Advancing the Fourier Neural Operator: From the FNO to the Wavelet Neural Operator and Beyond

Silas Langenbach

Born 22nd November 1999 in Kirchen, Germany

August 28, 2025

Master's Thesis Mathematics

Advisor: Prof. Dr. Jochen Garcke

Second Advisor: Prof. Dr. Barbara Verfürth

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Acknowledgments

Before we start with this thesis, I would like to take the opportunity to thank for the support I received while I worked on this thesis. First I would like to thank Prof. Dr. Jochen Garcke for his supervision, his support whenever problems arose and for his time for joint meetings. I would also like to express my gratitude towards my former colleges at PAI for their interest in my thesis and their input on machine learning topics. I also wish to thank my friends and family for their support and for providing either encouragement or distraction, depending on what was needed.

Acronyms

PDE partial differential equation
NN neural network
CNN convolutional neural network
FNN fully connected neural network
GNO graph neural operator
FNO Fourier neural operator
WNO wavelet neural operator
MWNO modified wavelet neural operator
FFT fast Fourier transform
DFT discrete Fourier transform
IDFT inverse discrete Fourier transform
FIR-filter finite impulse response filter
db$p$ Daubechies wavelet with p vanishing moments
CWT continuous wavelet transform
DWT discrete wavelet transform
FWT fast wavelet transform
IDWT inverse discrete wavelet transform
GELU Gaussian error linear unit
MSE mean squared error

Contents

Chapter 1

Introduction

Partial differential equations (PDEs) play a crucial role in mathematics, physics, engineering, and various other scientific fields. They provide a rigorous way to describe the evolution of spatial and temporal dynamics. Prominent examples include the transport of heat, the propagation of waves, and the diffusion of substances. Although PDEs are essential for describing these systems, many of them cannot be solved analytically. Therefore we must rely on numerical solvers like the finite element, finite volume, or finite difference methods.
A key motivation for the development of alternatives to the classical numerical solvers is the fact that in many applications we need to solve a PDE for multiple initial conditions. For instance, if we need to model the flow of traffic, the PDE must be solved repeatedly for multiple initial conditions since the traffic changes depending on road closures, the time of the day, the weekday, and school holidays.
Because classical solvers must be restarted every time you change a parameter, for example use a different initial condition, these methods are computationally expensive when solving a PDE for many different parameter settings.
An alternative approach is the usage of neural networks (NNs). Originally two different classes of neural networks were introduced to learn the solutions of PDEs, physics-informed NNs ([32]) and data-driven neural networks ([30], [31]). Both classes have downsides. For the use of physics-informed neural networks you need to know the physical realities of your data set before constructing the NN, but in many cases these are not known. Data-driven neural networks on the other hand do not capture the physical dynamics of the training data and fail to predict these dynamics for test data if the test data is outside of the training data distribution.
This is why a third form called neural operators was introduced. These neural operators learn the nonlinear mapping described by the PDE between two infinite dimensional function spaces, e.g. the space of initial conditions and the space of solutions of a PDE, and they rely on the universal operator approximation theorem from [29].
Among the first published neural operators were the DeepONet operator ([14], [15]) and the graph neural operator (GNO, [16]). Afterwards the Fourier Neural Operator (FNO) was published in [7] which will be one of the main topics of this thesis. The FNO applies the Fourier transform on the input data and then learns in both physical and frequency space.
As an improvement to the FNO, the Wavelet Neural Operator (WNO) was introduced in [1]. This operator uses a wavelet transform instead of a Fourier transform. This adjustment will be investigated later in this thesis.
Now we provide a brief outline of this thesis. We start by introducing the theoretical basis such as the Fourier and wavelet transforms and we study the network architecture of the

FNO. Additionally, we explore several strategies to further improve the accuracy of the FNO, including the introduction of the WNO, parameter tuning, modification of the network architecture and alternative approaches to the boundary handling during the discrete wavelet transform.
This thesis will be structured in six chapters. First, we start by briefly providing the theoretical background of the Fourier transform. Here we look at the continuous and discrete Fourier transform and the implementation of the fast Fourier transform (FFT).
The second chapter is focused on wavelets and will cover the derivation by showing filter banks, the wavelet equation, the continuous and discrete wavelet transform and the different methods of boundary treatments in the discrete wavelet transform.
In the next chapter, we will define the benchmark problems which are used for comparison of the different models. For that we will state the PDEs that are used to test the different neural operators and describe how the training data is generated.
Afterwards, we will introduce the FNO by stating the formal problem formulation, looking at the original network architecture of the FNO as it was introduced in [7] and look at the different numerical results from various papers and our own numerical results.
In the fifth chapter, we search for different approaches to improve the performance of the FNO. Here we will introduce the WNO and also discuss other changes to the network architecture in order to improve the ability of the WNO to correctly predict the solutions of the PDEs.
We end this thesis with a discussion of the results and an assessment of the potential of neural operators.
Our contribution to this topic is the use of boundary filter to treat the signal boundaries in the WNO instead of signal extensions like in the original WNO paper [1] and the use of a bias term and noise-augmented training data during the WNO model training as well as a comparison between the various models we will introduce during this thesis.

Chapter 2

Fourier Transform

The neural operators covered in this thesis rely on discrete versions of integral transformations to learn mappings between infinite-dimensional function spaces. One of the most widely known and used transformation in this context is the Fourier transform. It serves as the foundation of the FNO. The FNO aims to exploit the efficiency of the Fourier transform in capturing global patterns.
In this chapter we define the key concepts of the Fourier transform, present both the continuous and discrete Fourier transforms, and the Fast Fourier Transform (FFT) algorithm. These concepts will be used multiple times in this thesis, such that it is important to start here with the basic concepts.

2.1 Continuous Fourier Transform

The continuous Fourier transform is an integral operator that decomposes a function into its constituent frequencies. It provides a powerful tool for analyzing patterns in functions. The transformation maps a function from the spatial domain to the frequency domain which enables the representation of a function in terms of a basis of complex sine and cosine functions. The Fourier transform is defined by the following integral operator.

Definition 2.1.1
Let $f \in L^1(\mathbb{R}^n)$, then the Fourier transform $\mathcal{F}$ splits f into its frequency components $\hat{f}(y)$ which are defined by

$$\mathcal{F}f(y) := \hat{f}(y) := \int_{\mathbb{R}^n} f(x)e^{-iy\cdot x}\,dx.$$

The inverse Fourier transform reconstructs the function f from its frequency components:

$$(\mathcal{F}^{-1}\hat{f})(x) := \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} \mathcal{F}f(y)e^{iy\cdot x}\,dy.$$

The proof that the inverse Fourier transform is really the inverse of the Fourier transform is a known result. We will not cover this in this thesis since we will focus on the discrete version but the proof can be found for example in [17].

2.2 Discrete Fourier Transform

In practice, we need to implement our algorithms for computers and that is why we need discrete versions of our concepts. In the case of the Fourier transform we define a discrete

Fourier transform which does not use a continuous input signal f but a discrete vector x and transforms this into a vector X where each element represents a frequency which is a component of the input signal.

Definition 2.2.1
For a discrete sequence $x = \{x_n\}_{n=0}^{N-1}$ we define the discrete Fourier transform (DFT) as

$$\mathcal{F}_k(x) := X_k := \sum_{n=0}^{N-1} x_n e^{-2\pi i \frac{nk}{N}} \text{ for } k = 0, ..., N-1.$$

For the reconstruction of the signal x we can define the inverse discrete Fourier transform (IDFT) as

$$x_k = \mathcal{F}_k^{-1}(X) = \frac{1}{N} \sum_{n=0}^{N-1} e^{2\pi i \frac{nk}{N}} \cdot X_n \text{ for } k = 0, ..., N-1.$$

Given the DFT and IDFT defined as above we now want to prove that the IDFT is really the inverse of the DFT.

Proof. Let $\tilde{x}_k$ denote the result of applying the IDFT to $\{X_n\}$:

$$\begin{aligned}
\tilde{x}_k &= \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{2\pi i \frac{nk}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} \left(\sum_{m=0}^{N-1} x_m e^{-2\pi i \frac{mn}{N}} \right) e^{2\pi i \frac{nk}{N}} \\
&= \frac{1}{N} \sum_{m=0}^{N-1} x_m \sum_{n=0}^{N-1} e^{2\pi i \frac{n(k-m)}{N}} \text{ with } k, m = 0, ..., N-1.
\end{aligned}$$

The inner sum evaluates to:

$$\sum_{n=0}^{N-1} e^{2\pi i \frac{n(k-m)}{N}} = \begin{cases} N & \text{if } k = m \\ 0 & \text{otherwise.} \end{cases}$$

- **Case $k = m$:** The exponent vanishes ($e^0 = 1$), so the sum equals N.
- **Case $k \neq m$:** The sum is a geometric series with ratio $r = e^{2\pi i \frac{(k-m)}{N}} \neq 1$:

$$\sum_{n=0}^{N-1} r^n = \frac{1 - r^N}{1 - r} = \frac{1 - e^{2\pi i (k-m)}}{1 - e^{2\pi i \frac{(k-m)}{N}}} = 0,$$

since $e^{2\pi i (k-m)} = 1$ for integers k, m.

Thus, only the term where $m = k$ survives:

$$\tilde{x}_k = \frac{1}{N} \cdot x_k \cdot N = x_k.$$

Therefore $\mathcal{F}^{-1}(\mathcal{F}(x)) = \mathbf{x}$. The other direction $\mathcal{F}(\mathcal{F}^{-1}(X)) = X$ can be done analogously. This proofs that the IDFT is really the inverse of the DFT. □

2.3 Fast Fourier Transform

In many programs the DFT is called very often such that it is important to optimize the runtime of our programs. Therefore we need a fast algorithm to calculate the discrete Fourier transform. This can be achieved by the Fast Fourier Transform (FFT).
The fundamental result for constructing the FFT algorithm is presented in the following theorem from [10].

Theorem 2.3.1
Assume two discrete data sets of length N $x_0, ..., x_{N-1}$ and $x_N, ..., x_{2N-1}$. Then the discrete Fourier transform of the data set $x_0, x_N, x_1, x_{N+1}, ..., x_{N-1}, n_{2N-1}$ can be calculated from the transformed subsets as in the following:

$$\begin{aligned}
&\frac{1}{2}(\mathcal{F}_k(x_0, ..., x_{N-1}) + e^{-i\pi\frac{k}{N}}\mathcal{F}_k(x_N, ..., x_{2N-1})) \\
&= \mathcal{F}_k(x_0, x_N, x_1, x_{N+1}, ..., x_{N-1}, n_{2N-1}) \text{ for } k = 0, ..., N-1 \\
&\frac{1}{2}(\mathcal{F}_k(x_0, ..., x_{N-1}) - e^{-i\pi\frac{k}{N}}\mathcal{F}_k(x_N, ..., x_{2N-1})) \\
&= \mathcal{F}_{N+k}(x_0, x_N, x_1, x_{N+1}, ..., x_{N-1}, n_{2N-1}) \text{ for } k = 0, ..., N-1.
\end{aligned}$$

Proof. For $k = 0, ..., N-1$ we have

$$\begin{aligned}
&\mathcal{F}_k(x_0, x_N, x_1, x_{N+1}, ..., x_{N-1}, n_{2N-1}) \\
&= \frac{1}{2N}(\sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{2nk}{2N}} + \sum_{n=0}^{N-1} x_{N+n} e^{-i2\pi\frac{(2n+1)k}{2N}}) \\
&= \frac{1}{2N}(\sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{nk}{N}} + e^{\frac{-ik\pi}{N}} \sum_{n=0}^{N-1} x_{N+n} e^{-i2\pi\frac{nk}{N}}) \\
&= \frac{1}{2}(\mathcal{F}_k(x_0, ..., x_{N-1}) + e^{-i\pi\frac{k}{N}}\mathcal{F}_k(x_N, ..., x_{2N-1}))
\end{aligned}$$

The second equation can be achieved analog. □

For stating the algorithm we define an operator which reverses the bit representation of integers.

Definition 2.3.2
Let $q \in \mathbb{N}_0$ and $n = \sum_{l=0}^{q-1} b_l 2^l$ the unique binary representation for a number $n \in M_q := \{0, ..., 2^q - 1\}$ and bits $b_l \in \{0, 1\}$. Then we define the bit reversing operator as

$$\sigma_q : M_q \to M_q, \sum_{l=0}^{q-1} b_l 2^l \mapsto \sum_{l=0}^{q-1} b_{q-1-l} 2^l.$$

The case $q = 0$ is defined for technical reasons with $M_0 = \{0\}$ and $\sigma_0(0) = 0$.

Algorithm 1 Fast Fourier Transform (FFT) Algorithm

Input: x_k for $k = 0, \ldots, N-1$

for $k = 0$ to $N-1$ **do**
    $f_k = \mathcal{F}(\{x_k\})$
end for

for $k = 0$ to $N-1$ **do**
    $d(k) = f_{\sigma_q(k)}/N$
end for

for $r = 0$ to $q-1$ **do**
    $M = 2^r$
    $\theta = e^{i2\pi/M}$
    for $k = 0$ to $M-1$ **do**
        for $j = 0$ to $2^{q-r-1} - 1$ **do**
            $x = \theta^k d(2jM + M + k)$
            $d(2jM + M + k) = d(2jM + k) - x$
            $d(2jM + k) = d(2jM + k) + x$
        end for
    end for
end for

Output: $d(k) = d_k$ for $k = 0, \ldots, N-1$

Now we can define the algorithm for the case $N = 2^q$ in Algorithm 1. This algorithm allows us to apply the DFT efficiently. We will exploit this in the FNO where we need to apply the FFT for each data sample and each training epoch multiple times. This leads to a substantial increase in the number of Fourier transforms, which highlights the importance of an efficient implementation of the DFT.

The algorithm has a time complexity of $O(NlogN)$ which can be found in [10].

Chapter 3

Wavelets

In contrast to the Fourier transform, an alternative approach for transforming signals is the so-called wavelet transform. While the Fourier transform provides a global representation of the input function using sinusoidal basis functions, it lacks the ability to capture localized features. The wavelet transform addresses this issue by using basis functions that are localized in spatial and frequency domain. These basis functions will be called wavelets and will be introduced later in this chapter. An important difference between wavelets and the Fourier basis is that wavelets have compact support, which allows the wavelet transform to represent local variations more accurately.
In this chapter we will derive the wavelet transform by introducing filters and filter banks and the dilation and wavelet equations, which are necessary to get to the wavelet basis. This wavelet basis is then used to define the wavelet transform. As with the Fourier transform, we will define a continuous and discrete wavelet transform and show an efficient implementation of the discrete wavelet transform. The derivation of the wavelets will be primarily based on [6].

3.1 Filters

Filters are discrete objects that are applied to an input vector and they capture various features in the input signals. First, we define a few basic objects so that we can introduce filters more formal and precisely.

Definition 3.1.1
A signal x is an infinite vector. We can also view x as a function $x : \mathbb{Z} \to \mathbb{R}$

Example 3.1.2
One example of a signal is the so-called unit impulse δ which is defined as

$$\delta(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{else} \end{cases}$$

Definition 3.1.3
A filter H is a linear operator. It acts on a signal $x \in \ell^2$ and convolute it with a fixed vector with coefficients h

$$Hx(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) = h * x.$$

For H to be well defined, we need $h \in \ell^2$.

Example 3.1.4
If we transform the unit impulse we get

$$H\delta(n) = \sum_{k=-\infty}^{\infty} h(k)\delta(n-k) = h(n).$$

This is why h is also called impulse response.

Definition 3.1.5
A filter with $h(n) = 0 \, \forall n < 0$ is called causal.
A filter with $h(n) \neq 0$ for only finitely many n is called finite impulse response filter (FIR filter).

FIR filters are of special interest for practical application since calculating the output of a filter with infinite length is practically infeasible.

Example 3.1.6
Two important causal FIR filters are the moving average H_0 and the moving difference H_1. We will also call these filters Haar filters, since they will be used later to derive Haar wavelets. The filters are defined by their impulse responses

$$h_0(n) = \begin{cases} \frac{1}{2} & \text{if } n = 0 \text{ or } n = 1 \\ 0 & \text{otherwise,} \end{cases}$$

$$h_1(n) = \begin{cases} \frac{1}{2} & \text{if } n = 0 \\ -\frac{1}{2} & \text{if } n = 1 \\ 0 & \text{otherwise.} \end{cases}$$

If we apply these filter to an input signal x we get

$$H_0x(n) = \frac{1}{2}x(n) + \frac{1}{2}x(n-1) \tag{3.1.1}$$

$$H_1x(n) = \frac{1}{2}x(n) - \frac{1}{2}x(n-1) \tag{3.1.2}$$

These names are intuitive since (3.1.1) shows that $H_0x(n)$ is just the average of $x(n)$ and $x(n-1)$ and (3.1.2) gives us $H_1x(n)$ as half of the difference between $x(n)$ and $x(n-1)$.

Lemma 3.1.7 (Matrix Representation)
Applying a filter H with impulse response h to a signal x can be written as an infinite matrix vector product.

Proof. Define the matrix M as $M_{ij} = h(i-j)$.
Then we have

$$Mx(n) = \sum_{j\in\mathbb{Z}} M_{nj}x(j) = \sum_{j\in\mathbb{Z}} h(n-j)x(j) = (h*x)(n)$$

□

Depending on the context we will use H_0 and H_1 as a notation for the lowpass and highpass filter or their corresponding matrices.

Example 3.1.8
We can now write the application of the moving average H_0 and moving difference H_1 as matrix multiplications. For H_0 we get

$$y = h_0 * x \Leftrightarrow \begin{bmatrix} \vdots \\ y(-1) \\ y(0) \\ y(1) \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \ddots & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & \\ & & \frac{1}{2} & \frac{1}{2} & & \\ & & & \frac{1}{2} & \frac{1}{2} & \\ & & & & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ x(-1) \\ x(0) \\ x(1) \\ \vdots \end{bmatrix}$$

and analogously for H_1 we have

$$y = h_1 * x \Leftrightarrow \begin{bmatrix} \vdots \\ y(-1) \\ y(0) \\ y(1) \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \ddots & & & & \\ & -\frac{1}{2} & \frac{1}{2} & & & \\ & & -\frac{1}{2} & \frac{1}{2} & & \\ & & & -\frac{1}{2} & \frac{1}{2} & \\ & & & & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ x(-1) \\ x(0) \\ x(1) \\ \vdots \end{bmatrix}.$$

3.2 Filter Banks

A filter bank consists of two filters, a lowpass and a highpass filter. As the names suggests, the lowpass filter removes the high frequency components from a signal and keeps the low frequencies while a highpass filter does the exact opposite. In our previous example, we had the moving average, which is a lowpass filter. It removes the difference between two consecutive entries of the input signal and with that, it removes the high frequencies. Only changes in the signal over several entries of the signal, which are the low frequency components, can still be observed after the transformation.
Separately neither the lowpass filter nor the highpass filter is invertible, but we will show in the following, that you can retrieve the input signal from the combined results of the highpass and lowpass filter. But if you need two signals to store the information of your one input signal you have effectively doubled the needed storage. This is an issue in the practical implementation of filter banks. The solution for that is downsampling.
Downsampling means that we only keep half of the entries of H_0x and H_1x. We will see later that we can still recover x after dropping half of the entries.

Definition 3.2.1
We denote the downsampling operator as $(\downarrow 2)$. For an input signal x we define the downsampling operator as

$$((\downarrow 2)y)(n) = y(2n).$$

So the downsampling operator drops the odd-indexed components of the vector and keeps the even-indexed components.

Definition 3.2.2
For normalization we define new matrices C and D as

$$\begin{aligned} C &= \sqrt{2}H_0 && \text{(lowpass filter)} \\ D &= \sqrt{2}H_1 && \text{(highpass filter).} \end{aligned}$$

which are rescaled versions of the infinite matrices corresponding to the filters.

This normalization factor is introduced to make the rows and columns of the matrices unit vectors. We can see that in our example of the Haar filters, where now each row of the rescaled matrix has the euclidean norm

$$\sqrt{(\frac{\sqrt{2}}{2})^2 + (\pm\frac{\sqrt{2}}{2})^2} = 1.$$

The coefficients of more complex filters will also be normalized in such a way that the rows of C and D will be unit vectors.

We now combine both operations on one signal, downsampling and normalization, as multiplication with one matrix. For that we define the following new matrices.

Definition 3.2.3
For combining the application of the lowpass filter C and the downsampling operator, we define the rectangular matrix L as

$$L = (\downarrow 2)C$$

Here we apply the downsampling operator on the columns of C which effectively drops every second row of the matrix.
Analogously, we define the matrix B for the highpass filter.

$$B = (\downarrow 2)D$$

To represent our whole filter bank, in the following also called analysis bank, as one matrix, we combine both matrices into one:

$$\begin{bmatrix} (\downarrow 2)C \\ (\downarrow 2)D \end{bmatrix} = \begin{bmatrix} L \\ B \end{bmatrix}$$

Example 3.2.4
In the case of the Haar filter, these matrices are

$$L = (\downarrow 2)C = \begin{bmatrix} \ddots & \ddots & & & & & \\ & & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & & \\ & & & & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \\ & & & & & & \ddots & \ddots \end{bmatrix}$$

and

$$B = (\downarrow 2)D = \begin{bmatrix} \ddots & \ddots & & & & & & \\ & & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & & & \\ & & & & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & \\ & & & & & & \ddots & \ddots \end{bmatrix}.$$

Note here that the rows of C and D are orthonormal to each other which will also be true for other filters we will introduce later.
If we now combine both matrices, we get the infinite matrix

$$\begin{bmatrix} (\downarrow 2)C \\ (\downarrow 2)D \end{bmatrix} = \begin{bmatrix} L \\ B \end{bmatrix} = \begin{bmatrix} \ddots & \ddots & & & & & & \\ & & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & & & \\ & & & & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & \\ & & & & & & \ddots & \ddots \\ \ddots & \ddots & & & & & & \\ & & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & & & \\ & & & & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & \\ & & & & & & \ddots & \ddots \end{bmatrix}.$$

This matrix represents the whole analysis bank of the Haar filters. Due to the orthonormality of its rows, this matrix is orthonormal as a whole.

$$\begin{bmatrix} L^T & B^T \end{bmatrix} \begin{bmatrix} L \\ B \end{bmatrix} = L^T L + B^T B = \frac{1}{2} I + \frac{1}{2} I = I$$

The inverse of the analysis bank is called synthesis bank and in case of an orthonormal filter bank we can represent the synthesis bank as

$$\begin{bmatrix} L \\ B \end{bmatrix}^{-1} = \begin{bmatrix} L^T & B^T \end{bmatrix}$$

$$\begin{bmatrix} L^T & B^T \end{bmatrix} \begin{bmatrix} L \\ B \end{bmatrix} = L^T L + B^T B = \frac{1}{2} I + \frac{1}{2} I = I$$

The orthonormality of the Haar filter bank can easily be seen. If we want this characteristic for more involved filters, we need to choose the impulse responses of the filters, such that the orthonormality is given by construction.

3.2.1 Orthonormal filter banks

The analysis bank consists of two steps: filtering and downsampling. To invert this process, we need to define an upsampling operator $(\uparrow 2)$ and then find matrices $\tilde{C}$ and $\tilde{D}$ to reconstruct the signal in the following way:

Definition 3.2.5
For an input signal x we define the upsampling operator as

$$((\uparrow 2)x)(n) = \begin{cases} x(\frac{n}{2}) & \text{if } n \text{ is even} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 3.2.6
Let x be our input signal and $y_l = Lx$ and $y_h = Bx$ the results of the analysis bank. The synthesis bank now gives us

$$\begin{aligned} x_{rec} &= \tilde{C}(\uparrow 2)y_l + \tilde{D}(\uparrow 2)y_h \\ &= \tilde{C}(\uparrow 2)(\downarrow 2)Cx + \tilde{D}(\uparrow 2)(\downarrow 2)Dx. \end{aligned}$$

We say that a filter bank allows perfect reconstruction if there exists an $l \geq 0$ such that we have

$$x(n) = x_{rec}(n-l)$$

$\forall n \in \mathbb{N}$.

Definition 3.2.7
We call a filter bank orthonormal if setting the synthesis filters as

$$\begin{aligned} \tilde{C} &= C^T \\ \tilde{D} &= D^T \end{aligned}$$

gives us a perfect reconstruction.

The question that remains is how we can assure that we choose our filters in such a way that in the end we get an orthonormal filter bank.
We focus on causal FIR-filter, since these are the filters used in practical applications. Assume we have a lowpass filter with impulse response $c(0), \ldots, c(N)$ and a highpass filter with impulse response $d(0), \ldots, d(N)$ with $c(0), c(N), d(0), d(N) \neq 0$. All $c(n), d(n)$ are set to 0 for $n < 0$ and $n > N$. Then we can summarize the analysis bank in the matrix

$$H = \begin{bmatrix} L \\ B \end{bmatrix} = \begin{bmatrix} c(N) & c(N-1) & c(N-2) & \ldots & c(0) & & & \\ & & c(N) & c(N-1) & c(N-2) & \ldots & c(0) & \\ & & & & & & \ddots & \ddots \\ d(N) & d(N-1) & d(N-2) & \ldots & d(0) & & & \\ & & d(N) & d(N-1) & d(N-2) & \ldots & d(0) & \\ & & & & & & \ddots & \ddots \end{bmatrix}.$$

To have an orthogonal filter bank we need to fulfill the conditions $H^T H = I$ and $HH^T = I$. The following conclusions are stated in [6].
If we write this out in the block form we get

$$\begin{bmatrix} L^T & B^T \end{bmatrix} \begin{bmatrix} L \\ B \end{bmatrix} = L^T L + B^T B = I \tag{3.2.1}$$

$$\begin{bmatrix} L \\ B \end{bmatrix} \begin{bmatrix} L^T & B^T \end{bmatrix} = \begin{bmatrix} LL^T & LB^T \\ LB^T & BB^T \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{3.2.2}$$

To fulfill the condition (3.2.2) the coefficients need to fulfill several conditions. First the rows of L need to be orthonormal. This can be written as a scalar product in the following way:

$$LL^T = I \Leftrightarrow \sum_{n \in \mathbb{Z}} c(n)c(n-2k) = \delta(k). \tag{3.2.3}$$

Analogously, the coefficients of the highpass filter need to fulfill

$$BB^T = I \Leftrightarrow \sum_{n\in\mathbb{Z}} d(n)d(n-2k) = \delta(k). \tag{3.2.4}$$

In addition to that each row of B needs to be orthogonal to each row of L. That means

$$LB^T = 0 \Leftrightarrow \sum_{n\in\mathbb{Z}} c(n)d(n-2k) = 0. \tag{3.2.5}$$

We now have conditions (3.2.1), (3.2.3), (3.2.4), and (3.2.5) for the orthonormality of the filter bank.
(3.2.3) and (3.2.4) directly rule out filters of odd length. For example if we assume $N+1 = 3$ and test $k = 1$, then we have

$$(c(0), c(1), c(2))^T \cdot (0, 0, c(0))^T = c(0)c(2) \neq 0$$

which is a contradiction to (3.2.3). For that reason we need to have an even filter length. Assume now we have coefficients $c(0), \ldots, c(N)$, which fulfill (3.2.3). Then we can choose the coefficients for the highpass filter as

$$d(k) = (-1)^k c(N-k)$$

We can now check if these coefficients fulfill the other conditions.

$$\begin{aligned}
\sum_{n\in\mathbb{Z}} d(n)d(n-2k) &= \sum_{n\in\mathbb{Z}} (-1)^n c(N-n)(-1)^{(n-2k)} c(N-(n-2k)) \\
&= \sum_{n\in\mathbb{Z}} (-1)^{(N-n)+(N-n-2k)} c(n)c(n+2k) \\
&= \sum_{n\in\mathbb{Z}} (-1)^{2(N-n-k)} c(n)c(n+2k) \\
&= \sum_{n\in\mathbb{Z}} c(n-2k)c(n) = \delta(k)
\end{aligned}$$

which is true by assumption. To assure (3.2.2) we also need to check (3.2.5).

$$\begin{aligned}
\sum_{n\in\mathbb{Z}} c(n)d(n-2k) &= \sum_{n\in\mathbb{Z}} c(n)(-1)^{(n-2k)} c(N-(n-2k)) \\
&= \sum_{n\in\mathbb{Z}} c(N+2k-n)(-1)^{(N-n)} c(N-(N+2k-n-2k)) \\
&= \sum_{n\in\mathbb{Z}} c(N-(n-2k))(-1)^{n+1} c(n)) \\
&= -\sum_{n\in\mathbb{Z}} c(N-(n-2k))(-1)^n c(n)) \\
&= -\sum_{n\in\mathbb{Z}} c(n)d(n-2k) \\
\Rightarrow \sum_{n\in\mathbb{Z}} c(n)d(n-2k) &= 0
\end{aligned}$$

The last thing we need to check is (3.2.1).

$$\begin{aligned}
(L^TL + B^TB)_{ij} &= \sum_{k\in\mathbb{Z}} L_{ki}L_{kj} + \sum_{k\in\mathbb{Z}} B_{ki}B_{kj} \\
&= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-j) + \sum_{k\in\mathbb{Z}} d(2k-i)d(2k-j) \\
&= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-j) + (-1)^{(i+j)} \sum_{k\in\mathbb{Z}} c(N-(2k-i))c(N-(2k-j)) \\
&= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-j) + (-1)^{(i+j)} \sum_{k\in\mathbb{Z}} c(2k-i-N)c(2k-j-N) \\
&= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-j) + (-1)^{(i+j)} \sum_{k\in\mathbb{Z}} c(2k-i+1)c(2k-j+1)
\end{aligned}$$

Now let $i = j$:

$$\begin{aligned}
(L^TL + B^TB)_{ij} &= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-i) + c(2k-i+1)c(2k-i+1) \\
&= \sum_{k\in\mathbb{Z}} c(k-i)c(k-i) \\
&= \sum_{k\in\mathbb{Z}} c(k)c(k) \\
&= 1
\end{aligned}$$

For $i \neq j$ we look at two cases separately. First let $j - i = 2d$ be even.

$$\begin{aligned}
(L^TL + B^TB)_{ij} &= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-j) + \sum_{k\in\mathbb{Z}} c(2k-i+1)c(2k-j+1) \\
&= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-2d-i) + \sum_{k\in\mathbb{Z}} c(2k-i+1)c(2k-2d-i+1) \\
&= \sum_{k\in\mathbb{Z}} c(k-i)c(k-2d-i) \\
&= \sum_{k\in\mathbb{Z}} c(k)c(k-2d) = 0
\end{aligned}$$

Lastly we look at the case that $i - j = 2d + 1$ is odd. So w.l.o.g. assume i es even.

$$\begin{aligned}
(L^TL + B^TB)_{ij} &= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-j) + (-1)^{(i+j)} \sum_{k\in\mathbb{Z}} c(2k-i+1)c(2k-j+1) \\
&= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-j) - \sum_{k\in\mathbb{Z}} c(2k-i+1)c(2k-j+1) \\
&= \sum_{k\in\mathbb{Z}} c(2k-i)c(2k-2d-1-i) - \sum_{k\in\mathbb{Z}} c(2k-i+1)c(2k-2d-i) \\
&= \sum_{k\in\mathbb{Z}} c(2k)c(2k-2d-1) - \sum_{k\in\mathbb{Z}} c(2k+1)c(2k-2d) \\
&= \sum_{k\in\mathbb{Z}} c(2k)c(2k-2d-1) - \sum_{k\in\mathbb{Z}} c(2k+2d+1)c(2k) \\
&= 0
\end{aligned}$$

Now we have shown that every condition for an orthogonal filter bank is fulfilled if we assume (3.2.3). We can summarize these results in a theorem.

Theorem 3.2.8
Let $c(0), \dots, c(N)$ a lowpass filter of even length and choose the highpass filter as

$$d(k) = (-1)^k c(N-k).$$

Then the resulting filter bank is orthogonal if and only if

$$\sum_{n\in\mathbb{Z}} c(n)c(n-2k) = \delta(k)$$

Definition 3.2.9
Filter where the impulse response of the lowpass and highpass filter fulfill the condition

$$d(k) = (-1)^k c(N-k).$$

are called conjugate mirror filter.

Now that we can check if a filter bank is orthogonal, we also directly know how the synthesis bank of such a filter bank looks like. For orthogonal filter banks with matrix representation

$$H = \begin{bmatrix} L \\ B \end{bmatrix}$$

we have the synthesis bank

$$H^T = \begin{bmatrix} L^T & B^T \end{bmatrix}$$

For better understanding we will now do an example with our known Haar filters.

Example 3.2.10
First we check if the filter bank is orthogonal:
The lowpass filter has length 2 and $d(k) = (-1)^k c(N-k)$. So the assumptions of the theorem above are fulfilled.
Now let $k = 0$:

$$\sum_{n\in\mathbb{Z}} c(n)c(n-2k) = \sum_{n\in\mathbb{Z}} c(n)c(n) = 2(\frac{\sqrt{2}}{2})^2 = 1$$

Since the filter has only length 2, for every other k we have

$$\sum_{n\in\mathbb{Z}} c(n)c(n-2k) = 0.$$

Therefore we now know that the Haar filter bank is an orthogonal filter bank. So the synthesis bank consists of C^T and D^T. Let us now take $x = (1,2,3,4,5,6)^T$ as the input signal.
This signal has compact support so we extend it with $x(n) = 1$ for all $n < 0$ and $n > 6$. Since all of these entries are 0 we can This gives us the output signals

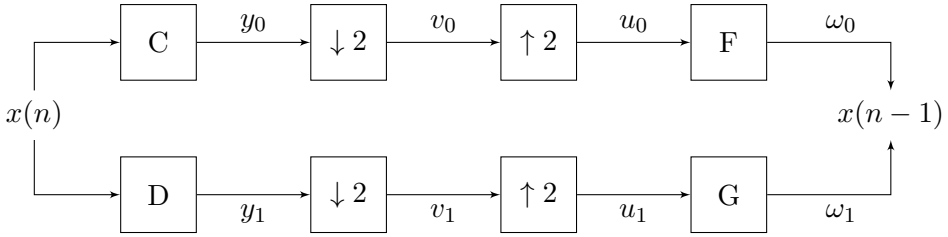
$$\begin{aligned} y_l(1) &= Lx(1) \\ &= Cx(2) \\ &= \frac{\sqrt{2}}{2}(1+2) \\ &= \frac{3\sqrt{2}}{2} \end{aligned}$$

We can calculate the other entries analogously and get $y_l(2) = \frac{7\sqrt{2}}{2}$ and $y_l(3) = \frac{11\sqrt{2}}{2}$. For the highpass filter we get $y_h = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$. From these two output signals we should be able to reconstruct our input signal.
For that we apply the inverse matrices C^T and D^T and apply the upsampling operator to the matrices. Previously applying the downsampling operator to a matrix meant to drop every second row, we now drop every second column. That means we get

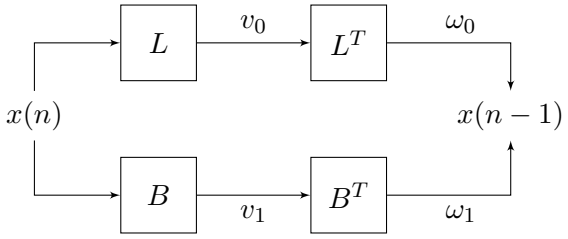
$$\begin{aligned} x &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} y_l + \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} y_h \\ &= (1, 2, 3, 4, 5, 6)^T \end{aligned}$$

In the result we have reconstructed our signal from the downsampled outputs of the lowpass filter and highpass filter.

We can summarize the previous section in a scheme



where $F = C^{-1}$ and $G = D^{-1}$ or we can write it more compact as



where the shift in the index is a result of applying the infinite filters and the inverses rigorously, but with this simple shift, no information is lost.

3.3 Dilation Equation

In this section we will present the dilation equation which is the first step in constructing wavelets. Wavelets are a special kind of basis for a function space which is used to deconstruct a signal and store the information in spatial and frequency domain separately. Wavelets arise from filters with impulse response h_l and h_h for the lowpass and highpass filter by solving the dilation equation which we will define now and the wavelet equation which will be introduced in the next section.

Definition 3.3.1
For a lowpass impulse response h_l the dilation equation is

$$\phi(t) = 2\sum_{k=0}^{N} h_l(k)\phi(2t-k)$$

or in terms of the rescaled coefficients of the filter bank it is

$$\phi(t) = \sqrt{2}\sum_{k=0}^{N} c(k)\phi(2t-k).$$

The solution to this equation is called scaling function.

Example 3.3.2
For the example of the Haar filters we have the lowpass coefficients $h_l(0) = h_l(1) = \frac{1}{2}$. Therefore we have

$$\begin{aligned}\phi(t) &= 2\sum_{k=0}^{N} h_l(k)\phi(2t-k)\\ &= \phi(2t) + \phi(2t-1).\end{aligned}$$

The solution to that is the box function

$$\phi(t) = \begin{cases} 1 & \text{for } 0 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

which we can confirm easily by plugging the function into the dilation equation.

The scaling behavior of the scaling function is strongly related to the scaling of the impulse response h_l as we can see in the following theorem from [6].

Theorem 3.3.3

$$\text{If } \int_{-\infty}^{\infty} \phi(t)\,dt = 1 \text{ then } \sum_{n=0}^{N} h_l(n) = 1$$

Proof. First we see that

$$\begin{aligned}1 &= \int_{-\infty}^{\infty} \phi(u)\,du\\ &= 2\int_{-\infty}^{\infty} \phi(2t-k)\,dt \quad (\text{set } u = 2t-k).\end{aligned}$$

So we have $\int_{-\infty}^{\infty} \phi(2t-k)\,dt = \frac{1}{2}$.

Integrating both sides of the dilation equation gives us

$$\begin{aligned} 1 &= \int_{-\infty}^{\infty} \phi(t)\,dt \\ &= 2\int_{-\infty}^{\infty} \sum_k h_l(k)\phi(2t-k)\,dt \\ &= 2\sum_k h_l(k)\int_{-\infty}^{\infty} \phi(2t-k)\,dt \\ &= 2\sum_k h_l(k)\frac{1}{2} \\ \Rightarrow &\sum_k h_l(k) = 1 \end{aligned}$$

□

3.4 Wavelet Equation

After stating the dilation equation and solving it for the scaling function, we can now use the highpass coefficients and define the wavelet equation. The solution of the wavelet equation will be called mother wavelet and is fundamental to the construction of the wavelet basis.

Definition 3.4.1
Let h_h be the impulse response of the highpass filter and let $\phi(t)$ the scaling function corresponding to the lowpass coefficients h_l. Then

$$\omega(t) = 2\sum_{k=0}^{N} h_h(k)\phi(2t-k)$$

is called wavelet equation. In terms of the rescaled coefficients of the filter bank this equation becomes

$$\omega(t) = \sqrt{2}\sum_{k=0}^{N} d(k)\phi(2t-k)$$

where ω is called mother wavelet.

Example 3.4.2
In the case of the Haar filter we have the highpass coefficients $h_h(0) = \frac{1}{2}$ and $h_h(1) = -\frac{1}{2}$ so the wavelet equation simplifies to

$$\omega(t) = \phi(2t) - \phi(2t-1).$$

If we plug in the scaling function of the Haar filter we get the Haar wavelet

$$\omega(t) = \begin{cases} 1 & \text{if } 0 \le t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \le t < 1 \\ 0 & \text{otherwise.} \end{cases}$$

If we have a wavelet ω, which is called mother wavelet, we now construct the rest of the set of wavelets by shifting and compressing the mother wavelet.

Definition 3.4.3
Define $\omega_{jk}(t) = \omega(2^j t - k)$ for $j \in N_0, 0 \leq k < 2^j$.
Then we can define the wavelet basis as $\{\omega_{jk}\}$

Now we can examine the properties of this basis. We will do that for the example of the Haar wavelets. The more general case can be looked up in [6] on page 186.

Theorem 3.4.4
The Haar wavelets in combination with a new scaling factor are defined as $\omega_{jk}(t) = 2^{j/2}\omega(2^j t - k)$. Now the rescaled Haar wavelets form an orthonormal basis:

$$\int_{-\infty}^{\infty} \omega_{jk}\omega_{JK}\, dt = \delta(j - J)\delta(k - K).$$

Proof. First we take a look at one rescaled Haar wavelet.

$$\omega_{jk}(t) = 2^{j/2}\omega(2^j t - k) = \begin{cases} 2^{\frac{j}{2}} & \text{if } \frac{k}{2^j} \leq t < \frac{k+\frac{1}{2}}{2^j} \\ -2^{\frac{j}{2}} & \text{if } \frac{k+\frac{1}{2}}{2^j} \leq t < \frac{k+1}{2^j} \\ 0 & \text{otherwise} \end{cases}$$

That also means that $supp(\omega_{jk}) = [\frac{k}{2^j}, \frac{k+1}{2^j})$.
We now treat the three different cases separately.

1. $j = J$ and $k = K$

$$\begin{aligned} \int_{-\infty}^{\infty} \omega_{jk}\omega_{JK}\, dt &= \int_{-\infty}^{\infty} \omega_{jk}^2\, dt \\ &= \int_{\frac{k}{2^j}}^{\frac{k+1}{2^j}} 2^{2\frac{j}{2}}\, dt \\ &= 2^j \frac{1}{2^j} \\ &= 1 \end{aligned}$$

2. $j = J$ and $k \neq K$

$$\int_{-\infty}^{\infty} \omega_{jk}\omega_{JK}\, dt = 0 \text{ since } \operatorname{supp}(\omega_{jk}) \cap \operatorname{supp}(\omega_{jK}) = \emptyset$$

3. $j \neq J$ Without loss of generality we assume $j < J$. Then either we have $\operatorname{supp}(\omega_{jk}) \cap \operatorname{supp}(\omega_{JK}) = \emptyset$, then we automatically have

$$\int_{-\infty}^{\infty} \omega_{jk}\omega_{JK}\, dt = 0$$

or we have $\operatorname{supp}(\omega_{JK}) \subseteq [\frac{k}{2^j}, \frac{k+\frac{1}{2}}{2^j})$ or $\operatorname{supp}(\omega_{JK}) \subseteq [\frac{k+\frac{1}{2}}{2^j}, \frac{k+1}{2^j})$. Then we have

$$\begin{aligned} \int_{-\infty}^{\infty} \omega_{jk}\omega_{JK}\, dt &= \int_{-\infty}^{\infty} \pm\omega_{jk}\, dt \\ &= 0 \end{aligned}$$

□

We now have shown that the Haar filters form an orthonormal basis but we also want to construct other orthonormal wavelet basis in the next section.

3.5 Daubechies Wavelets

Up to this point, we looked at the concept of wavelets only in an abstract way and using the example of the Haar wavelets, which are the simplest form of wavelets. Haar wavelets have the advantage that they have a closed form, the derivation is straightforward, they have a compact support, and they form an orthonormal basis. On the other hand, one issue with Haar wavelets is that they are not continuous.
In the following section, we want to construct a class of wavelets that also have compact support to be able to catch localized patterns and they should form a orthonormal basis to make the wavelet transform possible, but they should have a prescribed number p of vanishing moments (Definition 3.5.1). These wavelets will be able to detect changes and patterns over longer ranges in the input signals since their impulse responses will be longer than the two entries of the impulse responses of the Haar filters.
To avoid confusion we need to clarify the notation. In this section we will call wavelets ψ and use ω for the variable in the Fourier domain.

Definition 3.5.1
A wavelet ψ has p vanishing moments if

$$\int \psi(x)x^j \, dx = 0 \quad j = 0, 1, ..., p-1.$$

We now want to construct an orthonormal wavelet basis which has compact support of minimal size and p vanishing moments. The resulting family of wavelets is called Daubechies wavelets, which was first introduced by Ingrid Daubechies in [12]. We follow the construction of this class of wavelets from [13]. To tackle this construction we first take a look at two results from [13]. First proposition 7.2 from [13] gives us a connection between the compact support of a wavelet and the support of the filter and the scaling function.

Theorem 3.5.2
The scaling function ϕ has a compact support if and only if h_l has a compact support and their support is equal. If the support of h_l and ϕ is
$[N_1, N_2]$ then the support of ψ is $[(N_1 - N_2 + 1)/2, (N_2 - N_1 + 1)/2]$.

This shows that if we want to get a wavelet with compact support, we need to look for a lowpass filter with compact support. This is fulfilled by a causal finite impulse response filter. Assuming a causal FIR lowpass filter h,

$$\hat{h}(\omega) = \sum_{n=0}^{N-1} h[n]e^{-in\omega}$$

is a trigonometric polynomial. In the next step we need Theorem 7.4 from [13].

Theorem 3.5.3
Let ψ and ϕ be a wavelet and a scaling function that generate an orthogonal basis. Suppose that $|\psi(t)| = \mathcal{O}((1+t^2)^{-p/2-1})$ and $|\phi(t)| = \mathcal{O}((1+t^2)^{-p/2-1})$. Then the four following statements are equivalent:

- The wavelet ψ has p vanishing moments.
- $\hat{\psi}(\omega)$ and its first $p-1$ derivatives are zero at $\omega = 0$.
- $\hat{h}(\omega)$ and its first $p-1$ derivatives are zero at $\omega = \pi$.
- For any $0 \leq k < p$,
$$q_k(t) = \sum_{n=-\infty}^{\infty} n^k \phi(t-n)$$
is a polynomial of degree k.

The assumptions of this theorem are fulfilled since we assume wavelets with compact support.
To ensure that the wavelet we seek has p vanishing moments Theorem 3.5.3 shows that we need to choose h in such a way, that $\hat{h}$ has a zero of order p at $\omega = \pi$.
To ensure the zero of order p we write $\hat{h}$ as

$$\hat{h}(\omega) = \sqrt{2}(\frac{1+e^{-i\omega}}{2})^p R(e^{-i\omega}) \tag{3.5.1}$$

since the factor $(\frac{1+e^{-i\omega}}{2})^p$ is polynomial of minimal degree with a zero of order p at $\omega = \pi$.
To create an orthonormal wavelet, the filter also needs to fulfill

$$|\hat{h}(\omega)|^2 + |\hat{h}(\omega+\pi)|^2 = 2 \tag{3.5.2}$$

according to [13]. The deduction of this equation can also be looked up in section 5.2 in [6].

Before we can go into the final steps of the construction of the Daubechies filter, we need the Bezout theorem on polynomials from [18] which can also be found in Theorem 7.5 in [13].

Theorem 3.5.4
Let $Q_1(y)$ and $Q_2(y)$ be two polynomials of degrees n_1 and n_2 with no common zeroes. Then there exists two unique polynomials P_1, P_2 of degrees $n_2 - 1$ and $n_1 - 1$ such that

$$Q_1(y)P_1(y) + Q_2(y)P_2(y) = 1.$$

The proof of this theorem can be found in [18].
Up to this point we know the conditions on $\hat{h}$ to assure that the corresponding wavelets form an orthonormal basis, have p vanishing moments and compact support. Now we need to construct R from (3.5.1) to have a close form of $\hat{h}$. This will be the result of the constructive proof of the following theorem which can be found as Theorem 7.5 in [13].

Theorem 3.5.5
Let h be a real conjugate mirror filter, such that $\hat{h}(\omega)$ has p zeros at $\omega = \pi$. Then h has at least $2p$ non-zero coefficients. Daubechies filters have $2p$ non-zero entries.

Proof. Since we chose a real filter $h(n)$, $|\hat{h}(\omega)|^2$ is an even function and can be written as a polynomial in $\cos\omega$. This also means that $|R(e^{-i\omega})|$, where R is defined in (3.5.1), is a polynomial in $\cos\omega$. Due to the trigonometric identity

$$\cos(\omega) = 1 - 2\sin^2(\frac{\omega}{2})$$

we can also write

$$|R(e^{-i\omega})| = P(\sin^2(\frac{\omega}{2}))$$

where P is a polynomial. Combining this with (3.5.1), we get

$$|\hat{h}(\omega)|^2 = 2(\cos\frac{\omega}{2})^{2p} P(\sin^2\frac{\omega}{2}).$$

By substituting $y = \sin^2(\frac{\omega}{2})$, we can rewrite (3.5.2) as

$$(1-y)^p P(y) + y^p P(1-y) = 1$$

for any $y = \sin^2(\omega/2) \in [0,1]$. We now need to find a polynomial $P(y) \geq 0$ to minimize the number of non-zero terms of the finite Fourier series $\hat{h}(\omega)$ to get wavelets of minimal size. Since $(1-y)^p$ and y^p are two polynomials of degree p with no common zeros, the existence and uniqueness of the polynomials $P_1(y)$ and $P_2(y)$ such that

$$(1-y)^p P_1(y) + y^p P_2(y) = 1$$

is guaranteed by Theorem 3.5.4. The searched polynomials are $P_2(y) = P_1(1-y) = P(1-y)$ with

$$P(y) = \sum_{k=0}^{p-1} \binom{p-1+k}{k} y^k.$$

This polynomial is of degree $p-1$ and $P(y) \geq 0$.
The next step is to construct a polynomial with minimal degree m

$$R(e^{-i\omega}) = \sum_{k=0}^{m} r_k e^{-ik\omega} = r_o \prod_{k=0}^{m} (1 - a_k e^{-i\omega})$$

such that $|R(e^{-i\omega})|^2 = P(\sin^2\frac{\omega}{2})$. Since we have real coefficients, we have $R^*(e^{-i\omega}) = R(e^{i\omega})$ and

$$\begin{aligned} |R(e^{-i\omega})|^2 &= R(e^{i\omega})R(e^{-i\omega}) \\ &= P(\sin^2\frac{\omega}{2}) \\ &= P(\frac{2 - e^{i\omega} - e^{-i\omega}}{4}) \\ &= Q(e^{-i\omega}). \end{aligned}$$

If we substitute $z = e^{-i\omega}$ and extend the function to the whole complex plane we get

$$R(z)R(z^{-1}) = r_0^2 \prod_{k=0}^{m} (1 - a_k z)(1 - a_k z^{-1}) = P(\frac{2 - z - z^{-1}}{4}) = Q(z).$$

Due to the real coefficients of $Q(z)$ we know that if c_k is a root, then c_k^* is also a root. We also see that Q is a function of $z + z^{-1}$ and this shows that if c_k is a root, then $\frac{1}{c_k}$ and $\frac{1}{c_k*}$ are

also roots. To design $R(z)$ in a way that it satisfies the equation above, we choose each root a_k among a pair $(c_k, \frac{1}{c_k})$ and also include a_k^* to assure real coefficients. Due to Theorem 3.5.4 this yields a polynomial of degree $m = p - 1$ with $r_0 = 2^{p-1}$. The resulting filter h has $N = p + m + 1 = 2p$ non-zero coefficients.
The minimum phase solution $R(e^{-i\omega})$ is obtained if we also choose a_k out of $(c_k, \frac{1}{c_k})$ such that $|a_k| \leq 1$ ([19]).
We now have constructed R and can calculate the lowpass Daubechies filter h for a given number p of vanishing moments from

$$\hat{h}(\omega) = \sqrt{2}(\frac{1 + e^{-i\omega}}{2})^p R(e^{-i\omega}).$$

Since we assumed a conjugate mirror filter we get the highpass filter from the lowpass filter and get to the wavelets from there. □

The proof shows how we can construct the filter with minimal support for p vanishing moments.

p	**n**	$\mathbf{h_p[n]}$
2	0	0.482962913145
	1	0.836516303738
	2	0.224143868042
	3	-0.129409522551
3	0	0.332670552950
	1	0.806891509311
	2	0.459877502118
	3	-0.135011020010
	4	-0.085441273882
	5	0.035226291882
4	0	0.230377813309
	1	0.714846570553
	2	0.630880767930
	3	-0.027983769417
	4	-0.187034811719
	5	0.030841381836
	6	0.032883011667
	7	-0.010597401785

Table 3.1: Low-pass filter coefficients $h_p[n]$ for Daubechies wavelets with $p = 2, 3, 4$.

The following theorem which is proposition 7.4 from [13] shows that the wavelets which correspond to the filters above, have minimal support. The wavelets for $p = 2, 3, 4$ are shown in Figure 3.1 as an example.

Theorem 3.5.6
If ψ is a wavelet with p vanishing moments that generates an orthonormal basis of $L_2(\mathbb{R})$, then the length of the support of ψ is at least $2p-1$. A Daubechies wavelet has a minimum size support equal to $[-p+1, p]$. The support of the corresponding scaling function ϕ is $[0, 2p-1]$.

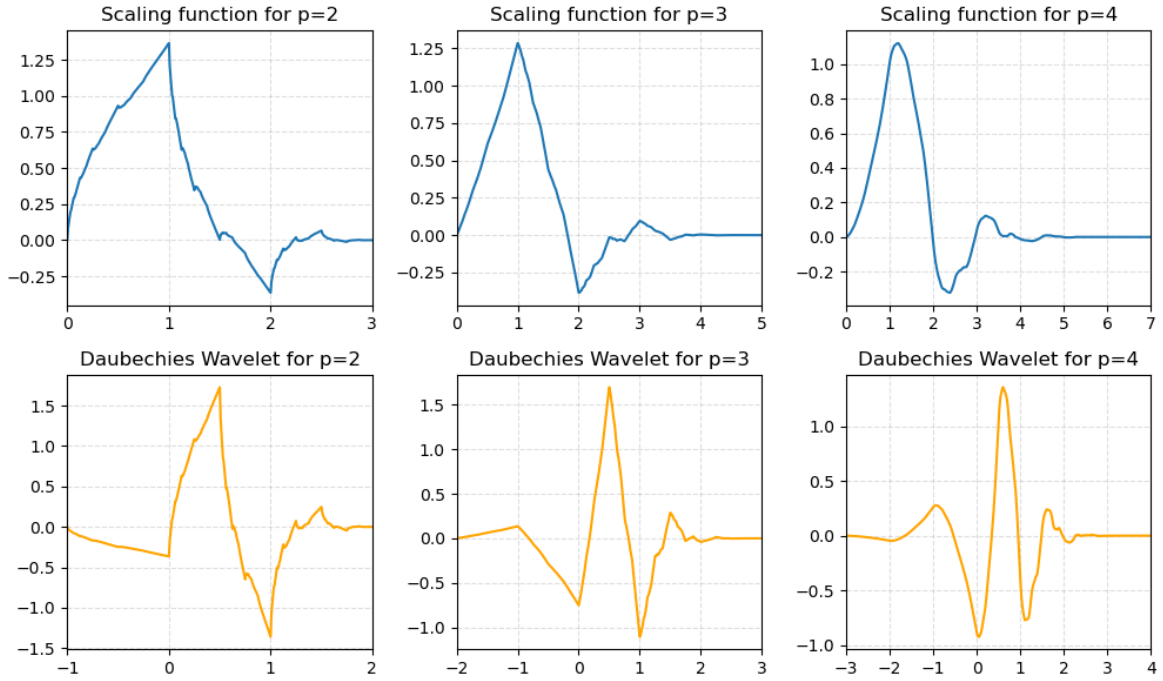


Figure 3.1: Scaling functions and Daubechies wavelets for $p = 2, 3, 4$ vanishing moments.

3.6 Wavelet transform

Now we can get to the wavelet transform. The continuous wavelet transform decomposes a given function at different scales and allows us to analyze the frequencies both in time and spatial domain simultaneously. Unlike with the Fourier transform we can use the localized basis functions to see at which spatial regions which frequency occurs.
In the following example we look once again at the Haar wavelets to get an intuition for the concept of the wavelet transform.

Example 3.6.1
Our goal is to decompose the signal from Figure 3.2 using the Haar wavelets.

The principle ansatz is to get a representation of our input function in terms of the Haar wavelet basis $\omega_{j,k}$ where $\omega_{j,k}(t) = \omega(2^j - k)$ for $j \in \mathbb{N}_0, 0 \leq k < 2^j$ and

$$\omega(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq t < 1 \\ 0 & \text{otherwise.} \end{cases}$$

In order to make this example more readable, we omit the scaling factor $2^{j/2}$ here.
First, we look at the coarsest scale $j = 0$. We see that the mean of our input signal f in the

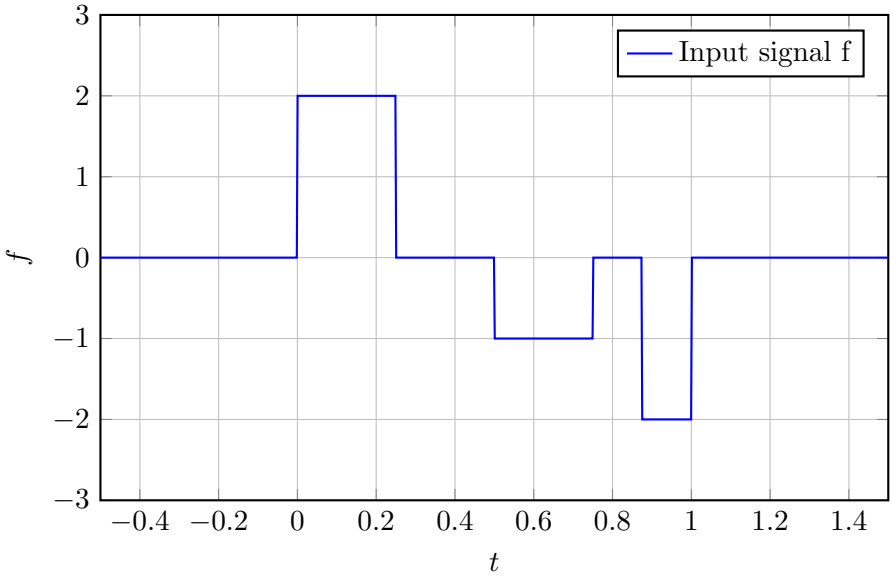
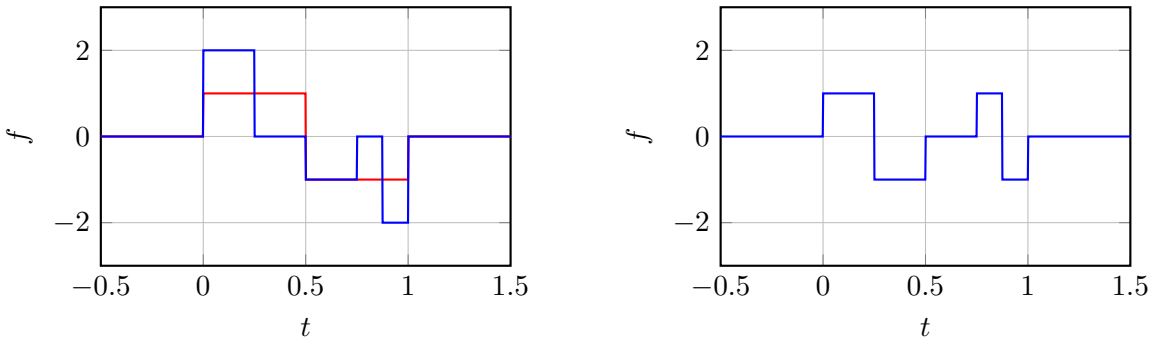


Figure 3.2: Input signal f

interval $[0, 0.5)$ is 1 and in $[0.5, 1)$ it is -1. Therefore we now subtract $\omega_{0,0}$ once, which means that our coefficient is $b_{00} = 1$

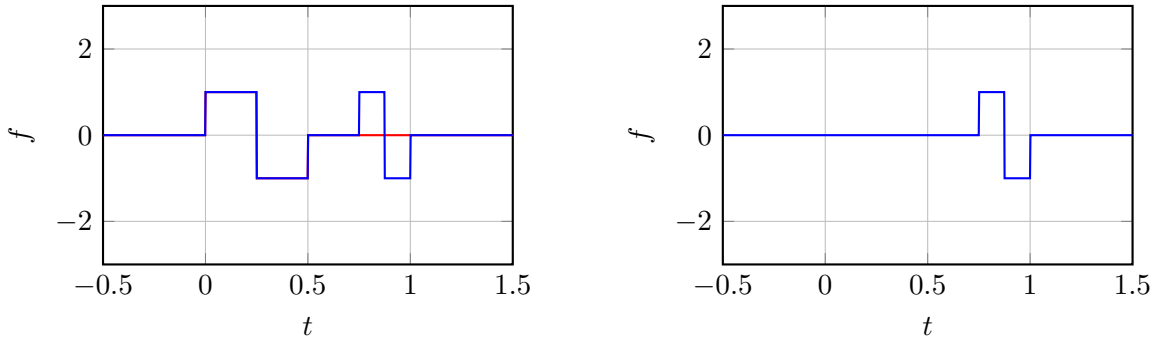


(a) f and $\omega_{0,0}$

(b) $f - \omega_{0,0}$

Figure 3.3: First level of decomposition

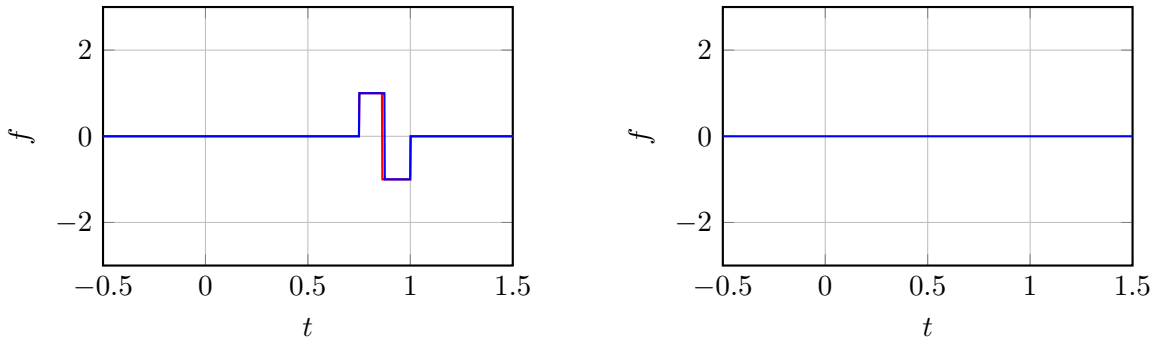
Now we analyze the result at the next finer resolution and look at intervals with width 0.25. In the first interval we see that the signal is 1 in $[0, 0.25)$ and -1 in $[0.25, 0.5)$. Therefore $b_{1,0} = 1$. in the second interval $[0.5, 1)$ we don't see differences in the mean of the first and second subinterval.



(a) $f - \omega_{0,0}$ and $\omega_{1,0}$ (b) $f - \omega_{0,0} - \omega_{1,0}$

Figure 3.4: Second level of decomposition

Lastly we want to look at the next finer scale and we see the step in the interval $[0.75, 1)$ so $b_{2,3} = 1$



(a) $f - \omega_{0,0} - \omega_{1,0}$ and $\omega_{2,3}$ (b) $f - \omega_{0,0} - \omega_{1,0} - \omega_{2,3}$

Figure 3.5: Third level of decomposition

We see now that we can express f as $f = \omega_{0,0} + \omega_{1,0} + \omega_{2,3}$, so we have a basis representation of f which can be stored in the form of the wavelet coefficients $b_{0,0}, b_{1,0}, b_{2,3}$.

One important note on this example, which also holds true in general, is that the wavelet transform only works for functions with zero mean, but this is not really an issue since you can just transform your input signal as

$$\tilde{f} = f - \frac{1}{|D|} \int_D f(t)\, dt$$

where D is the domain of f and we can store the mean of the function to be able to reconstruct the signal.
Now we want to be able to define the continuous wavelet transform in a more formal and more general way than in the example above.

Let $\{\omega_{jk}\}$ be a wavelet basis of $L_2(\mathbb{R})$ and $f \in L_2(\mathbb{R})$. Then we can analyze the function in terms of our wavelet like in [6] (1.38):

$$b_{jk} = \int_{-\infty}^{\infty} f(t)\omega_{jk}(t)\, dt$$

The coefficients b_{jk} are the result of the wavelet transform and are sufficient to reconstruct the input function f

$$f(t) = \sum_{j,k} b_{jk}\omega_{jk}(t).$$

Alongside the continuous wavelet transform, there is also a discrete version. Here we have a discrete signal x as an input and we use discrete wavelets, stored in a matrix A, which will be constructed in the following. Then the analysis and synthesis are

Analysis:	$b = Ax$
Synthesis:	$x = Sb.$

In case of an orthonormal wavelet basis we have $S = A^{-1} = A^T$.

3.6.1 Fast wavelet transform

The fast wavelet transform is an algorithm which calculates the discrete wavelet transform (DWT) efficiently.
Here we will focus on a version that can be implemented. Therefore we will not use an infinite wavelet basis, but only use wavelets up to a certain level J of detail. That means we have a basis $\{\omega_{jk} | 0 \leq j \leq J, 0 \leq k < 2^j\}$.
We will start the analysis of the input signal at the finest level of details. We apply the lowpass filter C and highpass filter D to the input signal and apply the downsampling operator afterwards.

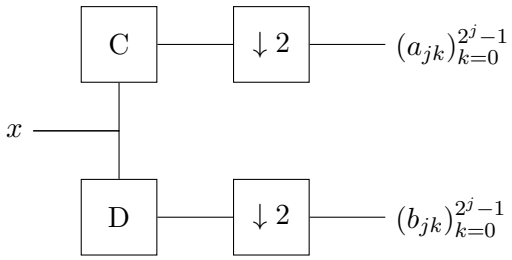


Figure 3.6: First level of fast wavelet transform

In the case of the Haar wavelets we can directly see a recursive connection between the detail level j and $j-1$:

$$a_{j-1,k} = \frac{1}{\sqrt{2}}(a_{j,2k} + a_{j,2k+1})$$
$$b_{j-1,k} = \frac{1}{\sqrt{2}}(a_{j,2k} - a_{j,2k+1}).$$

That means we can express the output of the filter at level $j-1$ by applying the filter to the output of the filters at level j. This is used by the algorithm also for other wavelets.

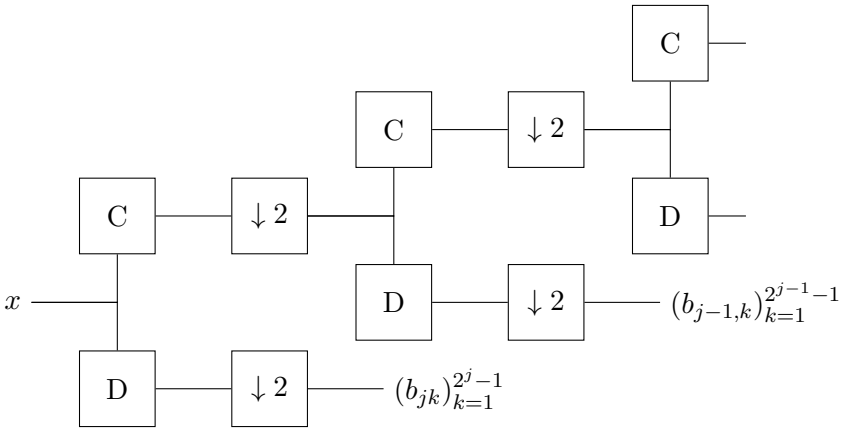


Figure 3.7: Second level of the fast wavelet transform

After each step we store the output of the highpass filter and after the last step we also store the output of the lowpass filter $a_{0,0}$.
These information are sufficient to reconstruct the signal. The analysis started at the finest detail level j and ended at the coarsest level 0. The reconstruction will do it the other way around. Starting at level 0 we have the coefficients $a_{0,0}$ and $b_{0,0}$ which is enough to reconstruct the output of the highpass filter at the level 1 which was $(a_{1k})_{k=0}^{1}$. We also stored $(b_{1k})_{k=0}^{1}$ so we can reconstruct $(a_{2k})_{k=0}^{3}$. Like that we can reconstruct our input to the finest detail level and apply the inverted analysis bank to recreate the input signal x. In the case of orthonormal filters we use the inverse matrices for reconstruction

$$((\downarrow 2)C)^{-1} = L^{-1} = L^T$$
$$((\downarrow 2)D)^{-1} = B^{-1} = B^T.$$

This sequence of analysis and reconstruction can be shown like in the following scheme where we do a discrete wavelet analysis and synthesis with maximum detail level $j = 1$.

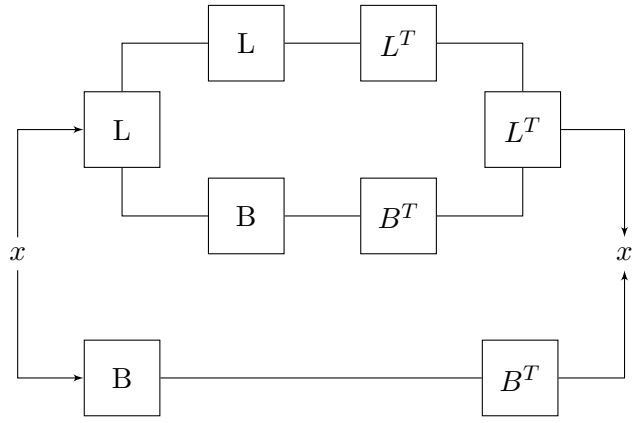


Figure 3.8: Fast wavelet transform and reconstruction

We know that the discrete wavelet transform has the structure $c = Ax$ where $c = \begin{bmatrix} a & b \end{bmatrix}^T$ are our coefficients and x is the input. Now we can ask ourselves how A looks like. We start

at the finest level j. This has a signal length $L = 2^j$. We know the application of the filter bank can be written as

$$\begin{bmatrix} a_j \\ b_j \end{bmatrix} = \begin{bmatrix} L_{2^j} \\ B_{2^j} \end{bmatrix} x. \tag{3.6.1}$$

In the second step we keep b_j and apply the filter bank to a_j which has length 2^{j-1}.

$$\begin{bmatrix} a_{j-1} \\ b_{j-1} \end{bmatrix} = \left[\begin{array}{c|c} \begin{matrix} L_{2^{j-1}} \\ B_{2^{j-1}} \end{matrix} & \\ \hline & I_4 \end{array}\right] \begin{bmatrix} a_j \\ b_j \end{bmatrix} = \left[\begin{array}{c|c} \begin{matrix} L_{2^{j-1}} \\ B_{2^{j-1}} \end{matrix} & \\ \hline & I_4 \end{array}\right] \begin{bmatrix} L_{2^j} \\ B_{2^j} \end{bmatrix} x$$

From here we can easily see the pattern. For example if we have $j = 3$, we can write A as

$$A = \left[\begin{array}{c|cc} \begin{matrix} L_2 \\ B_2 \end{matrix} & & \\ \hline & I_2 & \\ & & I_4 \end{array}\right] \left[\begin{array}{c|c} \begin{matrix} L_4 \\ B_4 \end{matrix} & \\ \hline & I_4 \end{array}\right] \begin{bmatrix} L_8 \\ B_8 \end{bmatrix}$$

or more general

$$A = \left[\begin{array}{c|c} \begin{matrix} L_{2^1} \\ B_{2^1} \end{matrix} & \\ \hline & I_{2^j-2^1} \end{array}\right] \left[\begin{array}{c|c} \begin{matrix} L_{2^2} \\ B_{2^2} \end{matrix} & \\ \hline & I_{2^j-2^2} \end{array}\right] \dots \left[\begin{array}{c|c} \begin{matrix} L_{2^{j-1}} \\ B_{2^{j-1}} \end{matrix} & \\ \hline & I_{2^j-2^{j-1}} \end{array}\right] \begin{bmatrix} L_{2^j} \\ B_{2^j} \end{bmatrix}.$$

The time complexity of the fast wavelet transform for signals of length $L = 2^n$ is $O(L)$ ([6]) which makes the scaling behavior even better than the FFT with its time complexity of $O(L \log L)$.

3.6.2 Boundary treatment

We started this thesis by defining the application of filter to infinite signals. In practice you usually have only a signal of finite length $x(0), \dots, x(L-1)$. We defined the application of a filter as a convolution of the input signal with a coefficient vector. For a causal FIR filter this means

$$(Hx)(n) = \sum_{k=0}^{N} h(k)x(n-k).$$

This can lead to problems on the boundary when not defined signal entries $x(n)$ with $n < 0$ or $n \geq L$ are used in the convolution. We will now discuss two approaches to this problem, signal continuation and boundary filters. A similar discussion of the boundary treatment is provided in [3] and yields a the basis for this chapter.

Signal continuation

A simple and widely used approach to solve this issue is signal continuation. Here, the signal is extended by assigning values to the undefined entries of x to certain values up to the point that we can calculate all entries of the filter output which uses original entries of x.
One simple example to this would be padding where you append a constant value, here 0, to both ends of the input signal by defining a new signal

$$\tilde{x}(n) = \begin{cases} 0 & \text{if } n < 0 \\ x(n) & \text{if } 0 \leq n < L \\ 0 & \text{if } n \geq L \end{cases}.$$

Now we can cut the matrix of the FWT such that only the rows and columns, which are applied to real values of x, are used and we have an input $\tilde{x}$ with the right dimensions. There are also other signal continuation techniques like periodic extension, symmetric extension or constant extension, which can be seen in Section 3.6.2.

(a) padding with zeros

(b) periodic extension

(c) symmetric extension

(d) constant extension

Figure 3.9: Different approaches to signal continuation

In the implementation of the WNO from [1] the symmetric signal extension mode was used which is formally defined by

$$\tilde{x}(n) = \begin{cases} x(-n+1) & \text{if } n < 0 \\ x(n) & \text{if } 0 \leq n < L \\ x(2L-n) & \text{if } n \geq L. \end{cases}$$

Boundary filters

The second approach for handling issues at the signal boundaries is boundary filters. In contrast to the signal extension, where you adjust the input signal, boundary filters do not enlarge or change the input signal, but change the filter bank.
Here we start with our infinite filter bank $\begin{bmatrix} L & B \end{bmatrix}^T$ and want to construct an orthogonal $L \times L$ matrix which we can apply to the finite signal.
We begin with our known filter bank:

$$\begin{bmatrix} L \\ B \end{bmatrix} = \begin{bmatrix} c(N) & c(N-1) & c(N-2) & \dots & c(0) & & \\ & & c(N) & c(N-1) & c(N-2) & \dots & c(0) \\ & & & & & \ddots & \ddots \\ d(N) & d(N-1) & d(N-2) & \dots & d(0) & & \\ & & d(N) & d(N-1) & d(N-2) & \dots & d(0) \\ & & & & & \ddots & \ddots \end{bmatrix}.$$

The first step is defining the block Toeplitz matrix. This matrix is constructed by interleaving the rows of L and B like

$$H_b = \begin{bmatrix} \ddots & \ddots & & & & & & & \\ & c(N) & c(N-1) & c(N-2) & \dots & c(0) & & & \\ & d(N) & d(N-1) & d(N-2) & \dots & d(0) & & & \\ & & & c(N) & c(N-1) & c(N-2) & \dots & c(0) & \\ & & & d(N) & d(N-1) & d(N-2) & \dots & d(0) & \\ & & & & & & & \ddots & \ddots \end{bmatrix}$$

or more formal as

$$\begin{cases} (H_b)_{2i,j} & = L_{i,j}, \\ (H_b)_{2i+1,j} & = B_{i,j}. \end{cases}$$

The second step is extracting all columns of the Toeplitz Band H_b which are not affected by the not defined input signal entries. So the i-th row is unaffected if $(H_b)_{ij} = 0$ for all $j < 0$ and $j \geq L$. This leads to the finite matrix

$$H_{in} = \begin{bmatrix} c(N) & c(N-1) & \dots & c(0) & & \\ d(N) & d(N-1) & \dots & d(0) & & \\ & & \ddots & \ddots & & \\ & & \ddots & \ddots & & \\ & & c(N) & c(N-1) & \dots & c(0) \\ & & d(N) & d(N-1) & \dots & d(0) \end{bmatrix}$$

which has L columns and $(L - N + 1)$ rows. Since we started with an orthonormal filter bank, the columns of the matrix are still orthonormal, so we have

$$(H_{in})(H_{in})^T = I.$$

The third step is to add $\frac{N+1}{2}$ rows, that use entries from the Toeplitz matrix, to the top and bottom of the matrix. The additional rows can be written as

$$H_t = \begin{bmatrix} c(1) & c(0) & 0 & \dots & & \\ d(1) & d(0) & 0 & \dots & & \\ c(3) & c(2) & c(1) & c(0) & 0 & \dots \\ d(3) & d(2) & d(1) & d(0) & 0 & \dots \\ & \ddots & & \ddots & & \\ c(N-2) & c(N-3) & \dots & c(1) & c(0) & 0 & \dots \\ d(N-2) & d(N-3) & \dots & d(1) & d(0) & 0 & \dots \end{bmatrix}$$

and

$$H_d = \begin{bmatrix} \dots & 0 & c(N) & c(N-1) & \dots & c(3) & c(2) \\ \dots & 0 & d(N) & d(N-1) & \dots & d(3) & d(2) \\ & & \ddots & & \ddots & & \\ & \dots & 0 & c(N) & c(N-1) & c(N-2) & c(N-3) \\ & \dots & 0 & d(N) & d(N-1) & d(N-2) & d(N-3) \\ & & & \dots & 0 & c(N) & c(N-1) \\ & & & \dots & 0 & d(N) & d(N-1) \end{bmatrix}.$$

These matrices have $\frac{N+1}{2}$ rows and L columns. They have a block structure $H_t = \begin{bmatrix} L & 0 \end{bmatrix}$ and $H_d = \begin{bmatrix} 0 & R \end{bmatrix}$. Our new matrix has now the following structure

$$\begin{bmatrix} \begin{bmatrix} L & 0 \end{bmatrix} \\ H_{in} \\ \begin{bmatrix} 0 & R \end{bmatrix} \end{bmatrix}$$

Due to (3.2.3) - (3.2.5) we know that the rows of L and R are orthogonal to the rows of H_{in}. If we assume $L >> N$, we also have that the rows of R are orthogonal to the rows of L.
The last step is to orthonormalize the rows of L and R, for example using the Gram-Schmidt orthogonalization. This yields the matrices L' and R' with orthonormal rows, giving the orthogonal matrix

$$H_{boundary} = \begin{bmatrix} \begin{bmatrix} L' & 0 \end{bmatrix} \\ H_{in} \\ \begin{bmatrix} 0 & R' \end{bmatrix} \end{bmatrix} \tag{3.6.2}$$

which we can now use as our filter bank and apply the discrete wavelet transform to our signal of length L.

3.7 2D Signals

Up to this point we only discussed one-dimensional signals $f \in L_2(\mathbb{R})$ or $x \in \ell_2$. Now we want to extend this theory to 2D signals and follow the construction in [3].
There are two different types of wavelet filters in two dimensions, separable and non-separable filters. Separable filters are products of one-dimensional filters and that is why they are quite easy to construct. Non-separable filters are harder to construct and therefore they are out of the scope of this thesis.
We will now construct the two-dimensional wavelet transform in the separable case for a discrete input signal $X \in \mathbb{R}^{(N+1)\times(N+1)}$ and also take a $(N+1)\times(N+1)$ matrix representing the one-dimensional wavelet transform e.g. we can use the previously constructed matrix $H := H_{boundary}$ to apply boundary filters. Now we apply the wavelet transform to each column of X separately which gives us a matrix Y_{col} where the i-th column of Y_{col} is the output signal of the application of the filter bank H to the i-th column of X.

$$Y_{col} := HX$$

Now we apply the filter bank to the rows of Y_{col} and that gives us the output Y of the two-dimensional signal X.

$$Y := (HY_{col}^T)^T = HXH^T$$

In the one-dimensional case the output signal is split into a high frequency part and a low frequency part. In the two-dimensional case, we split the output signal along each axis. That means that we can split the two-dimensional output signal into four parts. High frequency for the rows and high frequency for the columns, low frequency for rows and columns as well as the two mixed parts. This is portrayed in the figure Figure 3.10 where a is the low frequency part, d is the high frequency part and b and c are the mixed parts.

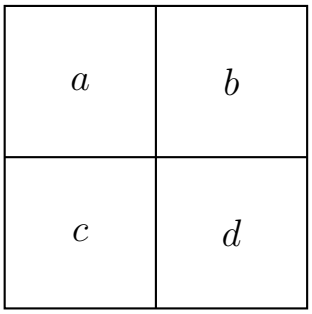


Figure 3.10: Output signal split into four parts

As in the one-dimensional case we can now apply the filter bank again to the output signal iteratively and get a wavelet transform for a two-dimensional input signal.

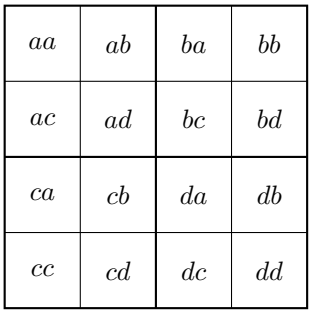


Figure 3.11: Second level wavelet transform to the whole signal

Analogously to the one-dimensional case, we also can only apply the filter to the low frequency parts in the next iteration and can construct the fast wavelet transform similar to the 1D scenario.

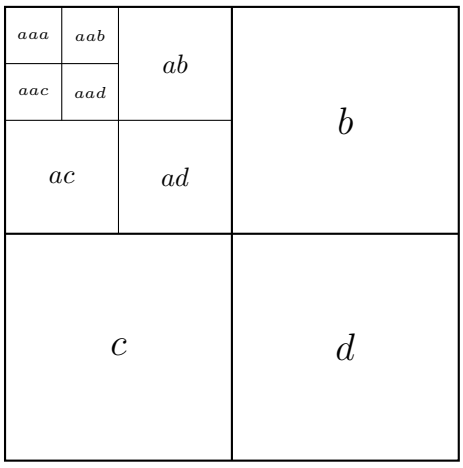


Figure 3.12: Third level FWT

Now we have the theoretical toolbox to build neural operators which uses the Fourier or wavelet transform to learn the nonlinear mapping between two infinite dimensional function spaces. This will allow us to learn and approximating PDE solutions for many different initial conditions.

Chapter 4

Benchmark Problems

To evaluate the model performance, the FNO and the variations that will be introduced in the later chapters are tested on benchmark problems. We present the PDEs which will be used for the various comparisons. We use the initial conditions and their corresponding solutions of the PDEs as training and test data. In this work the one-dimensional Burgers equation, the two-dimensional Darcy flow equation, the two-dimensional incompressible Navier–Stokes equation and the Euler equations will be used as benchmark problems. Not every model will be tested on every problem, but at some points we will look at the problem that best illustrates the differences between the models at that moment. In the following the problems are stated and the data generation is described.
The first three problems are PDEs which are stated similar in [1].

4.1 The 1D Burgers Equation

The 1D Burgers equation serves as a simplified model for a range of systems involving convection and diffusion, such as traffic flow, gas dynamics, and turbulence modeling. The formal formulation is typically stated as

$$\begin{aligned} \partial_t u(x,t) + \frac{1}{2}\partial_x u^2(x,t) &= \nu \partial_{xx} u(x,t), && x \in (0,1),\, t \in (0,1] \\ u(0,t) &= u(1,t), && x \in (0,1),\, t \in (0,1] \\ u(x,0) &= u_0(x), && x \in (0,1) \end{aligned}$$

where $u_0(x) \in L^2_{per}((0,1);\mathbb{R})$ is the initial condition from the set of periodic L^2 functions, and $\nu > 0$ is the viscosity. Depending on the viscosity parameter, the solution can exhibit smooth evolution or develop sharp gradients and discontinuities, making it a valuable test for evaluating the capacity of machine learning models to capture nonlinear transport phenomena.

In this work we assume $\nu = 0.1$ and want to learn the operator $\mathcal{D} : u_0(x) \mapsto u(x,1)$. The used data set consists of 1100 samples of initial conditions and the corresponding solutions of the Burger equations. They have a maximum resolution of 2^{13} which can be downsampled by a subsampling factor r such that the used resolution is $h = \frac{2^{13}}{2^r}$.
The data generation process is described in appendix A.3.1 in [7]. They generated the initial condition according to $u_0 \sim \mathcal{N}(0, 625(-\Delta + 25I)^{-2})$ with periodic boundary condition and solved the equation with a split step method. First the heat equation part is solved exactly in the Fourier domain and afterwards the non-linear part solved in Fourier space with a forward Euler method. The data is generated with a spatial mesh with resolution $2^{13} = 8192$.

4.2 The 2D Darcy Flow Equation

The Darcy flow equation is used to model the flow of fluids through a porous medium. In two spatial dimensions, it is given by

$$\begin{aligned} -\nabla \cdot (a(x,y)\nabla u(x,y)) &= f(x,y) && x,y \in (0,1) \\ u(x,y) &= u_0(x,y) && (x,y) \in \partial(0,1)^2 \end{aligned}$$

where $u(x,y)$ is the pressure field, $a(x,y)$ is the spatially varying permeability, $f(x,y)$ is a source term and $u_0(x,y)$ is the initial condition. For the practical use of the equation we restrict the domain to a unit box $x \times y \in (0,1)^2$ with zero Dirichlet boundary conditions. For data generation (compare Appendix A.3.2 in [7]) the coefficients $a(x,y)$ were drawn according to $a \sim \psi_{\#}\mathcal{N}(0,(-\Delta + 9I)^{-2})$ and the boundary condition was a zero Neumann boundary condition on the Laplacian of u. The push-forward in the data generation was defined pointwise and the mapping $\psi : \mathbb{R} \to \mathbb{R}$ returns 12 for the positive part of the real line and 3 for negative numbers. $f(x,y) = 1$ was assumed throughout the simulation. The solutions u were calculated with a second-order finite difference scheme on a 421×421 grid and solution for different grids were obtained by downsampling.

4.3 2D Navier-Stokes Equation

The Navier-Stokes equation is used in fluid flow problems and is classified as a second order nonlinear parabolic PDE. Here the incompressible Navier-Stokes equation is given in the vorticity-velocity form:

$$\begin{aligned} \partial_t \omega(x,y,t) + u(x,y,t) \cdot \nabla \omega(x,y,t) &= \nu \Delta \omega(x,y,t) + f(x,y), && x,y \in (0,1),\ t \in (0,T], \\ \nabla \cdot u(x,y,t) &= 0, && x,y \in (0,1),\ t \in [0,T], \\ \omega(x,y,0) &= \omega_0(x,y), && x,y \in (0,1). \end{aligned}$$

The viscosity of the fluid is given by $\nu \in \mathbb{R}$ and $f(x,y)$ is the source function and the functions u and ω are the velocity and vorticity fields. The initial condition is given by $\omega_0(x,y)$ and the viscosity is set to $\nu = 10^{-3}$.
We use the vorticity field at the time steps $t \in [0,10)$ as the input for our models and we want to do two different experiments. Once we only aim to predict the vorticity field just at time step $t = 10$ and once we want to predict the vorticity field for all time steps $t \in [10,20]$ like it is common in the literature ([7], [1]). In the following we will refer to these two variants as the single-step Navier-Stokes problem and the multi-step Navier-Stokes problem.
During the data generation in Appendix A.3.3 from [7] a fixed forcing

$$f(x,y) = 0.1(\sin(2\pi(x+y)) + \cos(2\pi(x+y)))$$

is assumed and $\omega_0(x,y)$ is drawn according to $\mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{-2.5})$ with periodic boundary conditions. To solve this the authors of [7] used the stream-function formulation and a pseudo-spectral method and they solved the equation on a 256×256 grid.

4.4 Euler Equations

The 1D compressible Euler equations in conservation form is given by:

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+P\\ (E+P)u\end{pmatrix}=0.$$

Here ρ represents the density, u the velocity, P is the pressure and E is the total energy. The equations are also used in fluid dynamics. The equation will be applied to a Riemann Problem, a 1D initial value with a discontinuity at the center and piecewise constant initial conditions. This formulation is stated similarly in [9]. We will not do our own numerical experiments with this problem, but refer to results from the literature.

Chapter 5

Fourier Neural Operator

Often PDEs are hard or even impossible to solve analytically. Usually we would now use a numerical solver like the finite element, finite difference or finite volume method. These can be computationally expensive, especially if you want to solve the PDE multiple times for different initial conditions.
The approach we now want to use to solve this issue is using a neural network. These need to be trained only once, which may require large computational resources, but after the training process, the application of the neural network is much faster than using one of the previously mentioned numerical techniques.
In this case we begin by introducing the Fourier Neural Operator (FNO) which was presented in [7]. We start this chapter by stating the formal problem formulation our model will be trained to solve. Afterwards we introduce the network architecture and the algorithm of the model. We conclude the chapter by summarizing the results from [7] and examining the problems of the FNO with our own numerical experiments and the results from [9].

5.1 Problem Formulation

Like in [1], we formulate the PDE as an operator $\mathcal{D} : \mathcal{A} \mapsto \mathcal{U}$, where $\mathcal{A}$ and $\mathcal{U}$ are normed vector spaces with functions taking values in $\mathbb{R}^{d_a}$ and $\mathbb{R}^{d_u}$. The functions are assumed to be sufficiently smooth to admit the required partial derivatives, so $\mathcal{A} := \mathcal{C}(D; \mathbb{R}^{d_a})$ and $\mathcal{U} := \mathcal{C}(D; \mathbb{R}^{d_u})$ where D is a given domain $D \subset \mathbb{R}^n$. We now want to approximate the operator $\mathcal{D}$ with a neural network

$$\tilde{\mathcal{D}} : \mathcal{A} \times \Phi_{NN} \mapsto \mathcal{U}$$

where Φ_{NN} are the weights and biases of the neural network.
The input functions $a \in \mathcal{A}$ and output functions $u \in \mathcal{U}$ are continuous functions but for practical implementation, we need to approximate them by finite dimensional input and output signals. Therefore we use a discretization of D using n_D points $\{x_j\}_{j=1}^{n_D} \subset D$ and assume that we have access to the evaluations of all input and output functions a and u at all points in $\{x_j\}_{j=1}^{n_D}$. So we have a data set

$$\{a_i \in \mathbb{R}^{n_D \times d_a}, u_i \in \mathbb{R}^{n_D \times d_u}\}_{i=1}^{N}$$

where N is the number of input-output pairs we have to train and test the neural network.

5.2 Network Architecture

The Fourier Neural Operator is a special neural network to solve partial differential equations for multiple initial conditions. It is introduced in the paper [7]. For better comparison with the WNO we use the notation from [1].
The construction of the FNO will be shown in the following section. It uses a regular convolutional neural network (CNN) as well as layers which use the Fourier transformed input signal as an input.

The first step is lifting the input signal $a(x) \in \mathbb{R}^{d_a}$ to a higher dimensional space with dimension d_v. To do that we use a transformation $P(a(x)) : \mathbb{R}^{d_a} \mapsto \mathbb{R}^{d_v}$.
We define $v_0(x) = P(a(x))$.
After that we define an update operator $G : \mathbb{R}^{d_v} \mapsto \mathbb{R}^{d_v}$ and a number l of update steps. Applying the operator means we define $v_{j+1} = G(v_j) \, \forall j = 0, \dots l-1$.
The update operator is defined as

$$v_{j+1} := g((K(a;\phi) * v_j)(x) + Cv_j(x)). \tag{5.2.1}$$

Here C is a linear transformation from $\mathbb{R}^{d_v}$ to $\mathbb{R}^{d_v}$ which will be represented by a convolutional neural network.
$K : \mathcal{A} \times \Phi \mapsto \mathcal{L}(\mathcal{U},\mathcal{U})$ is an integral operator on $\mathcal{A}$ with parameter $\phi \in \Phi$ which is defined as

$$(K(a;\phi) * v_j)(x) := \int_{D \subset \mathbb{R}^d} k(a(x), x, y; \phi) v_j(y) \, dy$$

This integral operator is also called Fourier layer when we talk about the network architecture. In practice this is split into two steps. First we apply a transformation $T(v_j(x))$ and apply a kernel in the transformed space $R_\phi * T(v_j(x))$. Then we apply the inverse transformation $T^{-1}(R_\phi * T(v_j(x)))$.
The function g is a non-linear activation function which is not applied if $j = l - 1$. The last step is applying a backward transformation $Q(v_l(x)) : \mathbb{R}^{d_v} \mapsto \mathbb{R}^{d_v}$ where the output $Q(v_l(x))$ is our solution $u(x)$.

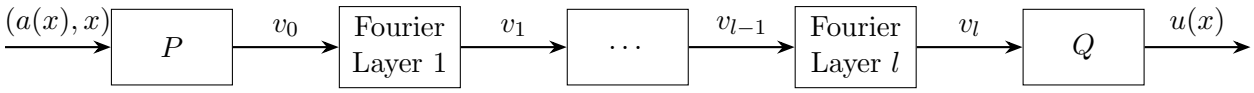


Figure 5.1: FNO Network Architecture

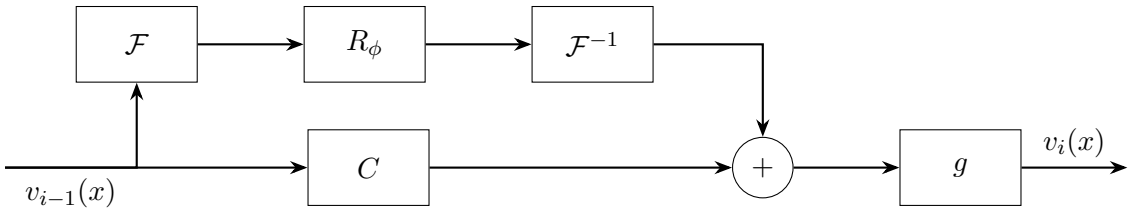


Figure 5.2: i-th Fourier layer of the FNO

In our example from [7] we use a shallow fully connected neural network as the transformation P. C is a convolutional neural network (CNN), T is the discrete Fourier transform

$\mathcal{F}$, the applied kernel R_ϕ are weights from the neural network, g is a activation function, in our case the Gaussian error linear unit (GELU)

$$\text{GELU}(x) = \frac{1}{2}x(1 + \text{erf}(\frac{x}{\sqrt{2}}))$$

where erf is the gaussian error function and Q is another shallow FNN.
In Algorithm 2 we write this procedure in an algorithm as pseudo-code.

Algorithm 2 Algorithm of the FNO

Input: N-samples of the pair $\{a(x) \in \mathbb{R}^{n_D \times d_a}, u(x) \in \mathbb{R}^{n_D \times d_u}\}$, coordinates $x \in D$, and network hyperparameter θ_{NN}.

1: Stack the inputs: $\{a(x), x\} \in \mathbb{R}^{n_D \times 2d_a}$.
2: **for** epoch $= 1, \ldots,$ epochs **do**
3: Uplift the input using transformation $P(\cdot)$: $v_0(x) = P(\{a(x), x\})$.
4: **for** $j = 1, \ldots, l$ **do**
5: Decompose the input using Fourier transform: $\mathcal{F}(v_j(x))$.
6: Parameterize the NN kernel k_ϕ in the frequency space: $R_\phi * \mathcal{F}(v_j(x))$.
7: Reconstruct the convolved input: $v^1_{j+1}(x) = \mathcal{F}^{-1}(R_\phi * F(v_j(x)))$.
8: Perform the linear transform: $v^2_{j+1}(x) = Cv_j(x)$ using a CNN C.
9: Add the outputs of steps 7 and 8: $\tilde{v}_{j+1}(x) = (v^1_{j+1} + v^2_{j+1})(x)$.
10: **if** $j \neq l$ **then**
11: Apply the activation to complete the iteration: $v_{j+1} = g(\tilde{v}_{j+1}(x))$.
12: **end if**
13: **end for**
14: Compute the final output: $\hat{u}(x) \in \mathbb{R}^{n_D \times d_u} = Q(v_l(x))$ where $Q(\cdot) : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$ is an FNN.
15: Compute the loss: $L(u, \hat{u})$.
16: Compute the gradient of the loss: $\frac{\partial L(u,\hat{u})}{\partial \theta_{\text{NN}}}$.
17: Update the parameters of the network using the gradient.
18: **end for**

Output: Predicted solution $\hat{u} \in U$, parameters of NN θ_{NN}.

5.3 Numerical Results

In [7] the FNO was tested on the 1D Burgers equation and the 2D Darcy flow and it was compared against several other networks trained for solving these equations. For comparison the following models were used in [7].

- NN: Neural Network for point-wise feedforward operations
- RBM: Reduced Basis Method [22]
- FCN: Neural network architecture that utilities Fully Convolutional Networks [23]
- PCANN: A neural network using principal component analysis on the data [24]
- GNO: Graph Neural Operator [16]
- MGNO: Multipole Graph Neural Operator [25]
- LNO: neural operator based on low rank decomposition of kernels [15]

The simulations were done with different resolutions and the FNO was the best operator among all tested models for all resolutions.

Network	**s=256**	**s=512**	**s=1024**	**s=2048**	**s=4096**	**s=8192**
NN	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO	**0.0149**	**0.0158**	**0.0160**	**0.0146**	**0.0142**	**0.0139**

Table 5.1: Table 3 from [7]: Relative errors for Burgers equation at different resolutions

In Section 5.3 we can see that the FNO performed about 29% better than the LNO which was in most cases the second best operator. Another thing that you can see in the data is that the performance is consistent across different spatial resolutions.
In the case of the 2D Darcy Flow equation, we see in Section 5.3 that the advantage of the FNO becomes even more clear since the error of the FNO is 58% smaller than the error of the second best operator, the RBM. Here the FNO performs best for the highest resolution but the difference is small and for the other resolutions, the performance of the FNO is consistent.

According to section 5.4 of [7] the FNO was even able to predict the solutions of the PDEs for higher resolution than the operator was trained on. In this setting the FNO was trained on the Navier-Stokes equation with a spatial resolution of 64×64 and a temporal resolution of 20 and it was able to transfer to a resolution of 256×256×80. Among the other tested benchmark operators U-Net, TF-Net and ResNet it was the only model that was able to do zero-shot super resolution.
Once trained, the model can predict the outcome of the problems much faster than classical solvers. In [7] the FNO had an inference time of 0.005s for the Navier-Stokes problem on a 256×256 grid while the pseudo-spectral method had a runtime of 2.2s. That shows that the

Network	**s=85**	**s=141**	**s=211**	**s=421**
NN	0.1716	0.1716	0.1716	0.1716
FCN	0.0253	0.0493	0.0727	0.1097
PCANN	0.0299	0.0298	0.0298	0.0299
RBM	0.0244	0.0251	0.0255	0.0259
GNO	0.0346	0.0332	0.0342	0.0369
LNO	0.0520	0.0461	0.0445	–
MGNO	0.0416	0.0428	0.0428	0.0420
FNO	**0.0108**	**0.0109**	**0.0109**	**0.0098**

Table 5.2: Table 4 from [7]: Relative errors for Darcy Flow at different resolutions.

FNO can significantly speed up calculations if you need to solve a problem for many initial conditions.
We now want to take a deeper look at the numerical results of the FNO and test it with different parameters for the one-dimensional Burgers equation. The model should learn the solution operator that maps from the initial condition to the solution at the fixed final time $T = 1$.
To be able to compare the results to the WNO and use the same data loader in the implementation. For these reasons we do not use the implementation from [7] but use the code basis from the WNO [2] and replace the discrete wavelet transform with the real fast Fourier transform.
The model is trained using the Adam optimizer for 500 epochs with a weight decay of 10^{-6} and an initial learning rate of 10^{-3}. The learning rate is scheduled to be halved every 50 epochs. During training the relative L^2-error is the loss function and a batch size of 20 is used.
The model used the $1D$ version of the FNO parameterized by the modes, which is the number of Fourier modes retained in the spectral convolution, the width, which is the dimension of the linear transformation denoted as d_v in Section 5.2 and the resolution h. We performed a grid search over the parameters with

$$\begin{aligned} r &\in \{10^2, 10^3, 10^4\} \\ modes &\in \{10, 12, 14\} \\ width &\in \{32, 40, 64\} \end{aligned}$$

The results are portrayed in table Table 5.3

The results clearly show that the performance depends on the width since for each resolution and width, the largest width yields the best results.
The results for the other two parameters are harder to interpret since there are parameter configurations where 12 modes perform better than 14 and other combinations of resolution and width where 14 modes are better than 12 modes. 10 modes seem too few since 12 and 14 modes perform consistently better in our tests.
Which resolution is the best also depends on the other parameters which becomes clear if you look at the resolution of 512. If you combine this with 10 modes and a width of 32 you get the worst observed results and on the other hand with 14 modes and a width of 64 you get the best observed results.
We now take a look at the results of the best model from above. From the 100 test samples, we look at the three with the smallest error and the three samples with the largest error.

Resolution	Modes	Width	Mean Error in %
512	10	32	0.1190
512	10	40	0.0751
512	10	64	0.0622
512	12	32	0.0838
512	12	40	0.0692
512	12	64	0.0641
512	14	32	0.1235
512	14	40	0.0750
512	14	64	0.0507
1024	10	32	0.0915
1024	10	40	0.0741
1024	10	64	0.0623
1024	12	32	0.0871
1024	12	40	0.0674
1024	12	64	0.0564
1024	14	32	0.0867
1024	14	40	0.0733
1024	14	64	0.0545
2048	10	32	0.0919
2048	10	40	0.0747
2048	10	64	0.0620
2048	12	32	0.0874
2048	12	40	0.0662
2048	12	64	0.0559
2048	14	32	0.0858
2048	14	40	0.0741
2048	14	64	0.0549

Table 5.3: FNO model performance for Burgers equation sorted by resolution, modes, and width.

Sample	Error
2	0.00016
53	0.00016
70	0.00017
⋮	⋮
76	0.00259
96	0.00385
94	0.00453

Table 5.4: Test errors per test sample ordered by test error

We see here that the error can differ significantly from sample to sample such that the largest error is more than 28 times larger than the smallest error. Samples with small errors have smaller maximal absolute values in the initial condition and PDE solution than samples

with large errors.
To back up this observation we calculate the range

$$R_x := \max_x(u(x,0)) - \min_x(u(x,0))$$
$$R_y := \max_x(u(x,1)) - \min_x(u(x,1))$$

of our samples.

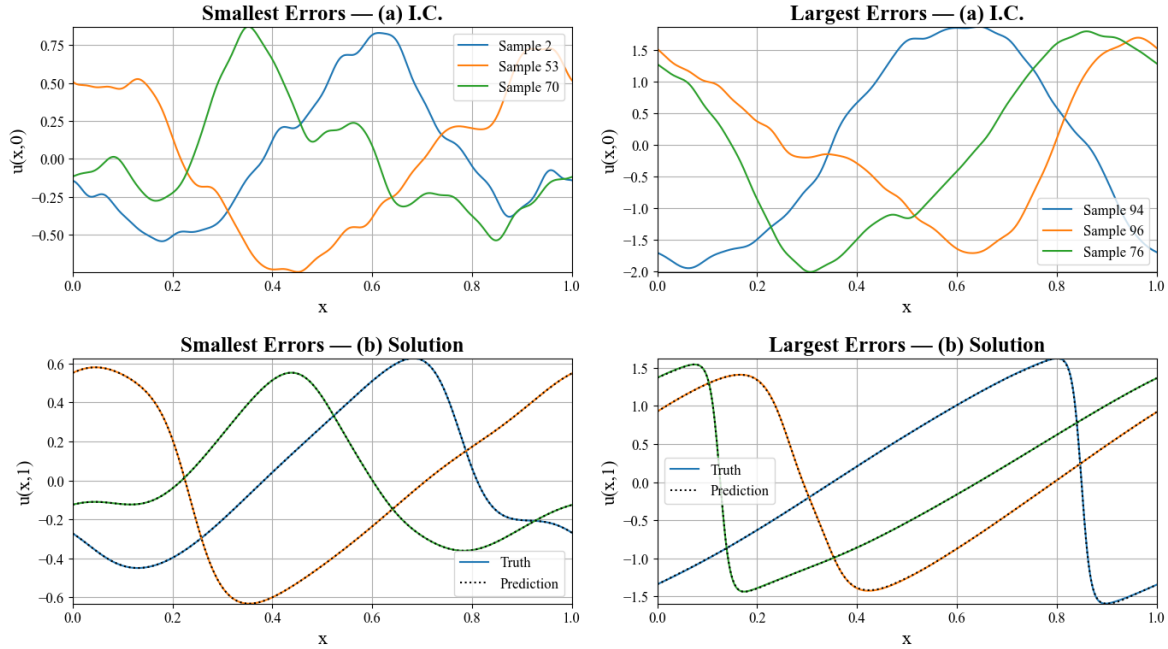


Figure 5.3: Comparison of the three smallest and largest prediction errors for the FNO model. In the top row, we see the initial conditions and on the bottom, we see the corresponding PDE solutions. On the left we see the test samples the model performed best on and on the right are the worst test samples.

We now look at the connection of the error with the range of the initial condition and the solution. To assess the linear dependence between the error E and the signal ranges R_x and R_y, we compute the Pearson correlation coefficients between E and each of the two quantities. The results were:

$$\text{Corr}(E, R_x) = 0.71$$
$$\text{Corr}(E, R_y) = 0.67$$

These values indicate a relevant positive correlation between the relative error and both the range of the input and the range of the target output. In particular, the model tends to produce larger relative errors for samples with a wider spread in the input or solution values. This observation was also reproduced in a similar setting in [8], where it was also shown that the FNO struggles with discontinuous inputs. In [8] the FNO was tested on the wavefront tracking scheme from [20] and the Godunov scheme from [21]. For each problem, it was trained on an easy data set with few discontinuities and a small range and a complex data set with more discontinuities and a greater range of the input. In Table 2 and Table 3 of [8]

you can see that the test error of the FNO is smaller for the easy data set and nearly twice as high for the complex data set. The exact numbers are depicted in Table 5.5.

Problem	**Easy Data Set**	**Complex Data Set**
Wavefront Tracking Scheme	0.078	0.127
Godunov Scheme	0.064	0.114

Table 5.5: Relative L_2 loss of the FNO for easy and complex data sets from [8].

This shows that the FNO is not able to predict with the same precision for complex inputs as it is for the easy data set even though the models were trained with 8040 complex and 2960 easy samples for the wavefront tracking scheme and 8040 complex and 3000 easy samples for the Godunov scheme. So even though the model trained on the complex data had more samples, it could not achieve the same small error as the model trained on the easy data.

Chapter 6

Network Variations

We now have introduced the FNO and have seen that the FNO performs better than the previously known neural operators. But also we have seen that the FNO struggles with discontinuities and big ranges in the input data. In the following we will try different approaches to improve the FNO in ways such that it either improves the general performance or specifically tackles the weaknesses of the FNO.

6.1 Bias

The nature of the FNO is that the operator learns the data in a spectral domain, which by definition learns patterns in the data globally. The problem with that is that the FNO struggles to detect sudden shifts and discontinuities as we have seen above.
One option to improve the operator here is by adding a bias as it was introduced in [9]. The authors modified the architecture of the FNO by adding a bias term to each Fourier layer. The biases are element-wise constants and should capture and sharpen the corners of discontinuities which the model could not capture in spectral domain or with the linear weighting terms.

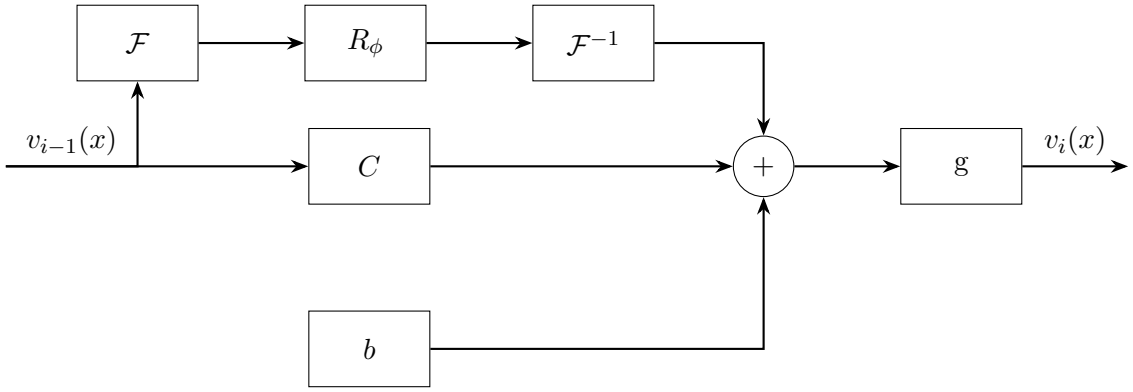


Figure 6.1: Fourier layer with bias

The authors also changed the uplifting operator P from a fully connected neural network to a convolutional neural network to reduce the number of network parameters and they experimented by training the model with different loss functions like the usual L_2 norm but also the L_1 and Sobolev norm.

The tests in [9] showed that the modified FNO with the L_1 loss function had a means squared error (MSE) of 0.058% while the original FNO had a MSE of 0.2%. But even if you compare the baseline FNO with the modified FNO with L_2 loss function the modified version only had a MSE of about 0.082%, which is less then half the error of the original FNO by [7].
In Figure 6.3 we compare the exact solution to the predictions of the original FNO and the modified FNO with L_1 loss function for discontinuous initial conditions.
The modified FNO performed way better at the edges of discontinuities in comparison to the original FNO. If we used the L_2 or Sobolev loss function the modified FNO still outperforms the original but we get more noise around the edges as we can see in the appendix of [9].

We have seen that with the modifications the FNO performed better around corners with discontinuous initial conditions and it had the better overall MSE. One downside is that the model lost its upscaling ability. The authors of [9] trained the models on a data set with resolution 256 and the original FNO was able to keep nearly the same average MSE for upscaled test data sets with a resolution up to 8196. The modified FNO with L_2 loss function was not able to keep the same average MSE in the early stages of upscaling. However it was still able to outperform the original FNO for all resolutions up to the maximum of 8196 as we can see in Figure 6.2. We also see that both models clearly outperform the U-Net operator.

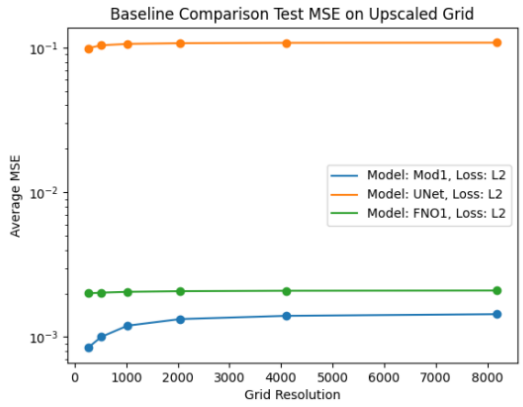


Figure 6.2: Upscaling performance of models trained on 256-resolution data, evaluated on up to 8196-resolution data (Figure 3a from [9]).

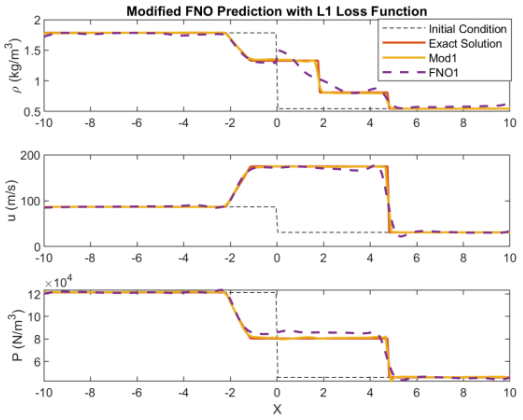


Figure 6.3: Comparison of the predictions of the original and the modified FNO at discontinuities (Figure 4a from [9]).

6.2 Wavelet Neural Operator

Based on the FNO another neural operator was introduced in [1]. While the FNO exploits the period and global nature of the Fourier transform to learn mapping between the initial conditions and the solutions of PDEs, it can have problems to model localized features or discontinuities. To address these problems, the wavelet neural operator (WNO) was proposed in [1]. It is based on the model architecture of the FNO but the core difference lies in the choice of basis in the transform. The FNO uses the Fourier basis, while the WNO uses a wavelet basis, which has the advantage of an localized function basis which enables the detection of localized patterns and sharp transitions.

6.2.1 Network Architecture

The network architecture of the WNO follows the same principles as the FNO: lifting the input to a higher dimensional space, iteratively applying an integral kernel and a nonlinear activation function and then transforming back to the initial dimension to get the solution. The difference is that the WNO does not use the Fourier transform $\mathcal{F}$ but the wavelet transform $\mathcal{W}$ and then the convolution is carried out on the wavelet coefficients instead of the Fourier coefficients.

Algorithm 3 Algorithm of the WNO

Input: N-samples of the pair $\{a(x) \in \mathbb{R}^{n_D \times d_a}, u(x) \in \mathbb{R}^{n_D \times d_u}\}$, coordinates $x \in D$, and network hyperparameters θ_{NN}.

1: Stack the inputs: $\{a(x), x\} \in \mathbb{R}^{n_D \times 2d_a}$.
2: **for** epoch = 1, ..., epochs **do**
3: Uplift the input using transformation $P(\cdot)$: $v_0(x) = P(\{a(x), x\})$.
4: **for** $j = 1, \ldots, l$ **do**
5: Decompose the input using wavelet transform: $\mathcal{W}(v_j(x))$.
6: Parameterize the NN kernel k_ϕ in the wavelet domain: $R_\phi * \mathcal{W}(v_j(x))$.
7: Reconstruct the convoluted input: $v^1_{j+1}(x) = \mathcal{W}^{-1}(R_\phi * \mathcal{W}(v_j(x)))$.
8: Perform the linear transform: $v^2_{j+1}(x) = Cv_j(x)$ using a CNN C.
9: Add the outputs of steps 7 and 8: $\tilde{v}_{j+1}(x) = (v^1_{j+1} + v^2_{j+1})(x)$.
10: **if** $j \neq l$ **then**
11: Apply the activation to complete the iteration: $v_{j+1} = g(\tilde{v}_{j+1}(x))$.
12: **end if**
13: **end for**
14: Compute the final output: $\hat{u}(x) \in \mathbb{R}^{n_D \times d_u} = Q(v_l(x))$ where $Q(\cdot) : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$ is an FNN.
15: Compute the loss: $L(u, \hat{u})$.
16: Compute the gradient of the loss: $\frac{\partial L(u,\hat{u})}{\partial \theta_{\text{NN}}}$.
17: Update the parameters of the network using the gradient.
18: **end for**

Output: Predicted solution $\hat{u} \in U$, parameters of NN θ_{NN}.

6.2.2 Numerical Results

The WNO performed better than the FNO in four out of five tested problems. Only in the one-dimensional Burgers equation, the FNO performed better. The WNO, on the other hand, outperforms the FNO in every two-dimensional problem. The exact results from [1] are given in Table 6.1. Here we can see that the FNO performed slightly better than the WNO for the Burgers equation and the WNO slightly better than the FNO for the Darcy flow equation and significantly better for the Navier-Stokes, Allen-Cahn and Wave-advection equations.

Besides the overall better performance, the WNO has two major advantages compared to the FNO.
The first point is that the WNO was able to learn the problem also for more complex geometry as well. While the FNO is only able to train and run on a rectangular domain, the WNO could also be used with a triangular domain. The WNO predictions had a mean relative L_2

Problem	FNO	WNO
Burgers Equation	$\approx 1.6\%$	$\approx 1.75\%$
Darcy Flow Equation	$\approx 1.08\%$	$\approx 0.84\%$
Navier-Stokes	$\approx 1.28\%$	$\approx 0.31\%$
Allen-Cahn	$\approx 0.93\%$	$\approx 0.21\%$
Wave-advection	$\approx 47.7\%$	$\approx 0.62\%$

Table 6.1: Test errors of the WNO and FNO from table 3 in [1]

error of approximately 0.77% for the Darcy flow equation on a triangular domain, which is even slightly better than the performance on a rectangular grid.
The second point is that the wavelet transform stores spatial information, which allows the WNO to perform well in modeling sharp jumps in the solution fields. The model was tested for a variation of the Burgers equation where discontinuities occurred in the solution field. The exact formulation of the used PDE is

$$\begin{aligned} \partial_t u(x,t) + \frac{1}{2}\partial_x u^2(x,t) &= \frac{0.01}{\pi}\partial_{xx} u(x,t), && x \in [0,1],\ t \in [0,1] \\ u(x=-1,t) &= u(x=1,t) = 0, && t \in [0,1] \\ u(x,0) &= -\sin(\pi x) + \zeta \sin(\pi x), && x \in [0,1] \end{aligned}$$

where ζ is randomly chosen from [0,0.5]. The WNO now not only learns the velocity field at $t = 10$ but for $t \in (10, 50]$ or, more formally, it learns the operator

$$u|_{(-1,1)\times[0,10]} \mapsto u|_{(-1,1)\times[10,50]}.$$

The model was trained with a spatial resolution of 512 and a time step of 0.02.
The model achieved a mean relative L_2 error of 0.23% and the predictions in Figure 5 of [1] shows that the model was able to predict sharp jumps.
On the other hand the paper [8], which showed that the FNO struggles with complex data sets, also tested the WNO and got similar results shown in Table 6.2. The results of the WNO are slightly better than the results of the FNO but the general trend is the same here. The WNO also struggles with the more complex data sets in this examples, which shows that the WNO is not generally able to predict discontinuities or handle a wide range in the input data.

Problem	Easy Data Set	Complex Data Set
Wavefront Tracking Scheme	0.076	0.126
Godunov Scheme	0.063	0.113

Table 6.2: Relative L_2 loss of the WNO for easy and complex data sets from [8].

As a remark regarding the runtime of the WNO, we would like to point out that the WNO exhibits a significantly slower runtime compared to the FNO, even though the discrete wavelet transform theoretically scales better than the FFT. In [8] the authors measured the runtime of the FNO and WNO for the wavefront tracking scheme and the Godunov scheme for both easy and complex data sets and yielded the results in Table 6.3.

	Wavefront Tracking Scheme		Godunov Scheme	
Problem	**Easy Data Set**	**Complex Data Set**	**Easy Data Set**	**Complex Data Set**
FNO	6.91s	18.78s	42.98s	111.34s
WNO	5.04s	14.56s	42.46s	119.45s

Table 6.3: Training Time per Epoch of the FNO and WNO ([8])

6.3 Boundary Wavelets

The WNO from [1] uses symmetric padding to continue their signals in the discrete wavelet transform. This leads to synthetic signals at the edges of the domain which do not represent the physical or mathematical reality of the PDE. This could interfere with the WNOs ability to learn the patterns of the data around the edges. If we test the WNO on simple boundary conditions like the zero boundary conditions in the Darcy flow problem, this is not a big issue since the prediction of the boundary values is simple regardless of the exact boundary treatment.
But we can observe bigger errors near the boundary if we train the WNO on problems with more involved boundary conditions like we see in Figure 6.4.

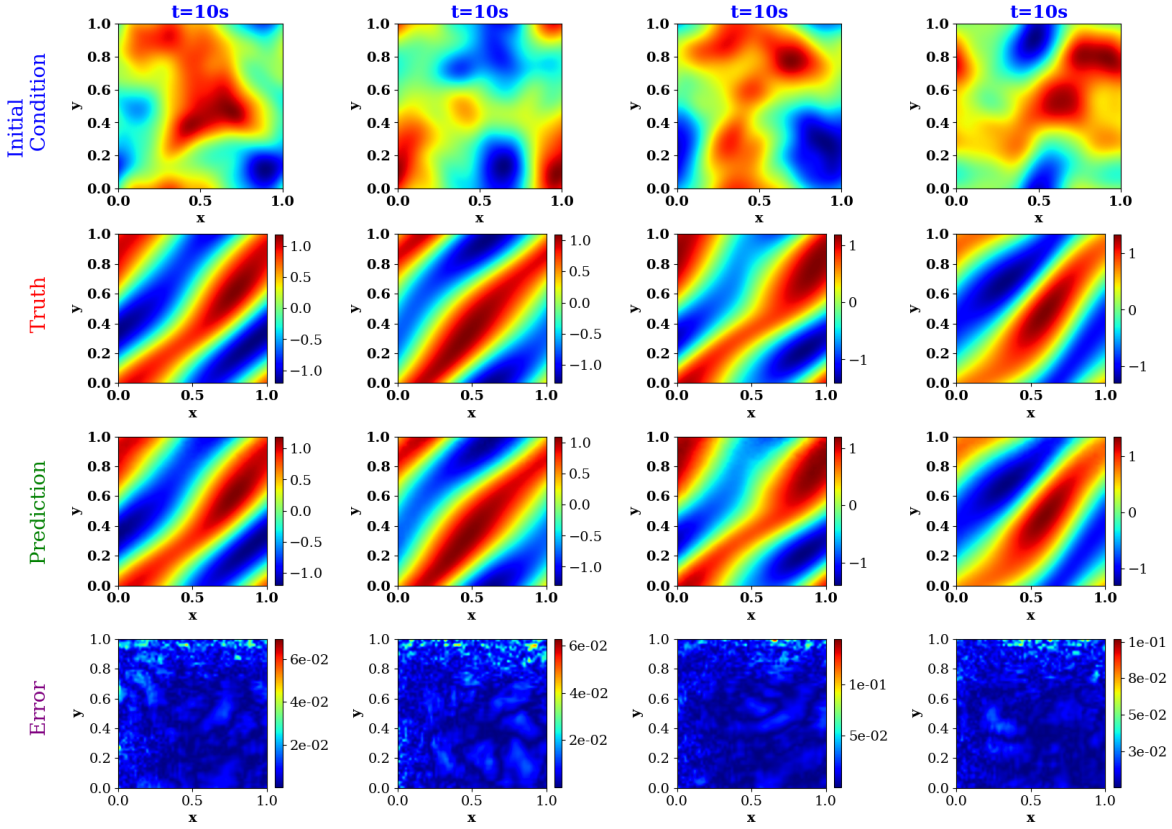


Figure 6.4: Navier-Stokes equation with a spatial resolution of 64×64. We train the model with the vorticity data for $t < 10$ and predict the vorticity field at $t = 10$. In the first row we see the initial condition of the vorticity field, the second row is the truth at $t = 10$, the third row depicts the prediction of the WNO and the fourth row shows the error of the WNO. Each column represents a sample of the test data set.

We can clearly see that for each sample the biggest errors occur at or near the edges. For this reason we created a version of the WNO which uses boundary wavelets in the discrete wavelet transform. Like that we do not need to expand the signals with artificial entries at the edges but we can rely purely on the original data. For the implementation we used the python library ptwt which was published on [5] and used the *Matrixwavedec* function to perform the DWT. We need to make sure in the implementation, that the DWT input has even length, otherwise the python function needs to add an entry to run.
The model with boundary wavelets outperforms the original WNO on the Navier-Stokes equation which we can see on the left side in Figure 6.5.

Boundary Treatment Method	**Test Error**
Symmetric padding	0.071%
Boundary filter	0.027%

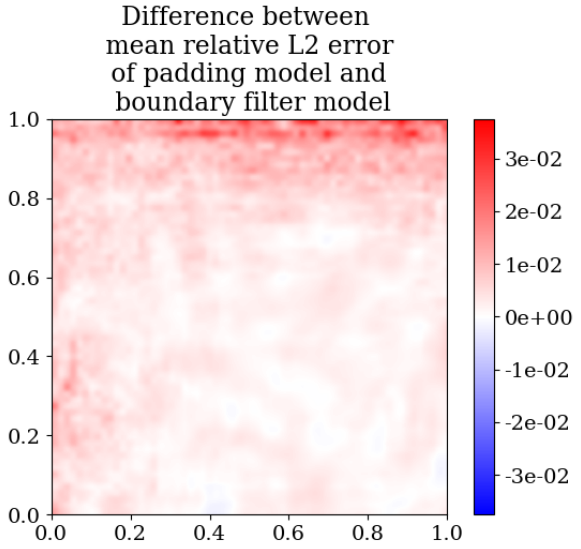


Figure 6.5: (**Left**) Test error comparison of the WNO with signal extension (padding) and boundary filter on the example of the Navier-Stokes equation.
(**Right**) Navier-Stokes equation with a spatial resolution of 64×64. We train the model with the vorticity data for $t < 10$ and predict the vorticity field at $t = 10$. The plot shows the mean difference over all test samples of the error of the WNO with signal extension and the WNO with boundary filter in the spatial domain. The regions with positive (red) values are the regions where the WNO with boundary filters performed better.

We do not only observe that the overall accuracy improved with the boundary filter but we also that the improvements mainly occur on the edges of the domain which is depicted in Figure 6.6 and on the right of Figure 6.5.

This shows that using boundary filter instead of extending the signal with artificial entries can improve the overall performance of the model and especially the error near the edges of the domain if the boundary conditions of the PDE are sufficiently complex.

Although the model accuracy at the boundary increases, the model also becomes more susceptible to error propagation. To show this we simulate the Navier-Stokes equation not only for the next time step but the next ten time steps by always including the results of the current time step to solve for the next one. The results can be seen in Figure 6.7 where we can see that the model struggles with predicting the correct values at the corners when we use our simulated data as our input. We have seen that using boundary filters help predict the value at the boundary once but it is not flexible such that we clearly see an error propagation. The WNO with signal extension has more parameters to adjust the model around the edges

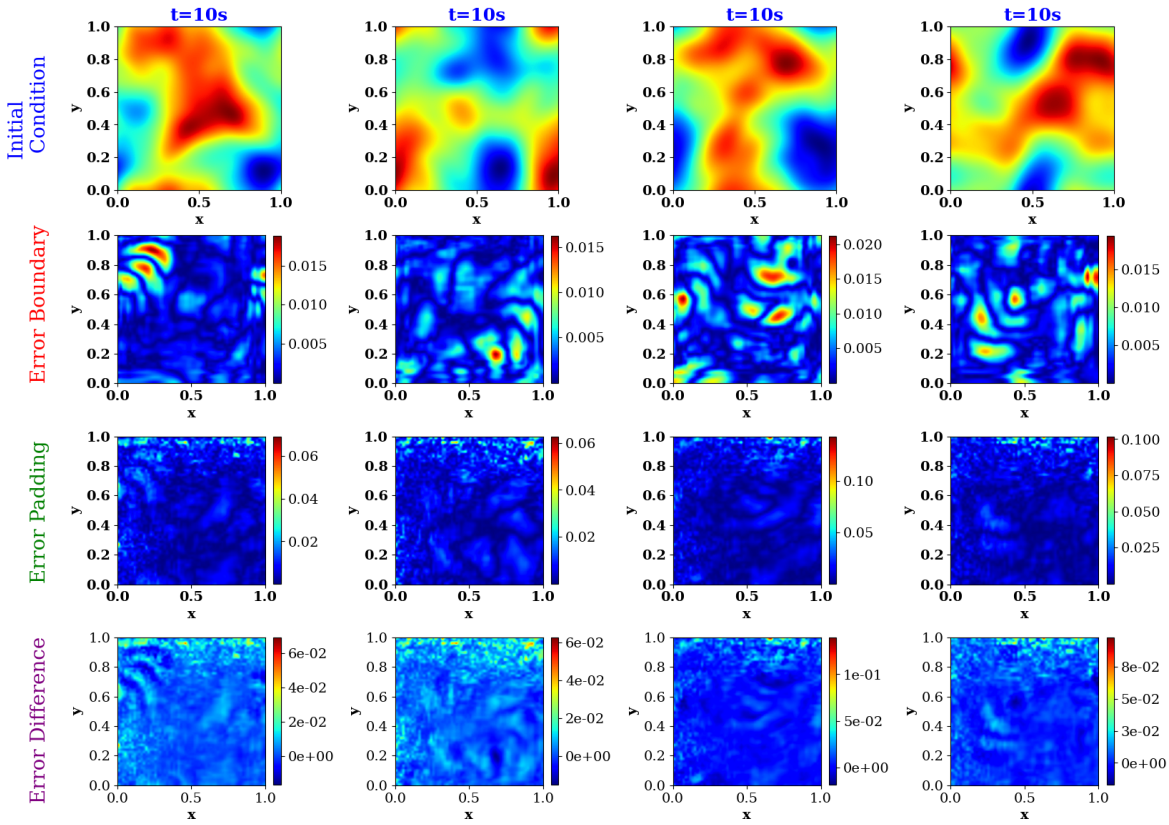


Figure 6.6: Navier-Stokes equation with a spatial resolution of 64×64. We train the model with the vorticity data for $t < 10$ and predict the vorticity field at $t = 10$. In the first row we see the initial condition of the vorticity field, the second row depicts the error of the WNO with boundary filter, the third row shows the error of the WNO with symmetric signal extension and the forth row shows the difference between the error of the WNO with signal extension and the WNO with boundary filter where red and yellow values indicate that the boundary filters performed better. Each column represents a sample of the test data set.

and we can hypothesize that this can help the model avoid error propagation.

The results in Figure 6.8 show that the error propagation is more randomly distributed if we use signal extension and does not focus on corners or edges. On the other hand we see in Figure 6.7 that while we use boundary filters, the error around the edges becomes worse at later time steps.

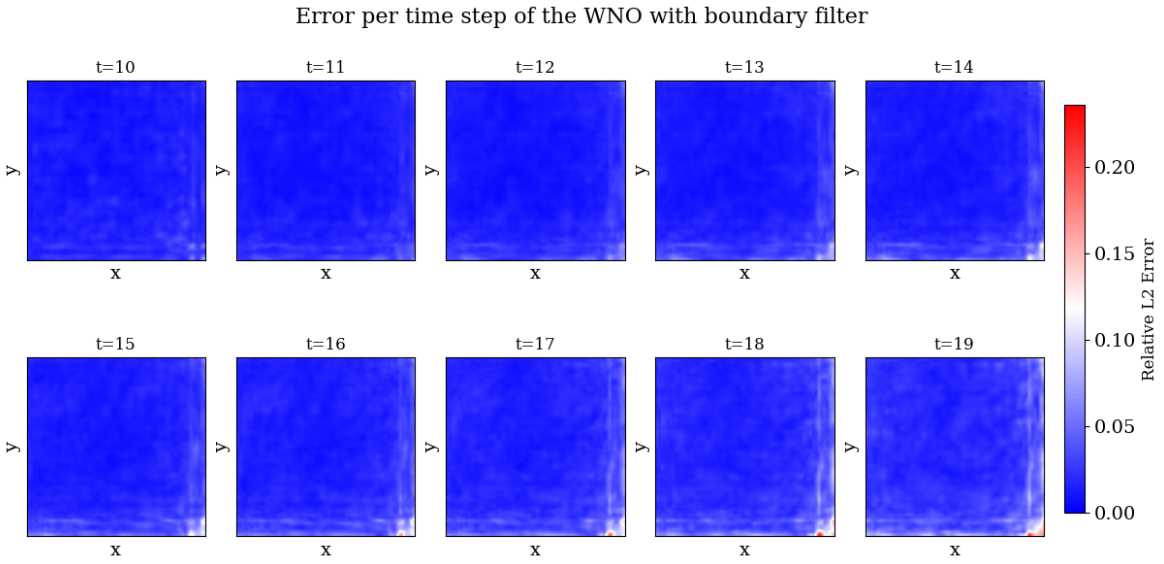


Figure 6.7: Navier-Stokes equation with a spatial resolution of 64×64. We train the model with the vorticity data for $t < 10$ and predict the vorticity field at $t = 10, ..., 19$. We can see here the mean error of the WNO with boundary filter over all test data samples for each time step.

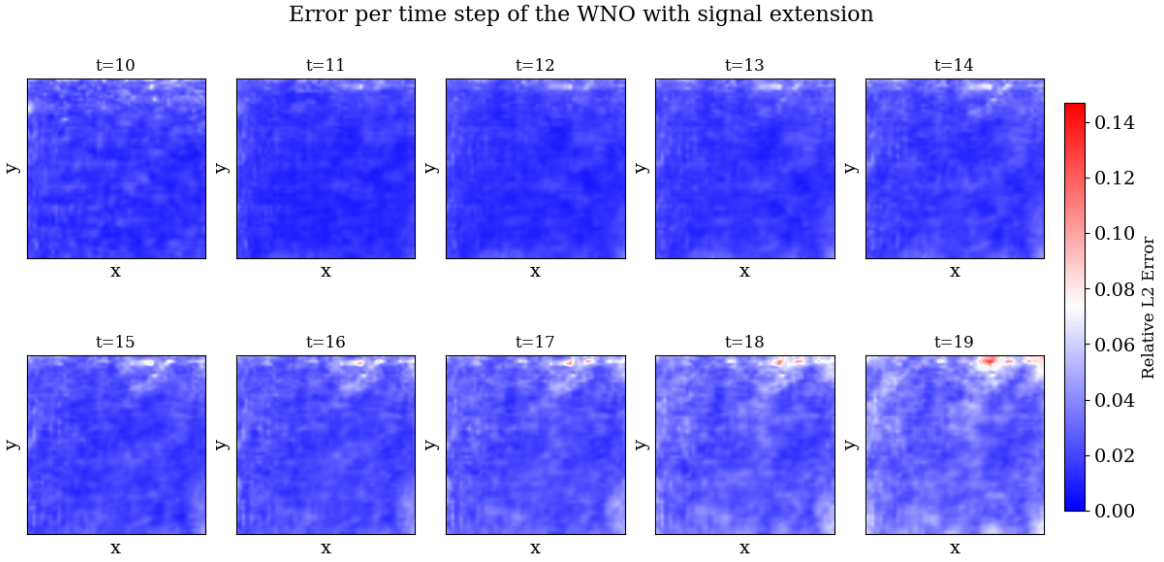


Figure 6.8: Navier-Stokes equation with a spatial resolution of 64×64. We train the model with the vorticity data for $t < 10$ and predict the vorticity field at $t = 10, ..., 19$. We can see here the mean error of the WNO with signal extension over all test data samples for each time step.

6.4 Convolution in Fourier Domain

The authors of [1] have provided another version of the WNO, which can be found as version 3.0.0 in [2], where they changed the way the weights are applied to the wavelet coefficients in the wavelet layer. Instead of applying them directly to the wavelet coefficients, they apply the Fourier transform to the wavelet coefficients and then learn the weights on the Fourier transformed wavelet coefficients.
For a more formal understanding of this change we look at the update operator of the neural operator which was introduced in Section 5.2

$$v_{j+1} := g((K(a;\phi) * v_j)(x) + Cv_j(x)).$$

Now we take a deeper look at the integral operator K. In the first version of the WNO the kernel consists of the discrete wavelet transform $\mathcal{W}$, the convolution with the weights and the inverse discrete wavelet transform $\mathcal{W}^{-1}$. Now we first apply the wavelet transform W, then the Fourier transform $\mathcal{F}$, apply the weights in the Fourier space and then apply the inverse transforms $\mathcal{F}^{-1}$ and $\mathcal{W}^{-1}$. The full algorithm is then presented in Algorithm 4.

Algorithm 4 Algorithm of the WNO updated Version

Input: N-samples of the pair $\{a(x) \in \mathbb{R}^{n_D \times d_a}, u(x) \in \mathbb{R}^{n_D \times d_u}\}$, coordinates $x \in D$, and network hyperparameters θ_{NN}.

1: Stack the inputs: $\{a(x), x\} \in \mathbb{R}^{n_D \times 2d_a}$.
2: **for** epoch $= 1, \ldots,$ epochs **do**
3: Uplift the input using transformation $P(\cdot)$: $v_0(x) = P(\{a(x), x\})$.
4: **for** $j = 1, \ldots, l$ **do**
5: Decompose the input using wavelet transform: $\mathcal{W}(v_j(x))$.
6: Apply discrete Fourier transform to the wavelet coefficients: $\mathcal{F}(\mathcal{W}(v_j(x)))$.
7: Parameterize the NN kernel k_ϕ in the Fourier domain: $R_\phi * \mathcal{F}(\mathcal{W}(v_j(x)))$.
8: Reconstruct the convolved input: $v^1_{j+1}(x) = \mathcal{W}^{-1}(\mathcal{F}^{-1}(R_\phi * \mathcal{F}(\mathcal{W}(v_j(x)))))$.
9: Perform the linear transform: $v^2_{j+1}(x) = Cv_j(x)$ using a CNN C.
10: Add the outputs of steps 8 and 9: $\tilde{v}_{j+1}(x) = (v^1_{j+1} + v^2_{j+1})(x)$.
11: **if** $j \neq l$ **then**
12: Apply the activation to complete the iteration: $v_{j+1} = g(\tilde{v}_{j+1}(x))$.
13: **end if**
14: **if** $j = l$ **then**
15: $v_{j+1} = \tilde{v}_{j+1}(x)$
16: **end if**
17: **end for**
18: Compute the final output: $\hat{u}(x) \in \mathbb{R}^{n_D \times d_u} = Q(v_l(x))$ where $Q(\cdot) : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$ is an FNN.
19: Compute the loss: $L(u, \hat{u})$.
20: Compute the gradient of the loss: $\frac{\partial L(u,\hat{u})}{\partial \theta_{\text{NN}}}$.
21: Update the parameters of the network using the gradient.
22: **end for**

Output: Predicted solution $\hat{u} \in U$, parameters of NN θ_{NN}.

Since this modification in the network architecture can have a significant influence on the model performance for different parameters, we perform a parameter tuning. Similar to the

Wavelet	Level	Test Error
db2	1	0.09
db4	1	0.118
db6	1	0.136
db2	3	0.19
db4	3	0.29
db6	3	0.52
db2	5	0.906
db4	5	0.9825
db6	5	3.9

Table 6.4: Test errors by wavelet type and decomposition level

Fourier Neural Operator we evaluate the Wavelet Neural Operator on the one-dimensional Burgers equation. Since their are no published numerical results of this WNO version, we need to make experiments ourselves. For these experiments, we use the WNO variant where convolution is applied in the Fourier domain after wavelet decomposition.
The models are trained on the same problem and the same data as the FNO. We train the model for 500 epochs with an initial learning rate of 10^{-3} which is halved every 50 epochs. The batch size is 20 and the weight decay is 10^{-6}.
We first evaluate the model across different wavelet types and levels of wavelet decomposition while keeping a constant width of 40 and a subsampling factor $r = 2^3$.
The results indicate that the model performs best with low-level decompositions and wavelets with shorter filters, such as db2. This is supported by the fact that with the db2 wavelet and a level 1 decomposition, the model yields the best results.
This is the combination of parameters that produces the most detailed representation of the input data with the most coefficients of the input data. This suggests that retaining more wavelet coefficients at finer scales is beneficial, possibly because they capture high-frequency details that are preserved through the subsequent Fourier transform. The Fourier transform now replaces the higher levels of the wavelet decomposition and catches the low-frequency patterns in the data.
This implies that in this version of the WNO, we need to use different parameters than in the original version from [1] which was trained with a level 6 decomposition and db6 wavelets.
Tests with different parameters showed that increasing the resolution did not improve the model performance significantly. Using the WNO model with db2 wavelets and a width of 40, we test different spatial resolutions with results shown in Table 6.5.

Resolution	Test Error in %
1024	0.09
2048	0.087
4096	0.085

Table 6.5: Test errors for different resolutions

Although increasing the resolution yields small improvements on the order of 10^{-5}, these are negligible relative to the overall error magnitude.
Increasing the width improves the model accuracy more significantly. Up to this point the models had a width of 40 and an error of 0.09% and increasing the width to 64 and 128 decreases the error to 0.076 and 0.07 respectively. So the changes are in the order of 10^{-4} but

comes with the downside that it increases the number of model parameters and with that the storage and time efficiency of training and saving the model. The model with width 40 has about 3.3 million parameters while the model with width 64 has approximately 8.5 million parameters and the model with width 128 has about 33.9 million parameters.
It is remarkable that the accuracy improved significantly in comparison to the original WNO which had a L_2 test error of 1.75%.

6.5 Noise-Augmented Training Data

A common issue in machine learning is that the model performs better on the training data than on the test data ([11]). To decrease the gap between the training and the test error we try to avoid overfitting. For this reason we take a look at Section 5.1 and change the training data to

$$\{a_i \in \mathbb{R}^{n_D \times d_a}, u_i + \epsilon \in \mathbb{R}^{n_D \times d_u}\}_{i=1}^N$$

where ϵ is randomly drawn from the normal distribution $\mathcal{N}(0, \sigma(u)^2)$ where $\sigma(u)$ is the standard deviation of $\{u_i\}_{i=1}^N$. To avoid overfitting to the new noise-augmented data, we redraw ϵ after each epoch during the training process. The formal algorithm of the WNO with noisy data is shown in Algorithm 5.

Algorithm 5 Algorithm of the WNO

Input: N-samples of the pair $\{a(x) \in \mathbb{R}^{n_D \times d_a}, u(x) \in \mathbb{R}^{n_D \times d_u}\}$, coordinates $x \in D$, and network hyperparameters θ_{NN}.

1: Stack the inputs: $\{a(x), x\} \in \mathbb{R}^{n_D \times 2d_a}$.
2: **for** epoch = 1, ..., epochs **do**
3: Uplift the input using transformation $P(\cdot)$: $v_0(x) = P(\{a(x), x\})$.
4: **for** $j = 1, \ldots, l$ **do**
5: Decompose the input using wavelet transform: $\mathcal{W}(v_j(x))$.
6: Parameterize the NN kernel k_ϕ in the wavelet domain: $R_\phi * \mathcal{W}(v_j(x))$.
7: Reconstruct the convolved input: $v_{j+1}^1(x) = \mathcal{W}^{-1}(R_\phi * \mathcal{W}(v_j(x)))$.
8: Perform the linear transform: $v_{j+1}^2(x) = Cv_j(x)$ using a CNN C.
9: Add the outputs of steps 7 and 8: $\tilde{v}_{j+1}(x) = (v_{j+1}^1 + v_{j+1}^2)(x)$.
10: **if** $j \neq l$ **then**
11: Apply the activation to complete the iteration: $v_{j+1} = g(\tilde{v}_{j+1}(x))$.
12: **end if**
13: **end for**
14: Compute the final output: $\hat{u}(x) \in \mathbb{R}^{n_D \times d_u} = Q(v_l(x))$ where $Q(\cdot) : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$ is an FNN.
15: Draw $\epsilon \sim \mathcal{N}(0, \sigma(u)^2)$
16: Compute the loss: $L(u + \epsilon, \hat{u})$.
17: Compute the gradient of the loss: $\frac{\partial L(u+\epsilon, \hat{u})}{\partial \theta_{\text{NN}}}$.
18: Update the parameters of the network using the gradient.
19: **end for**

Output: Predicted solution $\hat{u} \in U$, parameters of NN θ_{NN}.

In Figure 6.9 we see the train and test error of the WNO during a training process with the original data of the 2D Darcy flow problem. We can see that the model performs better

on the training data, which is usual since the model already knows the training data, but have never seen the test data. The question is whether we can improve the model performance with the test data by adding noise to the training data. As we increase the standard deviation of the noise ϵ it becomes harder for the model to perform on the training data. That means the training error will increase but we avoid overfitting and can improve on the test error. If we choose a too large standard deviation, we may obtain underfitting which means that the model is not able anymore to learn the problem sufficiently well. A parameter fitting for the standard deviation of ϵ yields the result that the standard deviation of the training data u is the best choice. With this setting we train the model on the noise-augmented data and get the result Figure 6.10. We see that the training error is worse than in Figure 6.9 but the test error improves, leading to a better model accuracy. This shows that adding noise to the data can improve the overall model performance.

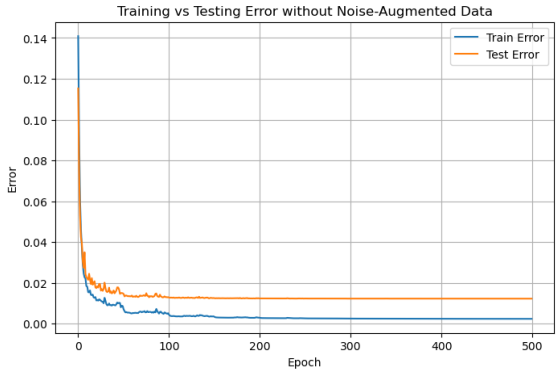


Figure 6.9: Train and test error of the WNO during training for the 2D Darcy flow equation process with the original data.

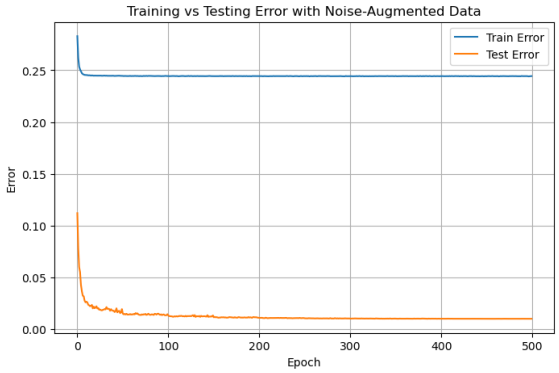


Figure 6.10: Train and test error of the WNO during training process for the 2D Darcy flow equation with the noise-augmented data.

6.6 Combinations

We have introduced several changes to the FNO network architecture which either tackled special issues of the FNO or tried to improve the overall model accuracy. We added a bias, changed the integral transformation to the wavelet transform and got to the WNO, changed the way the WNO handled signal boundaries, analyzed two different ways the WNO applied the weights of the neural network, and used noise augmented training data. Each of these adaptations could improve the model performance in some scenarios but we only applied one improvement at a time. Now we try to combine the different changes to the model to get the best model possible with the given tools. We start by testing the created models on the single step Navier-Stokes equation because this problem has non-zero boundary condition which allows us to test the model's capabilities to predict the PDE outcome on the edges.
First we try to combine the original WNO from Section 6.2 and [1] and add a bias like in Section 6.1. Analogous to Figure 6.1 we get the wavelet layer Figure 6.11.

The modified wavelet layer trained on the Navier- Stokes equation has a relative L_2 error of 0.046% which is a significant improvement over the test error of 0.071% from the original WNO (Figure 6.5), but it is still worse than the WNO with boundary filters which has a test error of 0.027%. We see in Figure 6.12 and Figure 6.5 that the improvements with the boundary filters and the bias terms were in similar regions and the model performed better at the edges and especially the top corner. If we now try the WNO with boundary filters and

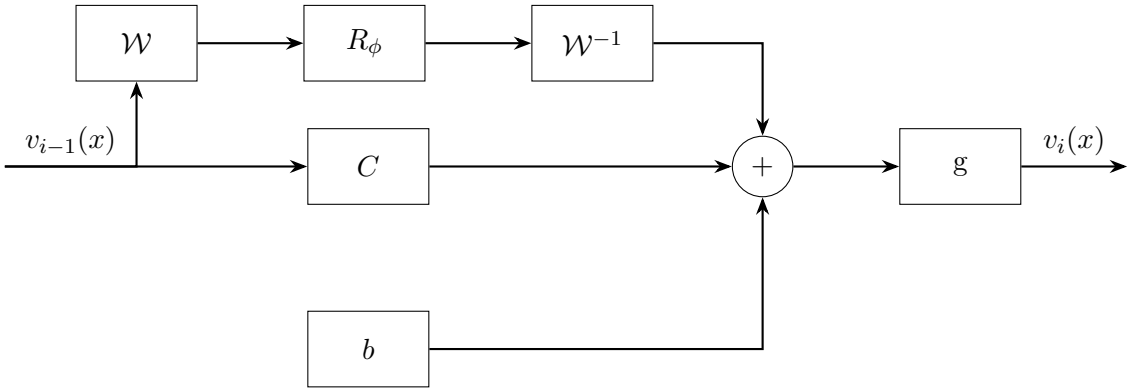


Figure 6.11: Wavelet layer with bias

the bias terms, we can further improve the model accuracy and get a test error of 0.025%, but the improvement is small in comparison to the WNO with just boundary filters. A possible explanation for that is that both changes could improve the WNO in the same spatial regions such that the strengths of both changes cannot add up.
We also see in Figure 6.13 that the difference between the test and training error is very small which is a strong indication that the model does not overfit in this situation. This suggests that there is no basis for using noise-augmented training data.
If we try to use it anyway, the accuracy of the model decreases. Adding the random noise like in Section 6.5 to the model leads to an increased test error of 0.074%. Even if we use a small random $\epsilon \sim \mathcal{N}(0, 0.05\,\sigma(u)^2)$ the model accuracy drops to 0.033%.

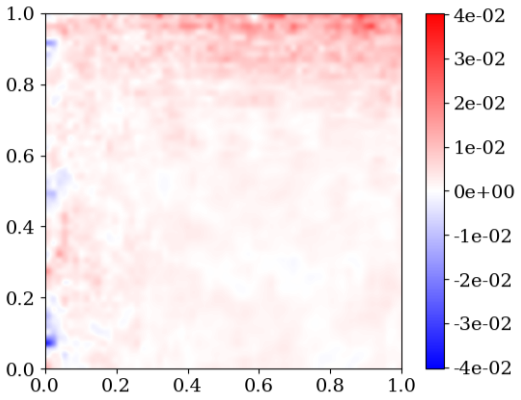


Figure 6.12: Difference of the mean test error for the single-step Navier-Stokes equation for the WNO without bias term and the WNO with bias term. In red areas the WNO with bias performed better than the original WNO.

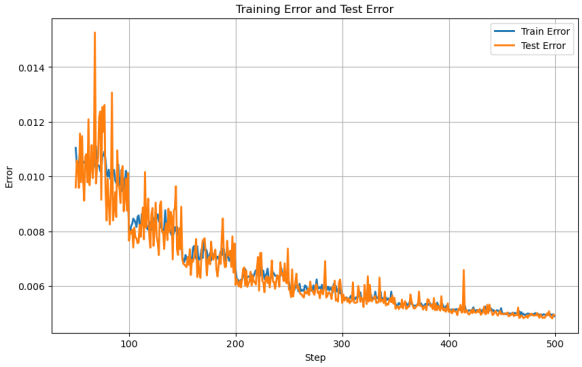


Figure 6.13: Training and test error of the WNO with boundary filters and bias term in the context of the single-step Navier-Stokes equation.

If we test the WNO with the convolution in the Fourier domain (Section 6.4), we get a relative L_2 error of 0.007% which is more than three times better than the best model up to this point. If we add a bias term the error gets even smaller to 0.005%. The gap between the test and training error is again too small to suspect overfitting and experiments with noise-augmented training data could not improve the model.

Convolution in	**Boundary Treatment**	**Bias**	**Noise-Augmentation**	**Test L_2 Error in %**
Spatial Domain	Signal Extension	No	No	0.071
Spatial Domain	Signal Extension	Yes	No	0.046
Fourier Domain	Boundary Filters	No	No	0.027
Fourier Domain	Boundary Filters	Yes	No	0.025
Fourier Domain	Boundary Filters	Yes	Yes	0.033

Table 6.6: L_2 Test errors of variations of the WNO for the single-step Navier-Stokes equation

Combining the convolution in the Fourier space with the use of boundary filters in the discrete wavelet transform yields a test error of 0.006% which is better than the model performance with signal extension. Adding the bias terms improves the test error to 0.004%. As in the previous scenario, adding noise to the training data could not improve the accuracy. An overview of the model performance on the Navier-Stokes problem for one time step is presented in Table 6.6 and we see the both boundary filters and the bias terms improve the model while noise augmented training data could not further increase the accuracy in this setting.
This means that the best possible combination of network variations of the FNO we could achieve is using the discrete wavelet transform instead of the Fourier transform and apply the NN weights to the Fourier transformed wavelet coefficients. We also add a bias term to the update operator and in this way we achieve the best model performance. The complete model is described in Algorithm 6 and we will refer to this model as the modified wavelet neural operator (MWNO).

We now apply this model for other problems to check if the improvements also carry over to the other problems. For this we train the model on the 1-D Burgers equation, the 2-D Darcy flow equation and the 2-D Navier-Stokes equation.
We see in Table 6.7 that the MWNO outperforms the other two models significantly for the Burgers equation, but does not reach the performance of the WNO for the Darcy Flow equation. If we simulate the Navier-Stokes equation for only one time step, we see that the MWNO improves upon the WNO but the FNO outperforms both of them, which is remarkable since the WNO is superior to the FNO if we simulate for several time steps.
For the Darcy Flow equation we can use noise-augmented training data and further improve the accuracy to get a test error of 0.94%, which is nearly as good as the performance of the WNO.

Problem	**WNO**	**FNO**	**MWNO**
Burgers Equation	1.75%	1.60%	0.06%
Darcy Flow	0.84%	1.08%	1.14%
Single Step Navier-Stokes Equation	0.071%	0.002%	0.025%
Multi Step Navier-Stokes Equation	0.31%	1.28%	0.002%

Table 6.7: Comparison of MWNO L_2 test errors with WNO and FNO literature results from [1] and own numerical results in the case of the single-step Navier-Stokes equation

Algorithm 6 Algorithm of the MWNO

Stack the inputs: $\{a(x), x\} \in \mathbb{R}^{n_D \times 2d_a}$.
for epoch = 1, ..., epochs **do**
    Uplift the input using transformation $P(\cdot)$: $v_0(x) = P(\{a(x), x\})$.
    for $j = 1, \ldots, l$ **do**
        Decompose the input using wavelet transform: $W(v_j(x))$.
        Apply discrete Fourier transform to the wavelet coefficients: $\mathcal{F}(W(v_j(x)))$.
        Parameterize the NN kernel k_ϕ in the wavelet domain: $R_\phi * \mathcal{F}(W(v_j(x)))$.
        Reconstruct the convolved input: $v^1_{j+1}(x) = W^{-1}(\mathcal{F}^{-1}(R_\phi * \mathcal{F}(W(v_j(x)))))$.
        Perform the linear transform: $v^2_{j+1}(x) = Cv_j(x)$ using a CNN C.
        Calculate the bias term: $v^3_{j+1}(x) = B(v_j(x))$
        Add the outputs of steps 8 and 9: $\tilde{v}_{j+1}(x) = (v^1_{j+1} + v^2_{j+1} + v^3_{j+1})(x)$.
        if $j \neq l$ **then**
            Apply the activation to complete the iteration: $v_{j+1} = g(\tilde{v}_{j+1}(x))$.
        end if
        if $j = l$ **then**
            $v_{j+1} = \tilde{v}_{j+1}(x)$
        end if
    end for
    Compute the final output: $\hat{u}(x) \in \mathbb{R}^{n_D \times d_u} = Q(v_l(x))$ where $Q(\cdot) : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$ is an FNN.
    Compute the loss: $L(u, \hat{u})$.
    Compute the gradient of the loss: $\frac{\partial L(u,\hat{u})}{\partial \theta_{\text{NN}}}$.
    Update the parameters of the network using the gradient.
end for

Output: Predicted solution $\hat{u} \in U$, parameters of NN θ_{NN}.

Chapter 7

Conclusion and Discussion

In this thesis, we began by revisiting the well-established Fourier transform as a foundational tool for analyzing signals and systems. To address some of its limitations, especially the lack of different basis for the function space during the transform and the lack of spatial resolution, we introduced the wavelet transform as an alternative that provides a more flexible representation, particularly for problems with localized or multi-scale features.
Based on the concept of the Fourier transform, we introduced the Fourier neural operator. We showed that the FNO is able to predict the solution of parametric PDEs up to a certain level and we have seen its inability to detect and model sharp edges and discontinuities. To tackle these issues and improve the overall performance of the FNO the main contribution of this thesis is presenting the wavelet neural operator and several changes to its network architecture.
In more detail we added a bias term to the network architecture which enabled us to detect sharp edges and discontinuities in the data sets.
Afterwards we changed the way how we treated the signal edges during the discrete wavelet transform in the WNO. Instead of expanding the signal, we used boundary filter. This allowed us to avoid artificial signals and rely exclusively on data, that reflects the mathematical-physical reality of the corresponding equations.
Then we looked at a different way to apply the weights of the NN to the wavelet coefficients, which was introduced by the authors of [1] in an update of their code [2]. Here we Fourier transformed the wavelet coefficients and then applied the NN weights. That led to a change of our network parameters, but it also improved the model performance significantly depending on the equation we worked on.
In some scenarios we could observe a big difference between the training and the test error during the model training process which is an indicator for overfitting. In these cases, adding random noise to the training data could improve the model performance in terms of the test error.
Lastly we tried to combine the changes to the network architecture and could improve the model performance for two benchmark problems and we could observe that different models performed best, depending on the problem.
One insight we found in this thesis is that we have no model that is better than the rest for all problems. The MWNO outperformed the FNO and WNO on the Burgers equation and Navier-Stokes equation while the WNO was the best on the Darcy flow equation and the FNO hat the best result with the single step Navier-Stokes equation. Also noise augmented training data was only helpful in the context of the Darcy flow problem, but could not improve the model accuracy for the other equations. This clearly indicates that we have not yet

found a model architecture that is the best fit for all problems. We also do not know yet for which classification of PDEs Fourier or wavelet based models suit better or how the model performances depend on the distribution of the training data sets.
Furthermore, we did only try to improve the FNO and WNO in this thesis but we did not try to construct a completely new network architecture. While the FNO performed better than most other models for most problems (Section 5.3, Section 5.3), we do not know if there is a better way to construct a model which is able to perform across a wide range of benchmark problems.
Lastly, the reliance on simulated data instead of analytical solutions represents a limitation, especially if we have a problem formulation where error propagation could play a role. Future work could aim to validate these models on real-world data sets or PDEs with analytical solutions for many initial conditions to make sure that they do not suffer from numerical errors in the training data.
In summary, this thesis contributes with a summary of the theoretical methods for the Fourier and wavelet transform, the network architecture of the FNO and WNO and ways to improve the model performances with network architecture modifications. While our improvements demonstrate the possibilities to further improve the models while maintaining the basic concepts of the FNO, the broader landscape of operator learning remains rich with opportunities for further exploration of different machine learning models for advancements in PDE solving.

List of Figures

Bibliography

[1] Tripura, T. and Chakraborty, S.: *Wavelet Neural Operator for solving parametric partial differential equations in computational mechanics problems*. Computer Methods in Applied Mechanics and Engineering, Band 404, Seite 115783, 2023. Elsevier.

[2] Tripura, T. and Chakraborty, S.:Code from *Wavelet Neural Operator for solving parametric partial differential equations in computational mechanics problems*. Computer Methods in Applied Mechanics and Engineering, Band 404, Seite 115783, 2023. Elsevier. URL: https://github.com/csccm-iitd/WNO

[3] Blanke, F.: *Wavelets for diffusion models*, Masters Thesis, Institut für Numerische Simulation, Universität Bonn, 2024.

[4] Blanke, F.: *Randbehandlung bei Wavelets für Faltungsnetzwerke*, Bachelor Thesis, Institut für Numerische Simulation, Universität Bonn2021

[5] Blanke, F.: Python source code of *Wavelets for diffusion models*, Masters Thesis, Institut für Numerische Simulation, Universität Bonn, 2024. URL: https://github.com/v0lta/PyTorch-Wavelet-Toolbox

[6] Strang, G. and Nguyen T.: *Wavelets and Filter Banks*, Wessesley-Cambridge Press, 1996

[7] Z.Li, N.Kovachki, K.Azizzadenesheli, B.Liu, A.Bhattacharya, A.Stuart, and A.Anandkumar: *Fourier neural operator for parametric partial differential equations*, *arXiv preprint arXiv:2010.08895*, 2020.

[8] Chauhan, P., Choutri, S. E., Ghattassi, M., Masmoudi, N., Jabari, S. E.: *Neural operators struggle to learn complex PDEs in pedestrian mobility: Hughes model case study*, *arXiv preprint arXiv:2504.18267v1*, 2025.

[9] Brodine, T. R., Hokaj, I. M. : *Learning Euler Equation Discontinuities with a Fourier Neural Operator*, CS230: Deep Learning, Stanford University, CA, Fall 2022

[10] Robert P.: *Numerische Mathematik kompakt - Grundlagenwissen für Studium und Praxis*, 4., aktualisierte Auflage, Vieweg+Teubner, Wiesbaden, 2010.

[11] Ian Goodfellow, I., Bengio, Y., Courville, A.: *Deep Learning*, MIT Press, 2016. URL: http://www.deeplearningbook.org

[12] Daubechies, I.: *Orthonormal Bases of Compactly Supported Wavelets*, Communications on Pure and Applied Mathematics, Vol. XLI 909-996, Copyright John Wiley & Sons, Inc. 1988

[13] Mallat, S.: *A Wavelet Tour of Signal Processing*, Second Edition, Academic Press, 1999

[14] Lu,L., Jin, P., Karniadakis, G.E.: *Deeponet: Learning nonlinear operators for identifying differential equations based on the approximation theorem of operators*, 2019, arXiv preprint arXiv:1910.03193.

[15] Lu,L., Meng, X., Cai, M., Mao, Z., Goswami, S., Zhang, Z., Karniadakis, G.E.: *A comprehensive and fair comparisonof two neural operators (with practical extensions) based on fair data*, Comput. Methods Appl. Mech. Engrg. 393, 2022

[16] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A., *Neural operator: Graph kernel network for partial differential equations*, 2020, arXiv preprint arXiv:2003.03485.

[17] Königsberger, K.: *Analysis 2*, fünfte korrigierte Auflage, Springer-Verlag, Berlin Heidelberg, 2004.

[18] Daubechies, I.: *Ten Lectures on Wavelets*, SIAM, Philadelphia PA, 1991.

[19] Oppenheim, A.V., Shafer, R.W.: *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[20] Goatin, P., Mimault, M.: *The wave-front tracking algorithm for hughes' model of pedestrian motion*, SIAM Journal on Scientific Computing, 35 (3) (2013), pp. B606-B622

[21] Godunov, S.K., Bohachevsky, I.: *Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics* Matematičeskij Sbornik, 47 (3) (1959), pp. 271-306

[22] DeVore, R.A.: *Chapter 3: The Theoretical Foundation of Reduced Basis Methods*, SIAM, 2014. DOI: 10.1137/1.9781611974829.ch3. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611974829.ch3

[23] Zhu, Y., Zabaras, N.: *Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification.* Journal of Computational Physics, 2018. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0021999118302341

[24] Bhattacharya, K., Kovachki, N.B., Stuart, A.M.: *Model reduction and neural networks for parametric PDE(s)*, Preprint, 2020.

[25] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Multipole graph neural operator for parametric partial differential equations, 2020a.

[26] He, K., Zhang, X., Ren, S., Sun, J.: *Deep residual learning for image recognition.*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016.

[27] Ronneberger, O., Fischer, P., Brox, T.: *U-Net: Convolutional networks for biomedical image segmentation.*, International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), pp. 234–241. Springer, 2015.

[28] Wang, R., Kashinath, K., Mustafa, M., Albert, A., Yu, R.: *Towards physics-informed deep learning for turbulent flow prediction*, Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), pp. 1457–1466, 2020.

[29] Chen, T., Chen, H. *Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems*, IEEE Transanctions on Neural Networks Volume 6 pp. 911-917, 1995.

[30] Raissi, M., *Deep hidden physics model: Deep learning of nonlinear partial differential equations*, Journal of Machine Learning Research, Volume 19, 2018

[31] Rudy, S.H., Brunton, S.L., Proctor, J.L., Kutz, J.N.: *Data-driven discovery of partial differential equations*, Science Advences, Volume 3, 2017.

[32] Raissi, M., Perdikaris, P., Karniadakis, G.E.: *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics Volume 378 pp. 686-707, 2019.