

BSc Thesis Applied Mathematics

Multi-scale convolutional
neural networks using partial
differential equations

Annemarie Jutte

Supervisor: Y. E. Boink MSc

July, 2019

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science



Preface

This thesis is written to finalize my bachelor's programme Applied Mathematics at the University of Twente. During the thesis I discovered new sides of mathematics and got to explore the relation to machine learning. This showed me new aspects of the applied side of mathematics. Writing this thesis was my first time experiencing a project of this scale. This gave me new insights into how to structure such a project.

I would like to thank my supervisor Yoeri Boink for his guidance whenever it was needed. He always made time for questions and offered interesting insights and suggestions. His help made it possible for me to finish this project in a satisfying way.

Furthermore, I would like to thank my friends for their support, helpful discussions and distractions in times of need. I want to specifically thank Femke Boelens and Wisse van der Meulen for their eagle-eyed proofreading. Finally, I want to thank my family for always supporting me.

To the reader, I hope you enjoy reading this.

Annemarie Jutte
Enschede, 2019

Multi-scale convolutional neural networks using partial differential equations

A. M. P. Jutte

July, 2019

Abstract

Convolutional neural networks (CNN) are known for their superior performance over other classical methods in numerous applications. A downside of conventional CNNs is that they are trained on a specific image size and are only meant to be used for this image size. Training a network on high resolution images is expensive. Downscaling high resolution images such that they can be used for low resolution networks results in loss on information. Multi-scale CNNs are capable of performing on different image sizes. In this paper a multi-scale method is presented based on partial differential equations. Using partial differential equations, a continuous representation can be created for neural networks of specific structures. The results presented in this paper are focused on denoising applications. However, the method presented in this paper could have applications in for example the removal of artefacts from CT scans and the removal of timestamps from images.

Keywords: residual neural networks, partial differential equations, multi-scale, convolutional neural networks, denoising, discretization

1 Introduction

The field of deep learning has been progressing rapidly over the past couple of years. The recent interest in deep learning is caused by the revolutionary increase in both parallel computation power and memory for computers in the past years [6]. Furthermore, it has been shown that deep learning gives outstanding results, which further increases the use of deep learning.

Deep learning makes use of neural networks consisting of many layers. A neural network is a system that transforms data. The data that is feeded into the system is transformed by each layer until the network gives an output. The system is trained such that the output of this network should predict a certain attribute or result. The transformation in each layer can be seen as a simple function applied to the input data of the layer. Unlike for example linear regression, neural networks also use nonlinear functions to transform the data. In each layer a nonlinearity is used as part of the transformation. This nonlinearity is generally called the activation function. Each layer applies a transformation to the result of the past layers. This way the nonlinearities are ‘stacked’ onto each other such that rather complex functions can be constructed. This is part of the power of neural networks: each layer performs a rather simple transformation, but by putting all these layers together a strong and complex system is created. Neural networks have shown to be very efficient for a lot of applications. A special kind of neural network, called a convolutional neural network (CNN), can for example be used for image denoising, image recognition and speech recognition.

Residual neural networks (ResNets) are a special kind of CNNs, which have shown to be very effective. The big advantage of using ResNets is that it is relatively easy to improve the performance of the network. Namely, improving the network can simply be done by adding extra layers, while this does not hold for CNNs in general [9]. It has been shown that ResNets can be modelled using partial differential equations (PDEs). This can for example be used to analyze the stability of networks and create stable ResNets [19]. Furthermore, it has been suggested that the modelling of some ResNets as PDEs can be used to create multi-scale neural networks [8]. By the term multi-scale neural networks it is meant that these networks that can be used to perform on different discrete image sizes. Multi-scale could also mean that it is possible to use the network with different numbers of layers.

The problem that is the focus of this research is how to create multi-scale networks. The focus is specifically on creating a multi-scale network for the denoising of images. Multi-scale networks can be very useful, training a network can be very expensive. Being able to use the trained network on images of different resolutions saves from having to train the network for multiple resolutions. Furthermore, being able to train a network using low resolution images and being able to use the network on higher resolution images reduces the training costs. It is possible to rescale an image before passing it into a (not multi-scale) network. However, when an image has to be rescaled information is lost.

For neural networks of a specific form, each layer in the network can be viewed as a step in time, and each pixel of an image going through these layers can be seen as a discrete step in space. These steps can be viewed as discretizations of a continuous process, which can be represented by a partial differential equation. Describing the network in its discrete mathematical form allows the modeling of a continuous representation. The discrete form depends on the grid size and time step. Using the continuous representation it is possible to convert the network to a finer or rougher discrete model. The multi-scale method presented in this paper is based on this idea, suggested by Haber *et al.* [8].

Rescaling can be done both in spatial and temporal dimensions. Rescaling in spatial dimensions allows a network to work on different image resolutions. The focus of the method described in this paper is on rescaling in spatial dimensions. However, to keep the rescaled network stable it is necessary to rescale in temporal dimensions as well.

First some related theory is discussed to lay the foundations of this paper. Convolutional neural networks, residual neural networks and their relation to this application are discussed. Furthermore, the link between these differential equations and residual neural networks is described. Then the method itself is discussed. Furthermore, it is explained how to make sure the network stays stable before and after the rescaling. For the implementation, different ResNet structures are considered and described. Finally, the results are presented and discussed.

2 Related Theory

2.1 Convolutional Neural Networks

In the field of neural networks convolutional neural networks are generally used when working with images. Other applications of CNNs can be found for example in language classification [13] and speech recognition [1]. CNNs make use of so called convolutional layers, possibly in combination with the so-called fully connected layers which are used in regular neural networks. Unlike regular neural networks, convolutional layers make use of so-called "tied" weights [14]. These weights are applied to the input using a convolutional

operation with a kernel filled with parameters. Suppose the kernel $K(\boldsymbol{\theta})$ is an $P \times Q$ matrix depending on parameters $\boldsymbol{\theta}$. In a convolutional layer, given $N \times M$ input matrix X_k , a convolutional transformation is applied such that the element at position (i, j) of the output matrix X_{k+1} is given by: [6]

$$X_{k+1}(i, j) = (K(\boldsymbol{\theta}) * X_k)(i, j) = \sum_m \sum_n X_k(i - m, j - n)K(\boldsymbol{\theta})(m, n).$$

As a result of the sharing of parameters, convolutional neural networks make use of the relation of a pixel in the image to the other pixels in the image. The kernel can be rather small, meaning that the number of parameters is significantly smaller than for regular neural networks. For a regular neural network the number of parameters is directly proportional to the input size, for a CNN this is not the case since the kernel does not depend on the size of the input. So the structure of the layer is independent of the input size, which allows for different input sizes to be used [6].

This is why convolutional layers are very useful for the creation of multi-scale networks. However, the problem with simply applying the same kernel to images of different resolutions is that the kernel has been trained for a specific resolution. If this kernel is applied to an image with a higher or lower resolution, the kernel will treat the image at a local scale as being of the original resolution. Because of this the kernel might have trouble performing on the new image. The purpose of multi-scale convolutional neural networks is to find a solution for this. The goal is to find a method that adjusts the weights of the kernel such that the network can be applied to different resolutions.

2.2 Residual Neural Networks

Another special type of neural networks are the residual neural networks (ResNets), a ResNet can be a CNN but does not need to be. However, for this paper only convolutional ResNets are considered. The idea of ResNets was first proposed in the article by He *et al.* [9]. In a regular CNN only the transformation of the data resulting from the previous layer is passed on to the next, residual networks also have so-called "shortcut connections". These connect the input data for one layer to another point in the network. In the network first proposed by He *et al.* this is done by passing the data through two layers and then adding the input data to the transformed data, as pictured in Figure 1. The part of the network pictured in this figure is called a residual block. It is also possible to create residual blocks of any other number of layers. Generally, residual neural networks exist of a series of residual blocks, possibly combined with other layers.

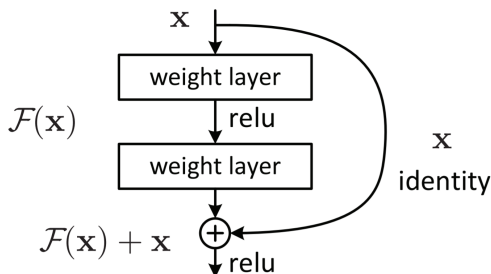


Figure 1: Block in a residual network [9].

This residual block can also be described in the form of a mathematical equation, let

F be transformation that is applied to the data, then:

$$Y_{k+1} = Y_k + F(Y_k).$$

The idea behind residual neural networks is that where ResNets generally do not improve when more layers are added, this is the case for residual neural networks. In Figure 1 the so-called ReLU function is used as activation function. This will be further discussed in Section 3.1. However, residual neural networks also work for different activation functions. The method proposed in this paper relies on the use of convolutional ResNets, as will be discussed in Section 3.

2.3 Discretization of Partial Differential Equations

For partial differential equations (PDEs) finite difference methods are widely used. Mainly because of its easy application, to first and higher degree partial differential equations, and the simplicity of the idea it is popular in use. The idea is a result of Taylor series [7]. There are multiple ways to discretize a partial differential equation using finite difference methods. The most simple choices are the forward, central and backwards finite difference methods. Suppose a uniform discretization is to be found for a two dimensional function $y(x_1, x_2)$. This discretization is to be done on a grid such that the difference between a point in the x_1 -direction x_{1j} and the next point x_{1j+1} is h . For the x_2 -direction the difference between each grid point x_{2j} and x_{2j+1} is k . This gives the grid points $\{(x_{1n}, x_{2m}) | n \in \{0, 1, 2, \dots, N\} \text{ and } m \in \{0, 1, 2, \dots, M\}\}$. Suppose at grid point (x_{1n}, x_{2m}) the function is approximated by: $Y_{n,m} \approx y(x_{1n}, x_{2m})$. A partial derivative of the first order can be approximated using a forward finite difference method by [18]:

$$y'(x) \approx \frac{Y_{n+1,m} - Y_{n,m}}{h}.$$

The forward finite difference of the first order is the same as the forward Euler discretization [3]. Higher order finite difference discretizations are used for this method as well, these can be found in appendix B.

2.4 Link between ResNets and PDE

The work by Haber *et al.* [8] on multi-scale networks shows that partial differential equations can be used to model certain residual networks as discretizations of optimal control problems. The authors show how a ResNet can be viewed as a forward Euler discretization of a PDE. However, for multi-scale solutions this article turns to methods from linear algebra instead of PDEs.

The work by Ruthotto *et al.* [19] proposes the link between ResNets and PDEs as well. In the paper it is first discussed how this can be done with respect to time, the same way as the previous article, by regarding the ResNet as a forward Euler discretization. The authors then go on to show how the weights of the kernels can be parameterized such that the convolution of the kernel and the data, passing through the kernel, can be seen as a discretization of another PDE. It is mentioned that the partial differential representation of the kernel could be used to create a multi-scale neural network, however no further information on how to do this and whether it would work is given in the paper. Furthermore, the parametrization of the kernel is only proposed for the one dimensional case. For networks that deal with images, the two dimensional case is relevant. In the paper a PDE that could be used as representation is proposed. The paper a good starting

point for this research. However, how to get to this representation or how to preserve stability is not explored. This is done in this paper.

This paper builds on the two previously mentioned articles. It will be shown how the partial differential representation of the kernel in two dimensions relates to the network. Furthermore, this partial differential approach to multi-scale networks is implemented and tested, which is not done in either article.

3 Methods

The purpose of this research is to find a way to create a multi-scale residual neural network using partial differential equations. In a neural network, each layer receives input from the previous layer, or if it is the first layer input from a dataset is given. This data is then transformed, generally by applying both linear and nonlinear transformations. The transformations rely on parameters that need to be "trained" by the network. After applying these transformations the layer passes the data on to the next layer. Special about residual neural networks is that not only the transformation is passed on from the previous layer to the next. The original data that was passed to a specific layer is also passed on to a successive layer, this data is called the residue. There can be several layers between this specific layer and the successive layer. The sequence of these layers (including the specific layer, excluding the successive layer) is called a residual block, see Section 2.2. Only residual networks with blocks that contain one convolutional layer will be considered in this work. The mathematical structure of these blocks allows for necessary manipulations to be applied as will be discussed shortly. These blocks will have the structure depicted in Figure 2. An activation function will be applied to the data transformed by the convolutional layer before the residue is added to the transformed data. The method shown in this section depends on the kernel size. The method could also be extended to different kernel sizes. However, in this paper only convolutional layers with kernels of 3 by 3 weights will be considered.

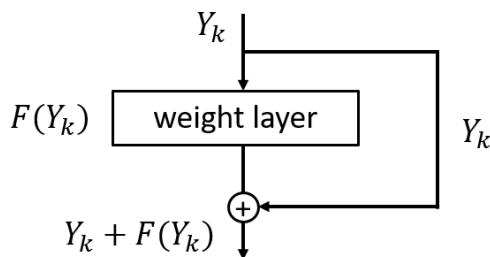


Figure 2: The residual blocks that will be considered.

A residual block of the form in Figure 2 can be represented in the following mathematical form:

$$Y_{k+1} = Y_k + F(Y_k) = Y_k + \sigma(K(\theta) * Y_k). \quad (1)$$

Here it is assumed that this neural network consists out of R residual blocks, such that $k \in \{0, 1, \dots, N\}$. Furthermore, there is a set of size s of training data $Y = [y_1, y_2, \dots, y_s]$, where $y_i \in \mathbb{R}^{\mathcal{N}_0 \times \mathcal{M}_0} \forall i \in \{1, 2, \dots, n\}$. For residual block number k , Y_k is the input and Y_{k+1} is the output data. This block can be considered as the state at time k . Here $Y_k \in \mathbb{R}^{\mathcal{N}_k \times \mathcal{M}_k}$, where $Y_0, Y_N \in \mathbb{R}^{\mathcal{N}_0 \times \mathcal{M}_0}$. This means the input and output of the network are required to have the same size, but this is not required for the layers in between.

Furthermore, $F(\boldsymbol{\theta})$ is the combination of linear and nonlinear transformations applied to Y_k , where $\boldsymbol{\theta}$ are the weights of the convolutional layer in the residual block. Within the weight layer a convolution is applied to the input with kernel $K(\boldsymbol{\theta}) \in \mathbb{R}^{3 \times 3}$, here $\boldsymbol{\theta} \in \mathbb{R}^9$ are the parameters that the filter depends on. Finally, σ is the activation function used.

As described in article [8], certain ResNets can be described using discretizations of partial differential equations. This can be deduced from the mathematical representation of a residual block. If the terms of equation (1) are rearranged and both sides are divided by some time step $\Delta t > 0$, the following is found:

$$\frac{Y_{k+1} - Y_k}{\Delta t} = \frac{1}{\Delta t} \sigma(K(\boldsymbol{\theta}) * Y_k). \quad (2)$$

The left side can be seen as the forward Euler discretization of the derivative of some function y with respect to t , see Section 2.3.

For reasons that will come apparent later it is necessary to be able to move the $\frac{1}{\Delta t}$ from outside the activation function to inside the activation function σ . This implies the function chosen as activation function should allow this.

3.1 Activation Functions

Different activation functions σ can be considered. Ideally the activation function will have some resemblance to a linear function with respect to multiplication, such that either:

$$\frac{1}{\Delta t} \sigma(x) = \sigma\left(\frac{1}{\Delta t} x\right) \quad \text{or} \quad \frac{1}{\Delta t} \sigma(x) \approx \sigma\left(\frac{1}{\Delta t} x\right). \quad (3)$$

Here it is important to note that the time step satisfies $\Delta t \geq 0$. So the equality or approximation only needs to hold for $\Delta t \geq 0$. A function that satisfies the case in which both sides are equal is called a positively homogeneous function of the first degree [5]. So the activation function should either be a positively homogeneous function of degree one or it should approximate one. Three activation functions that satisfy either of the conditions will be discussed: the hyperbolic tangent, the ReLU and the Leaky ReLU.

3.1.1 The Hyperbolic Tangent

The hyperbolic tangent is an activation function used in practice [12], it is given by:

$$\sigma(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)}.$$

The property of the hyperbolic tangent that could be useful for the special activation function required is that the hyperbolic tangent approximates a linear function around zero. This makes it a possible candidate, see equation (3). The downside of using this activation would be that, if it were to be used, the nonlinearity would be approximated by a linearity when using equation (3). However, if the activation function is treated as a linear function a very important aspect of the neural network is ignored. The idea of the activation function is to introduce a nonlinearity to the process. Furthermore, for small numbers the hyperbolic tangent indeed acts closely to a linear function. However, as the numbers increase the hyperbolic tangent soon starts to converge from the linear approximation. So the hyperbolic tangent would be a possible candidate, but it might be valuable to find another activation function that would not require to be linearized.

3.1.2 The ReLU Function

The ReLU function is an activation function of the following form:

$$\sigma(x) = \max(0, x),$$

[20]. The ReLU function is a homogeneous function of degree 1. When x is less than 0 the function acts as the linear function $f(x) = 0$, when x is bigger than 0 it acts like the linear function $f(x) = x$. This implies that the following holds for some constant $c \geq 0$, as it should for a homogeneous function of degree 1:

$$c\sigma(x) = \max(0, cx).$$

This means this function could be considered as activation function for the neural network. It is a nonlinear function that seems to serve the purposes and does not need to be linearized in order to use equation (3). However, when looking at the ResNet where the blocks are of the form:

$$Y_{k+1} = Y_k + \sigma(K(\theta) * Y_k),$$

it becomes apparent that for each layer a positive quantity (or zero) is added to the previous layer. This means that the network can only steer in positive direction. Once a layer transforms the data to a too high quantity, there is no way to restore the damage. A ResNet of, for example, the form:

$$Y_{k+1} = \sigma(Y_k + K(\theta) * Y_k),$$

does not have this problem. This is because the convolution, that can have a negative result is directly added to the old data. However, this gives the problem that the Y_k term cannot simply be brought to the other side. This gives difficulties to model this form as a differential equation, see equation (2).

3.1.3 The Leaky ReLU Function

The Leaky ReLU is an activation function based on the ReLU function previously discussed. It is of the form:

$$\sigma(x) = \begin{cases} x & \text{for } x \geq 0, \\ \alpha x & \text{for } x < 0. \end{cases}$$

Here, α is a parameter that determines the ‘size of the leak’. It determines the weight of the output given negative input. The Leaky ReLU function was first introduced in the article by Maas *et al.* [16]. It was introduced as solution for the problem that the ReLU unit becomes inactive if the output of one weight layer is constantly negative. This is a problem for gradient based optimization functions: when the gradient is zero the parameters are not updated. This way the ReLU unit remains negative and parts of the network could become inactive. The Leaky ReLU function solves this by still giving some output if the input is negative. In the article by Maas *et al.* [16] the Leaky ReLU was introduced with $\alpha = 0.01$. However, other values of α can be used as well [20].

As discussed in Section 3.1.2, for the specific residual block structure used it is necessary that the activation function can have negative values as output. For the Leaky ReLU this is indeed the case.

The Leaky ReLU function is a positively homogeneous function of degree 1, since the following holds for all $c \geq 0$:

$$c\sigma(x) = \sigma(cx) = \begin{cases} cx & \text{for } x \geq 0, \\ c\alpha x & \text{for } x < 0. \end{cases}$$

3.2 Rescaling

From now on the activation function $\sigma(x)$ will be assumed to be a LeakyReLU function with parameter α . Furthermore, it will be assumed that each block k ($k \in \{1, 2, \dots, N\}$) in a network of N layers is of the form:

$$Y_{k+1} = Y_k + \sigma(K(\boldsymbol{\theta}) * Y_k).$$

Since the activation function is homogeneous it is possible to rewrite the previous formula and incorporate the time step Δt :

$$\frac{Y_{k+1} - Y_k}{\Delta t} = \sigma\left(\frac{1}{\Delta t} K(\boldsymbol{\theta}) * Y_k\right). \quad (4)$$

In this form it is clear that the left side of the equation is in the form of a forward Euler discretization, see Section 2.3. So suppose the Y_k matrix is considered as a discretization of some continuous function $y(x_1, x_2, t)$ at time t_k . Let t_k be defined such that $t_k = t_{k-1} + \Delta t$ with $k \in \{1, \dots, K\}$. Furthermore, time $t_0 = 0$ and time $t_K = T$. For the spatial coordinates, let $x_{1n} = x_{1n-1} + h$ where $n \in \{0, \dots, N_k\}$, such that $x_{10} = 0$ and $x_{1N_k} = X_1$. Let $x_{2m} = x_{2m-1} + k$ where $m \in \{0, \dots, M_k\}$, such that $x_{20} = 0$ and $x_{2M_k} = X_2$. Then, each element at position (m, n) within the Y_k matrix represents the continuous function $y(x_1, x_2, t)$ at position (x_{1n}, x_{2m}) . The element within matrix Y_j at time t_j at position (x_{1n}, x_{2m}) will be denoted as $Y_{n,m}^j$. As defined earlier this section $K(\boldsymbol{\theta}) \in \mathbb{R}^{3 \times 3}$, where $\boldsymbol{\theta} \in \mathbb{R}^9$. Now suppose:

$$K(\boldsymbol{\theta}) = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \\ \theta_7 & \theta_8 & \theta_9 \end{bmatrix}.$$

Since Y_k is a $M \times N$ matrix the convolution in equation (4) results in a $M \times N$ matrix. Generally the size of the resulting matrix depends on edge handling, however it will be assumed a method is chosen that keeps the matrix sizes equal. Because of the edge handling, at the edges the resulting pixels will not be the result of the convolution with $K(\boldsymbol{\theta})$, but will depend on the method used. For more on edge handling see [10]. The method of edge handling influences only what happens at the outer most pixels because of the kernel size. It will be assumed that the influence of the chosen method is negligible.

Assume $c_{n,m}$ is an element in the matrix resulting from the convolution, in equation (4), at position (n, m) . Then the following holds for this element:

$$c_{n,m} = \frac{1}{\Delta t} (\theta_1 Y_{n-1,m+1} + \theta_2 Y_{n,m+1} + \theta_3 Y_{n+1,m+1} + \theta_4 Y_{n-1,m} + \theta_5 Y_{n,m} + \theta_6 Y_{n+1,m} + \theta_7 Y_{n-1,m-1} + \theta_8 Y_{n,m-1} + \theta_9 Y_{n+1,m-1}). \quad (5)$$

Now a transformation for the parameters $\boldsymbol{\theta}$ will be proposed which will allow for a link between the discrete equation given and a continuous representation, which is yet to be found, to exist. This transformation is based on the idea proposed in the article by Ruthotto and Haber [19]. For this transformation nine unknown variables β_i ($i = 1, 2, \dots, 9$) are introduced. To be able to convert from and to these new parameters at least nine equations are necessary. The transformation matrix needs to be nonsingular such that it is possible to convert from $\boldsymbol{\theta}$ to $\boldsymbol{\beta}$ and back again, using the matrix itself and its inverse.

The following transformation is proposed:

$$\Delta t \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{h^2 k} & \frac{1}{h^2 k^2} \\ 0 & 0 & \frac{1}{k} & 0 & \frac{1}{k^2} & \frac{-1}{hk} & \frac{-1}{hk^2} & \frac{-1}{h^2 k} & \frac{-1}{h^2 k^2} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{hk} & \frac{1}{hk^2} & \frac{1}{h^2 k} & \frac{1}{h^2 k^2} \\ 0 & 0 & 0 & \frac{1}{h^2} & 0 & 0 & 0 & \frac{-1}{h^2 k} & \frac{-1}{h^2 k^2} \\ 1 & \frac{-1}{h} & \frac{-1}{k} & \frac{-1}{h^2} & \frac{-2}{k^2} & \frac{1}{hk} & \frac{2}{hk^2} & \frac{2}{h^2 k} & \frac{2}{h^2 k^2} \\ 0 & \frac{1}{h} & 0 & \frac{1}{h^2} & 0 & \frac{-1}{hk} & \frac{-2}{hk^2} & \frac{-1}{h^2 k} & \frac{-1}{h^2 k^2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{h^2 k^2} \\ 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & \frac{-1}{hk^2} & 0 & \frac{-2}{h^2 k^2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{hk^2} & 0 & \frac{1}{h^2 k^2} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \\ \beta_9 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \\ \theta_8 \\ \theta_9 \end{bmatrix}. \quad (6)$$

The transformation is based on finite difference discretizations which can be found in appendix B. This transformation matrix has determinant $\frac{-1}{h^9 k^9}$, making it indeed nonsingular, since the grid sizes h and k are never 0. Suppose the transformation matrix is denoted as $M(h, k)$, where h and k are the grid sizes, the vector of β_i is denoted as $\boldsymbol{\beta}$ and the vector of θ_i is denoted as $\boldsymbol{\theta}$. Then the transformation can be written as:

$$\Delta t M(h, k) \boldsymbol{\beta} = \boldsymbol{\theta}. \quad (7)$$

It can be shown that applying this transformation to equation (5) results in a sum of finite difference discretizations. From these discretizations a continuous form can be deduced, see appendix C. This form is given by:

$$\begin{aligned} k(\boldsymbol{\theta}, y) = & \beta_1 y + \beta_2 \frac{\partial y}{\partial x_1} + \beta_3 \frac{\partial y}{\partial x_2} + \beta_4 \frac{\partial y^2}{\partial x_1^2} + \beta_5 \frac{\partial y^2}{\partial x_2^2} \\ & + \beta_6 \frac{\partial y^2}{\partial x_1 \partial x_2} + \beta_7 \frac{\partial y^3}{\partial x_1 \partial x_2^2} + \beta_8 \frac{\partial y^3}{\partial x_1^2 \partial x_2} + \beta_9 \frac{\partial y^4}{\partial x_1^2 \partial x_2^2}. \end{aligned} \quad (8)$$

Thus, what has been discovered is that given the discrete form:

$$Y_{k+1} = Y_k + \sigma(K(\boldsymbol{\theta}) * Y_k),$$

with gridsize h in x -direction, gridsize k in y -direction and time steps of size Δt , a transformation can be applied to the weights of the kernel. Namely the reverse of the previously defined transformation in equation (7). This uses the inverse of matrix $M(h, k)$, this inverse matrix can be found in appendix A. Using this, the reverse transformation is given by:

$$\boldsymbol{\beta} = \frac{1}{\Delta t} M(h, k)^{-1} \boldsymbol{\theta}. \quad (9)$$

Now it is possible to convert the parameters for the continuous form to the discrete parameters again, however it is not necessarily required that this discrete form has the same gridsizes and time step. This allows for a new discretization which can be used to create a multiscale network. Suppose previously the network worked for images of size $N \times M$ such that $h = \frac{1}{N}$ and $k = \frac{1}{M}$. Now it is desired to use the network on images of size $\tilde{N} \times \tilde{M}$, such that the new gridsizes are given by $\tilde{h} = \frac{1}{\tilde{N}}$ and $\tilde{k} = \frac{1}{\tilde{M}}$. Then the new weights for the discrete form $\tilde{\boldsymbol{\theta}}$ can be found by applying the transformation given by equation (7), such that:

$$\tilde{\boldsymbol{\theta}} = \Delta t M(\tilde{h}, \tilde{k}) \boldsymbol{\beta}.$$

Another possibility that arises from the previous steps is to rescale the time step. Previously the time step was $\frac{1}{\Delta t}$, now suppose a time step of $\frac{1}{\tilde{\Delta t}}$ is desired. As will be shown in Section 3.3 it is sometimes necessary to change the time step to preserve stability of the network. The continuous parameters β can be found as before. The new weights $\tilde{\theta}$ can be obtained by:

$$\tilde{\theta} = \Delta \tilde{t} M(\tilde{h}, \tilde{k}) \beta \quad (10)$$

Here it is important to note that when changing the time step the number of layers of the network also needs to be adjusted. Suppose the time step is divided by two, then the number of layers needs to be doubled. Combining equations (9) and (10) gives the following equation for the new weights:

$$\tilde{\theta} = \frac{\Delta \tilde{t}}{\Delta t} M(\tilde{h}, \tilde{k}) M^{-1}(h, k) \theta.$$

3.3 Stability

To have any certainty of the behaviour of the conversion of the network using the previously discussed method stability needs to be ensured. The method that will be used to ensure this is based on the method used in the book by Morton and Mayers [18]. Suppose the truncation error is given by $T_{n,m}^j$. The error of the discretization is given by the difference between the discretization and the actual values:

$$e_{n,m}^j := Y_{n,m}^j - y(x_{1n}, x_{2m}, t_j).$$

Furthermore, suppose the maximum value at time t_j over all positions x_{1n} and x_{2m} of the absolute error is denoted by E^j , so:

$$E^j = \max \{ |e_{n,m}^j| : n \in \{0, 1, \dots, N\}, m \in \{0, 1, \dots, M\} \}.$$

Then it can be shown that if the following conditions are satisfied:

Conditions 1

1. $\theta_i \geq 0$ and $\theta_i^n \geq 0$, $i \neq 5$,
2. $\sum_{i=1, i \neq 5}^9 \theta_i = \sum_{i=1, i \neq 5}^9 \theta_i^n = 0$,
3. $\Delta t < \frac{1}{\theta_5}$.

The following upper bound holds for the error at time t_{j+1} :

$$E^{j+1} \leq E^j + \Delta t |T_{n,m}^j|.$$

From this bound it is clear that the increase in error for each step in time can only increase by the truncation error by a factor of the time step. This means that the increase in error is bounded, given that the truncation error is bounded. Furthermore, this means that when following the conditions the error can be reduced by choosing smaller time steps. It can be assumed that the error at time $t_0 = 0$ is $E^0 = 0$. This can be assumed since the continuous representation of the discrete network that is being dealt with is simply a representation, an abstract concept. So it is assumed that the continuous representation is such that its initial condition is exactly satisfied by the discrete network. If this is the

case then indeed $E^0 = 0$. Now it can be shown using a proof by induction, as stated by [18], that the following holds for the error at time $t_n = n$:

$$E^n \leq n|T_{n,m}^j|\Delta t.$$

Since $|T_{n,m}^j|$ is bounded, see appendix E, $E^n \rightarrow 0$ as $t \rightarrow 0$. This means the discretization is stable and furthermore it means that the discretization is consistent. For a detailed derivation for the bound, see appendix D.

3.4 Regularization

In the neural network the parameters are trained using data. However, to be able to train the neural network there has to be an objective to train for. This objective is the minimization of the loss function $V(\boldsymbol{\theta})$ depending on the parameters of the network. During training, the parameters are optimized by minimizing the loss function. An example, and the loss function that will be used for this paper, is the mean squared error. Suppose the output of the network is the $N \times M$ matrix Y_N and the ground truth is the $N \times M$ matrix \hat{Y}_N . Let $y_{i,j}$ be the element at position (n, m) of Y_N and let $\hat{y}_{n,m}$ be the element at position (n, m) of \hat{Y}_N . Then the mean squared error loss function is given by:

$$V(\boldsymbol{\theta}) = \frac{1}{NM} \sum_{n=1}^N \sum_{m=1}^M (y_{n,m} - \hat{y}_{n,m})^2.$$

Now to ensure *Conditions 1* a regularization term is introduced which will be added to the loss function, such that the new loss function $L(\boldsymbol{\theta})$ is a sum of the old loss function $V(\boldsymbol{\theta})$ and a weighted regularization function $R(\boldsymbol{\theta})$:

$$L(\boldsymbol{\theta}) := V(\boldsymbol{\theta}) + \alpha R(\boldsymbol{\theta}),$$

where the parameter α determines the weight of the regularization. The regularization term is introduced such that, given desired conditions, penalties are incurred when these conditions are not satisfied. This way when minimizing the loss function including the regularization term it is valuable to minimize the regularization term as well, which is done by satisfying the conditions. The parameter α is introduced which determines the weight of the regularization term. The higher this parameter, the more valuable it is to minimize the regularization term. However, a rather high weight can result in more difficulty optimizing the weights for the objective of minimizing the loss function without regularization term. Before defining the regularization function $R(\boldsymbol{\theta})$, two functions will be defined that will be used to induce the penalties for not satisfying the conditions. The first function $\phi(x, n)$ is given by:

$$\phi(x, n) := \begin{cases} 0, & \text{for } x \geq n, \\ (x - n)^2, & \text{for } x < n. \end{cases} \quad (11)$$

Introducing this function (11) should result in penalties being given if the specified term does not satisfy an inequality condition. The other function that is to be defined is the function $\gamma(x, n)$, given by:

$$\gamma(x, n) := \begin{cases} 0, & \text{for } x = n, \\ (x - n)^2, & \text{for } x \neq n. \end{cases}$$

The penalty function γ induces penalties if the given term does not satisfy an equality condition, where the given term should equal n . Using these functions a regularization function can be introduced that induces penalties if the given conditions are not satisfied. This function will be defined as:

$$R(\boldsymbol{\theta}) := \sum_{i=1, i \neq 5}^9 \phi(\theta_i, 0) + \sum_{i=1, i \neq 5}^9 \phi(\tilde{\theta}_i, 0) + \gamma \left(\sum_{i=1}^n \theta_i, 0 \right).$$

4 Implementation

For the implementation the Python extension Keras is used on top of the TensorFlow extension. The training is done on the MNIST dataset [15], containing 60000 training and 10000 testing images of 28 by 28 pixels. The images are of handwritten letters. The pixels of the original images are on a 0 to 255 scale, the images that are used are on a 0 to 1 scale. The implementation will be a denoising ResNet. Before the network is trained noise is added to the images from the MNIST image set. The original images will be used as the ground truth. So the network will learn to transform the noisy images such that the transformed image approximates the ground truth, non-noisy, images.

The implementation satisfies the residual block structure given by equation (1). So for each block the input is stored, then transformed using a 3×3 convolution, before being put through a LeakyReLU activation. This transformed input is then added to the original input and passed to the next residual block.

The Adam optimization algorithm is used for the training of the network with a learning rate of 0.001. The loss function used is described in Section 3.4. The weights are initialized using a normal distribution with mean 0 and standard deviation 0.05. The LeakyReLU parameter is set to 0.3. The regularization parameter is set to $1 \cdot 10^3$. Training will be done in batches of 16 over 10 epochs.

4.1 Image preparation

The network is trained using the 28 by 28 pixel images from the MNIST dataset. Noise is added to these images such that they can be used to train with. The original images, taken directly from the dataset, will be used as the ground truth images. The noise is added to the images following a Gaussian distribution with mean 0 and standard deviation 0.25. As an example the ground truth for a particular entry from the data set is given by Figure 3a, adding noise to this image results in Figure 3b, for instance.

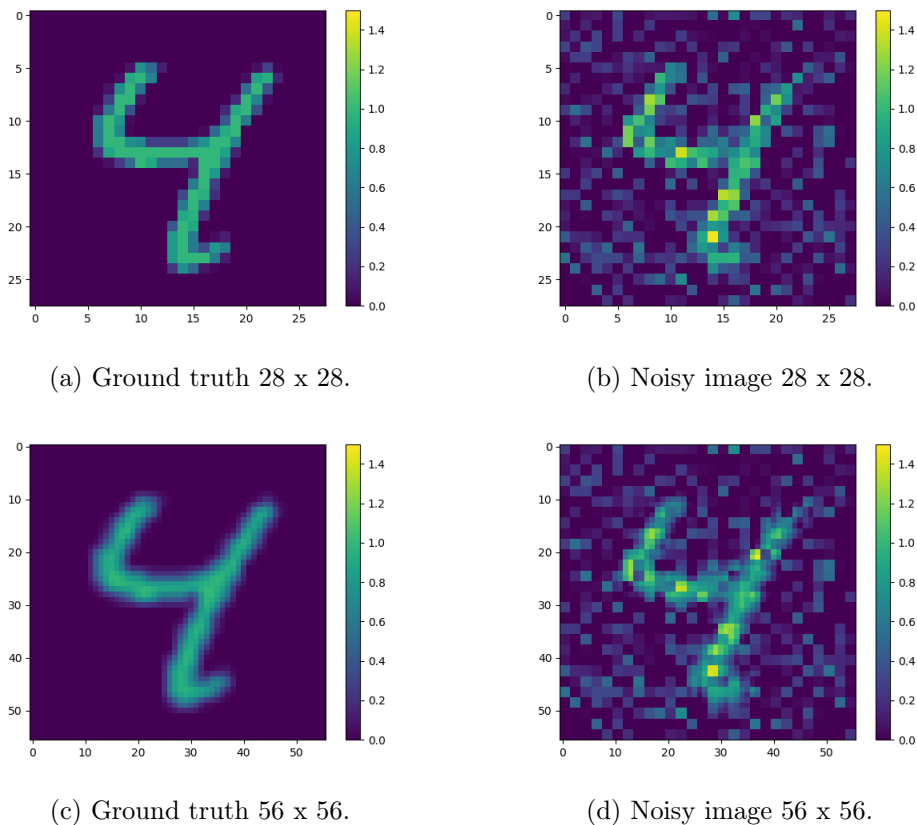


Figure 3: Sample from the dataset.

The images used for testing the multi-scale method are the images from the MNIST dataset, but rescaled. The rescaling is done using a bilinear interpolation method. The rescaled image serves as ground truth. In Figure 3c the rescaled image of a specific instance can be found. Noise is added to this image to obtain the multi-scale test image. This noise is on a 28 by 28 grid, it is then rescaled using a nearest neighbour interpolation method to match the rescaled image. Using the nearest neighbour interpolation method the grid of the noise virtually stays the same, but it allows for the noise to be added to the image. The 56 by 56 image after adding the noise can be found in Figure 3d.

The reason the noise is added on a 28 by 28 grid instead of the finer grid of the image has to do with the frequencies of the noise. From a continuous perspective, noise contains all frequencies. So in the continuous domain when noise is added this happens at an infinite number of frequencies. However, when discretizing the frequencies that remain present under this transformation decreases such that only the frequencies that lie within the gridsize are still present. When a network has been trained on a rough grid the network has been trained to only reconstruct the image for the frequencies detected by this grid size. If the network is rescaled, it is still only capable of detecting the frequencies of the rough scale.

4.2 Time rescaling

To preserve the stability of the network after rescaling it is necessary to rescale in the time direction. When this happens the number of layers needs to be changed. To do this the number of layers required is first determined. The size of the new time step is chosen such

that the stability is ensured for all kernels. For each kernel it is checked which time step is necessary and the smallest is chosen, see *Conditions 1*. It is then determined how many layers are necessary to ensure this time step. This number is rounded up to an integer, since it is only possible to repeat the layers a discrete number of times. The new time step is finally chosen such that it matches the increase in layers. New blocks are added between the existing (rescaled) blocks, such that they are evenly spread. If the time step is halved, a new block is added in between each existing block. The weights for these new blocks are linearly interpolated with respect to time.

4.3 Network structures

Networks of different structures are considered and tested.

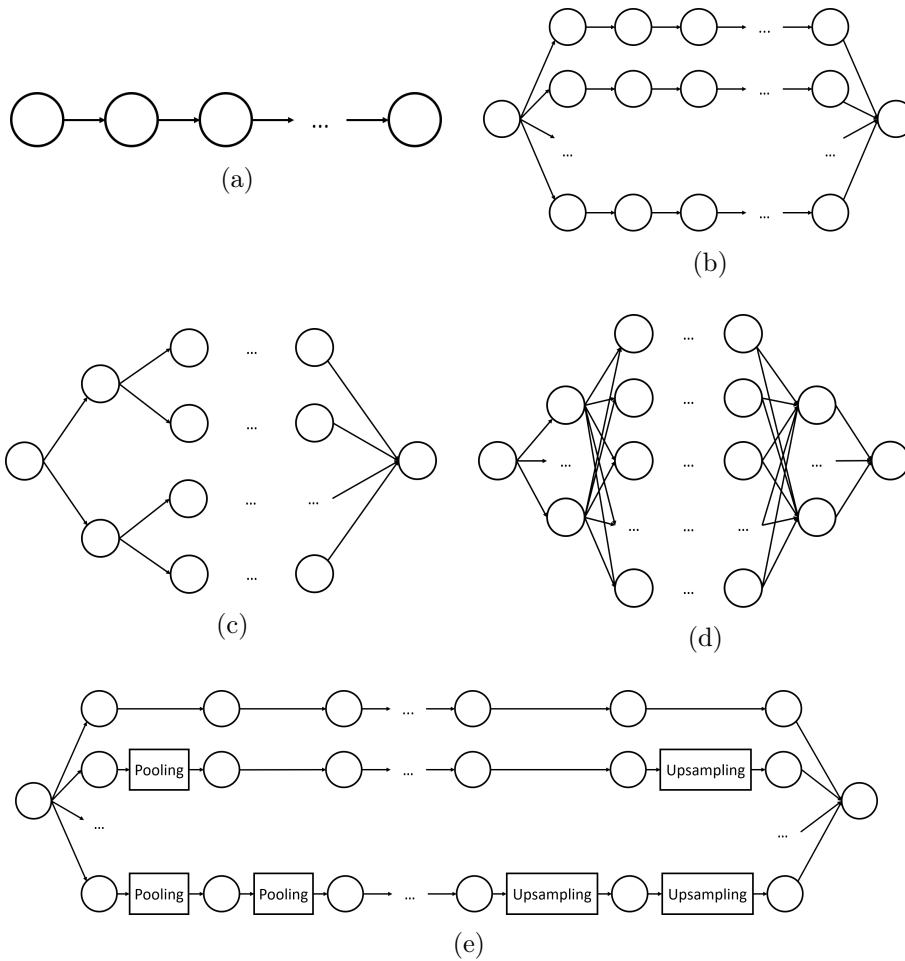


Figure 4: (a) Network with width one, (b) network with parallel branches, (c) network with binary branches, (d) network with connected layers, (e) network with pooling layers.

4.3.1 Network with layer width one

A network with layer width one is considered, see Figure 4a. This network consists of a series of blocks. When talking about blocks, blocks of the shape as discussed in Section 3 and depicted in Figure 2 are meant. This means each block contains one convolutional layer with one kernel. If it is assumed that the weights of each kernel are the same for every

block the stability proof holds, see Section 3.3. It is assumed that stability is preserved when the weights are not necessarily the same for each layer. Rescaling with respect to time is done as described in Section 4.2.

The advantage of this network is that it closely relates to the theory. Except for the nonconstant weights, the network is of the same shape as was theoretically analyzed. The downside is that it is a relatively simple network. A more complicated network could give better results, since it is capable of more complicated transformations. CNNs that consist of layers of more channels generally yield better results.

A version of this network with eight layers will be the main focus of the results.

4.3.2 Network with parallel branches

Another network to consider is one with parallel branches. It closely relates to the network with layer width one. It repeats this network on a number of parallel branches. A network of this form is depicted in Figure 4b. The start node is responsible for dividing the image by the number of branches. At the end node the results from each branch are added up. The division of the image at the start node is essential because the conditions on the kernel, see Section 3.3, restrict the kernel such that on average the image stays the same. If the image is not first divided while later the results of each branch are added, the values of the resulting image will always be too high. Rescaling can be done the same as for the network with layer width one, one can simply consider each branch as a network with layer width one and rescale this.

The advantage of a network of this is that it offers a wider network, while still maintaining simplicity. It offers the option for the network to transform the image differently along each path. While each path itself can still be considered as a simple network with a width of one. A network of this form is implemented with five branches, each containing six blocks.

4.3.3 Network with binary branches

A network that branches out like a binary tree can be considered. Each node, except for the last two layers, in this network passes output to two connected nodes, see Figure 4c. Each path from the starting node to a node in the one to final layer can separately be considered as a network of width one. The nodes are residual blocks of which the results are divided by two, for the same reason as discussed for the network with parallel branches, since in the last layer all the nodes of the previous layer are added again. Dividing does not interfere with the relation to PDEs, since this operation can be seen as part of the kernel.

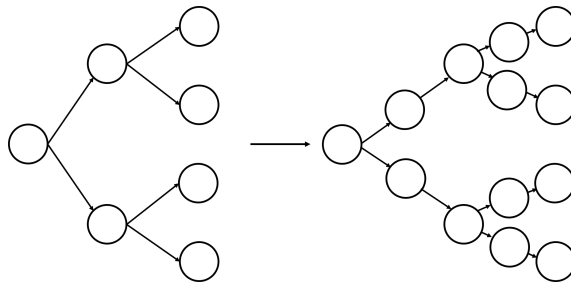


Figure 5: Adding additional layers for binary structure.

For rescaling with respect to time more layers need to be added, this is done by inserting

new residual blocks between the existing layers, as described in Section 4.2. This does make the network take on a slightly different form, see Figure 5. Just like the previous network with the parallel branches this network keeps the simplicity while still having the possibility to develop itself differently over the different paths of the network. A network of this form is implemented with five layers.

4.3.4 Network with connected nodes

Generally convolutional neural networks are of more complex structures than the previously discussed networks. Using a CNN that is generally used one would increase the number of channels in between some layers. These channels can be seen as extra nodes within the layer, see figure 4d, so extra depth is added. Throughout a network like this the number of channels is first increased and at the end decreased such that the process finishes with one channel. By increasing the number of channels the network is able to use a variety of filters on the image.

A problem with a network like this for the multi-scale network is the issue of stability. The method used to prove the stability of the method is not easily extended to a network like this. Furthermore, when changing the time step, thus increasing the number of layers, it is still uncertain how the new layers should behave. This is why it is decided not to use a network like this.

4.4 Network with pooling layers

Finally, a network with pooling layers is considered. A pooling layer decreases the image size. The output of the pooling layer is an image in which each pixel is a summary of a region in the previous layer [6]. This means an operation is applied to a region of $p_1 \times p_2$ pixels and in the output this region is represented by a single pixel q . An example of a pooling method, which will be used for this network, is average pooling. Suppose the image can be divided into R regions each consisting of P pixels, such that some region r is given by the pixels $\{p_1^r, p_2^r, \dots, p_P^r\}$. Then, after applying average pooling to this region r , the output returns a scalar s , such that:

$$s = \frac{1}{P} \sum_{i=0}^P p_i^r.$$

Generally, the regions r are either 2×2 or 3×3 pixels. So for example, if a pooling of size 2×2 is used, the output of the pooling layer is half the size of the input. Here the 2×2 regions are transformed into a scalar, so an 1×1 region. Here the scalar has the average value of the 2×2 region. For an example, see Figure 6 [17]. Later in the network the image can be upsampled, to restore the image to the original image size. This can for example be done using a nearest neighbour interpolation, which simply repeats the existing pixels.

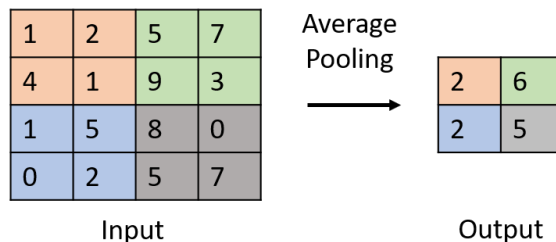


Figure 6: Example of a pooling layer.

One clear advantage of pooling is that it reduces the computation costs of training the network, since the image resolution is reduced. Another important advantage is that it allows for higher order features to be detected [4]. Finally, pooling also helps to make the network invariant to small translations within the input [6].

By first downsampling using the pooling layer and later upsampling information is lost. To account for this it is possible to create a branch within the network in which the image resolution does not change. Combining this branch with branches that contain pooling layers solves this problem.

An example of the structure of a network with pooling layers can be found in Figure 4e. In this network 2×2 average pooling layers are used. The structure of this network consists out of three branches. The first branch is a branch of the form of the network of width 1. The second branch uses one pooling layer, later followed by one upsampling layer. The third branch uses two pooling layers, later followed by two upsampling layers. The image is again divided by the number of branches, as it was for the parallel and binary networks. At the final node the results of the branches are added.

For the rescaling with respect to time, the pooling and upsampling layers are ignored and the network is rescaled the same way as the network with parallel tracks. It is assumed that the pooling and upsampling have no effect on the rescaling or the relation between the kernels.

5 Results

First the results of the network with a width of one will be presented. After this the results of the other implemented networks will be presented as well. For the exact implementation of the networks a request can be made at the author.

5.1 Network with layer width one

The network with a width of one, described in Section 4.3.1, is implemented. Training this network with 8 different layers for 10 epochs, resulted in a training loss of 0.0252 and a validation loss of 0.0249. The trained kernels can be found in appendix F. Rescaling the network for this resolution requires the time step to be halved. The rescaled network is tested on 1000 test samples.

Table 1: Mean squared error old, new and benchmark network.

MSE noisy images (56×56 pixels)	0.0627 ± 0.0002
MSE old network (28×28 pixels)	0.0248 ± 0.0002
MSE fine network (56×56 pixels)	0.0076 ± 0.0002
MSE old network (56×56 pixels)	0.0287 ± 0.0001
MSE new network (56×56 pixels)	0.0217 ± 0.0002

The results of the testing of the multi-scale network can be found in Table 1 within a 95% confidence interval. Each mean squared error represents the mean of the mean squared error of each of the 1000 test images that were processed. Old network means the network without the adjusted weights, new network is the network with the rescaled weights. The fine network is a network trained on the rescaled images of 56×56 pixels. This network is trained without regularization. It is of the same structure as the other network and also trained for 10 epochs.

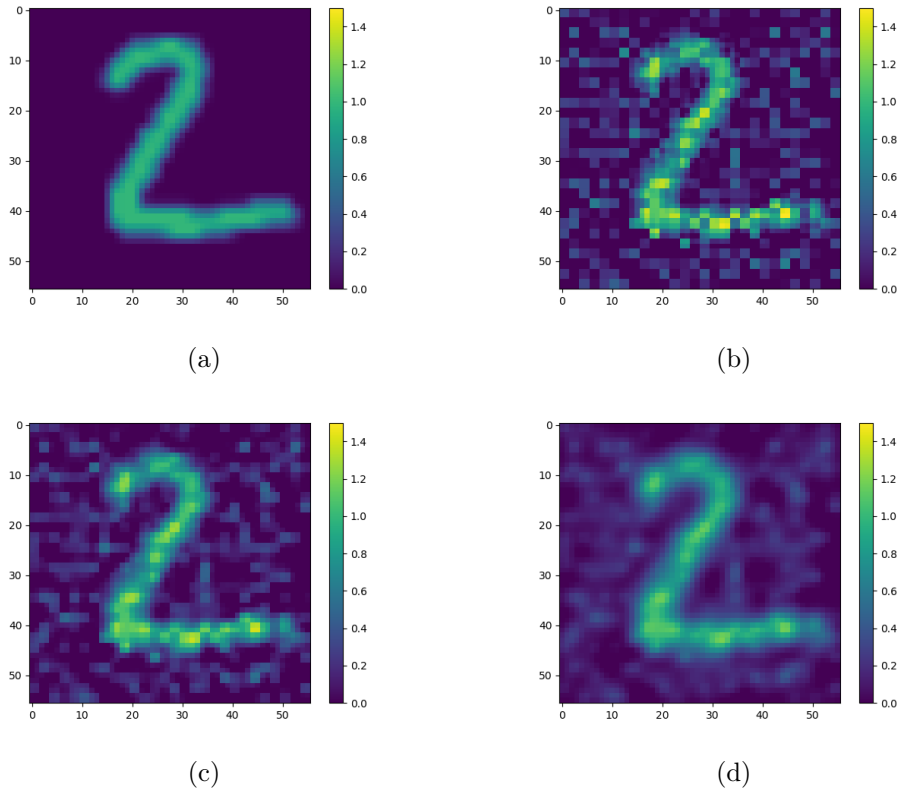


Figure 7: (a) Ground truth image, (b) noisy image, (c) image denoised by the original network, (d) image denoised by the new rescaled network.

For one of the images from the test set the ground truth, noisy and denoised images can be found in Figure 7. A slice of the pixels at row eight gives the graphs that can be found in Figure 8. These are graphs of the pixel intensities for the eight row of the images of 56 by 56 pixels.

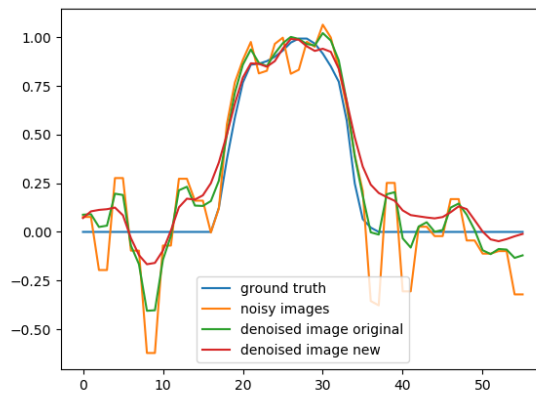


Figure 8: Slice at row 8 of 56 by 56 pixels.

5.2 Other structures

Table 2: Mean squared error parallel branches network.

MSE old parallel network (28×28 pixels)	0.0247 ± 0.0002
MSE old parallel network (56×56 pixels)	0.0292 ± 0.0002
MSE new parallel network (56×56 pixels)	0.0223 ± 0.0002

Training is done on the parallel and binary branches networks as well. See Section 4.3.2 for a description of the parallel network and Section 4.3.3 for a description of the binary network. The parallel branches network result in a training loss of 0.0296 and a validation loss of 0.0261. It turns out that no time rescaling is necessary for this configuration of the network. The results of the multi-scale method on the 1000 test images can be found in Table 2 with 95% confidence.

Table 3: Mean squared error binary branches network.

MSE old binary network (28×28 pixels)	0.0249 ± 0.0002
MSE old binary network (56×56 pixels)	0.0305 ± 0.0002
MSE new binary network (56×56 pixels)	0.0216 ± 0.0002

The training of the binary branches network results in a training loss of 0.0306 and a validation loss of 0.0251. The time step needs to be halved to rescaled it to a stable network for the 56×56 images. The results of the multi-scale method using this network can be found in Table 3 with 95% confidence as well.

Table 4: Mean squared error network with pooling layers.

MSE old binary network (28×28 pixels)	0.0236 ± 0.0003
MSE old binary network (56×56 pixels)	0.0220 ± 0.0002
MSE new binary network (56×56 pixels)	0.0220 ± 0.0002

Finally, training is done for the network with the pooling layers. See Section 4.4 for more on this network. This resulted in a training loss of 0.0311 and a validation loss of 0.0243. To rescale this network for the 56×56 pixel images it is necessary to halve the time step. The results of the network with the pooling layers can be found in Table 4, with 95% confidence.

6 Discussion

6.1 Results

First the results of the network of width one are discussed. The performance of the other structures will be discussed in Section 6.1.1. The new multi-scale network performed significantly better during the testing than the original network on the high resolution images. From this it can be concluded that the multi-scale method indeed improves the multi-scale results.

However, it does not come close to the results of the network that was trained on the high resolution images. It has to be noted that this network was trained without

regularization. So this network did not need to satisfy the stability conditions. This means this network was less restricted, possibly making it easier to get better results.

When visually comparing the results of the old and the new network, see Figures 7c and 7d, the background of the image resulting from the new network seems to be smoother than the background of the image resulting from the old network. Furthermore, the colour of the two seems to be smoother as well. This smoothing in both the number and the background seems to have a some effect on the border between the two as well. The whole image looks more blurry than the image that went through the original network. So it seems like the new network has smoothed the whole overall image more than the original network.

This can be seen in Figure 8 as well. The plot of the slice of the noisy image has some fluctuations. The original denoised network smooths this a bit and the new denoised network smooths it even more.

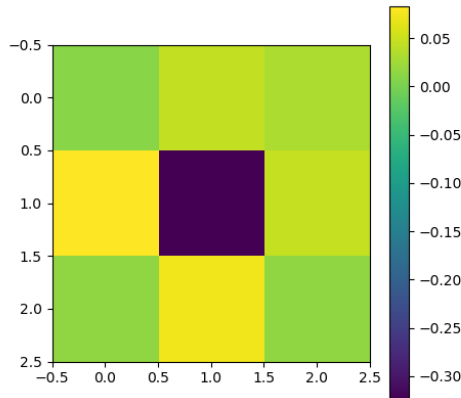


Figure 9: Kernel in the second convolutional layer.

All kernels within this network are quite close to each other, see appendix F. This probably is the result of the conditions placed upon the kernel. The kernel of the second convolutional layer can be found in Figure 9. It can be seen that the corner weights lie close to each other, as do the outer middle weights. Here the corner weights are lower than the middle ones. Comparable patterns can be found for the other kernels. A kernel of this form resembles a smoothing filter. This makes sense since smoothing the image would result in noise reduction. This smoothing is also reflected by the results, as previously discussed.

The need for stability turns out to put many restrictions on the weights. These restrictions on the weights might rule out kernels that would give better results. It might be possible to prove the stability in another, less restrictive way. This could be something to explore in further research.

In Figure 8 it can be seen that the network (both rescaled and original) seems to be better at handling negative noise than positive noise. The negative noise seems to be smoothed more than the positive noise. This could have something to do with the chosen value for α in the LeakyReLU activation, see Section 3.1.3. This is because the value of α determines how much negativity is added to the transformed image. So a higher value allows for lower values in the resulting image. This possibly results in a better ability to cancel positive noise. The value chosen for α was 0.3, but for example 0.5 might have been

a better choice.

6.1.1 Network structures

After ruling out the connected structure three structures remained to consider, see Section 4.3. The main focus of the results section was the structure with network width one. This structure is the closest related to the theory. The results of the original networks for the networks with parallel and binary branches on the 28×28 pixel images are comparable to those of the basic structure. The results of the rescaled networks on the 56×56 pixel images are not significantly better than those for the basic structure. The parallel and binary branch structures therefore do not improve the results. The kernels trained for the other structures resembled those found in the network with width one, like Figure 9. The way the same kernel structure repeated itself in these networks seems to further indicate that the conditions imposed on the kernels cause the kernels to be trained to the same form.

The results of the network with the pooling layers, see Section 4.4, were a bit different. The network performs better on the 28×28 pixel images than all the other networks. Especially interesting is that the original network performs better on the 56×56 pixel images than the resolution it was trained for. Rescaling the network specifically for the 56×56 images yields about the same result as using the original network for 56×56 images. However, the rescaled network with layer width one performs better on the 56×56 images. The network with the pooling layers gives better results for the resolution it was trained for, however the rescaling method does not seem to have much use for the network of this form.

It could be possible that the original network and the rescaled network perform similarly on the 56×56 pixel images because of the different scales that are present in both networks. In the original network the use of the pooling layers yield a network that is trained to handle images of 28×28 , 14×14 and 7×7 pixels. After rescaling the network should be able to handle images of 56×56 , 28×28 and 14×14 . Two of these resolutions are the same, so the original network was already partially capable of dealing with images of these resolutions. The kernels that operate on them are different, but the network does know how to use these resolutions. This could make the original pooling network more robust for different resolutions.

The network again trains kernels similar to the previous networks. This shows that the stability conditions might be too restrictive for the kernels of this network as well. It seems like the pooling process allows the network to perform better, since the difference cannot be found in the kernels trained. It might be possible to improve the results more by creating pooling networks of different structures. This could be something to look into for further research.

6.2 Theory

In Section 3.3 conditions are imposed on the parameters of the network, see *Conditions 1*. One of these conditions specifies that the sum of the weights should always be zero. The continuous parameter β_1 from the continuous form of the kernel of the network, see equation (8), happens to be the sum of the weights:

$$\beta_1 = \sum_{i=1}^9 \theta_i = 0.$$

This means this parameter is always zero. So from this it can be concluded that the first term of the continuous representation always drops. This is the $\beta_1 y$ term. It makes sense for this term to drop. For example, suppose y is a continuous representation of the discrete network. Since the left side of the equation is a derivative the right side should equal 0. However, if the first term is equal to the function itself and β_1 is not zero this will not hold. All other terms will be zero since they are partial derivatives of the constant function.

The condition that imposes that the sum of the weights should be zero makes sense for the deblurring application. When the sum of the weights is zero, one can say that on average the image does not change. For the denoising application, Gaussian noise with mean zero is added to the image which needs to be removed. The Gaussian noise does not change the mean of the image on average. So applying this network that averages out the noise while keeping the mean of the image the same seems like a good strategy for denoising.

The kernel structure used throughout this paper is restricted to kernels of size 3×3 . The theory could also be extended to different kernel sizes. The way to do this is to increase or decrease the number of finite difference discretizations. The theory could be adapted in a way to for example cater for 2×2 or 4×4 kernels. This could be done by choosing the finite difference discretization that represent the kernel such that they span over all weights. The number of finite differences should be equal to the number of weights to ensure a unique transformation of the weights.

As discussed in Section 3.2 the edge handling is ignored throughout this research. However, when dealing with small images, for example as was done with the images of 28 by 28 pixels, the edges might have quite some influence on the outcome. It could be useful to further analyze the behaviour of the processes on the edges.

The multi-scale method might be too restrictive for the network structure and the kernels. The connected network structure discussed in Section 4.3 was ruled out since it did not fit the theory. Network structures are generally considered to be more effective. It might be necessary to adapt the theory to fit network structures like these as well.

6.2.1 Pooling layers

For the network with the pooling layers described in Section 4e the pooling layers and upscaling layers were ignored when rescaling. Better rescaling results might be obtained when taking these layers in account. It might be a possibility to consider the average pooling layer as a convolutional layer. The convolutional operation is essentially a linear combination of the pixels within a region. This is the same for the average pooling layer. However, there is a difference: a convolutional layer applies a convolution for every pixel. The pooling layer applies its operation only once within each region of pooling. The distance between the pixels at which a convolution is applied is called the stride. For the convolutions used in this paper the stride is 1. For the 2×2 pooling layer the stride is 2. Maybe the theoretical link between convolutional layers and pooling layers is easier made if the convolutional layers all have a stride of 2. In further research the relation between convolutional layers and average pooling layers could be further investigated.

Another problem with relating the theory developed in this paper to the pooling layers is that the resolution changes after these layers. So the convolutions performed after a pooling layer are on a different scale than those before. In the theoretical model it was possible to consider the network as a continuous process, where each layer is part of a step in the discretization of a PDE. It is the question how this is still possible when changing the image scale, since this means the grid on which the discretization is performed changes. This could be something else to investigate in further research.

6.3 Application

A problem that was discovered for the multi-scale denoising is that when starting with a rough grid image, rescaling the image and then separately adding noise to both images the developed multi-scale network kept performing worse than the results of simply putting the image through the original network. This has to do with the network being trained to recognise the noise on the rough image with specific frequencies, see Section 4.1. Applying the network to various gridsizes, but only being able to add noise at one resolution level (to the multi-scale images) severely restricts the applications of this method within denoising. Generally, the noise that needs to be removed from an image is at the same scale as the image. However, this method could have other applications that are more suitable.

The main problem with the noise, as discussed in Section 4.1, is that the underlying continuous image is not the same. Any application in which these are the same should be suitable to apply the multi-scale network.

For example in the removal of objects at a fixed ratio from images the method could be useful. One could think of time stamps in photographs. The time stamps are added with the same size, but some cameras are capable of taking images at different resolutions. This method would be suitable to remove these time stamps.

Another application would be to remove CT scan artefacts. When making a CT scan, multiple images are made. These images are meshed together to create the full scan. Because there is only a finite number of images this results in streaks, the artefacts, being left in the image. These artefacts are dependent on the angles the CT scanner made while scanning. So this is a continuous process, different resolutions will result in artefacts at the same places with respect to the continuous domain. So this could be a suitable application. For more on the removal of CT artefacts using CNN, see [11].

Important to note for the use of this method for any applications are the restrictions on the kernel. For denoising it is logical that the sum of the weights is zero, keeping the image equal on average. However, this might not be suitable for all applications. For example for the time stamp removal, this might not be desirable at all. Since for example when removing a dark stamp from a light image the average should probably not remain the same.

In this paper the 28 by 28 pixel images as training scale and 56 by 56 pixel images as test scale were used. Of course in actual applications only being able to use the second multiple of the training scale is not that interesting. It would be more useful if the method could be extended to other resolutions as well. A solution for this would be based on what has been done in this paper to ensure the positivity of the weights for the 56 by 56 pixels, see Section 3.4. If it is known before hand which scale would be desired, for example scales 48, 56 and 79. Then it would be possible to extend the regularization function such that the positivity constraint is enforced for 48 and 79 as well as 28 and 56.

7 Conclusion

The problem at hand was to create a multi-scale CNN using PDEs. The idea of a multi-scale CNN is that it is able to perform on input of different sizes. In this paper only convolutional networks with respect to images were considered. So the objective was to create a network that performs on images with different resolutions. To do this residual neural networks of a specific structure were considered.

This structure allows for a link between the ResNets and PDEs, since it is possible to treat these ResNets as discretizations of a PDE. Using these discretizations it is possible

to find a continuous representation of the ResNet, this representation depends on the resolution of the images the network originally was able to perform on. The resolution can be viewed as the grid of the discretization. Using this continuous representation, it is possible to go to different discretizations of this representation. These new discretizations can be of different grid sizes. This allows to find different residual networks, which are able to perform on these different grid sizes. This method also allows for rescaling with respect to time. This can be used to ensure stability.

The need for stability implies that it is not possible to simply train a network and rescale it. Conditions need to be imposed on the weights of the network. These conditions limit the behaviour of the network to specific forms. Furthermore, these conditions limit the number of resolutions the multi-scale network can be applied to.

The multi-scale network performs better than the original network on a finer grid than the network originally was trained for. Except when pooling layers are used, in this case the original network yields comparable results to the rescaled network. The network with the pooling layers does perform better overall.

Within the paper the multi-scale network was used for the denoising of images, because of the importance of this application. This choice turned out to have some downsides. The method only works if the continuous representation of the different resolutions is the same. For noise this is generally not the case. However, the method that has been developed could be significant for other applications. Examples of these applications the removal of artefacts in CT scans and the removal of time stamps from images.

References

- [1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.
- [2] Robert A Adams and Christopher Essex. *Calculus: A Complete Course*. Pearson, 2014.
- [3] Kendall Atkinson, Weimin Han, and David E Stewart. *Numerical solution of ordinary differential equations*, volume 108. John Wiley & Sons, 2011.
- [4] Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2559–2566. Citeseer, 2010.
- [5] VF Demyanov. Exhausters of a positively homogeneous function. *Optimization*, 45(1-4):13–29, 1999.
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [7] Christian Grossmann, Hans-Görg Roos, and Martin Stynes. *Numerical treatment of partial differential equations*, volume 154. Springer, 2007.
- [8] Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun. Learning across scales—multiscale methods for convolution neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [10] Carlo Innamorati, Tobias Ritschel, Tim Weyrich, and Niloy J Mitra. Learning on the edge: Explicit boundary handling in cnns. *arXiv preprint arXiv:1805.03106*, 2018.
- [11] Kyong Hwan Jin, Michael T McCann, Emmanuel Froustey, and Michael Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26(9):4509–4522, 2017.
- [12] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.
- [13] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.
- [14] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.
- [15] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Urs A Muller, Eduard Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.
- [16] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.
- [17] Sparsh Mittal. A survey of fpga-based accelerators for convolutional neural networks. *Neural computing and applications*, pages 1–31, 2018.
- [18] Keith W Morton and David Francis Mayers. *Numerical solution of partial differential equations: an introduction*. Cambridge university press, 2005.
- [19] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.
- [20] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Appendix

A Inverse transformation

The inverse transformation of the parametrization of the weights given by equation (6), is given by the following expression:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -h & 0 & h & -h & 0 & h & -h & 0 & h \\ k & k & k & 0 & 0 & 0 & -k & -k & -k \\ h^2 & 0 & 0 & h^2 & 0 & 0 & h^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & k^2 & k^2 & k^2 \\ -hk & 0 & hk & 0 & 0 & 0 & hk & 0 & -hk \\ 0 & 0 & 0 & 0 & 0 & 0 & -hk^2 & 0 & hk^2 \\ h^2k & 0 & 0 & 0 & 0 & 0 & -h^2k & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h^2k^2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \\ \theta_8 \\ \theta_9 \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \\ \beta_9 \end{bmatrix}.$$

B Finite differences

Several finite difference discretizations are used by the method presented in this paper. Different finite difference methods needed to be chosen to make it possible to parametrize the weights such that they can be seen as the discretization of a PDE. In the problem at hand only the values of a pixel can be considered. It is not possible to obtain the value halfway between two grid points. This means, for example, that the first order central finite difference cannot be used since this is given by:

$$\frac{\partial y}{\partial x} \approx \frac{y(x + \frac{1}{2}h) - y(x - \frac{1}{2}h)}{h},$$

instead the forward difference was used. This will be discussed shortly. However, for the second order finite difference the central method is used. This is because the forward method uses the pixel two step distant from the middle pixel, this pixel is not available in the convolution so it is not possible to use this. This results in a mix of forward and central differences with respect to space. This means that the entire approximation is not exactly centred at the middle. However, all discretizations are chosen such that the steps made in the discretization never skip over a grid point. For example a variation on the first order central difference could have been used:

$$\frac{\partial y}{\partial x} \approx \frac{y(x + h) - y(x - h)}{2h}.$$

It was specifically chosen not to use this. This ensures the approximations are all on the same grid.

A total of nine equations are necessary for the method to work. Each discretization that is chosen such that the points considered are on the grid and no grid points are skipped. The first order forward finite difference in x_1 direction is given by:

$$\begin{aligned} \frac{\partial y}{\partial x_1} &\approx \frac{y(x_1 + h, x_2) - y(x_1, x_2)}{h} \\ &= \frac{Y_{n+1,m} - Y_{n,m}}{h}. \end{aligned}$$

This discretization works the same in the x_2 direction only the terms are switched. The same goes for the other not mixed finite differences in this section. The first order central finite difference discretization is given by:

$$\frac{\partial y}{\partial x_1} \approx \frac{y(x_1 + \frac{1}{2}h, x_2) - y(x_1 - \frac{1}{2}h, x_2)}{h}.$$

The second order central finite difference makes use of the first order central finite difference by applying it twice, resulting in the following discretization:

$$\begin{aligned} \frac{\partial^2 y}{\partial x_1^2} &\approx \frac{\partial}{\partial x_1} \left[\frac{y(x_1 + \frac{1}{2}h, x_2) - y(x_1 - \frac{1}{2}h, x_2)}{h} \right] \\ &= \frac{y(x_1 + h, x_2) - 2y(x_1, x_2) + y(x_1 - h, x_2)}{h^2} \\ &= \frac{Y_{n+1,m} - 2Y_{n,m} + Y_{n-1,m}}{2h}. \end{aligned}$$

The mixed second order forward applies the first order forward finite difference in both directions, which yields:

$$\begin{aligned} \frac{\partial^2 y}{\partial x_1 \partial x_2} &\approx \frac{\partial}{\partial x_1} \left[\frac{y(x_1, x_2 + h) - y(x_1, x_2)}{k} \right] \\ &= \frac{y(x_1 + h, x_2 + k) - y(x_1 + h, x_2) - y(x_1, x_2 + k) + y(x_1, x_2)}{hk} \\ &= \frac{Y_{n+1,m+1} - Y_{n+1,m} - Y_{n,m+1} + Y_{n,m}}{hk}. \end{aligned}$$

The mixed third order finite difference is used as well. The discretization that is used is found by applying the first order forward finite difference to the second order central, such that:

$$\begin{aligned} \frac{\partial^3 y}{\partial x_1 \partial x_2^2} &\approx \frac{\partial}{\partial x_1} \left[\frac{y(x_1 + h, x_2) - 2y(x_1, x_2) + y(x_1 - h, x_2)}{h^2} \right] \\ &= \frac{y(x_1 + h, x_2 + k) - 2y(x_1 + h, y) + y(x_1 + h, x_2 - k)}{hk^2} \\ &\quad + \frac{-y(x_1, x_2 + k) + 2y(x_1, x_2) - y(x_1, x_2 - k)}{hk^2} \\ &= \frac{Y_{n+1,m+1} - 2Y_{n+1,m} + Y_{n+1,m-1} - Y_{n,m+1} + 2Y_{n,m} - Y_{n,m-1}}{hk^2}. \end{aligned}$$

Finally, the mixed fourth order finite difference is used. The one that is used is the derivative that is evaluated two times with respect to one direction and two times with respect to the other. It is found by applying the central finite difference of the first order

twice:

$$\begin{aligned}
\frac{\partial^4 y}{\partial x_1^2 \partial x_2^2} &\approx \frac{\partial^2}{\partial x_1^2} \left[\frac{y(x_1, x_2 + k) - 2y(x_1, x_2) + y(x_1, x_2 - k)}{k^2} \right] \\
&= \frac{y(x_1 + h, x_2 + k) - 2y(x_1 + h, x_2) + y(x_1 + h, x_2 - k)}{h^2 k^2} \\
&\quad + \frac{4y(x_1, x_2) - 2y(x_1, x_2 - k) - 2y(x_1, x_2 + k)}{h^2 k^2} \\
&\quad + \frac{y(x_1 - h, x_2 + k) - 2y(x_1 - h, x_2) + y(x_1 - h, x_2 - h)}{h^2 k^2} \\
&= \frac{Y_{n+1, m+1} - 2Y_{n+1, m} + Y_{n+1, m-1} - 2Y_{n, m_1}}{h^2 k^2} \\
&\quad + \frac{4Y_{n, m} - 2Y_{n, m+1} + Y_{n-1, m+1} - 2Y_{n-1, m} + y_{n-1, m-1}}{h^2 k^2}.
\end{aligned}$$

C Continuous representation

A continuous representation of the discrete kernel is to be found. When the discrete kernel is applied to some input Y , the element at position (n, m) of the resulting matrix C is given by:

$$\begin{aligned}
c_{n, m} = \frac{1}{\Delta t} &(\theta_1 Y_{n-1, m+1} + \theta_2 Y_{n, m+1} + \theta_3 Y_{n+1, m+1} + \theta_4 Y_{n-1, m} \\
&+ \theta_5 Y_{n, m} + \theta_6 Y_{n+1, m} + \theta_7 Y_{n-1, m-1} + \theta_8 Y_{n, m-1} + \theta_9 Y_{n+1, m-1}).
\end{aligned}$$

Applying the parametrization of the weights given by equation (6), gives the following representation:

$$\begin{aligned}
c_{n, m} = &Y_{n-1, m+1} \left[\frac{\beta_8}{h^2 k} + \frac{\beta_9}{h^2 k^2} \right] \\
&+ Y_{n, m+1} \left[\frac{\beta_3}{k} - \frac{\beta_6}{hk} + \frac{\beta_5}{k^2} - \frac{\beta_7}{hk^2} + \frac{2\beta_8}{h^2 k} - \frac{2\beta_9}{h^2 k^2} \right] \\
&+ Y_{n+1, m+1} \left[\frac{\beta_6}{hk} + \frac{\beta_7}{hk^2} + \frac{\beta_8}{h^2 k} + \frac{\beta_9}{h^2 k^2} \right] \\
&+ Y_{n-1, m} \left[\frac{\beta_4}{h^2} - \frac{\beta_8}{h^2 k} - \frac{2\beta_9}{h^2 k^2} \right] \\
&+ Y_{n, m} \left[\beta_1 - \frac{\beta_2}{h} - \frac{\beta_3}{k} - \frac{2\beta_4}{h^2} - \frac{2\beta_5}{k^2} + \frac{\beta_6}{hk} + \frac{2\beta_7}{hk^2} + \frac{2\beta_8}{h^2 k} + \frac{4\beta_9}{h^2 k^2} \right] \\
&+ Y_{n+1, m} \left[\frac{\beta_2}{k} + \frac{\beta_4}{h^2} - \frac{\beta_6}{hk} - \frac{2\beta_7}{hk^2} - \frac{\beta_8}{h^2 k} - \frac{2\beta_9}{h^2 k^2} \right] \\
&+ Y_{n-1, m-1} \left[\frac{\beta_9}{h^2 k^2} \right] \\
&+ Y_{n, m-1} \left[\frac{\beta_5}{k^2} - \frac{\beta_7}{hk^2} - \frac{2\beta_9}{h^2 k^2} \right] \\
&+ Y_{n+1, m-1} \left[\frac{\beta_7}{hk^2} + \frac{\beta_9}{h^2 k^2} \right].
\end{aligned}$$

Rearranging the terms by ordering the elements $Y_{n,m}^j$ gives the following form:

$$\begin{aligned}
c_{n,m} = & \beta_1 Y_{n,m} + \beta_2 \frac{Y_{n+1,m} - Y_{n,m}}{h} + \beta_3 \frac{Y_{n,m+1} - Y_{n,m}}{k} \\
& + \beta_4 \frac{Y_{n+1,m} - 2Y_{n,m} + Y_{n-1,m}}{h^2} + \beta_5 \frac{Y_{n,m+1} - 2Y_{n,m} + Y_{n,m-1}}{k^2} \\
& + \beta_6 \frac{Y_{n+1,m+1} - Y_{n+1,m} - Y_{n,m+1} + Y_{n,m-1}}{hk} \\
& + \beta_7 \frac{Y_{n+1,m+1} - 2Y_{n+1,m} + Y_{n+1,m-1} - Y_{n,m+1} + 2Y_{n,m} - Y_{n,m-1}}{hk^2} \\
& + \beta_8 \frac{Y_{n+1,m+1} - 2Y_{n,m+1} + Y_{n-1,m+1} - Y_{n+1,m} + 2Y_{n,m} - Y_{n-1,m}}{h^2k} \\
& + \beta_9 \left[\frac{Y_{n+1,m+1} - 2Y_{n+1,m} + Y_{n+1,m-1} - 2Y_{n,m-1}}{h^2k^2} \right. \\
& \quad \left. + \frac{4Y_{n,m} - 2Y_{n,m+1} + Y_{n-1,m+1} - 2Y_{n-1,m} + Y_{n-1,m-1}}{h^2k^2} \right].
\end{aligned}$$

This holds for every element in the matrix $C = K(\theta) * Y$. Every term in the previous equation relates to a finite difference discretization, see appendix B. Now by taking the limit as h and k approach zero each element in Y^j gets a continuous representation:

$$\begin{aligned}
k(\theta, y(x_n, y_m, t_j)) = & \beta_1 y(x_n, y_m, t_j) + \beta_2 \frac{\partial y(x_n, y_m, t_j)}{\partial x_1} + \beta_3 \frac{\partial y(x_n, y_m, t_j)}{\partial x_2} \\
& + \beta_4 \frac{\partial^2 y(x_n, y_m, t_j)}{\partial x_1^2} + \beta_5 \frac{\partial^2 y(x_n, y_m, t_j)}{\partial x_2^2} + \beta_6 \frac{\partial^2 y(x_n, y_m, t_j)}{\partial x_1 \partial x_2} \\
& + \beta_7 \frac{\partial^3 y(x_n, y_m, t_j)}{\partial x_1 \partial x_2^2} + \beta_8 \frac{\partial^3 y(x_n, y_m, t_j)}{\partial x_1^2 \partial x_2} + \beta_9 \frac{\partial^4 y(x_n, y_m, t_j)}{\partial x_1^2 \partial x_2^2}.
\end{aligned}$$

D Stability proof

A discretization of the following continuous partial differential equation will be considered:

$$\begin{aligned}
\frac{\partial y}{\partial t} = & \sigma \left(\beta_1 y + \beta_2 \frac{\partial y}{\partial x_1} + \beta_3 \frac{\partial y}{\partial x_2} + \beta_4 \frac{\partial^2 y}{\partial x_1^2} + \beta_5 \frac{\partial^2 y}{\partial x_2^2} \right. \\
& \left. + \beta_6 \frac{\partial^2 y}{\partial x_1 \partial x_2} + \beta_7 \frac{\partial^3 y}{\partial x_1 \partial x_2^2} + \beta_8 \frac{\partial^3 y}{\partial x_1^2 \partial x_2} + \beta_9 \frac{\partial^4 y}{\partial x_1^2 \partial x_2^2} \right),
\end{aligned}$$

here σ is a LeakyReLU activation function with parameter $0 < \alpha < 1$. The discretization that is chosen is of the following form:

$$\begin{aligned} \frac{Y_{n,m}^{j+1} - Y_{n,m}^j}{\Delta t} = & \sigma \left(\beta_1 Y_{n,m}^j + \beta_2 \frac{Y_{n+1,m}^j - Y_{n,m}^j}{h} + \beta_3 \frac{Y_{n,m+1}^j - Y_{n,m}^j}{h_2} \right. \\ & + \beta_4 \frac{Y_{n+1,m}^j - 2Y_{n,m}^j + Y_{n-1,m}^j}{h^2} + \beta_5 \frac{Y_{n,m+1}^j - 2Y_{n,m}^j + Y_{n,m-1}^j}{k^2} \\ & + \beta_6 \frac{Y_{n+1,m+1}^j - Y_{n,m+1}^j - Y_{n+1,m}^j + Y_{n,m}^j}{hk} \\ & + \beta_7 \frac{Y_{n+1,m+1}^j - 2Y_{n+1,m}^j + Y_{n+1,m-1}^j - Y_{n,m+1}^j + 2Y_{n,m}^j - Y_{n,m-1}^j}{hk^2} \\ & + \beta_8 \frac{Y_{n+1,m+1}^j - 2Y_{n,m+1}^j + Y_{n-1,m+1}^j - Y_{n+1,m}^j + 2Y_{n,m}^j - Y_{n-1,m}^j}{h^2k} \\ & + \beta_9 \left[\frac{Y_{n+1,m+1}^j - 2Y_{n+1,m}^j + Y_{n+1,m-1}^j - 2Y_{n,m+1}^j}{h^2k^2} \right. \\ & \left. + \frac{4Y_{n,m}^j - 2Y_{n,m+1}^j + Y_{n-1,m+1}^j - 2Y_{n-1,m}^j + Y_{n-1,m-1}^j}{h^2k^2} \right] \Bigg). \end{aligned}$$

For each derivative in the continuous PDE a finite difference method is used to obtain a discrete form. Some operators for these finite difference methods will be defined. For the first order forward finite difference the following will be used:

$$\begin{aligned} \Delta_{+x_1} v(x_1, x_2, t) & := v(x_1 + h, x_2, t) - v(x_1, x_2, t), \\ \Delta_{+x_2} v(x_1, x_2, t) & := v(x_1, x_2 + k, t) - v(x_1, x_2, t), \\ \Delta_{+t} v(x_1, x_2, t) & := v(x_1, x_2, t + \Delta t) - v(x_1, x_2, t). \end{aligned}$$

For the second order central difference with respect to either x_1 or x_2 , the following are defined:

$$\begin{aligned} \delta_{x_1}^2 v(x_1, x_2, t) & := v(x_1 + h, x_2, t) - 2v(x_1, x_2, t) + v(x_1 - h, x_2, t), \\ \delta_{x_2}^2 v(x_1, x_2, t) & := v(x_1, x_2 + k, t) - 2v(x_1, x_2, t) + v(x_1, x_2 - k, t). \end{aligned}$$

Furthermore, for the mixed second order forward difference the following operator will be used:

$$\Delta_{+x_1 x_2} v(x_1, x_2, t) := v(x_1 + h, x_2 + k, t) - v(x_1, x_2 + k, t) - v(x_1 + h, x_2, t) + v(x_1, x_2, t).$$

For the third order mixed derivatives first a second order central difference is applied, followed by a first order forward difference, so the following will be denoted:

$$\begin{aligned} \Delta_{+x_1} \delta_{x_2}^2 v(x_1, x_2, t) & := v(x_1 + h, x_2 + k, t) - 2v(x_1 + h, x_2, t) + v(x_1 + h, x_2 - k, t) \\ & \quad - v(x_1, x_2 + k, t) + 2v(x_1, x_2, t) - v(x_1, x_2 - k, t), \\ \Delta_{+x_2} \delta_{x_1}^2 v(x_1, x_2, t) & := v(x_1 + h, x_2 + k, t) - 2v(x_1, x_2 + k, t) + v(x_1 - h, x_2 + k, t) \\ & \quad - v(x_1 + h, x_2, t) + 2v(x_1, x_2, t) - v(x_1 - h, x_2, t). \end{aligned}$$

Finally, the fourth order mixed derivative used consists of a combination two central differences, so this will be denoted by:

$$\begin{aligned} \delta_{x_1}^2 \delta_{x_2}^2 v(x_1, x_2, t) & := v(x_1 + h, x_2 + k, t) - 2v(x_1 + h, x_2, t) + v(x_1 + h, x_2 - k, t) \\ & \quad - 2v(x_1, x_2 + k, t) + 4v(x_1, x_2, t) - 2v(x_1, x_2 - k, t) \\ & \quad + v(x_1 - h, x_2 + k, t) - 2v(x_1 - h, x_2, t) + v(x_1 - h, x_2 - k, t). \end{aligned}$$

The error of the approximation at (x_{1n}, x_{2n}, t_j) will be denoted by $e_{n,m}^j$, defined as:

$$e_{n,m}^j := Y_{n,m}^j - y(x_{1n}, x_{2m}, t_j). \quad (12)$$

It is desired that this error is bounded to ensure stability.

The truncation error $T(x_1, x_2, t)$ will be defined as:

$$\begin{aligned} T(x_1, x_2, t) := & \frac{\Delta_{+t}y(x_1, x_2, t)}{\Delta t} - \sigma \left(\beta_1 y(x_1, x_2, t) + \beta_2 \frac{\Delta_{+x_1}y(x_1, x_2, t)}{h} \right. \\ & + \beta_3 \frac{\Delta_{+x_2}y(x_1, x_2, t)}{k} + \beta_4 \frac{\delta_{x_1}y(x_1, x_2, t)}{h^2} \\ & + \beta_5 \frac{\delta_{x_2}y(x_1, x_2, t)}{k^2} + \beta_6 \frac{\Delta_{+x_1x_2}y(x_1, x_2, t)}{hk} \\ & + \beta_7 \frac{\Delta_{+x_1}\delta_{x_2}^2y(x_1, x_2, t)}{hk^2} + \beta_8 \frac{\Delta_{+x_2}\delta_{x_1}^2y(x_1, x_2, t)}{kh^2} \\ & \left. + \beta_9 \frac{\delta_{x_1}^2\delta_{x_2}^2y(x_1, x_2, t)}{k^2h^2} \right). \end{aligned} \quad (13)$$

Normally the activation function should be around the second term until the last term. However, this special truncation error is chosen for convenience.

It is known that $Y_{n,m}^j$ satisfies the discretization and $y(x_{1n}, x_{2m}, t_j)$ satisfies the forward difference. Such that the error, see equation (12), can be rewritten:

$$\begin{aligned} e_{n,m}^{j+1} = & Y_{n,m}^{j+1} - y(x_{1n}, x_{2m}, t_j + \Delta t) \\ = & Y_{n,m}^j + \sigma \left(\beta_1 \Delta t Y_{n,m}^j + \beta_2 \frac{\Delta t}{h} \Delta_{+x_1} Y_{n,m}^j + \beta_3 \frac{\Delta t}{k} \Delta_{+x_2} Y_{n,m}^j \right. \\ & + \beta_4 \frac{\Delta t}{h^2} \delta_{x_1} Y_{n,m}^j + \beta_5 \frac{\Delta t}{k^2} \delta_{x_2} Y_{n,m}^j + \beta_6 \frac{\Delta t}{hk} \Delta_{+x_1x_2} Y_{n,m}^j \\ & + \beta_7 \frac{\Delta t}{hk^2} \Delta_{+x_1} \delta_{x_2}^2 Y_{n,m}^j + \beta_8 \frac{\Delta t}{kh^2} \Delta_{+x_2} \delta_{x_1}^2 Y_{n,m}^j + \beta_9 \frac{\Delta t}{k^2h^2} \delta_{x_1}^2 \delta_{x_2}^2 Y_{n,m}^j \left. \right) \\ & - y(x_{1n}, x_{2m}, t_j) - \Delta_{+t}y(x_{1n}, x_{2m}, t_j). \end{aligned}$$

From now on, the combination of all finite differences in the previous equation will be denoted as (since $Y_{n,m}^j = \tilde{y}(x_{1n}, x_{2m}, t_j)$ for some approximation \tilde{y} of y):

$$\begin{aligned} D_{x_1, x_2, t}v(x_1, x_2, t) := & \beta_1 v(x_1, x_2, t) + \beta_2 \frac{1}{h} \Delta_{+x_1} v(x_1, x_2, t) + \beta_3 \frac{1}{k} \Delta_{+x_2} v(x_1, x_2, t) \\ & + \beta_4 \frac{1}{h^2} \delta_{x_1} v(x_1, x_2, t) + \beta_5 \frac{1}{k^2} \delta_{x_2} v(x_1, x_2, t) \\ & + \beta_6 \frac{1}{hk} \Delta_{+x_1x_2} v(x_1, x_2, t) + \beta_7 \frac{1}{hk^2} \Delta_{+x_1} \delta_{x_2}^2 v(x_1, x_2, t) \\ & + \beta_8 \frac{1}{kh^2} \Delta_{+x_2} \delta_{x_1}^2 v(x_1, x_2, t) + \beta_9 \frac{1}{k^2h^2} \delta_{x_1}^2 \delta_{x_2}^2 v(x_1, x_2, t). \end{aligned} \quad (14)$$

Using this notation and the definition of the error (12), the following can be seen:

$$\begin{aligned} e_{n,m}^{j+1} = & Y_{n,m}^j - y(x_{1n}, x_{2m}, t_j) + \sigma \left(\Delta t D_{x_1, x_2, t} Y_{n,m}^j \right) - \Delta_{+t}y(x_{1n}, x_{2m}, t_j) \\ = & e_{n,m}^j + \sigma \left(\Delta t D_{x_1, x_2, t} e_{n,m}^j + \Delta t D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j) \right) - \Delta_{+t}y(x_{1n}, x_{2m}, t_j). \end{aligned} \quad (15)$$

Furthermore, using the definition 14 for the definition of the truncation error, see equation (13), the truncation error is given by:

$$T(x_1, x_2, t) = \frac{\Delta_{+t}y(x_1, x_2, t)}{\Delta t} - \sigma(D_{x_1, x_2, t}y(x_1, x_2, t)). \quad (16)$$

Now four cases will be considered:

- Case 1: $D_{x_1, x_2, t}e_{n, m}^j + D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) \geq 0$ and $D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) \geq 0$
- Case 2: $D_{x_1, x_2, t}e_{n, m}^j + D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) \geq 0$ and $D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) < 0$
- Case 3: $D_{x_1, x_2, t}e_{n, m}^j + D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) < 0$ and $D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) < 0$
- Case 4: $D_{x_1, x_2, t}e_{n, m}^j + D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) < 0$ and $D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) \geq 0$

Case 1 In this case the activation function of the term within the activation function is the term itself, since a LeakyReLU is used on a positive term, so:

$$e_{n, m}^{j+1} = e_{n, m}^j + \Delta t D_{x_1, x_2, t}e_{n, m}^j + \Delta t D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) - \Delta_{+t}y(x_{1n}, x_{2m}, t_j).$$

For the truncation error as previously defined the following holds:

$$T(x_{1n}, x_{2m}, t_j) = \frac{\Delta_{+t}y(x_{1n}, x_{2m}, t_j)}{\Delta t} - D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j).$$

Thus, in this case the following holds:

$$e_{n, m}^{j+1} = e_{n, m}^j + \Delta t D_{x_1, x_2, t}e_{n, m}^j - \Delta t T(x_{1n}, x_{2m}, t_j).$$

Taking the absolute value and using the triangle inequality results in the following:

$$|e_{n, m}^{j+1}| \leq |e_{n, m}^j + \Delta t D_{x_1, x_2, t}e_{n, m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|.$$

Case 2 In this case the LeakyReLU is again used on a positive term, so:

$$e_{n, m}^{j+1} = e_{n, m}^j + \Delta t D_{x_1, x_2, t}e_{n, m}^j + \Delta t D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) - \Delta_{+t}y(x_{1n}, x_{2m}, t_j).$$

However, for the term within the activation function of the truncation error is negative, see equation (16), which means that:

$$T(x_{1n}, x_{2m}, t_j) = \frac{\Delta_{+t}y(x_{1n}, x_{2m}, t_j)}{\Delta t} - \alpha (D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j)).$$

Substituting this into the equation for the error (15) gives:

$$e_{n, m}^{j+1} = e_{n, m}^j + \Delta t D_{x_1, x_2, t}e_{n, m}^j + (1 - \alpha)\Delta t D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) - \Delta t T(x_{1n}, x_{2m}, t_j).$$

Taking the absolute value of both sides and applying the triangle inequality gives:

$$|e_{n, m}^{j+1}| \leq |e_{n, m}^j + \Delta t D_{x_1, x_2, t}e_{n, m}^j + (1 - \alpha)\Delta t D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j)| + \Delta t |T(x_{1n}, x_{2m}, t_j)|. \quad (17)$$

Since it is given that $D_{x_1, x_2, t}e_{n, m}^j + D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) \geq 0$ and $D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) < 0$, it is clear that:

$$-D_{x_1, x_2, t}e_{n, m}^j \leq D_{x_1, x_2, t}y(x_{1n}, x_{2m}, t_j) < 0. \quad (18)$$

Now this case will be split in two subcases:

- Case 2.1: $e_{n,m}^j + \Delta t D_{x_1,x_2,t} e_{n,m}^j + (1 - \alpha) D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j) \geq 0$
- Case 2.2: $e_{n,m}^j + \Delta t D_{x_1,x_2,t} e_{n,m}^j + (1 - \alpha) D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j) < 0$

Case 2.1 In this case the first term of the right hand side of equation (17) is the absolute value of a positive number. From equation (18) it is known that $D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j) < 0$, combining this yields:

$$|e_{n,m}^{j+1}| \leq |e_{n,m}^j + \Delta t D_{x_1,x_2,t} e_{n,m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|$$

Case 2.2 Now, the first term of the right hand side of equation (17) is the absolute value of a negative number. By equation (18) it is known that $-D_{x_1,x_2,t} e_{n,m}^j \leq D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j)$. Combining these equations gives:

$$\begin{aligned} |e_{n,m}^{j+1}| &\leq |e_{n,m}^j + \Delta t D_{x_1,x_2,t} e_{n,m}^j - (1 - \alpha) \Delta t D_{x_1,x_2,t} e_{n,m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)| \\ &= |e_{n,m}^j + \alpha \Delta t D_{x_1,x_2,t} e_{n,m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)| \end{aligned}$$

Case 3 In this case the LeakyReLU in the error equation 15 is applied to a negative term, which implies:

$$e_{n,m}^{j+1} = e_{n,m}^j + \alpha \Delta t (D_{x_1,x_2,t} e_{n,m}^j + D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j)) - \Delta_{+t} y(x_{1n}, x_{2m}, t_j). \quad (19)$$

The term within the activation function in the truncation error (16) is negative as well, such that:

$$T(x_{1n}, x_{2m}, t_j) = \frac{\Delta_{+t} y(x_{1n}, x_{2m}, t_j)}{\Delta t} - \alpha (D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j)). \quad (20)$$

Combining equations (19) and (20) yields:

$$e_{n,m}^{j+1} = e_{n,m}^j + \alpha \Delta t D_{x_1,x_2,t} e_{n,m}^j - \Delta t T(x_{1n}, x_{2m}, t_j).$$

By taking the absolute value of this expression an upper bound for the error is found in this case as well:

$$|e_{n,m}^{j+1}| \leq |e_{n,m}^j + \alpha \Delta t D_{x_1,x_2,t} e_{n,m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|.$$

Case 4 For this case, just like case 3, the activation function in the equation for the error is applied to a negative term, so again:

$$e_{n,m}^{j+1} = e_{n,m}^j + \alpha \Delta t (D_{x_1,x_2,t} e_{n,m}^j + D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j)) - \Delta_{+t} y(x_{1n}, x_{2m}, t_j).$$

However, this time the activation function in the truncation error is applied to a positive term, such that:

$$T(x_{1n}, x_{2m}, t_j) = \frac{\Delta_{+t} y(x_{1n}, x_{2m}, t_j)}{\Delta t} - D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j).$$

Substituting this equation into that of the error (15), gives:

$$e_{n,m}^{j+1} = e_{n,m}^j + \alpha \Delta t D_{x_1,x_2,t} e_{n,m}^j + (\alpha - 1) \Delta t D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j) - \Delta t T(x_{1n}, x_{2m}, t_j).$$

Taking the absolute value of both sides and using the triangle inequality, the following is obtained:

$$|e_{n,m}^{j+1}| = |e_{n,m}^j + \alpha \Delta t D_{x_1,x_2,t} e_{n,m}^j + (\alpha - 1) \Delta t D_{x_1,x_2,t} y(x_{1n}, x_{2m}, t_j)| + \Delta t |T(x_{1n}, x_{2m}, t_j)|. \quad (21)$$

Now, since $D_{x_1, x_2, t} e_{n, m}^j + D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j) < 0$ and $D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j) \geq 0$ are given, it can be seen that:

$$0 \leq D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j) < -D_{x_1, x_2, t} e_{n, m}^j. \quad (22)$$

This case will again, like case 2, be split into two subcases:

- Case 4.1: $e_{n, m}^j + \alpha \Delta t D_{x_1, x_2, t} e_{n, m}^j + (\alpha - 1) \Delta t D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j) \geq 0$
- Case 4.2: $e_{n, m}^j + \alpha \Delta t D_{x_1, x_2, t} e_{n, m}^j + (\alpha - 1) \Delta t D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j) < 0$

Case 4.1: From equation (22) it is known that $D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j) < -D_{x_1, x_2, t} e_{n, m}^j$. Furthermore it was given that the first term on the right hand side of equation (21) is positive, combining this yields:

$$\begin{aligned} |e_{n, m}^{j+1}| &= |e_{n, m}^j + \alpha \Delta t D_{x_1, x_2, t} e_{n, m}^j - (\alpha - 1) \Delta t D_{x_1, x_2, t} e_{n, m}^j + \Delta t |T(x_{1n}, x_{2m}, t_j)| \\ &= |e_{n, m}^j + \Delta t D_{x_1, x_2, t} e_{n, m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|. \end{aligned}$$

Case 4.2: From equation (22) it is given that $0 \leq D_{x_1, x_2, t} y(x_{1n}, x_{2m}, t_j)$. Combining this with the given that the first term of the right hand side of equation (21) is negative, the following can be obtained:

$$|e_{n, m}^{j+1}| = |e_{n, m}^j + \alpha \Delta t D_{x_1, x_2, t} e_{n, m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|.$$

For every case one of the two following bounds hold for the error at time t_{j+1} at position (x_{1n}, x_{2m}) :

1. Bound 1: $|e_{n, m}^{j+1}| \leq |e_{n, m}^j + \Delta t D_{x_1, x_2, t} e_{n, m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|$
2. Bound 2: $|e_{n, m}^{j+1}| \leq |e_{n, m}^j + \alpha \Delta t D_{x_1, x_2, t} e_{n, m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|$

Where bound 1 holds for case 1, case 2.1 and case 4.1 and bound 2 holds for case 2.2, case 3 and case 4.2. Substituting the definition of $D_{x_1, x_2, t}$ and rewriting the result the following is found for bound 1:

$$\begin{aligned} |e_{n, m}^{j+1}| &= |e_{n, m}^j + \beta_1 e_{n, m}^j + \beta_2 \frac{\Delta t}{h} \Delta_{+x_1} e_{n, m}^j + \beta_3 \frac{\Delta t}{k} \Delta_{+x_2} e_{n, m}^j \\ &\quad + \beta_4 \frac{\Delta t}{h^2} \delta_{x_1} e_{n, m}^j + \beta_5 \frac{\Delta t}{k^2} \delta_{x_2} e_{n, m}^j + \beta_6 \frac{\Delta t}{hk} \Delta_{+x_1 x_2} e_{n, m}^j \\ &\quad + \beta_7 \frac{\Delta t}{hk^2} \Delta_{+x_1} \delta_{x_2}^2 e_{n, m}^j + \beta_8 \frac{\Delta t}{kh^2} \Delta_{+x_2} \delta_{x_1}^2 e_{n, m}^j \\ &\quad + \beta_9 \frac{\Delta t}{k^2 h^2} \delta_{x_1}^2 \delta_{x_2}^2 e_{n, m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|. \end{aligned}$$

Rearranging these terms yields:

$$\begin{aligned}
|e_{n,m}^{j+1}| = & \left| \left[1 + \Delta t \left(\beta_1 - \frac{\beta_2}{h} - \frac{\beta_3}{k} - \frac{2\beta_4}{h^2} - \frac{2\beta_5}{k^2} + \frac{2\beta_7}{hk^2} + \frac{2\beta_8}{h^2k} + \frac{4\beta_9}{h^2k^2} \right) \right] e_{n,m}^j \right. \\
& + \Delta t \left[\frac{-\beta_6}{4hk} + \frac{\beta_8}{h^2k} + \frac{\beta_9}{h^2k^2} \right] e_{n-1,m+1}^j \\
& + \Delta t \left[\frac{\beta_3}{k} + \frac{\beta_5}{k^2} - \frac{\beta_7}{hk^2} + \frac{2\beta_8}{h^2k} - \frac{2\beta_9}{h^2k^2} \right] e_{n,m+1}^j \\
& + \Delta t \left[\frac{\beta_6}{4hk} + \frac{\beta_7}{hk^2} + \frac{\beta_8}{h^2k} + \frac{\beta_9}{h^2k^2} \right] e_{n+1,m+1}^j \\
& + \Delta t \left[\frac{\beta_4}{h^2} - \frac{\beta_8}{h^2k} - \frac{2\beta_9}{h^2k^2} \right] e_{n-1,m}^j \\
& + \Delta t \left[\frac{\beta_2}{k} + \frac{\beta_4}{h^2} - \frac{2\beta_7}{hk^2} - \frac{\beta_8}{h^2k} - \frac{2\beta_9}{h^2k^2} \right] e_{n+1,m}^j \\
& + \Delta t \left[\frac{\beta_6}{4hk} + \frac{\beta_9}{h^2k^2} \right] e_{n-1,m-1}^j \\
& + \Delta t \left[\frac{\beta_5}{k^2} - \frac{\beta_7}{hk^2} - \frac{2\beta_9}{h^2k^2} \right] e_{n,m-1}^j \\
& \left. + \Delta t \left[\frac{-\beta_6}{4hk} + \frac{\beta_7}{hk^2} + \frac{\beta_9}{h^2k^2} \right] e_{n+1,m-1}^j \right| + \Delta t |T(x_{1n}, x_{2m}, t_j)|.
\end{aligned} \tag{23}$$

The terms in front of the error terms at time t_j are exactly the parametrizations of the weights θ defined in equation (6) in Section 3.2. This means that equation (23) can be rewritten as:

$$\begin{aligned}
|e_{n,m}^{j+1}| = & \left| (1 + \Delta t\theta_5)e_{n,m}^j + \Delta t\theta_1e_{n-1,m+1}^j + \Delta t\theta_2e_{n,m+1}^j + \Delta t\theta_3e_{n+1,m+1}^j \right. \\
& + \Delta t\theta_4e_{n-1,m}^j + \Delta t\theta_6e_{n+1,m}^j + \Delta t\theta_7e_{n-1,m-1}^j \\
& \left. + \Delta t\theta_8e_{n,m-1}^j + \Delta t\theta_9e_{n+1,m-1}^j \right| + \Delta t |T(x_{1n}, y_m, t_j)|.
\end{aligned}$$

Using the triangle inequality, this gives the following:

$$\begin{aligned}
|e_{n,m}^{j+1}| \leq & |(1 + \Delta t\theta_5)e_{n,m}^j| + |\Delta t\theta_1e_{n-1,m+1}^j| + |\Delta t\theta_2e_{n,m+1}^j| + |\Delta t\theta_3e_{n+1,m+1}^j| \\
& + |\Delta t\theta_4e_{n-1,m}^j| + |\Delta t\theta_6e_{n+1,m}^j| + |\Delta t\theta_7e_{n-1,m-1}^j| \\
& + |\Delta t\theta_8e_{n,m-1}^j| + |\Delta t\theta_9e_{n+1,m-1}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|.
\end{aligned} \tag{24}$$

Now an important assumption will be made, nonnegativity will be assumed for all weights except θ_5 and nonnegativity of $1 + \Delta t\theta_5$ is assumed. These will be two conditions that need to be imposed for stability:

- $\theta_i \geq 0$ for $i \neq 5$
- $1 + \Delta t\theta_5 \geq 0$

If these conditions are met, equation (24) can be rewritten such that:

$$\begin{aligned}
|e_{n,m}^{j+1}| \leq & (1 + \Delta t\theta_5)|e_{n,m}^j| + \Delta t\theta_1|e_{n-1,m+1}^j| + \Delta t\theta_2|e_{n,m+1}^j| + \Delta t\theta_3|e_{n+1,m+1}^j| \\
& + \Delta t\theta_4|e_{n-1,m}^j| + \Delta t\theta_6|e_{n+1,m}^j| + \Delta t\theta_7|e_{n-1,m-1}^j| \\
& + \Delta t\theta_8|e_{n,m-1}^j| + \Delta t\theta_9|e_{n+1,m-1}^j| + \Delta t\Delta t |T(x_{1n}, x_{2m}, t_j)|.
\end{aligned} \tag{25}$$

Now suppose the maximum absolute error at time t_j is defined as E^j :

$$E^j := \max \{ |e_{n,m}^j| : n \in (1, 2, \dots, N), m \in (1, 2, \dots, M) \}.$$

Combining the definition of E^j with equation (25) yields the following:

$$\begin{aligned} |e_{n,m}^{j+1}| \leq & (1 + \Delta t \theta_5) E^j + \Delta t \theta_1 E^j + \Delta t \theta_2 E^j + \Delta t \theta_3 E^j + \Delta t \theta_4 E^j \\ & + \Delta t \theta_6 E^j + \Delta t \theta_7 E^j + \Delta t \theta_8 E^j + \Delta t \theta_9 E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|. \end{aligned}$$

Rewriting this expression gives the following form:

$$|e_{n,m}^{j+1}| \leq \left(1 + \Delta t \sum_{i=1}^9 \theta_i \right) E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|. \quad (26)$$

Doing the same for bound two, which was given by:

$$|e_{n,m}^{j+1}| \leq |e_{n,m}^j + \alpha \Delta t D_{x_1, x_2, t} e_{n,m}^j| + \Delta t |T(x_{1n}, x_{2m}, t_j)|,$$

and assuming $\theta_i \geq 0$ for $i \neq 5$ and $1 + \alpha \Delta t \theta_5 \geq 0$ gives the bound:

$$|e_{n,m}^{j+1}| \leq \left(1 + \alpha \Delta t \sum_{i=1}^9 \theta_i \right) E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|.$$

Summarizing, the following has been found so far, given $\theta_i \geq 0$ for $i \neq 5$:

- Bound 1: $|e_{n,m}^{j+1}| \leq \left(1 + \Delta t \sum_{i=1}^9 \theta_i \right) E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|$, given $1 + \Delta t \theta_5 \geq 0$
- Bound 2: $|e_{n,m}^{j+1}| \leq \left(1 + \alpha \Delta t \sum_{i=1}^9 \theta_i \right) E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|$, given $1 + \alpha \Delta t \theta_5 \geq 0$

Another condition will be imposed on the weights, the sum of the weights should add up to zero:

- $\sum_{i=1}^9 \theta_i = 0$.

This changes the previous bounds to:

- Bound 1: $|e_{n,m}^{j+1}| \leq E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|$, given $1 + \Delta t \theta_5 \geq 0$
- Bound 2: $|e_{n,m}^{j+1}| \leq E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|$, given $1 + \alpha \Delta t \theta_5 \geq 0$

Thus these bounds reduce to the same bound. However the conditions under which these hold are still different. Rewriting both conditions, gives the following forms:

$$\Delta t \leq \frac{1}{-\theta_5} \quad \text{and} \quad \Delta t \leq \frac{1}{-\alpha \theta_5}. \quad (27)$$

This uses the given that $\theta_5 \leq 0$, since all other weights are required to be positive and the sum of the weights should be 0. Now since $0 < \alpha < 1$ and $\theta_5 \leq 0$, the first condition from equation (27) is the strongest. The other condition is satisfied as well as the first is. Thus, obtained so far is the bound:

$$|e_{n,m}^{j+1}| \leq E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|. \quad (28)$$

Given conditions:

- $\theta_i \geq 0$ for $i \neq 5$
- $\sum_{i=1}^9 \theta_i = 0$
- $\Delta t \leq \frac{1}{-\theta_5}$

Since equation (28) holds for all $n \in \{1, 2, \dots, N\}$ and $m \in \{1, 2, \dots, M\}$ it also holds for the position at which $|e_{n,m}^j|$ reaches its maximum at time t_{j+1} , such that:

$$E^{j+1} \leq E^j + \Delta t |T(x_{1n}, x_{2m}, t_j)|.$$

Using an induction argument it can be shown that, given $E^0 = 0$, this implies:

$$E^{j+1} \leq n \Delta t |T(x_{1n}, x_{2m}, t_j)|.$$

Now, since the truncation error is bounded, see appendix E:

$$E^{j+1} \rightarrow 0 \quad \text{as } \Delta t \rightarrow 0.$$

Given the following conditions hold:

- $\theta_i \geq 0$ for $i \neq 5$
- $\sum_{i=1}^9 \theta_i = 0$
- $|\Delta t \leq \frac{1}{-\theta_5}$

Meaning stability is ensured given these conditions.

E Truncation error

A bound for the truncation error can be found. Each term in the truncation error can be expanded using Taylor series expansions.

E.1 Taylor series expansions

The Taylor series expansions around (x_1, x_2) for each of the finite differences used in the discretization are to be obtained. For the first order finite difference the following can be found:

$$\begin{aligned} \Delta_{+x_1} y(x_1, x_2, t) &= y(x_1 + h, x_2, t) - y(x_1, x_2, t) \\ &= y(x_1, x_2, t) + h y_{x_1} + \frac{1}{2} h^2 y_{x_1 x_1} + \frac{1}{6} h^3 y_{x_1 x_1 x_1} + \dots - y(x_1, x_2, t) \\ &= h y_{x_1} + \frac{1}{2} h^2 y_{x_1 x_1} + \frac{1}{6} h^3 y_{x_1 x_1 x_1} + \dots \\ \Delta_{+x_2} y(x_1, x_2, t) &= k y_{x_2} + \frac{1}{2} k^2 y_{x_2 x_2} + \frac{1}{6} k^3 y_{x_2 x_2 x_2} + \dots \\ \Delta_{+t} y(x_1, x_2, t) &= \Delta t y_t(x_1, x_2, t) + \frac{1}{2} \Delta t^2 y_{tt} + \frac{1}{6} \Delta t^3 y_{ttt} + \dots \end{aligned}$$

Now Lagrange remainder terms can be introduced, which denote the truncation error for the specific Taylor approximation, for more on the Lagrange remainder see [2]. Let

$\xi_{\Delta x_1} \in (x_1 - h, x_1 + h)$, $\gamma_{\Delta x_2} \in (x_2 - k, x_2 + k)$ and $s_t \in (t, t + \Delta t)$. These can be chosen such that:

$$\begin{aligned}\Delta_{+x_1}y(x_1, x_2, t) &= hy_{x_1}(x_1, x_2, t) + \frac{1}{2}\Delta ty_{tt}(\xi_{\Delta x_1}, x_2, t) \\ \Delta_{+x_2}y(x_1, x_2, t) &= ky_{x_2}(x_1, x_2, t) + \frac{1}{2}k^2y_{x_2x_2}(x_1, \gamma_{\Delta x_2}, t) \\ \Delta_{+t}y(x_1, x_2, t) &= \Delta ty_t(x_1, x_2, t) + \frac{1}{2}\Delta t^2y_{tt}(x_1, x_2, s_t)\end{aligned}$$

For the second order central differences the following can be found, introducing remainders $\xi_{\delta x_1} \in (x_1 - h, x_1 + h)$ and $\gamma_{\delta x_2} \in (x_2 - k, x_2 + k)$:

$$\begin{aligned}\delta_{x_1}^2y(x_1, x_2, t) &= y(x_1 + h, x_2, t) - 2y(x_1, x_2, t) + y(x_1 - h, x_2, t) \\ &= y(x_1, x_2, t) + hy_{x_1} + \frac{1}{2}h^2y_{x_1x_1} + \frac{1}{6}h^3y_{x_1x_1x_1} + \frac{1}{24}h^4y_{x_1x_1x_1x_1} + \dots \\ &\quad - 2y(x_1, x_2, t) + y(x_1, x_2, t) - hy_{x_1} + \frac{1}{2}h^2y_{x_1x_1} \\ &\quad - \frac{1}{6}h^3y_{x_1x_1x_1} + \frac{1}{24}h^4y_{x_1x_1x_1x_1} + \dots \\ &= h^2y_{x_1x_1}(x_1, x_2, t) + \frac{1}{12}h^4y_{x_1x_1x_1x_1}(\xi_{\delta x_1}, x_2, t), \\ \delta_{x_2}^2y(x_1, x_2, t) &= k^2y_{x_2x_2}(x_1, x_2, t) + \frac{1}{12}k^4y_{x_2x_2x_2x_2}(x_1, \gamma_{\delta x_2}, t).\end{aligned}$$

For the second order mixed forward difference the following can be found, letting $\xi_{\Delta+x_1x_2}^{(1)} \in (x_1, x_1 + h)$, $\xi_{\Delta+x_1x_2}^{(2)} \in (x_1, x_1 + h)$, $\gamma_{\Delta+x_1x_2}^{(1)} \in (x_2, x_2 + h)$ and $\gamma_{\Delta+x_1x_2}^{(2)} \in (x_2, x_2 + h)$:

$$\begin{aligned}\Delta_{+x_1x_2}y(x_1, x_2, t) &= y(x_1 + h, x_2 + k, t) \\ &\quad - y(x_1, x_2 + k, t) - y(x_1 + h, x_2, t) + y(x_1, x_2, t) \\ &= hky_{x_1x_2}(x_1, x_2, t) + \frac{1}{2}h^2ky_{x_1x_1x_2}(\xi_{\Delta+x_1x_2}^{(1)}, \gamma_{\Delta+x_1x_2}^{(1)}, t) \\ &\quad + \frac{1}{2}hk^2y_{x_1x_2x_2}(\xi_{\Delta+x_1x_2}^{(2)}, \gamma_{\Delta+x_1x_2}^{(2)}, t).\end{aligned}$$

For the third order mixed central difference followed by forward difference the following holds, if $\xi_{\Delta+x_1\delta_{x_2}^2} \in (x_1, x_1 + h)$ and $\gamma_{\Delta+x_1\delta_{x_2}^2} \in (x_2 - k, x_2 + k)$:

$$\begin{aligned}\Delta_{+x_1}\delta_{x_2}^2y(x_1, x_2, t) &= y(x_1 + h, x_2 + k, t) - 2y(x_1 + h, x_2, t) + y(x_1 + h, x_2 - k, t) \\ &\quad - y(x_1, x_2 + k, t) + 2y(x_1, x_2, t) - y(x_1, x_2 - k, t) \\ &= hk^2y_{x_1x_2x_2}(x_1, x_2, t) + \frac{1}{2}h^2k^2y_{x_1x_1x_2x_2}(\xi_{\Delta+x_1\delta_{x_2}^2}, \gamma_{\Delta+x_1\delta_{x_2}^2}, t),\end{aligned}$$

and let $\xi_{\Delta+x_2\delta_{x_1}^2} \in (x_1 - h, x_1 + h)$ and $\gamma_{\Delta+x_2\delta_{x_1}^2} \in (x_2, x_2 + k)$, such that:

$$\Delta_{+x_2}\delta_{x_1}^2y(x_1, x_2, t) = hk^2y_{x_1x_2x_2}(x_1, x_2, t) + \frac{1}{2}h^2k^2y_{x_1x_1x_2x_2}(\xi_{\Delta+x_2\delta_{x_1}^2}, \gamma_{\Delta+x_2\delta_{x_1}^2}, t).$$

Given some:

- $\xi_{\delta_{x_1}^2\delta_{x_2}^2}^{(1)} \in (x_1 - h, x_1 + h)$
- $\xi_{\delta_{x_1}^2\delta_{x_2}^2}^{(2)} \in (x_1 - h, x_1 + h)$

- $\gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)} \in (x_2 - k, x_2 + k)$
- $\gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)} \in (x_2 - k, x_2 + k)$

the following holds for the fourth order mixed central difference:

$$\begin{aligned}
\delta_{x_1}^2 \delta_{x_2}^2 y(x_1, x_2, t) &:= y(x_1 + h, x_2 + k, t) - 2y(x_1 + h, x_2, t) + y(x_1 + h, x_2 - k, t) \\
&\quad - 2y(x_1, x_2 + k, t) + 4y(x_1, x_2, t) - 2y(x_1, x_2 - k, t) \\
&\quad + y(x_1 - h, x_2 + k, t) - 2y(x_1 - h, x_2, t) + y(x_1 - h, x_2 - k, t) \\
&= hk y_{x_1 x_1 x_2 x_2}(x_1, x_2, t) + \frac{1}{12} h^4 k^2 y_{x_1 x_1 x_1 x_1 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, t) \\
&\quad + h^2 k^4 \frac{1}{12} y_{x_1 x_1 x_2 x_2 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, t).
\end{aligned}$$

E.2 Bound for truncation error

The truncation error is previously defined in Section 3.3 as:

$$T(x_1, x_2, t) = \frac{\Delta_{+t} y(x_1, x_2, t)}{\Delta t} - \sigma(D_{x_1, x_2, t} y(x_1, x_2, t)).$$

Substituting the found Taylor expansions into the equation for the truncation error, the following is obtained:

$$\begin{aligned}
T(x_1, x_2, t) &= y_t(x_1, x_2, t) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, t) - \sigma(\beta_1 y(x_1, x_2, t)) \\
&\quad + \beta_2 (y_{x_1}(x_1, x_2, t) + \frac{1}{2} h y_{x_1 x_1}(x_1, x_2, t)) \\
&\quad + \beta_3 (y_{x_2}(x_1, x_2, t) + \frac{1}{2} k y_{x_2 x_2}(x_1, x_2, t)) \\
&\quad + \beta_4 (y_{x_1 x_1}(x_1, x_2, t) + \frac{1}{12} h^2 y_{x_1 x_1 x_1 x_1}(x_1, x_2, t)) \\
&\quad + \beta_5 (y_{x_2 x_2}(x_1, x_2, t) + \frac{1}{12} k^2 y_{x_2 x_2 x_2 x_2}(x_1, x_2, t)) \\
&\quad + \beta_6 (y_{x_1 x_2}(x_1, x_2, t) + \frac{1}{2} h y_{x_1 x_1 x_2}(\xi_{\Delta_{+x_1 x_2}}^{(1)}, \gamma_{\Delta_{+x_1 x_2}}^{(1)}, t) \\
&\quad + \frac{1}{2} k y_{x_1 x_2 x_2}(\xi_{\Delta_{+x_1 x_2}}^{(2)}, \gamma_{\Delta_{+x_1 x_2}}^{(2)}, t)) \\
&\quad + \beta_7 (y_{x_1 x_2 x_2}(x_1, x_2, t) + h \frac{1}{2} y_{x_1 x_1 x_2 x_2}(\xi_{\Delta_{+x_1} \delta_{x_2}^2}, \gamma_{\Delta_{+x_1} \delta_{x_2}^2}, t)) \\
&\quad + \beta_8 (y_{x_1 x_1 x_2}(x_1, x_2, t) + k \frac{1}{2} y_{x_1 x_1 x_2 x_2}(\xi_{\Delta_{+x_1} \delta_{x_2}^2}, \gamma_{\Delta_{+x_2} \delta_{x_1}^2}, t)) \\
&\quad + \beta_9 (y_{x_1 x_1 x_2 x_2}(x_1, x_2, t) + \frac{1}{12} h^2 y_{x_1 x_1 x_1 x_1 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, t) \\
&\quad + \frac{1}{12} k^2 y_{x_1 x_1 x_2 x_2 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, t)).
\end{aligned}$$

Rearranging this equation gives:

$$\begin{aligned}
T(x_1, x_2, t) = & y_t(x_1, x_2, t) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - \sigma(\beta_1 y(x_1, x_2, t) \\
& + \beta_2 y_{x_1}(x_1, x_2, t) + \beta_3 y_{x_2}(x_1, x_2, t) + \beta_4 y_{x_1 x_1}(x_1, x_2, t) \\
& + \beta_5 y_{x_2 x_2}(x_1, x_2, t) + \beta_6 y_{x_1 x_2}(x_1, x_2, t) + \beta_7 y_{x_1 x_2 x_2}(x_1, x_2, t) \\
& + \beta_8 y_{x_1 x_1 x_2}(x_1, x_2, t) + \beta_9 y_{x_1 x_1 x_2 x_2}(x_1, x_2, t)) \\
& + (\beta_2 \frac{1}{2} h y_{x_1 x_1}(\gamma_{\Delta x_1}, x_2, t) + \beta_3 \frac{1}{2} k y_{x_2 x_2}(x_1, \gamma_{\Delta x_2}, t) \\
& + \beta_4 \frac{1}{12} h^2 y_{x_1 x_1 x_1 x_1}(\gamma_{\delta x_1}, x_2, t) \\
& + \beta_5 \frac{1}{12} k^2 y_{x_2 x_2 x_2 x_2}(x_1, \gamma_{\delta x_2}, t) \\
& + \beta_6 (\frac{1}{2} h y_{x_1 x_1 x_2}(\xi_{\Delta+x_1 x_2}^{(1)}, \gamma_{\Delta+x_1 x_2}^{(1)}, t) + \frac{1}{2} k y_{x_1 x_2 x_2}(\xi_{\Delta+x_1 x_2}^{(2)}, \gamma_{\Delta+x_1 x_2}^{(2)}, t)) \\
& + \beta_7 k \frac{1}{2} y_{x_1 x_1 x_2 x_2}(\xi_{\Delta+x_2 \delta_{x_1}^2}, \gamma_{\Delta+x_2 \delta_{x_1}^2}, t) \\
& + \beta_8 k \frac{1}{2} y_{x_1 x_1 x_2 x_2}(\xi_{\Delta+x_1 \delta_{x_2}^2}, \gamma_{\Delta+x_2 \delta_{x_1}^2}, t) \\
& + \beta_9 \frac{1}{12} (h^2 y_{x_1 x_1 x_1 x_1 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, t) \\
& + k^2 y_{x_1 x_1 x_2 x_2 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, t))).
\end{aligned} \tag{29}$$

For notational reasons the following are defined:

$$\begin{aligned}
T_s(y) & := \beta_1 y + \beta_2 y_{x_1} + \beta_3 y_{x_2} + \beta_4 y_{x_1 x_1} + \beta_5 y_{x_2 x_2} \\
& + \beta_6 y_{x_1 x_2} + \beta_7 y_{x_1 x_2 x_2} + \beta_8 y_{x_1 x_1 x_2} + \beta_9 y_{x_1 x_1 x_2 x_2}, \\
T_l(y) & := \beta_2 \frac{1}{2} h y_{x_1 x_1}(\gamma_{\Delta x_1}, x_2, t) + \beta_3 \frac{1}{2} k y_{x_2 x_2}(x_1, \gamma_{\Delta x_2}, t) \\
& + \beta_4 \frac{1}{12} h^2 y_{x_1 x_1 x_1 x_1}(\gamma_{\delta x_1}, x_2, t) + \beta_5 \frac{1}{12} k^2 y_{x_2 x_2 x_2 x_2}(x_1, \gamma_{\delta x_2}, t) \\
& + \beta_6 \frac{1}{2} (h y_{x_1 x_1 x_2}(\xi_{\Delta+x_1 x_2}^{(1)}, \gamma_{\Delta+x_1 x_2}^{(1)}, t) + k y_{x_1 x_2 x_2}(\xi_{\Delta+x_1 x_2}^{(2)}, \gamma_{\Delta+x_1 x_2}^{(2)}, t)) \\
& + \beta_7 \frac{1}{2} h y_{x_1 x_1 x_2 x_2}(\xi_{\Delta+x_2 \delta_{x_1}^2}, \gamma_{\Delta+x_2 \delta_{x_1}^2}, t) \\
& + \beta_8 \frac{1}{2} k y_{x_1 x_1 x_2 x_2}(\xi_{\Delta+x_1 \delta_{x_2}^2}, \gamma_{\Delta+x_2 \delta_{x_1}^2}, t) \\
& + \beta_9 \frac{1}{12} (h^2 y_{x_1 x_1 x_1 x_1 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, t) \\
& + k^2 y_{x_1 x_1 x_2 x_2 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, t)).
\end{aligned} \tag{30}$$

Using this notation equation (29) can be rewritten as the following:

$$T(x_1, x_2, t) = y_t(x_1, x_2, t) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - \sigma(T_s(y) + T_l(y)).$$

Now four cases will be considered:

- Case 1: $T_s(y) + T_l(y) \geq 0$ and $T_s(y) \geq 0$
- Case 2: $T_s(y) + T_l(y) \geq 0$ and $T_s(y) < 0$

- Case 3: $T_s(y) + T_l(y) < 0$ and $T_s(y) \geq 0$
- Case 4: $T_s(y) + T_l(y) < 0$ and $T_s(y) < 0$

Case 1: In this case the truncation error equals:

$$T(x_1, x_2, t) = y_t(x_1, x_2, t) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - (T_s(y) + T_l(y)). \quad (31)$$

Now, since $T_s(y) \geq 0$:

$$T_s(y) = \sigma(T_s(s)).$$

Substituting this back into equation (31) yields:

$$T(x_1, x_2, t) = y_t(x_1, x_2, t) - \sigma(T_s(y)) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - T_l(y). \quad (32)$$

Here the first part satisfies the partial differential equation at hand, such that the truncation error is given by:

$$T(x_1, x_2, t) = \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - T_l(y). \quad (33)$$

Taking the absolute value of both sides yields:

$$\begin{aligned} |T(x_1, x_2, t)| &= \left| \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - T_l(y) \right| \\ &\leq \left| \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) \right| + |T_l(y)|. \end{aligned}$$

Case 2: In this case the truncation error is the same as equation (31). Taking the absolute value of this expression yields:

$$|T(x_1, x_2, t)| = \left| y_t(x_1, x_2, t) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - (T_s(y) + T_l(y)) \right|. \quad (34)$$

Now two subcases will be considered:

- Case 2.1: $y_t(x_1, x_2, t) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - (T_s(y) + T_l(y)) \geq 0$
- Case 2.2: $y_t(x_1, x_2, t) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - (T_s(y) + T_l(y)) < 0$

Case 2.1: Since $T_s(y) < 0$ and $T_s(y) + T_l(y) + \geq 0$:

$$|T_s(y)| \leq |T_l(y)|. \quad (35)$$

Using that $y_t(x_1, x_2, t) - \sigma(T_s(y)) = y_t(x_1, x_2, t) - \alpha(T_s(y)) = 0$, since it satisfies the partial differential equation and combining this with equation (34), the following is obtained:

$$\begin{aligned} |T(x_1, x_2, t)| &= |(\alpha - 1)T_s(y) + \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) - T_l(y)| \\ &\leq (1 - \alpha)|T_s(y)| + |T_l(y)| + \left| \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) \right|. \end{aligned}$$

Combining this with equation (35), yields the following upper bound:

$$\begin{aligned} |T(x_1, x_2, t)| &\leq (1 - \alpha)|T_l(y)| + |T_l(y)| + \left| \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) \right| \\ &\leq 2|T_l(y)| + \left| \frac{1}{2}\Delta ty_{tt}(x_1, x_2, s_t) \right|. \end{aligned}$$

Case 2.2: It is given that $T_s(y) < 0$ and $0 < \alpha < 1$, such that:

$$-T_s(y) > -\alpha T_s(y).$$

Since it is now given that equation (34) is negative, the following upper bound is obtained:

$$|T(x_1, x_2, t)| = |y_t(x_1, x_2, t) - \alpha T_s(y) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - T_l(y)|.$$

Like the previous case 2.1, $y_t(x_1, x_2, t) - \alpha(T_s(y)) = 0$ since it satisfies the PDE, such that:

$$|T(x_1, x_2, t)| = \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) \right| + |T_l(y)|.$$

Case 3: From the given conditions for this case, the following truncation error follows:

$$|T(x_1, x_2, t)| = |y_t(x_1, x_2, t) - \alpha T_s(y) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - \alpha T_l(y)|. \quad (36)$$

Now two subcases will be considered again:

- Case 3.1: $y_t(x_1, x_2, t) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - (T_s(y) + T_l(y)) \geq 0$
- Case 3.2: $y_t(x_1, x_2, t) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - (T_s(y) + T_l(y)) < 0$

Case 3.1: Since it is given that $|T_s + T_l| < 0$ and $T_s \geq 0$, the following holds:

$$|T_s(y)| < |T_l(y)|. \quad (37)$$

It is known that $y_t(x_1, x_2, t) - T_s(y) = y_t(x_1, x_2, t) - \sigma(T_s(y)) = 0$ since $T_s(y) > 0$ and this satisfies the PDE, combining this with the inequality (37) and expression for the truncation error, see equation (36), gives:

$$\begin{aligned} |T(x_1, x_2, t)| &= |(1 - \alpha)T_s(y) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - \alpha T_l(y)| \\ &\leq (1 - \alpha)|T_s(y)| + \alpha|T_l(y)| + \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) \right| \\ &\leq |T_l(y)| + \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) \right|. \end{aligned}$$

Case 3.2: Since $T_s(y) < 0$ and $0 < \alpha < 0$, the following holds:

$$-T_s(y) < -\alpha T_s(y).$$

Since it is given that, in this case, the term within the absolute operator in equation (37) is negative, the following holds:

$$|T(x_1, x_2, t)| \leq |y_t(x_1, x_2, t) - T_s(y) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - \alpha T_l(y)|.$$

The first part within the absolute signs of the right hand of the equation again satisfies the PDE, such that:

$$\begin{aligned} |T(x_1, x_2, t)| &\leq \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - \alpha T_l(y) \right| \\ &\leq \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) \right| + \alpha|T_l(y)|. \end{aligned}$$

Case 4: In this case the truncation error is given by:

$$|T(x_1, x_2, t)| = |y_t(x_1, x_2, t) - \alpha T_s(y) + \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - \alpha T_l(y)|. \quad (38)$$

Since $T_s(y) < 0$, $y_t(x_1, x_2, t) - \alpha T_s(y) = y_t - \sigma T_s(y) = 0$, since it satisfies the PDE. Combining this with equation (38) yields:

$$\begin{aligned} |T(x_1, x_2, t)| &= \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) - \alpha T_l(y) \right| \\ &\leq \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) \right| + \alpha |T_l(y)|. \end{aligned}$$

For each of the cases an upper bound has been found for the absolute error. Cases 3.2. and 4 led to the strictest upper bound, each of the other cases satisfies this bound as well. So from now on this bound will be discussed since everything that follows from this holds for all cases. This bound is:

$$|T(x_1, x_2, t)| \leq \left| \frac{1}{2} \Delta t y_{tt}(x_1, x_2, s_t) \right| + \alpha |T_l(y)|.$$

Substituting the definition of $T_l(y)$ as given by equation (30) and applying the triangle inequality, yields:

$$\begin{aligned} |T(x_1, x_2, t)| &\leq \frac{1}{2} \Delta t |y_{tt}(x_1, x_2, s_t)| + \alpha |\beta_2| \frac{1}{2} h |y_{x_1 x_1}(\gamma_{\Delta x_1}, x_2, t)| \\ &\quad + |\beta_3| \frac{\alpha}{2} k |y_{x_2 x_2}(x_1, \gamma_{\Delta x_2}, t)| \\ &\quad + |\beta_4| \frac{\alpha}{12} h^2 |y_{x_1 x_1 x_1 x_1}(\gamma_{\delta x_1}, x_2, t)| + |\beta_5| \frac{\alpha}{12} k^2 |y_{x_2 x_2 x_2 x_2}(x_1, \gamma_{\delta x_2}, t)| \\ &\quad + |\beta_6| \frac{\alpha}{2} (h |y_{x_1 x_1 x_2}(\xi_{\Delta+x_1 x_2}^{(1)}, \gamma_{\Delta+x_1 x_2}^{(1)}, t)| \\ &\quad + k |y_{x_1 x_2 x_2}(\xi_{\Delta+x_1 x_2}^{(2)}, \gamma_{\Delta+x_1 x_2}^{(2)}, t)|) \\ &\quad + |\beta_7| \frac{\alpha}{2} h |y_{x_1 x_1 x_2 x_2}(\xi_{\Delta+x_2 \delta_{x_1}^2}, \gamma_{\Delta+x_2 \delta_{x_1}^2}, t)| \\ &\quad + |\beta_8| \frac{\alpha}{2} k |y_{x_1 x_1 x_2 x_2}(\xi_{\Delta+x_1 \delta_{x_2}^2}, \gamma_{\Delta+x_1 \delta_{x_2}^2}, t)| \\ &\quad + |\beta_9| \frac{\alpha}{12} (h^2 |y_{x_1 x_1 x_1 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(1)}, t)| \\ &\quad + k^2 |y_{x_1 x_1 x_2 x_2 x_2}(\xi_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, \gamma_{\delta_{x_1}^2 \delta_{x_2}^2}^{(2)}, t)|). \end{aligned}$$

Assuming each of the absolute values in the previous equation is bounded by some term, gives the truncation error a bound of the following form:

$$\begin{aligned} |T(x_1, x_2, t)| &\leq \frac{1}{2} \Delta t M_{tt} + \frac{\alpha}{2} |\beta_2| h M_{x_1 x_1} + \frac{\alpha}{2} |\beta_3| k M_{x_2 x_2} + \frac{\alpha}{12} |\beta_4| h^2 M_{x_1 x_1 x_1 x_1} \\ &\quad + \frac{\alpha}{12} |\beta_5| k^2 M_{x_2 x_2 x_2 x_2} + \frac{\alpha}{2} |\beta_6| [h M_{x_1 x_1 x_2} + k M_{x_1 x_2 x_2}] \\ &\quad + \frac{\alpha}{2} |\beta_7| h M_{x_1 x_1 x_2 x_2} + \frac{\alpha}{2} |\beta_8| k M_{x_1 x_1 x_2 x_2} \\ &\quad + \frac{\alpha}{12} |\beta_9| [h^2 M_{x_1 x_1 x_1 x_2 x_2} + k^2 M_{x_1 x_1 x_2 x_2 x_2}]. \end{aligned}$$

Thus, the truncation error is bounded.

F Kernels

The kernels trained for the eight layer network containing one kernel per layer can be found in Figure 10. Trained on images of 28 by 28 pixels.

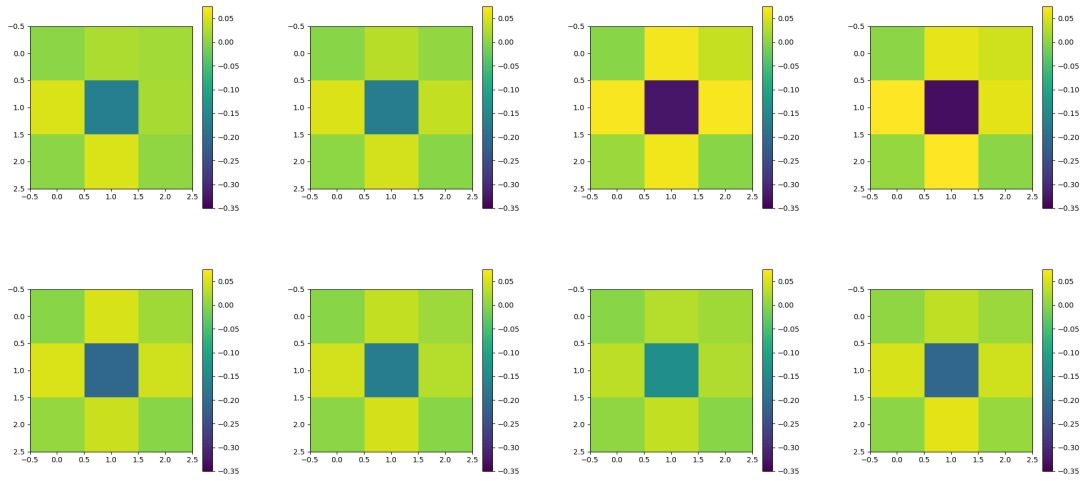


Figure 10: Kernels of the convolutional layers in ascending order (left to right, top to bottom).