inner product of two matrices is defined as follows:

The kernel is shifted across the input matrix and produces a feature map entry in each step. The number of elements the kernel is moved along the matrix in each step is called the step-size. Increasing the size of the input matrix through a border of constant entries is called padding. A visualization of this process can be found in Figure XYZ.

The resulting maps of the C_0 kernels are then summed up element wise into a $n \times m$

output matrix. This is repeated until there are C_1 channels created. Thus the final output is again three dimensional of the form $C_1 \times n \times m$ and the whole process can be defined as such:

Definition 4.12. *Die Funktion P_k definiert durch*

$$P_k : \mathbb{R}^{n\times m} \to \mathbb{R}^{(n+2k)\times(m+2k)}$$
$$P_k(A) = A^*$$

heißt padding-Funktion, wobei $A^*_{[1:2k,:]} = A^*_{[n+1:n+2k,:]} = A^*_{[:,0:2k]} = A^*_{[:,m+1:m+2k]} = 0$.

Eine padding-Funktion P_k fügt also an eine bestehende Matrix A an allen Seiten genau k Nullzeilen an

Definition 4.13. *Die Funktionen S^h, S^v definiert durch*

$$S^i_{s,t} : \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m}, i \in \{h, v\}$$
$$S^h_{s,t}(A_{[i:j,p:q]}) = A_{[i+t*s:j+t*s,p:q]}$$
$$S^v_{s,t}(A_{[i:j,p:q]}) = A_{[i:j,p+t*s:q+t*s]}$$

heißen stride-Funktionen.

Definition 4.14. *Sei der Input-Vektor $\boldsymbol{X} \in \mathbb{R}^{h\times w\times d}$ gegeben, $h, w, d \in \mathbb{N}$. Seien $F^i = (\boldsymbol{W}_{1,i}, \ldots, \boldsymbol{W}_{d,i}) \in \mathbb{R}^{n\times n\times d}$ Filter, $i = 1, \ldots, k$, mit Bias $b^i \in \mathbb{R}^{n\times n\times n}$. Weiterhin sei $s \in \mathbb{N}$ die Schrittweite zu den entsprechenden stride-Funktionen S^h, S^v und P_t eine padding-Funktion mit Parameter $t \in \mathbb{N}$, σ eine Aktivierungsfunktion die elementweise angewandt wird. Eine Faltungs-Ebene $\ell^{conv}_{k,F_i,s,\sigma}$ ist definiert als Funktion*

$$\ell^{conv}_{(n,k),s,t,\sigma} : \mathbb{R}^{h\times w\times d} \to \mathbb{R}^{(\frac{(h-n+2*t)}{s}+1)\times(\frac{(w-n+2*t)}{s}+1)\times k} =: \mathbb{R}^{h^*\times w^*\times k} \tag{4.15}$$

$$\ell^{conv}_{(n,k),s,t,\sigma}(\boldsymbol{X})_{[p,q,i]} = \sigma(\sum_{j=1}^{d}(\sum_{1\le a\le n, 1\le b\le n}(S^h_{s,p}(S^v_{s,q}(P_t(\boldsymbol{X})_{[1:n,1:n,\ j]})) \circ F^i_j)_{[a,b]}) \tag{4.16}$$

Convolutional layers are capable of processing more dimensional input better than affine layers and can lead to drastically improved results.

Next we need to introduce so called pooling layers. Instead of calculating a inner product of a kernel with the input, a pooling layer only has a kernel size parameter to define the window on which its function is applied on in each step. Contrary to convolutions they always keep the number of channels that is passed on as output constant and only change the dimensions of the channel matrices. Similar to convolutions they are further defined through a step- and padding-size parameter, but their

applied functions are fixed and do not require training.
An example would be the so called max-pooling, which takes the maximum of all numbers inside the kernel window and returns it as an output entry. A short overview of pooling functions is given here:

Definition 4.15. *Sei der Input-Vektor $\boldsymbol{X} \in \mathbb{R}^{h\times w\times d}$ gegeben, $h, w, d \in \mathbb{N}$. Sei weiter $\phi : R^{n\times n\times d}$ die pooling-Funktion der Ebene und $s \in \mathbb{N}$ die Schrittweite der stride-Funktionen S^h, S^v. Dann ist die daraus konstruierte pooling-Ebene $\ell^{pool}_{n,s}$ wie folgt definiert.*

$$\ell^{pool}_{n,s} : \mathbb{R}^{h\times w\times d} \to \mathbb{R}^{(\frac{(h-n)}{s}+1)\times(\frac{(w-n)}{s}+1)\times d}$$
$$\ell^{pool}_{n,s}(\boldsymbol{X})_{[p,q,i]} := \phi(S^h_{s,p}(S^v_{s,q}(\boldsymbol{X}_{[1:n,1:n,i]})))$$

In der Praxis werden 3 Arten von pooling-Ebenen Verwendet: max-pooling [**?**], average-pooling und L_2-pooling [9]. Der Name steht dabei für die Form von ϕ. Ein Beispiel für eine solche Ebene ist in Abbildung dargestellt.

Definition 4.16. *Sei $A = (a_{i,j}) \in \mathbb{R}^{n\times m}$. Wir nennen eine pooling-Ebene $\ell^{pool}_{n,s}$*

- *max-pooling-Ebene (kurz max-pool), falls*

$$\phi(A) = \max_{i,j} a_{i,j} \tag{4.17}$$

- *avgerage-pooling-Ebene (kurz avg-pool), falls*

$$\phi(A) = \frac{1}{n\times m}\sum_{i,j} a_{i,j} \tag{4.18}$$

- *L_2-pooling-Ebene (kurz l2-pool), falls*

$$\phi(A) = \sqrt{\sum_{i,j} a^2_{i,j}} \tag{4.19}$$

Pooling layers are often used to reduce dimensions without introducing additional parameters into the network.

4.2 Sophisticated Network Architectures

ResNet RoR DenseNet FractalNet

5 Application of Multigrid Methods in CNNs

MGNet

6 Implementation and empirical Analysis

6.1 Chosen Network Architectures

6.2 Realization in Python

6.3 Analysis

7 Conclusion

Bibliography

[1] N.S. Bakhvalov. On the convergence of a relaxation method with natural constraints on the elliptic operator. *USSR Computational Mathematics and Mathematical Physics*, 6(5):101 – 135, 1966.

[2] Achi Brandt. *Multi-Level Adaptive Technique (MLAT) for Fast Numerical Solution to Boundary Value Problems*, volume 18, pages 82–89. 02 1973.

[3] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[4] Dan Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Flexible, High Performance Convolutional Neural Networks for Image Classification.*, pages 1237–1242, 07 2011.

[5] Dan Cireșan, Ueli Meier, and Juergen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. *arXiv e-prints*, page arXiv:1202.2745, February 2012.

[6] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column deep neural networks for image classification, 2012.

[7] R.P. Fedorenko. A relaxation method for solving elliptic difference equations. *USSR Computational Mathematics and Mathematical Physics*, 1(4):1092 – 1096, 1962.

[8] R.P. Fedorenko. The speed of convergence of one iterative process. *USSR Computational Mathematics and Mathematical Physics*, 4(3):227 – 235, 1964.

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

[10] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc., 2009.

[11] Wolfgang Hackbusch. A multi-grid method applied to a boundary value problem with variable coefficients in a rectangle. 12 1977.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learn-

ing for image recognition. *CoRR*, abs/1512.03385, 2015.

[13] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, June 1949.

[14] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[15] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[16] A.G. Ivakhnenko, A.G. Ivakhnenko, V.G. Lapa, V.G. Lapa, V.G.A. LAPA, and R.N. McDonough. *Cybernetics and Forecasting Techniques*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[18] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Warren S. McCulloch and Walter Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*, page 15–27. MIT Press, Cambridge, MA, USA, 1988.

[21] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[22] B. Rost and Christian Sander. Prediction of protein secondary structure at better than 70accuracy. *Journal of molecular biology*, 232 2:584–99, 1993.

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

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[25] K. Stüben. *Algebraic Multigrid (AMG): An Introduction with Applications*. GMD-Report. GMD-Forschungszentrum Informationstechnik, 1999.

[26] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabi-

novich. Going deeper with convolutions, 2014.

[28] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[29] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975.

[30] Jinchao Xu and Ludmil Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26, 2017.

[31] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.

[32] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. *CoRR*, abs/1611.05725, 2016.