ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Fourier Analysis of Activations in Neural Networks

Bachelor Thesis

Felipa Schwarz

Tuesday 19th October, 2021

Advisors: Prof. Dr. Markus Püschel, Chris Wendler
Department of Computer Science, ETH Zürich

Abstract

Within the last decade, deep learning methods have replaced the state-of-the-art in various domains such as computer vision and natural language processing. Nevertheless, the interpretability of deep neural activations has remained an open case of research. In this work, we analyze the activations of deep neural networks using causal signal processing (CSP). CSP is a novel signal processing theory for data indexed by the vertices of a directed acyclic graph like the computational graph of a neural network. In particular, we collect activation patterns generated by convolutional neural networks (CNNs) trained on image recognition, then, apply visualization and clustering techniques on the activations patterns and their respective Fourier coefficients in CSP to observe the behavior of neural network signals in the causal Fourier-domain. Finally, we give interpretations and provide formal reasoning for our results.

Contents

Chapter 1

Introduction

In recent years neural networks have attracted tremendous interest in research [26]. They have proven themselves by solving machine learning tasks and not only outperforming, but reaching far beyond human capabilities in numerous disciplines [4]. Larger data sets, faster computations, and the growing complexity of the systems allowed them to accelerate their performance and take artificial intelligence to the next level [26]. What was left behind is our understanding of them [26]. Intermediate computations generate thousands, even millions of activations yet their meaning surpass our scope of comprehension. However, for many applications like self-driving cars [1], healthcare [2], cyber security [1], or solely the ethics of AI [8], the interpretability of neural networks is indispensable. Extensive research goes on the quest for a deeper understanding of the self-learning black box [26].

In this thesis, we are going to analyze the activations of neural networks using recent advances in signal processing (SP). SP is an integral component of data science and provides complex data processing techniques [7]. Core SP concepts like translation, invariant filters and associated Fourier transform serve as a basis for extracting, transforming, and analyzing data [7].

In essence, a neural network is a directed acyclic computational graph and can be modeled as a layered mesh of consecutive dependencies. Feeding an input in $\mathbb{R}^{\alpha}$ into a neural network with β neurons yields an activation pattern in $\mathbb{R}^{\alpha+\beta}$, which can be considered as a signal. The substantial hurdle in investigating such signals by applying classical SP techniques for discrete-time signals is that they neglect the causal dependencies embedded in the computational graph of a neural network [20]. While some central properties of classical SP such as linearity smoothly transfer to this irregular domain, other fundamental concepts like shifts in time raise questions. In classical discrete time signal processing, we can shift a signal by a number of steps on the time axes. Yet, how does one even interpret a shift on irregular domains such as graphs?

Recent works on the theory of SP broaden their foundation by porting classical SP concepts to data indexed by partially ordered sets (posets) [3]. The latter having the suitable property of capturing the causality arising from the underlying structure of a neural network.

In this work, we entered uncharted territory by applying causal signal processing (CSP) on activations in neural networks.

1. We trained convolutional neural networks with up to 2^{15} neurons on image recognition using PyTorch [13], a Python machine learning library.
2. We implemented the extraction of the directed graph embedded in a Neural Network and the components of the CSP framework.
3. We collected and transformed samples of activations.
4. We performed an exploratory analysis of data.

This manuscript is organized into four chapters. In Chapter 2 we set the necessary foundations and notation. We introduce causal signal processing theory and the basics of neural networks. In Chapter 3 we draw the relationship between CSP and neural networks and formulate our approach. In Chapter 4 we outline our implementation including its computational complexities. In Chapter 5 we report a collection of our results and insights. We visualize and investigate the Fourier transform, activation patterns, and their Fourier coefficients. Later, we cluster the data and measure its performance against spectral graph signal processing [20]. Finally, we provide a numerical analysis of our computations.

Chapter 2

Background and Notation

Symbol	Meaning
$\mathcal{V}$	set of vertices in the computational graph of a NN
$\mathcal{V}_i$	set of vertices in the ith layer of the computational graph of a NN
$\mathcal{E}$	set of directed edges between vertices $\mathcal{V}$
$\mathcal{E}_i$	set of incoming edges into vertices $\mathcal{V}_i$
ℓ	number of layers in a neural network
n	number of vertices $\lvert\mathcal{V}\rvert$
n_i	number of vertices in layer i $\lvert\mathcal{V}_i\rvert$
m	number of edges $\lvert\mathcal{E}\rvert$
m_i	number of incoming edges in layer i $\lvert\mathcal{E}_i\rvert$
$deg^-(x)$	number of incoming edges of vertex x
$deg^+(x)$	number of outgoing edges of vertex x
$A_{[i,j]}$	reference to entry $A_{i,j}$ of matrix A
x_v	reference to the component of vector x, associated with vertex v
$A_{u,v}$	reference to the entry of matrix A, where the row and column is associated with vertex u and v, respectively
η	fixed total order of elements in $\mathcal{V}$
η_v	index of vertex $v \in \mathcal{V}$ in a fixed total order η of elements in $\mathcal{V}$
$\mathbf{e}_i$	ith standard basis vector in $\mathbb{R}^n$
$x \vee y$	elementwise logical OR of indicator vectors x and y
$x \wedge y$	elementwise logical AND of indicator vectors x and y
$\iota_{x \leq y}$	characteristic function of $x \leq y$, $\iota_{x \leq y} = 1$ if $x \leq y$ and 0 otherwise
$(\iota_{x \leq y})_{y \in \mathcal{V}}$	vector of the element-wise characteristic function of $x \leq y$ for all $y \in \mathcal{V}$ in order η

In this chapter, we first introduce the definitions and concepts of causal signal processing (CSP). This theory originates from [17] and an up till now unpublished follow-up paper [3]. In [17] and [3] further concepts from classical signal processing are ported to data-domains modeling causal dependence

upon events.

For those unfamiliar with neural networks we explain the fundamental idea behind them and define common terminologies. This knowledge is necessary to understand their relation to causal signal processing.

We assume prior knowledge of basic linear algebra and graph theory. Familiarity with the basics of machine learning is beneficial.

2.1 Causal Signal Processing

We start by defining our data-domain, the poset-domain, and its notation. We then fix [3] notion of the Fourier-domain to define a Fourier transform and filtering operation analogous to classical signal processing.

2.1.1 Poset-Domain

In the following, we specify terminology that describes our data-domain, i.e. the structure of our data. But first, let us begin with a brief digression:

Causality: Causality studies the influence among elements. In particular, it is concerned with the relationship between cause and effect. In this regard causality intrinsically defines a partial order on elements, where one element x is considered smaller than another element y if x causes y. Formally, we represent causal elements by the vertices of a graph, in which an edge (x, y) encodes the causal relationship 'x is a direct cause of y' [3]. Reference [14] defines a causal structure as a directed acyclic graph.

Definition 2.1 (DAG) *A **directed acyclic graph (DAG)** is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \dots, v_n\}$ is the set of vertices and $\mathcal{E} = \{(v_i, v_j) |\ v_i, v_j \in \mathcal{V}\}$ is the set of directed edges without cycles, i.e. there is no path $\langle v_1, v_2, \dots, v_k \rangle$ s. t. $(v_i, v_{i+1}) \in \mathcal{E}$ and $v_1 = v_k$.*

Example 2.2 *Let $G = (V, E)$ be a graph with vertices $V = \{x_1, x_2, y_1, y_2, z_1\}$ and edges $E = \{(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2), (y_1, z_1), (y_2, z_1)\}$. G is directed and acyclic as can be seen in Figure 2.1a.*

Equivalently, we can represent the set of causal variables as a partially ordered set.

Definition 2.3 (Poset) *A **partial order** on a set $\mathcal{P}$ is a binary relation that is reflexive, antisymmetric, and transitive. The combination of a set $\mathcal{P}$ and a partial order relation $\preceq$, denoted $(\mathcal{P}, \preceq)$, is called a **partially ordered set (poset)**.*

Recall that a relation ρ on a set $\mathcal{A}$ is called

$$\begin{array}{lcll} \textbf{\textit{Reflexive}} & \iff & a \,\rho\, a & \forall\, a \in \mathcal{A} \\ \textbf{\textit{Antisymmetric}} & \iff & (a \,\rho\, b \wedge b \,\rho\, a) \implies a = b & \forall\, a, b \in \mathcal{A} \\ \textbf{\textit{Transitive}} & \iff & (a \,\rho\, b \wedge b \,\rho\, c) \implies a \,\rho\, c & \forall\, a, b, c \in \mathcal{A} \end{array}$$

We use $a \prec b$ to denote $(a \preceq b \wedge a \neq b)$.

In contrast to totally ordered sets, like a time-axis or the integers, not all elements of posets are necessarily comparable, i.e. there might exist $a, b \in \mathcal{P}$ such that $a \not\preceq b$ and $b \not\preceq a$.

Each poset is associated with a DAG, the so-called cover graph.

Definition 2.4 (Cover graph) *A **cover graph** of a poset $(\mathcal{P}, \preceq)$ is defined as the directed graph $\mathcal{C} = (\mathcal{P}, \mathcal{E})$, where $\mathcal{E} = \{(v_i, v_j) \mid v_i \text{ is covered by } v_j,\ v_i, v_j \in \mathcal{P}\}$. A vertex v_i is **covered** by v_j iff $v_i \prec v_j$ and there is no vertex v_k such that $v_i \prec v_k \prec v_j$.*

We go through the next examples to compare the different representations of causality that we just learned.

Example 2.5 *Let $(P, \leq)$ be a poset with $P = V$ and $v_x \leq v_y$ if $(v_x, v_y) \in E$ from example 2.2. Here, $(P, \leq)$ is the corresponding poset to DAG G. The directed acyclic graph G is also the cover graph of $(P, \leq)$.*

Example 2.6 *Let us define an extension of graph G by $G' = (V, E')$, where $E' = (x_1, z_1) \cup E$. G' is still a DAG as it contains no cycles. $(P, \leq)$, as defined in example 2.5, is the corresponding poset to both DAGs G and G'. However, G' is not a cover graph since $(x_1, z_1) \in E'$ but x_1 is not covered by z_1 because of y_1 or y_2. We say, G is the **transitive reduction** of G'.*

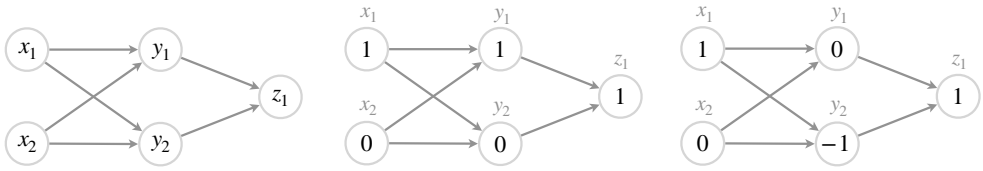
A poset uniquely defines a cover graph which is a DAG by definition. Conversely, multiple DAGs can yield the same poset. Specifically, all DAGs with the same transitive reduction yield the same poset. We obtain a transitively reduced DAG G by removing all edges (v_i, v_j) for which we have a vertex v_k such that $\langle v_i, \ldots, v_k \rangle$ and $\langle v_k, \ldots, v_j \rangle$ are valid paths in G. A transitively reduced DAG is a cover graph of a poset.

Definition 2.7 (Causal graph) *Let $\mathcal{P}$ be a set of causally related elements. The **causal graph** of $\mathcal{P}$ is the cover graph of the poset $(P, \preceq)$ with $x \preceq y$ iff x causes y.*

If we assign a value to every element in a causal structure, we obtain a signal.

Definition 2.8 (Signal) *A **signal** is defined as the data indexed by the elements of a finite poset $(\mathcal{P}, \preceq)$:*

$$s : \mathcal{P} \to \mathbb{R},\ p \mapsto s_p \tag{2.1}$$



(a) Directed acyclic graph (DAG) G

(b) Signal $s = (1, 0, 1, 0, 1)$ on G

(c) Fourier coefficients $\widehat{s} = (1, 0, 0, -1, 1)$ of s

Figure 2.1: A signal of a DAG assigns a value to each vertex. The Moebius function yields the Fourier coefficients of the signal.

We will use s as $s = (s_p)_{p \in \mathcal{P}} \in \mathbb{R}^{|\mathcal{P}|}$.

Example 2.9 $s = (s_{x_1}, s_{x_2}, s_{y_1}, s_{y_2}, s_{z_1}) = (1, 0, 1, 0, 1)$ *is a signal on DAG G. Signal s is illustrated in Figure 2.1b.*

For the order of values in s we fix one total order compatible with the partial order of the poset, e.g. a topological order of the cover graph.

Definition 2.10 (Total order) *We denote a fixed* ***total order*** *of poset elements by the bijective map*

$$\eta : \mathcal{P} \to \{1, \ldots, |\mathcal{P}|\},\ p \mapsto \eta_p \tag{2.2}$$

where

$$p \preceq q \implies \eta_p \leq \eta_q \tag{2.3}$$

is satisfied for all $p, q \in \mathcal{P}$.

So far we have specified our data-domain which we refer to as the poset-domain. We will now introduce [3] notion of the Fourier-domain.

2.1.2 Fourier-Domain

We start by defining our notion of the Fourier transform in CSP [3]. We justify it by showing its analogy to the classical signal processing Fourier transform.

Classical Fourier Transform

In classic one-dimensional discrete time signal processing the Fourier transform of a signal x is defined as

$$\widehat{x}_k = \sum_{n=1}^{N} x_n \cdot e^{-\frac{i2\pi kn}{N}}\ , \tag{2.4}$$

where i is the imaginary unit, N is the number of samples and $\widehat{x}_k$ denotes the k^{th} base frequency, i.e., Fourier coefficient. In other words, the Fourier transform of x is the linear expansion of complex exponentials, decomposing a signal x into its base frequencies $\widehat{x}$.

Frequencies Imagine we hear someone simultaneously playing three keys on the piano, two with more power and one with less. What we hear is the wave of the signal in the time-domain. The Fourier transform of this signal yields the base frequencies, which will have three peaks, two higher and one lower. These peaks correspond to the three notes. In this sense, the classical Fourier transform decomposes a piano sound into its constituent frequencies. In this particular example, the Fourier transform recovers 'pure information'.

The zeta function extends the discussion of causality from the previous chapter.

Definition 2.11 (Zeta function) *The **zeta function** encodes the order relation of two elements x and y.*

$$\zeta(x,y) = \begin{cases} 1 & \text{if } y \leq x \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

Translating this to our causal structure, the zeta function states whether there exists a directed path from vertex y to vertex x or whether y causes x.

Causes - events - effects Recall our intuition of causality. Every vertex x in our causal DAG represents an **event**. Therefore a DAG can be seen as a chain or rather a network of reactions. One event x causes another event y iff there is a path from x to y. The set of vertices that can reach x along some path are the **causes** of x. The set of vertices reachable from x are the **effects** of x. Let's consider the causal model proposed by [3] in which the signal value s_x is defined as the cumulative value of $x's$ causes. This dependency can be written as

$$s_x = \sum_{y \leq x,\ y \in \mathcal{P}} \widehat{s}_y = \sum_{y \in \mathcal{P}} \zeta(x,y)\ \widehat{s}_y\ , \tag{2.6}$$

where $\widehat{s}_y$ is the non-cumulative, unobserved strength of a cause y [3]. If an event has no causes it is considered a root cause. A root cause is not influenced by any other events and so is its value. Therefore, the unobserved strength $\widehat{s}_x$ of a root cause x is equal to its observed value s_x.

Conversely, we can reconstruct the strength $\widehat{s}_x$ of a cause x by using the inverse of the zeta function.

Definition 2.12 (Moebius inversion) *The **Moebius function** is the inverse of the zeta function and defined as*

$$\begin{aligned} \mu(x,x) &= 1\ , \\ \mu(x,y) &= -\sum_{y \leq z < x} \mu(z,y). \end{aligned}$$

We get

$$\widehat{s}_x = \sum_{y \in \mathcal{P}} \mu(x,y) s_y. \tag{2.7}$$

In other words, the Moebius function states in which way one needs to linearly combine the observed values s_y such that one obtains the unobserved values of strength $\widehat{s}_x$. Similar to the classical Fourier transform, the Moebius function recovers information from a superposition of elementary information [3]. Using the Moebius function we define the causal Fourier transform.

Definition 2.13 (Fourier transform) *The **Fourier transform** of a signal s in the poset-domain is defined as*

$$\widehat{s} = \mathcal{F} s \tag{2.8}$$

*with the **Fourier transform matrix** with entries*

$$\mathcal{F}_{x,y} = \mu(y,x)\ \iota_{y \leq x} \tag{2.9}$$

*for all $x, y \in \mathcal{P}$. We also refer to $\widehat{s}$ as **Fourier coefficients** or **spectrum** [3].*

Example 2.14 *The Fourier transform matrix of G is*

$$\mathcal{F} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 \\ 1 & 1 & -1 & -1 & 1 \end{bmatrix} \tag{2.10}$$

where we used the same topological order as in example 2.9.

Example 2.15 $\widehat{s} = (\widehat{s}_{x_1}, \widehat{s}_{x_2}, \widehat{s}_{y_1}, \widehat{s}_{y_2}, \widehat{s}_{z_1}) = (1, 0, 0, -1, 1)$ *are the Fourier coefficients of signal s from example 2.9. $\widehat{s}$ is illustrated in Figure 2.1c.*

Definition 2.16 (Inverse Fourier transform) *The corresponding **inverse Fourier transform** of Fourier coefficients $\widehat{s}$ is*

$$s = \mathcal{F}^{-1} \widehat{s} \tag{2.11}$$

*with the **inverse Fourier transform matrix** $\mathcal{F}^{-1}$, which is the inverse of $\mathcal{F}$ or explicitly*

$$\mathcal{F}^{-1}_{x,y} = \zeta(x,y) = \iota_{y \leq x} \tag{2.12}$$

for all $x, y \in \mathcal{P}$ [3].

Example 2.17 *The inverse Fourier transform matrix of G is*

$$\mathcal{F}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.13}$$

Note, that $\mathcal{F}^{-1}$ does not distinguish between DAGs G and G' from example 2.5 and 2.6, respectively. This because G and G' have the same causal graph. Their transitive reductions yield the same cover graph, and hence poset. In other words, $\mathcal{F}^{-1}$ encodes the **transitive closure** of a DAG. What should be taken from this observation is that a Fourier transform of a signal s on graphs with the same causal graph, yields the same Fourier coefficients $\widehat{s}$.

Causal Convolution

This section underlines how causal signal processing integrates the causality embedded in our poset-domain while elaborating its relation to the classical Fourier transform.

Definition 2.18 (Causal shift) *In our interpretation a **causal shift** by an arbitrary element $q \in \mathcal{P}$ shifts every signal s to sum common causes of the two elements q and y. This shift can be performed by matrix $\mathcal{T}_q \in \mathbb{N}^{n\times n}$ [3]:*

$$\mathcal{T}_q s = \mathcal{C}_q \widehat{s} \tag{2.14}$$

*where matrix $\mathcal{C}_q \in \{0,1\}^{n\times n}$ denotes the **common causes matrix** given by*

$$\mathcal{C}_{q(x,y)} = \zeta(x,y) \wedge \zeta(q,y) = \mathcal{F}^{-1}_{(x,y)} \wedge \mathcal{F}^{-1}_{(q,y)}. \tag{2.15}$$

The second equality holds by definition of $\mathcal{F}^{-1}$. Equivalently we can rewrite the rows of $\mathcal{C}_q$ as

$$\mathcal{C}_{q(x,:)} = \mathcal{F}^{-1}_{(x,:)} \wedge \mathcal{F}^{-1}_{(q,:)} \tag{2.16}$$

By the expansion $\mathcal{T}_q s = \mathcal{C}_q \widehat{s} = \mathcal{C}_q \mathcal{F} s$ of equation (2.14) we derive

$$\mathcal{T}_q = \mathcal{C}_q \mathcal{F} \tag{2.17}$$

Example 2.19 *The common causes matrix C_{y_1} of vertex y_1 in DAG G is given by*

$$C_{y_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.18}$$

The corresponding interpretation on the graph is illustrated in Figure 2.2. We observe that every row $C_{q(v,:)}$ contains the common causes of v and y_1.

Example 2.20 *The corresponding shift matrix T_{y_1} is given by*

$$T_{y_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.19}$$

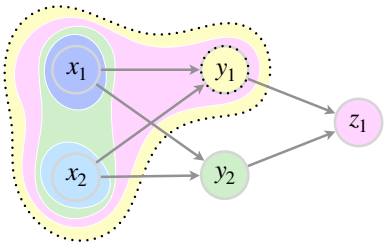


Figure 2.2: Causes of y_1 are circled by the dotted line. Common causes of y_1 and a vertex in G are the colored fields within the dotted line.

Shifting signal s from example 2.9 by y_1 we get $T_{y_1}s = (1,0,1,1,2) = C_{y_1}\widehat{s}$.

Shifting events Recall that the value of event x is the sum of the unobserved strengths of *all* causes of x. Now if we shift an event x by event q then x's value is only the sum of unobserved strengths of those causes which also cause q, i.e, their common causes. Consequently, if x and q have no causes in common, meaning that there no event y affects both x and q, then the shifted value of x is 0. On the other hand, if q is the event x itself or q is an effect of x, then the value of x is not altered by the shift. The shift $\mathcal{T}_q$ removes all unobserved strengths in a signal s that are not causes of q or q itself.

In classical discrete time signal processing, the Fourier transform is the expansion of a signal s in terms of the complex exponentials, which are the eigenvectors of all shifts. We show that the same applies to the causal Fourier transform basis vectors.

Lemma 2.21 (Fourier basis) *The columns of our inverse Fourier transform matrix, i.e. the vectors that span our **Fourier basis***

$$\mathrm{f}^y = \mathcal{F}^{-1}_{(:,y)} \tag{2.20}$$

are simultaneous eigenvectors of all shifts [3].

Proof Let T_q be the shift matrix of an arbitrary element $q \in \mathcal{V}$ and $\mathrm{f}^y = \mathcal{F}^{-1}_{(:,y)}$ an arbitrary Fourier basis vector. We obtain

$$T_q\mathrm{f}^y \overset{(2.17)}{=} (C_qF)\mathrm{f}^y \overset{(2.20)}{=} C_q(\mathcal{F}\mathcal{F}^{-1}_{(:,y)}) = C_q\mathrm{e}_y = C_{q(:,y)}\ , \tag{2.21}$$

where e_y denotes the y^{th} standard basis vector.

For the next step, we think of how we can construct C_q using equation (2.16). We can take the row $F^{-1}_{(q,:)}$ and scan F^{-1} from top to bottom, like a sliding window, while performing the element-wise logical and operation between

row $F^{-1}_{(q,:)}$ and a row in F^{-1}. As a result we obtain C_q. What does this mean for a column $C_{q(:,y)}$? If our sliding window is 0 at $F^{-1}_{(q,y)}$, the column $C_{q(:,y)}$ will be the null vector. If our sliding window is 1 at $F^{-1}_{(q,y)}$ it will simply copy the entries from $F^{-1}_{(:,y)}$, which is equal to f^y, in column y.

It follows

$$T_q \mathrm{f}^y \overset{(2.21)}{=} C_{q(:,y)} = \begin{cases} 1 \cdot \mathrm{f}^y & \text{if } F^{-1}_{(q,y)}\,, \\ 0 \cdot \mathrm{f}^y & \text{otherwise}\,, \end{cases} \tag{2.22}$$

meaning that f^y is an eigenvector to the eigenvalue 1 if $F^{-1}_{(q,y)}$, i.e. $y \leq q$, and an eigenvector to the eigenvalue 0 otherwise. □

Lemma 2.21 implies that our Fourier basis simultaneously diagonalizes all shift matrices. Formally, for a particular T_q and its eigenvalues

$$\lambda_y = \begin{cases} 1 & y \leq q \\ 0 & otherwise \end{cases} \tag{2.23}$$

for all $y \in \mathcal{P}$ we get

$$\mathcal{F}\mathcal{T}_q\mathcal{F}^{-1} = \begin{bmatrix} \ddots & & \\ & \lambda_y & \\ & & \ddots \end{bmatrix} \in \{0,1\}^{n \times n}. \tag{2.24}$$

In classical discrete time signal processing filtering, i.e. convolution, is the process of representing an input signal s as a linear combination of complex exponentials [20]. Likewise

Definition 2.22 (Causal convolution) *A **causal filter** is defined as a linear combination of shifts $h = \sum_{q \in \mathcal{P}} h_q \mathcal{T}_q$ and provides us with a notion of **causal convolution** defines as*

$$h * s = \left(\sum_{q \in \mathcal{P}} h_q \mathcal{T}_q \right) s \tag{2.25}$$

[3].

Filtering events: Rewriting the convolution (2.25) as

$$h * s = \left(\sum_{q \in \mathcal{P}} h_q \mathcal{C}_q \mathcal{F} \right) s = \left(\sum_{q \in \mathcal{P}} h_q \mathcal{C}_q \right) \widehat{s} \tag{2.26}$$

gives us some intuition of the meaning behind it. For every $q \in \mathcal{P}$ a filter h assigns a factor h_q of importance to the causes of q and q itself. The importance of an event x is the sum of factors h_q of importance it has received.

After convolution with h, the new unobserved strength of an event x is the original strength multiplied with its importance.

The convolution theorem is a fundamental property of the Fourier transform in classical discrete time signal processing. It states that the convolution of a signal in the time domain corresponds to point-wise multiplication in the Fourier domain. We show that the same applies in CSP.

Theorem 2.23 *The convolution of a signal in the poset-domain corresponds element-wise multiplication in the Fourier-domain.*

$$h * s = \mathcal{F}^{-1}(\bar{h} \odot \mathcal{F}s) \,, \tag{2.27}$$

where $\odot$ denotes the element-wise multiplication.

Proof

$$\begin{aligned}
h * s &= \left(\sum_{q \in \mathcal{P}} h_q \mathcal{T}_q \right) s \\
&= \mathcal{F}^{-1} \mathcal{F} \left(\sum_{q \in \mathcal{P}} h_q \mathcal{T}_q \right) \mathcal{F}^{-1} \mathcal{F} s \\
&\overset{\text{lin.}}{=} \mathcal{F}^{-1} \left(\sum_{q \in \mathcal{P}} h_q \mathcal{F} \mathcal{T}_q \mathcal{F}^{-1} \right) \mathcal{F} s \\
&\overset{(2.24)}{=} \mathcal{F}^{-1} \left(\sum_{q \in \mathcal{P}} h_q \begin{bmatrix} \ddots & & \\ & \lambda_y & \\ & & \ddots \end{bmatrix} \right) \mathcal{F} s \\
&= \mathcal{F}^{-1} \left(\sum_{q \in \mathcal{P}} \begin{bmatrix} \vdots \\ h_q \lambda_y \\ \vdots \end{bmatrix} \odot \mathcal{F} s \right) \\
&= \mathcal{F}^{-1} \left(\bar{h} \odot \mathcal{F} s \right)
\end{aligned}$$

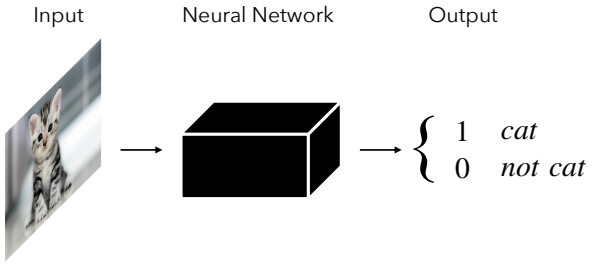
□

2.2 Neural Networks

This section covers the basics of neural networks. The objective is to understand the underlying structure of a neural network. We start by looking at its fundamental concept. Later we explain how classic deep neural networks are built and introduce corresponding terminology which is relevant for the next chapters. [9] and [11] were used as a reference for this section.

Machine learning (ML) has given rise to a new popular predictive methodology. Neural networks, whose name is inspired by the architecture and learning process of a human brain [11].

Imagine you take a picture of your cat. While your human brain sees an image of a *cat* to a computer this is just a matrix with a collection of numbers. It can not identify the image as a *cat* unless it is explicitly told so. If you take another picture of your cat the next day you will again see your *cat*. To a computer this a another matrix of *different* values. How could it identify the second image as a *cat*? Explicitly writing the code for a *cat* classifier is hard. This is where neural networks come in. We can find a *cat* classifier by fitting the weights of a neural network to data through optimization, also referred to as the training phase. Given enough data, i.e. sample images of *cats* and *not cats*, they can be trained to classify an arbitrary new image as a *cat* or *not cat*.



Cat-image-https://blog.mystart.com/my-cats-adorable-cat-kitten-hd-wallpapers/, 24-09-2021

Figure 2.3: An input image is fed into the neural network. The network evaluates all pixels values of the image to determine whether it is a *cat* or *not*. Here, the output is given in form of a one-hot encoding which is typical for classification tasks [9].

2.2.1 Neural Networks are Computational Graphs

Essentially every neural network represents one (large) mathematical function. The general structure of the function is predefined by the architecture of the network. The exact function however is obtained by step-wise refinement during the training process.

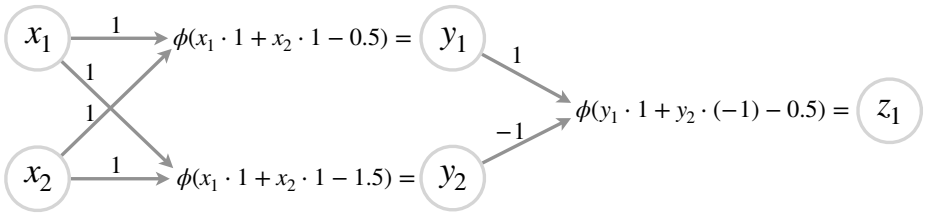


Figure 2.4: Computational graph of equation (2.28). x_1 and x_2 are input vertices. y_1 and y_2 are intermediate vertices. For input $x \in \mathbb{R}^2$, y_1 and y_2 detect the logical x_1 OR x_2 and logical x_1 AND x_2, respectively. The output vertex z_1 builds upon y_1 and y_2 to compute the more complicated function $f(x) = x_1 XOR x_2$.

Definition 2.24 (Computational Graph) *A **Computational Graph** is a way to express and evaluate a mathematical function in the terms of graph theory. It is a directed graph where every vertex corresponds to a mathematical operation. Incoming edges deliver the necessary variables for the operation. The result of the operation is forwarded along the outgoing edges as an input variable of further operations. The vertices without incoming edges are defined as **input vertices**. The vertices without outgoing edges are defined as **output vertices**. All remaining ones are **intermediate vertices**.*

Example 2.25 *Consider the function*

$$f(x) = \varphi\left(\varphi(x_1 + x_2 - 0.5) - \varphi(x_1 + x_2 - 1.5) - 0.5\right) \tag{2.28}$$

with

$$\varphi(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.29}$$

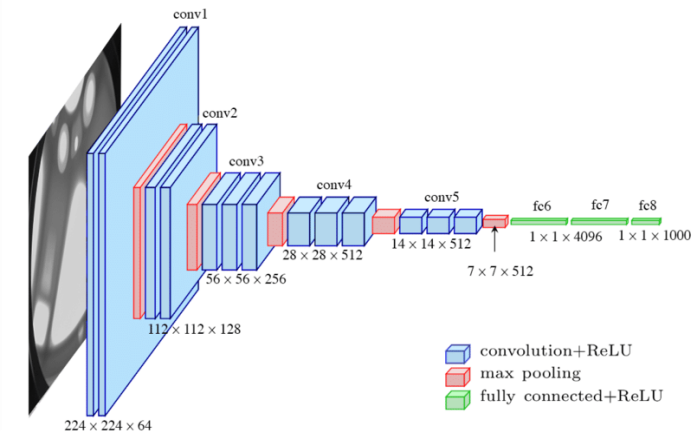
One can easily verify that f effectively classifies a given input $x = [x_1, x_2] \in \{0,1\}^2$ as

$$f(x) = \begin{cases} 1 & \text{if } x_1 \text{ XOR } x_2 \\ 0 & \text{else} \end{cases} \tag{2.30}$$

*Equivalently we can depict this classification in form of a **computational graph** as can be seen in Figure 2.4.*

A neural network is a computational graph. We could say the computational graph in Figure 2.4 is a tiny neural network that computes x_1 XOR x_2. However, when we refer to neural networks, we use a different terminology.

Definition 2.26 (Neuron) *The vertices of the computational graph of a neural network are called **neurons**. We refer to the input vertices of the computational graph as **input neurons**, the intermediate vertices as **intermediate neurons**, and output vertices as **output neurons**.*



https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b, 24-09-2021

Figure 2.5: The architecture of the VGG-16 [21]. The image on the left is the input. Every one of the 16 cuboids represents one layer. The output layer is the last cuboid on the right.

Definition 2.27 (Input) *The **input** of a neural network are the values assigned to the input neurons.*

Definition 2.28 (Output) *The **output** of a neural network are the values computed at the output neurons.*

Definition 2.29 (Activation) *For a given input, the weighted sum computed at each individual neuron in a neural network is called **activation**.*

Definition 2.30 (Activation pattern) *The **activation pattern** generated by an input is the collection of activations of all neurons in the neural network.*

2.2.2 Deep Neural Networks

Typically neural networks are composed of building blocks called layers. A network with at least one layer between the input and output is a deep neural network [9]. Figure 2.5 shows the architecture of the neural network VGG-16 [21]. It is composed of 16 layers of three different types and an input layer. In this section, we shortly describe the layer types used in the networks of our experiments. In the following, understanding the examples of the corresponding layer types is more important than their formal equations. The definitions originate from [13].

Definition 2.31 (Linear layer) *For incoming data $x \in \mathbb{R}^{n_{i-1}}$ a **linear layer** applies an affine transformation of the form*

$$y = xA^T + b \tag{2.31}$$

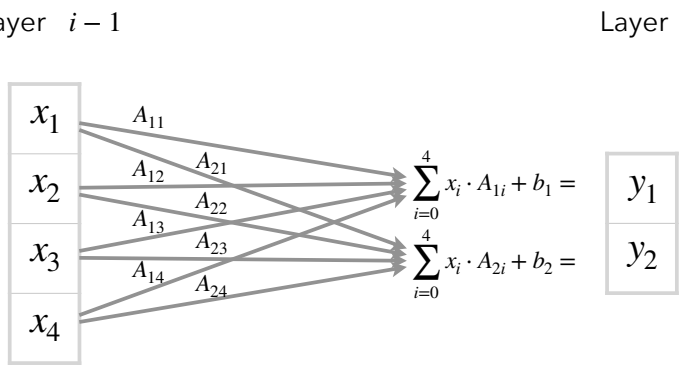


Figure 2.6: Linear layer i with input $x \in \mathbb{R}^4$ and output $y \in \mathbb{R}^2$

where $A \in \mathbb{R}^{n_i \times n_{i-1}}$ and $b \in \mathbb{R}^{n_i}$, hence $y \in \mathbb{R}^{n_i}$ [13].

A linear layer is also called a **fully connected layer**. Looking at its computational graph, we see that every input neuron is connected to every output neuron.

Example 2.32 *Figure 2.6 illustrates the computational graph of a linear layer i with input $x \in \mathbb{R}^4$ and output $y \in \mathbb{R}^2$.*

When classifying images, it is not helpful to look at one pixel at a time. In order to identify larger patterns, we take neighbourhoods of 3×3 or 5×5 pixels and summarize their importance in a single activation.

Definition 2.33 (Maximum pooling layer) *A maximum pooling layer takes an input $x \in \mathbb{R}^{C \times H \times W}$ and applies a kernel of size $H_k \times W_k$ to produce output*

$$y_{c,i,j} = \max_{h \in \{1,\dots,H_k\}} \max_{w \in \{1,\dots,H_w\}} x_{c,\ H_k \cdot (i-1)+h,\ W_k \cdot (j-1)+w} \quad , \tag{2.32}$$

where y is in $\mathbb{R}^{C \times \frac{H}{H_k} \times \frac{W}{W_k}}$ [13].

Example 2.34 *Figure 2.7 gives an example for a maximum pooling layer i with input $x \in \mathbb{R}^{4 \times 4 \times 4}$ and a kernel of size 2×2. $y \in \mathbb{R}^{4 \times 2 \times 2}$ is the output of layer i. The kernel acts like a sliding window on the input. It considers all x_c for some $c \in \{1, \cdots C\}$ separately. It starts in the top left corner of x_c and copies the maximum value of x_c, screened by the kernel window, to the top left corner of the output y_c. Sliding to the side by $W_k = 2$ or up and down by $H_k = 2$ it repeats the process.*

Definition 2.35 (Average pooling layer) *An average pooling layer takes an input $x \in \mathbb{R}^{C \times H \times W}$ and applies a kernel of size $H_k \times W_k$ in the following way*

$$y_{c,i,j} = \frac{1}{H_k \cdot W_k} \sum_{h=1}^{H_k} \sum_{w=1}^{W_k} x_{c,\ H_k \cdot (i-1)+h,\ W_k \cdot (j-1)+w} \tag{2.33}$$

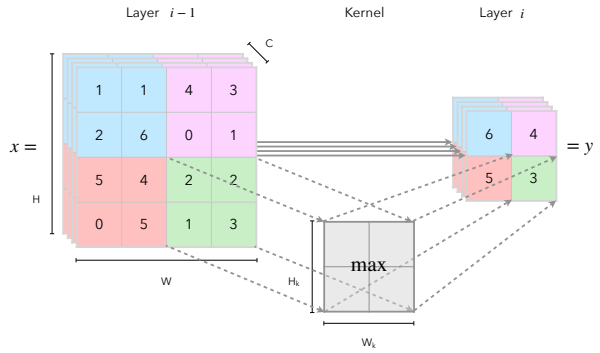


Figure 2.7: Input $x \in \mathbb{R}^{4\times4\times4}$ and output $y \in \mathbb{R}^{4\times2\times2}$ of a maximum pooling layer with a 2×2 kernel.

[13].

Example 2.36 *Figure 2.8 gives an example of an average pooling layer. The setup is the same as in example 2.34 except that the type of layer i to an average pooling layer. The definition of the kernel changes. It takes the average of all values screened by the kernel and copies it to the output.*

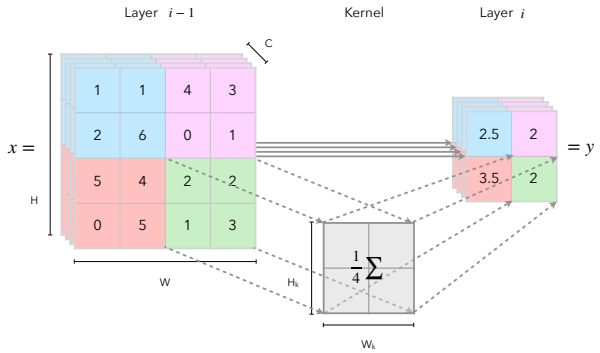


Figure 2.8: Input $x \in \mathbb{R}^{4\times4\times4}$ and output $y \in \mathbb{R}^{4\times2\times2}$ of an average pooling layer with a 2×2 kernel.

Definition 2.37 (Convolutional layer) *In the simplest case, a **convolutional layer** takes an input $x \in \mathbb{R}^{C_{in}\times H_{in}\times W_{in}}$ and applies $C_{out} \cdot C_{in}$ kernels $K_1, \ldots, K_{C_{in}\cdot C_{out}}$ each of size $H_k \times W_k$ to produce an output $y \in \mathbb{R}^{C_{out}\times H_{out}\times W_{out}}$. The operations defined as*

$$y_k = b_k + \sum_{c=1}^{C_{in}} K_{k,c} \star x_c \quad , \tag{2.34}$$

where $y_k \in \{1, \ldots, C_{out}\}$, $b_k \in \mathbb{R}^{H_{out}\times W_{out}}$ and $\star$ denotes the valid cross-correlation operator. $\star$ depends on various other parameters not explained here for the sake of

brevity. In the simplest case,[1] *we get*

$$(K_{k,c} \star x_c)_{i,j} = \sum_{h=1}^{H_k} \sum_{w=1}^{W_k} K_{k,c,h,w} \cdot x_{c,\ (i-1)+h,\ (j-1)+w} \tag{2.35}$$

for $i \in \{1, \ldots, H_{out}\}$ *and* $j \in \{1, \ldots, W_{out}\}$.

Example 2.38 *Figure 2.9 illustrates a convolutional layer i with input* $x \in \mathbb{R}^{4\times4\times4}$ *and 8 kernels of size* 2×2. $y \in \mathbb{R}^{2\times3\times3}$ *is the output of layer i. First, for every pair of input* x_c *and corresponding kernel* $K_{k,c}$ *the sum of element-wise multiplications, is computed as in equation* (2.35). *The operation then sums up the* $C_{in} = 4$ *matrices of size* $H_{out} \times W_{out} = 3 \times 3$ *and adds a bias term* $b_k \in \mathbb{R}^{H_{out}\times W_{out}}$ *for every* $k \in \{1, \ldots, C_{out}\}$.

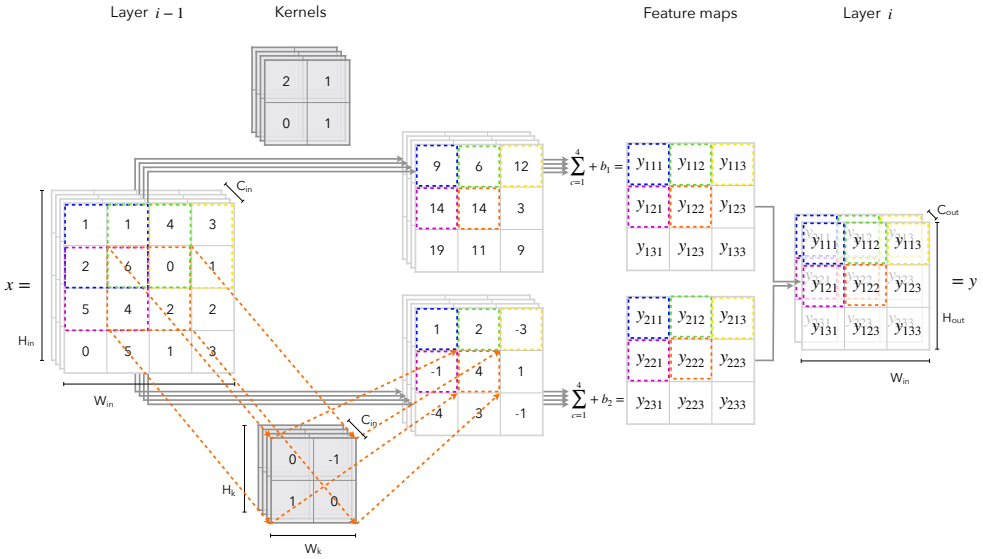


Figure 2.9: Illustration of a convolutional layer i with 8 kernels of size 2×2. The $4 \times 4 \times 4$ matrix x on the left is taken as an input. The output of the layer is the $2 \times 3 \times 3$ matrix y on the right. The dotted lines illustrate the computation of the cross-correlation $\star$.

During the training phase of a neural network, the values in the kernels and bias terms are being refined in a self-correcting manner. In a linear layer, this corresponds to the entries in matrix A and b. Recall that a linear layer has inputs $x \in \mathbb{R}^{n_i-1}$ and outputs $y \in \mathbb{R}^{n_i}$. In cases where a layer with output $x \in \mathbb{R}^{C_{out}\times H_{out}\times W_{out}}$ precedes a linear layer the x is reshaped into vector $x \in \mathbb{R}^{C_{out}\cdot H_{out}\cdot W_{out}}$.

[1] [13] stride = 1, dilation = 1 and padding = 0

Definition 2.39 (Channels) *Let $y \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$ be the output of a layer i. A submatrix $y_c \in \mathbb{R}^{H_{out} \times W_{out}}$ is referred to as **channel** or **output channel** of layer i. Accordingly, for the input $x \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$ of layer i, a submatrix $x_c \in \mathbb{R}^{H_{in} \times W_{in}}$ is referred to as **input channel**.*

Definition 2.40 (Feature map) *We refer to a **feature map** with respect to a channel as the set of values in that channel.*

Example 2.41 *The feature maps computed by the convolutional layer in Figure 2.9 are the values in the two matrices of size 3×3, i.e. the two output channels.*

Chapter 3

Causal Signal Processing on Neural Networks

In the last Section, we explained how a neural network takes values of pixels and transforms them across multiple layers. In Figure 2.4, we have seen that activations of neurons are influenced by activations of previous neurons. Unfortunately, the meanings behind the intermediate activations are not always as clear as in the carefully constructed example 2.25. While there are evident reasons to believe that large networks encode deeper information in the values of their intermediate layers, up till now, it remains challenging to reveal this information [26].

We now show how the causal signal processing framework on posets may provide a mathematical tool to analyze activation patterns of neural networks.

3.1 Computational Graph as Poset

The computational graph of a neural network is, for most architectures[1], a DAG. By $\mathcal{V}$ we denote the set of vertices in the computational graph, i.e. neurons in the neural network. The computational graph has directed edges where the activation of one neuron is a direct input to the activation of another neuron. As described in section 2.1, this directed acyclic graph, encoded by a neural network, contains a poset.

We formalize this with some notation that we will use throughout the remaining chapters of this thesis.

By ℓ we denote the number of layers in a neural network *including* the input

[1] Recurrent neural networks (RNN) have backward edges. Their computational graph is not acyclic. In this case we limit the RNN to t processing steps such that we can unroll it.

layer. Our set of vertices $\mathcal{V}$ is defined by

$$\mathcal{V} = \biguplus_{i=1}^{\ell} \mathcal{V}_i \tag{3.1}$$

, where $\uplus$ denotes the disjoint union and $\mathcal{V}_i$ the set of neurons in the ith layer. In particular, $\mathcal{V}_1$ is the set of input neurons and $\mathcal{V}_\ell$ is the set of output neurons. By n_i we denote the number of neurons in layer i, i.e. $n_i = |V_i|$.

Example 3.1 *In Figure 2.9 we observe that there are multiple intermediate computations or values within one layer i. There is some degree of freedom in which elements one considers as a vertex in the computational graph and which ones one ignores or, stated in graph theoretical-terms, contracts. In our definition $\mathcal{V}_i$ is the set of output elements in matrix $y \in \mathbb{R}^{2\times3\times3}$. Hence, we get $n_i = 2 \cdot 3 \cdot 3 = 18$. Accordingly, $\mathcal{V}_{i-1}$ is the set of elements in matrix $x \in \mathbb{R}^{4\times4\times4}$ and $n_{i-1} = 4 \cdot 4 \cdot 4 = 64$.*

The set of edges $\mathcal{E}$ is given by

$$\mathcal{E} = \biguplus_{i=1}^{\ell} \mathcal{E}_i = \biguplus_{i=2}^{\ell} \mathcal{E}_i \tag{3.2}$$

, where $\mathcal{E}_i$ denotes the set of edges *incoming* to neurons in $\mathcal{V}_i$. Since the input $\mathcal{V}_1$ has no incoming edges we have $\mathcal{E}_1 = \varnothing$ and $m_1 = |\mathcal{E}_1|$ which explains the second equality in (3.2). We have an edge from u to v, if the value at u is used in the intermediate calculations of neuron v.

Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed acyclic computational graph. $(\mathcal{V}, \preceq)$, with $u \preceq v$ if $(u, v) \in \mathcal{E}$, is the partially ordered set contained in the computational graph of a neural network.

3.2 Activation Pattern as Signal

Every input of a neural network yields an activation pattern. Every activation value in the activation pattern is associated with one neuron. Therefore we have values indexed by the elements of the poset $(\mathcal{V}, \preceq)$ of a neural network. An activation pattern is a signal s on this poset $(\mathcal{V}, \preceq)$.

Example 3.2 *The signal values associated with the neurons, i.e. elements in $\mathcal{V}$, in the convolutional layer of Figure 2.9 are all $y_{c,h,w}$ for $c \in \{1, 2\}$ and $h, w \in \{1, 2, 3\}$.*

Example 3.3 *In example 2.25 we constructed a tiny neural network computing the XOR of an input $x \in \{0, 1\}^2$. Figure 2.4 shows its computational graph. The corresponding DAG that we use in the signal processing framework is precisely the one in Figure 2.1a. Let $x = [x_1, x_2] = [1, 0]$ be the input. Feeding x into the computational graph in Figure 2.4 we get $y_1 = 1$, $y_2 = 0$ and $z_1 = 1$. This is the*

signal shown in Figure 2.1b. The corresponding Fourier coefficients are shown in Figure 2.1c.

Take a closer look at function (2.28). *At the end of the calculations at a neuron, a function* φ (2.29) *is applied. Such functions, so-called activation functions, are very typical in neural networks as they bound the values in the activations.*[2] *In this thesis, we consider the values* after *the activation function is applied.*

Project Outline In this thesis, we collect and explore neural activation patterns of various neural networks trained on image recognition. In particular we will:

- Train convolutional neural networks (CNNs) with up to 2^{15} neurons on image datasets MNIST, Cifar10 and FashionMNIST.[3]
- Extract their associated computational graphs and activation patterns generated by various input images.
- Compute their Fourier transform and coefficients.
- Perform a numerical analysis of our results.
- Visualize and cluster the data.

[2]Usually activation functions are differentiable, unlike ours

[3]The network specifications are listed in Table B.1 in the appendix.

Chapter 4

Implementation

We built and trained neural networks on image datasets using the open source machine learning framework PyTorch [13]. We then took input images and captured the activations generated after each layer of the neural network.

This chapter describes in which way we implemented the components of the CSP framework. We start by extracting the computational graph with its respective neurons and their connections in form of a directed graph. We then compute the Fourier transform, its inverse, and the Fourier coefficients. Finally, we give a short summary of the computational complexity of our implementation.

The following sections outline our implementation of the CSP framework. For faster implementations, we refer to [16]. It develops a novel approach that produces zeta and Moebius transforms with reduced overall memory consumption and runtime.

We used NetworkX [5], a python package for networks that provides us with a graph data-structure and functions for basic graph operations. For the data clustering, we used scikit-learn [15]. scikit-learn is an open-source machine learning library that provides tools for predictive data analysis.

4.1 Extracting the Computational Graph of a Neural Network

Given a trained neural network we need to construct its underlying computational graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. First, we create the set of vertices $\mathcal{V}$. Second, we insert the edges $\mathcal{E}$.

4.1.1 Extracting the Vertices

A convenient way to retrieve the set of vertices $\mathcal{V}$ is to directly feed a sample input into the neural network and collect its generated activation pattern [1]. The n_0 input values and $n - n_0$ activation values constitute one signal $s \in \mathbb{R}^n$. Every value in the signal is indexed by exactly one vertex in $\mathcal{V}$. Every vertex in $\mathcal{V}$ is associated with exactly one element of the signal. Therefore we can construct $\mathcal{V}$ by inserting a vertex for every value in s.

This can be executed in $\mathcal{O}(n)$.

4.1.2 Extracting the Edges

We propose two possible approaches to retrieve the set of directed edges $\mathcal{E}$[2]. Both approaches start with the empty set $\mathcal{E} = \{\}$ and insert incoming edges of a vertex ascending by layer, hence the input layer 1 can be skipped.

Approach 1: Layer type (specific)

The straightforward approach is to explicitly define the incoming edges for each layer type. Then, iterating through layers $i = \{2, \ldots, l\}$, we add edges $\mathcal{E}_i$ depending on the type of layer i.

For some layer types this can be trivial:

Example 4.1 *Let layer i be a linear layer, also known as a fully connected layer. Then*

$$\mathcal{E}_i = \{(u, v) \mid u \in \mathcal{V}_{i-1},\ v \in \mathcal{V}_i\} \tag{4.1}$$

is its set of edges.

The crucial drawback of this approach is the scalability with regard to the variety of layer types. For every new type of layer contained in a network, one has to define an edge-retrieving function. On top of this, the implementation of many layer types turns out to be tedious.

Example 4.2 *Parameters of a convolutional layer like: kernel size, stride, padding, dilation and the number of groups all affect the way edges are drawn.*

Approach 2: Jacobian matrix (generic)

Recall that each layer is essentially defined by a differentiable[3] function f_i that takes the activations from the previous layer[4] $i - 1$ as an input $x \in \mathbb{R}^{n_{i-1}}$ to compute the activations of the current layer i as an output $y \in \mathbb{R}^{n_i}$.

[1] The retrieval of this pattern depends on the neural network framework used.

[2] Depending on the NN framework used for implementation other convenient methods to define $\mathcal{E}$ may arise.

[3] Differentiability is essential for the training process of a NN.

[4] collection of previous layers in the case of residual neural networks (ResNets).

For simplicity, we display the Jacobian matrix for input vector x and output vector y. In example 4.4 we will extend both to two dimensional matrices. We use the matrix notation as defined in [22]. In general, the concept applies to any input and any output dimension.

Definition 4.3 (Jacobian Matrix) *Let*

$$f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, x \mapsto y$$

be a function where all first-order partial derivatives exist. Then the ***Jacobian matrix*** *represents all first-order partial derivatives of the function f_i.*

$$J(f_i) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_{n_{i-1}}} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_{n_{i-1}}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{n_i}}{\partial x_1} & \frac{\partial y_{n_i}}{\partial x_2} & \dots & \frac{\partial y_{n_i}}{\partial x_{n_{i-1}}} \end{bmatrix} \in \mathbb{R}^{n_{i-1} \times n_i}$$

We take the output activations of layer $i-1$ as an input x and compute the Jacobian matrix of the layer function f_i. We get:

$$J(f_i)_{(u,v)} \neq 0 \implies \text{vertex } u \text{ is an input of vertex } v \tag{4.2}$$

Using equation (4.2) we can define

$$\mathcal{E}_i = \{(u,v) \mid J(f_i)_{(u,v)} \neq 0\} \tag{4.3}$$

In particular, $J(f)_{(u,v)}$ contains the parameter in the kernel associated with edge (u,v).

Example 4.4 *Let layer i be a convolutional layer with one kernel k. Then f_i is the convolutional function* (2.34) *with kernel k. Furthermore, let x be the activation in layer $i-1$ with $n_{i-1} = 9$ activations. By definition of f_i the activation y in layer i has shape $= 2 \times 2$ with $n_{i-1} = 4$ activations.*

$$x = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \qquad k = \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 & y_2 \\ y_3 & y_4 \end{bmatrix}$$

$$J(f_i) = \begin{bmatrix} \begin{bmatrix} \frac{\partial y_1}{\partial x} \end{bmatrix} & \begin{bmatrix} \frac{\partial y_2}{\partial x} \end{bmatrix} \\ \begin{bmatrix} \frac{\partial y_3}{\partial x} \end{bmatrix} & \begin{bmatrix} \frac{\partial y_4}{\partial x} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \frac{\partial y_1}{\partial x_3} \\ \frac{\partial y_1}{\partial x_4} & \frac{\partial y_1}{\partial x_5} & \frac{\partial y_1}{\partial x_6} \\ \frac{\partial y_1}{\partial x_7} & \frac{\partial y_1}{\partial x_8} & \frac{\partial y_1}{\partial x_9} \end{bmatrix} & \begin{bmatrix} \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \frac{\partial y_2}{\partial x_3} \\ \frac{\partial y_2}{\partial x_4} & \frac{\partial y_2}{\partial x_5} & \frac{\partial y_2}{\partial x_6} \\ \frac{\partial y_2}{\partial x_7} & \frac{\partial y_2}{\partial x_8} & \frac{\partial y_2}{\partial x_9} \end{bmatrix} \\ \begin{bmatrix} \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} & \frac{\partial y_3}{\partial x_3} \\ \frac{\partial y_3}{\partial x_4} & \frac{\partial y_3}{\partial x_5} & \frac{\partial y_3}{\partial x_6} \\ \frac{\partial y_3}{\partial x_7} & \frac{\partial y_3}{\partial x_8} & \frac{\partial y_3}{\partial x_9} \end{bmatrix} & \begin{bmatrix} \frac{\partial y_4}{\partial x_1} & \frac{\partial y_4}{\partial x_2} & \frac{\partial y_4}{\partial x_3} \\ \frac{\partial y_4}{\partial x_4} & \frac{\partial y_4}{\partial x_5} & \frac{\partial y_4}{\partial x_6} \\ \frac{\partial y_4}{\partial x_7} & \frac{\partial y_4}{\partial x_8} & \frac{\partial y_4}{\partial x_9} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} k_1 & k_2 & 0 \\ k_3 & k_4 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & k_1 & k_2 \\ 0 & k_3 & k_4 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ k_1 & k_2 & 0 \\ k_3 & k_4 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & k_1 & k_2 \\ 0 & k_3 & k_4 \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{n_{i-1} \times n_i}$$

The fact that $J(f)_{(u,v)}$ holds the edge weight of edge (u, v) introduces special cases to consider:

Dead edges During the training process of a network edge weights can diminish to 0 or almost 0. In such cases, it is a matter of design choice whether one introduces some threshold value $\theta \in \mathbb{R}$ and replaces (4.3) by

$$\mathcal{E}_i = \{(u, v) \mid J(f_i)_{(u,v)} \geq \theta\} \tag{4.4}$$

Alternatively one can reassign all edge weights to 1 in function f_i before computing $J(f_i)$. In this thesis, we opt for the latter.

Input-value dependent edges Some edge weights depend on the specific activation values contained in x. Different images generate different signals, hence different activations. We want our computational graph to be independent of specific signals. It should represent any *possible* flow of values. This is why some layer types require a workaround.

Example 4.5 *Let us continue the example 4.4 but change layer i to a Maximum Pooling layer. We show that two different input activations x and x' yield different matrices $J(f_i(x))$ and $J(f_i(x'))$.*

$$x' = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad J(f_i(x')) = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix}$$

$$x = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad J(f_i(x)) = \begin{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{bmatrix}$$

Workaround: Changing function f_i from a Maximum Pooling function to Average Pooling function we get:

$$J(f_i) = \begin{bmatrix} \begin{bmatrix} 1/4 & 1/4 & 0 \\ 1/4 & 1/4 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1/4 & 1/4 \\ 0 & 1/4 & 1/4 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 1/4 & 1/4 & 0 \\ 1/4 & 1/4 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1/4 & 1/4 \\ 0 & 1/4 & 1/4 \end{bmatrix} \end{bmatrix}$$

for any input.

The algorithm of the first approach runs in $\mathcal{O}(m_i)$. This amounts to $\mathcal{O}(\sum_{i=2}^{l} m_i) = \mathcal{O}(m)$ for the entire graph.

The algorithm of the second approach runs in $\mathcal{O}(t_{jac} + n_{i-1} \cdot n_i)$. Where t_{jac} denotes the time required to compute the jacobian matrix. We assume that the jacobian matrix can be computed using the layer function f_i, hence $t_{jac} \in \mathcal{O}(n_{i-1} \cdot n_i)$. This amounts to $\mathcal{O}(\sum_{i=2}^{l} n_{i-1} \cdot n_i) = \mathcal{O}(n^2)$ for the entire graph. In the last step, we used $l \ll n$, which means that the number of layers l is negligible compared to n.

In practice, a hybrid version of the two approaches can be used since each layer is evaluated independently. One can implement the most frequently used layer types explicitly and back up with the second approach to broaden the range of analyzable networks.

4.2 Computing the inverse Causal Fourier Transform of a DAG

Given the cover graph, we can compute the inverse causal Fourier transform $\mathcal{F}^{-1}$. Recall that

$$\mathcal{F}^{-1}_{[\eta_x, \eta_y]} = \begin{cases} 1 & \text{if } y \leq x, \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

We can extend equation (4.5) for $\mathcal{F}^{-1}_{[\eta_x,:]} \in \{0,1\}^n$ as follows:

$$\begin{aligned}
\mathcal{F}^{-1}_{[\eta_x,\,:]} &= (\iota_{y\leq x})_{y\in\mathcal{V}} && (4.6)\\
&= (\iota_{y<x})_{y\in\mathcal{V}} \vee (\iota_{y=x})_{y\in\mathcal{V}} && (4.7)\\
&= \bigvee_{(y,\,x)\in\mathcal{E}} \underbrace{(\iota_{z\leq y})_{z\in\mathcal{V}}}_{(4.6)} \vee (\underset{0}{0},\ldots,\underset{\eta_x-1}{0},\underset{\eta_x}{1},\underset{\eta_x+1}{0},\ldots,\underset{\eta-1}{0}) && (4.8)
\end{aligned}$$

In (4.8) we used that all causes of x must either be direct causes y of x, or causes z of the direct causes y. We see that the row η_x of a vertex x in $\mathcal{F}^{-1}$ is recursively defined as the element-wise or of the rows of all its direct ancestor vertices and $(\iota_{y=x})_{y\in\mathcal{V}} \in \{0,1\}^n$. Equation (4.8) defines an iterative way to compute $\mathcal{F}^{-1}$ for ascending rows η_x.

The asymptotic time complexity to compute row η_x of a vertex x is $\mathcal{O}(deg^+(x)\cdot n)$ if the computational graph is stored as an adjacency list. This yields an $\mathcal{O}(\sum_{v\in\mathcal{V}} deg^+(v)\cdot n) = \mathcal{O}(mn)$ algorithm to define $\mathcal{F}^{-1}$.

4.3 Computing the Causal Fourier Transform of a DAG

For visualization purposes, we want to compute matrix $\mathcal{F}$. Taking the inverse $\mathcal{F}^{-1}$ lead to low numerical precision, so we used the explicit Moebius function.

Recall that

$$\begin{aligned}
\mathcal{F}_{[\eta_x,\eta_y]} &= \mu(y,x)\\
&= \begin{cases} 0 & \text{if } x \not\leq y\,,\\ 1 & \text{if } x = y\,,\\ -\sum_{y\leq z<x}\mu(y,z) & \text{otherwise.}\end{cases} && (4.9)
\end{aligned}$$

We observe a recursive equation (4.9). For ascending rows η_x this lets us compute all entries of $\mathcal{F}$ iteratively. The asymptotic complexity to compute one entry $\mathcal{F}_{[\eta_x,\eta_y]}$ depends on the number of elements z such that $y \leq z < x$ and the time it takes to retrieve them.

For all vertices z with $\eta_z \in \{\eta_y,\ldots,\eta_x-1\}$, we can lookup $y \leq z$ and $z \leq x$ in $\mathcal{F}^{-1}$. *Given* $\mathcal{F}^{-1}$, the lookup takes $\mathcal{O}(1)$ time per vertex which amounts to $\mathcal{O}(n)$ in total. Additionally, we can skip all lookups if $x \not\leq y$, i.e. $\mathcal{F}^{-1}_{[\eta_x,\eta_y]} = 0$. Filling all entries of $\mathcal{F}$ requires $\mathcal{O}(n^3+mn) = \mathcal{O}(n^3)$ time.

4.4 Computing the Fourier Coefficients

Given the inverse Fourier transform matrix $\mathcal{F}^{-1} \in \mathbb{Z}^{n \times n}$ and a signal $s \in \mathbb{R}^n$ we can compute the Fourier coefficients $\widehat{s} \in \mathbb{R}^n$ of the respective signal by solving the following equation for $\widehat{s}$.

$$\mathcal{F}^{-1}\widehat{s} = s$$

By definition of the zeta function and η, F^{-1} is lower triangular. Thus we can solve the linear system of equations in $\mathcal{O}(n^2)$.

4.5 Computational Complexities

We summarize the computational complexities derived in this chapter:

Algorithm	**Runtime**
Computing the vertices of the cover graph $\mathcal{V}$	$\mathcal{O}(n)$
Computing the edges of the cover graph $\mathcal{E}$	$\mathcal{O}(n^2)$
Computing the inverse Fourier transform $\mathcal{F}^{-1}$	$\mathcal{O}(nm)$
Computing the Fourier transform $\mathcal{F}$	$\mathcal{O}(n^3)$
Computing the Fourier coefficients $\widehat{s}$	$\mathcal{O}(nm + n^2)$
Computing the Fourier coefficients $\widehat{s}$ of k signals	$\mathcal{O}(nm + kn^2)$

Bounding m In common neural networks we have $\Theta(n_i) = \mathcal{O}(m_i) = \Omega(n_{i-1} \cdot n_i)$. Maximum pooling and average pooling layers are layer types for which we have $m_i = c \cdot n_i \in \mathcal{O}(n_i)$, $c \in \mathbb{N}$. c depends on the kernel size. A linear layer has exactly $n_{i-1} \cdot n_i$ edges. Ultimately, the number of edges in a neural network depends on its design.

Example 4.6 *Let us define a neural network of one single linear layer which takes $x \in \mathbb{R}^{\frac{n}{2}}$ as input and produces output $y \in \mathbb{R}^{\frac{n}{2}}$. By definition of a linear layer, the neural network has $\frac{n}{2} \cdot \frac{n}{2} = \mathcal{O}(n^2)$ edges.*

For this reason, we use $m \in \mathcal{O}(n^2)$.

Chapter 5

Analysis

This chapter is concerned with the analysis of the data that we retrieved. We will start by verifying the numerical precision of the data. Second, we examine the transform matrices and their structure in neural networks. Later we visualize the activation patterns and their corresponding Fourier coefficients. Finally, we use k-means clustering to search for patterns in the Fourier-domain.

Recall, that η defines the total order of the poset elements. The total order defined here is ascending by layer, channel, row, and column.

Example 5.1 *In Figure 2.9, this corresponds to* $\eta_{x_{4,4,4}} \leq \eta_{y_{1,1,1}} \leq \eta_{y_{1,1,2}} \cdots \leq \eta_{y_{1,2,1}} \cdots \leq \eta_{y_{2,1,1}}$. *Note that x is in layer* $i-1$ *and y in layer i.*

When we visualize signals in $\mathbb{R}^n$, we reshape them into the dimensions of the original activation pattern. This way, we can associate every signal value with its position in the neural network. We display layers and channels as separate matrices.

5.1 Numerical Analysis

Before we start inspecting the generated data, we assess its numerical precision. In order to do so, we compute an approximation $\widetilde{s}$ of a signal s as

$$\widetilde{s} := \mathcal{F}^{-1}\widehat{s}\,, \tag{5.1}$$

where $\widehat{s}$ is precomputed by solving the lower triangular linear system $\mathcal{F}^{-1}\widehat{s} = s$. We then calculate the absolute and relative errors.

Definition 5.2 (Errors) *For vector* $s \in \mathbb{R}^n$ *and approximation* $\widetilde{s} \in \mathbb{R}^n$

the ***absolute error*** *is defined as*

$$\varepsilon_{abs} := |s - \widetilde{s}| \tag{5.2}$$

			standard			column normalization			row normalization		
Network	n	m	$cond_2(\mathcal{F})$	$\|\varepsilon_{abs}\|_\infty$	$\|\varepsilon_{rel}\|_\infty$	$cond_2(\mathcal{F})$	$\|\varepsilon_{abs}\|_\infty$	$\|\varepsilon_{rel}\|_\infty$	$cond_2(\mathcal{F})$	$\|\varepsilon_{abs}\|_\infty$	$\|\varepsilon_{rel}\|_\infty$
1	3098	80640	$1 \cdot 10^{+06}$	$0 \cdot 10^{+00}$	$0 \cdot 10^{+00}$	$3 \cdot 10^{+05}$	$1 \cdot 10^{-12}$	$6 \cdot 10^{-10}$	$6 \cdot 10^{+04}$	$1 \cdot 10^{-11}$	$1 \cdot 10^{-08}$
2	8506	237600	$5 \cdot 10^{+07}$	$0 \cdot 10^{+00}$	$0 \cdot 10^{+00}$	$1 \cdot 10^{+07}$	$1 \cdot 10^{-10}$	$5 \cdot 10^{-08}$	$1 \cdot 10^{+06}$	$1 \cdot 10^{-10}$	$8 \cdot 10^{-08}$
3	9594	474240	$5 \cdot 10^{+08}$	$1 \cdot 10^{-07}$	$4 \cdot 10^{-05}$	$2 \cdot 10^{+07}$	$1 \cdot 10^{-07}$	$2 \cdot 10^{-05}$	$1 \cdot 10^{+07}$	$2 \cdot 10^{-10}$	$1 \cdot 10^{-07}$
4	9834	535840	$1 \cdot 10^{+13}$	$4 \cdot 10^{-05}$	$1 \cdot 10^{-01}$	$5 \cdot 10^{+11}$	$3 \cdot 10^{-05}$	$2 \cdot 10^{-01}$	$2 \cdot 10^{+11}$	$4 \cdot 10^{-07}$	$1 \cdot 10^{-03}$
5	11166	658024	$1 \cdot 10^{+13}$	$3 \cdot 10^{-05}$	$1 \cdot 10^{-02}$	$5 \cdot 10^{+11}$	$3 \cdot 10^{-05}$	$1 \cdot 10^{-02}$	$1 \cdot 10^{+11}$	$7 \cdot 10^{-07}$	$1 \cdot 10^{-04}$
6	23338	1802400	$3 \cdot 10^{+13}$	$9 \cdot 10^{-05}$	$1 \cdot 10^{+00}$	$1 \cdot 10^{+12}$	$9 \cdot 10^{-05}$	$1 \cdot 10^{+00}$	$6 \cdot 10^{+11}$	$4 \cdot 10^{-07}$	$5 \cdot 10^{-04}$
7	30282	2579264		$5 \cdot 10^{-05}$	$1 \cdot 10^{+04}$		$5 \cdot 10^{-05}$	$1 \cdot 10^{+04}$		$3 \cdot 10^{-07}$	$1 \cdot 10^{+02}$

Table 5.1: Condition number and maximum errors of 1000 samples on networks with n vertices and m edges.

*and the **relative error** is defined as*

$$\varepsilon_{rel} := \frac{|s - \widetilde{s}|}{|s|} \tag{5.3}$$

[6].

The relative error estimates the number of correct digits in an approximation $\widetilde{s}$ of s. If

$$\varepsilon_{rel} := \frac{|s - \widetilde{s}|}{|s|} \leq 10^{-k} \tag{5.4}$$

, then $\widetilde{s}$ has k correct digits, $k \in \mathbb{N}$ [6].

The *sensitivity* of a linear map $s \mapsto \mathcal{F}^{-1}s =: \widehat{s}$, evaluates the impact of small perturbations in s on the result $\widehat{s}$. The condition number quantifies this sensitivity.

Definition 5.3 (Condition number) *The **condition number** of a matrix $A \in \mathbb{R}^{n \times n}$ on the Euclidean norm is defined as*

$$cond_2(A) := \|A^{-1}\|_2 \|A\|_2 \,. \tag{5.5}$$

If $cond_2(A) \gg 1$, then small relative changes of data A or b may effect huge relative changes in $b \mapsto A^{-1}b =: \widehat{b}$ [6].

Table 5.1 shows the maximum errors $\|\varepsilon_{abs}\|_\infty$ and $\|\varepsilon_{rel}\|_\infty$ of 1000 random samples for each of our networks. For those networks trained on the same dataset, we used the same set of random samples. The errors of the data are severe. We experience relative errors of up to 4 orders of magnitude. Normalizing the columns or rows of the transform matrix $\mathcal{F}^{-1}$, mitigates the condition number of the linear system. In the case of row normalization this leads to lower absolute and relative errors.

We suspect that the errors first arise when solving $\mathcal{F}^{-1}\widehat{s} = s$ for $\widehat{s}$. In this process, rows of $\mathcal{F}^{-1}$ and components in s are multiplied by constants and later subtracted off each other. This can lead to cancellation, i.e. subtraction of almost equal numbers in s and an extreme amplification of relative errors

[6]. The second step (5.1) accumulates these errors. A component $\widetilde{s}_x$ sums up all errors in $\widehat{s}_y$ where $y \leq x$ in the partial order. The latter suggests that errors get larger towards deeper layers. In general, the deeper the neuron x lies in the neural network, the more causes y, with $y \leq x$ it has.

Figure 5.1 plots $|s_x - \widetilde{s}_x|$ for each neuron x separately. We see that the errors in Table 5.1 are largely caused by the neurons towards the end of the networks. Furthermore, we observe a step-wise increase along the horizontal axis η. These steps are related to the partial order of the neurons in the poset. In general, the larger the set of causes $|\{y \,|y \ \leq x\}|$, the larger is the error $|s_x - \widetilde{s}_x|$. Note, the total order η of the poset elements is defined on top of the partial order. Any valid total order η of the poset elements would yield a similar step-wise increase of the errors in Figure 5.1.

Moreover, the plot may explain why row normalization exhibits lower numerical errors than column normalization. Row normalization divides the rows of a matrix by their norm. The row norm of $F^{-1}_{x,:}$ is

$$\|F^{-1}_{x,:}\|_2 = \sqrt{|\{y \,|y \ \leq x\}|}. \tag{5.6}$$

If a neuron x has many causes, the row of $\mathcal{F}^{-1}$ will be divided by a large number. When (5.1) is computed for a component $\widetilde{s}_x$ it accumulates all errors in $\widehat{s}_y$, where $y \leq x$, *divided* by the row norm. This reduces the error in deep layers, where neurons have a large set of causes.

Column normalization divides the columns of a matrix by their norm. The column norm of $F^{-1}_{:,x}$ is

$$\|F^{-1}_{:,x}\|_2 = \sqrt{|\{z \,|x \ \leq z\}|}. \tag{5.7}$$

When (5.1) is computed for a component $\widetilde{s}_x$, not all $\widehat{s}_y$, where $y \leq x$, are divided by a large norm. $\widehat{s}_y$ is divided by the number of causes that y has. This number is small for neurons that lie in deeper layers. Figure 5.2 of the following section 5.2 helps to illustrate this numerical analysis.

5.2 Transform Matrices $\mathcal{F}$ and $\mathcal{F}^{-1}$

Different causal graphs yield different transform matrices. In this section we observe the structures of the transform matrices $\mathcal{F}$ and $\mathcal{F}^{-1}$ of CNNs.

The networks that we analysed throughout this project have up to 2^{15} neurons. To illustrate some of our insights on A4-paper, we create a network with just 71 neurons. Its specifications are listed in Table 5.2.

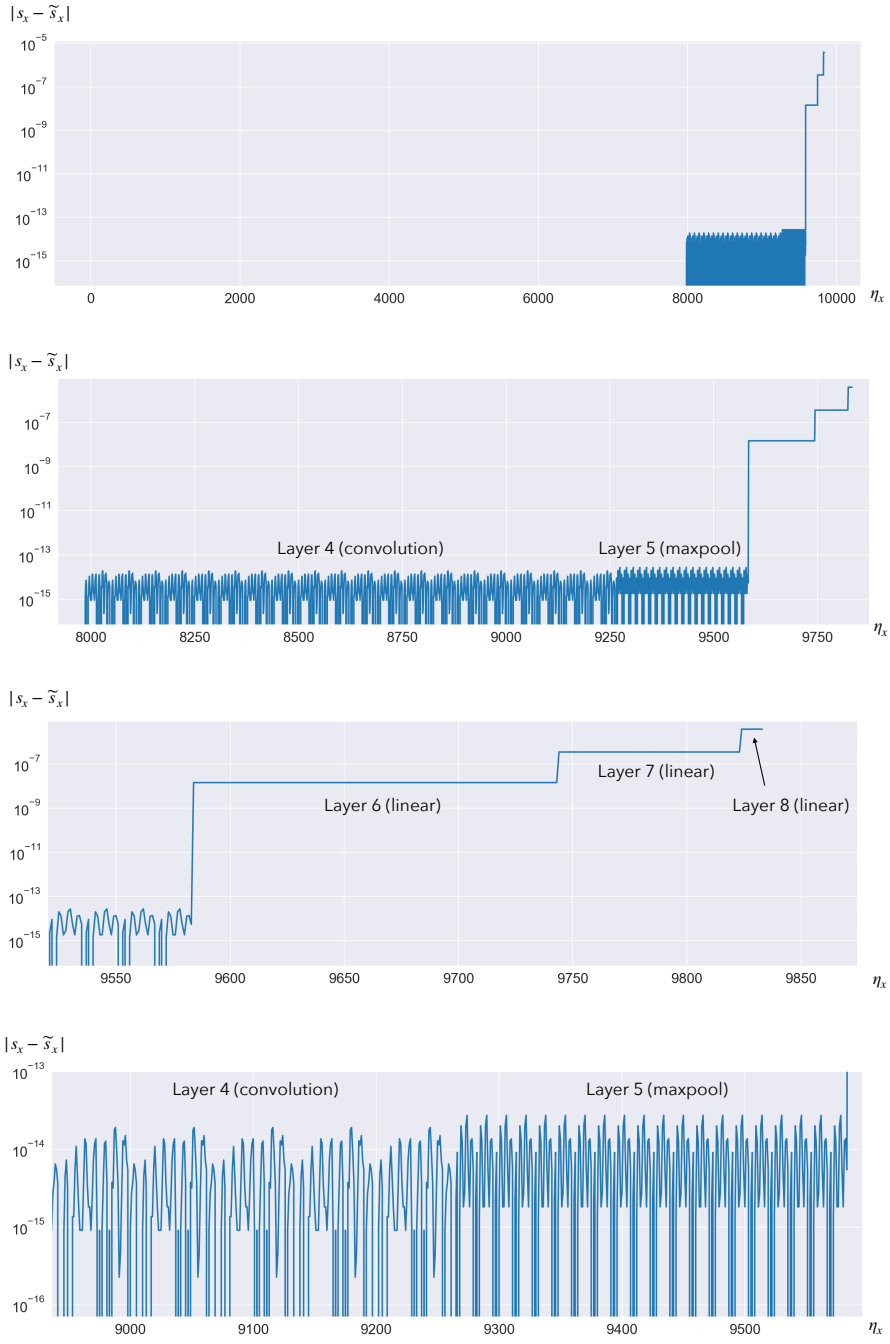


Figure 5.1: Maximum $|s_x - \widetilde{s}_x|$ of $N = 1000$ samples in Network 4. The horizontal axis is given in η_x. The magnitude of the error is closely related to the deepness of the neuron x in the neural network. In other words, to its number of causes.

Layer	Type	$\mathcal{V}_i$	Input width	Input height	Output width	Output height	Input channels	Output channels	Kernel width	Kernel height
1	Input	25			5	5		1		
2	Conv.	18	5	5	3	3	1	2	3	3
3	Conv.	16	3	3	2	2	2	4	2	2
4	Linear	8	16		8					
5	Linear	4	8		4					

Table 5.2: Network specifications

5.2.1 Inverse Fourier Transform Matrix $\mathcal{F}^{-1}$

Recall the definition of $\mathcal{F}^{-1}$. A row $\mathcal{F}^{-1}_{x,:}$ encodes all causes of x. A column $\mathcal{F}^{-1}_{:,x}$ encodes all effects of x. The causal relationships of neurons are defined by their respective layer types. Plotting the inverse Fourier transform matrices of our CNNs, we found very characteristic and recurring structures.

Figure 5.2 plots these typical patterns. The illustration is best understood by self-verifying the annotations. We summarize some key observations:

- $\mathcal{F}^{-1}$ can be clearly divided into its ℓ layers, both horizontally and vertically.
- The block matrices along the diagonal corresponding to one layer i, are identity matrices of dimension $n_i \times n_i$. This is the case, since there are no edges between two neurons in one layer.
- A convolutional operation in layer i is identified by the steps in layer $i-1$. We can find very detailed information of a layers specification such as the kernel width and height, the number of incoming and outgoing channels or input and output dimensions.
- The pattern corresponding to the indirect causes of a layer i, i.e. the neurons that lie in layers $\{1,\dots,i-2\}$ can be inferred using equation (4.8).
- A neuron in a linear layer i has edges to all neurons in $\mathcal{V}_{i-1}$. Every neuron in layers $\{1,\dots,i-2\}$ has at least one effect in layer $i-1$. Hence the causes of a neuron in layer i is the union of all $\mathcal{V}_j$ for $j \in \{0,\cdots,i-1\}$.

5.2.2 Fourier Transform Matrix $\mathcal{F}$

A row $\mathcal{F}_{x,:}$ describes in which way a signal s is linearly combined to obtain the Fourier coefficient $\widehat{s}_x$. Unlike the inverse transform matrix, the values

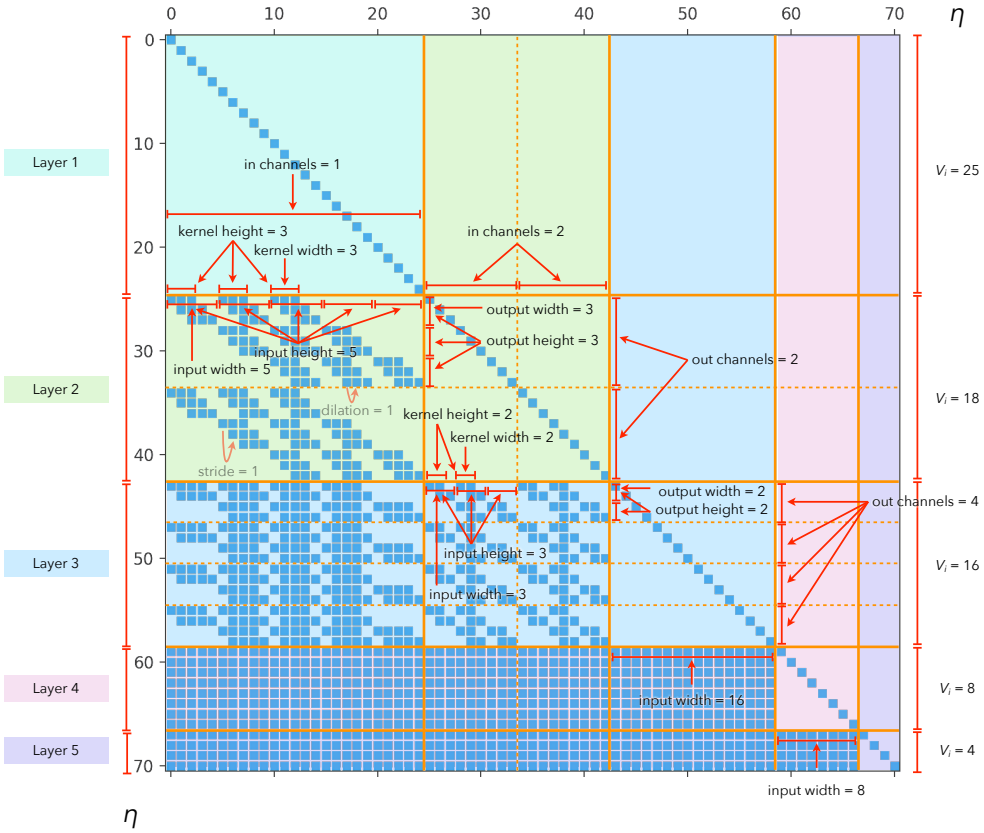


Figure 5.2: $\mathcal{F}^{-1} \in \{0,1\}^{71\times 71}$ of the CNN specified in Table 5.2. The blue squares indicate the value 1 in $\mathcal{F}^{-1}$. The total order η on neurons is defined ascending by layer, channel, row, and column in the activation pattern.

in $\mathcal{F}$ are integer values. Since the Moebius function is a recursively defined function and not explicit, it is not easy to guess which values we will find in $\mathcal{F}$.

Expanding the definition of $\mathcal{F}$ will help to reason about later observations:

$$\mathcal{F}_{x,y} = \begin{cases} 1 & \text{if } x = y\,, \\ -\sum_{y \le z < x} \mu(y,z) & \text{otherwise.} \end{cases} \tag{5.8}$$

$$= \begin{cases} 0 & \text{if } x \not\le y\,, \\ 1 & \text{if } x = y\,, \\ -1 & \text{if } (y,x) \in \mathcal{E}\,, \\ -\sum_{\substack{y \le w \le z,\\ (z,x)\in\mathcal{E}}} \mu(y,w) & \text{otherwise.} \end{cases} \tag{5.9}$$

Figure 5.3 and 5.4 illustrate the cases of equation (5.9).

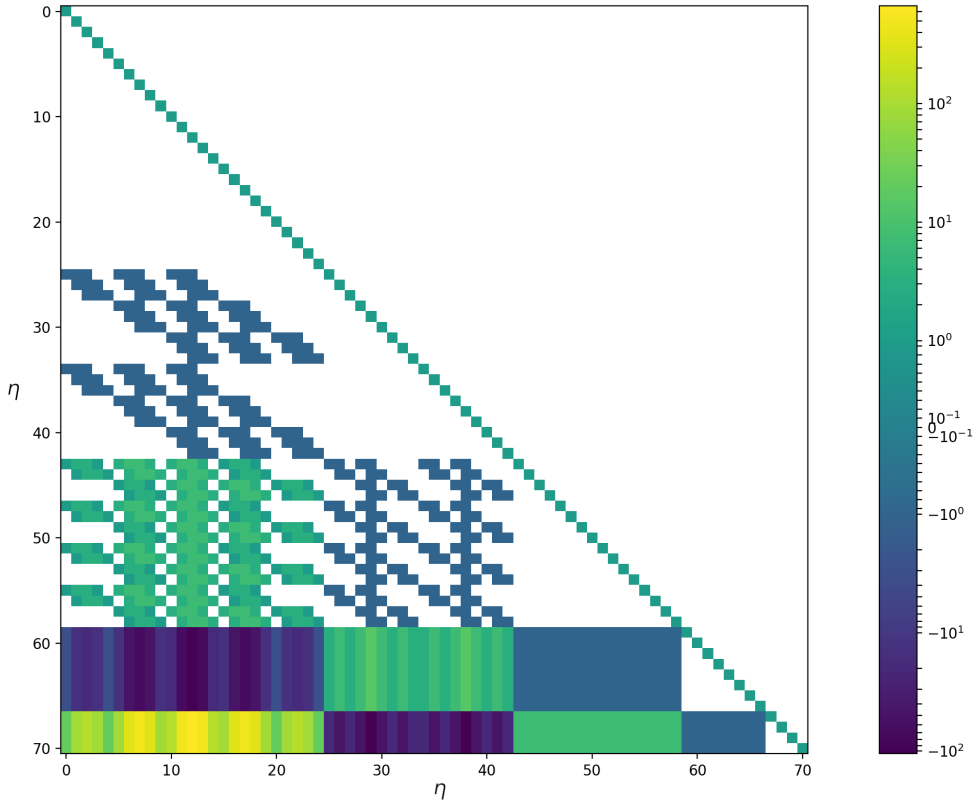


Figure 5.3: $\mathcal{F}$ of the CNN specified in Table 5.2.

We see, $\mathcal{F} = 0$ whenever $\mathcal{F}^{-1} = 0$, which corresponds to the first case in equation (5.9). Moreover for all values along the diagonal we have $\mathcal{F}_{x,x} = 1$. Given a neuron $x \in \mathcal{V}_i$ in layer i and a neuron $y \in \mathcal{V}_{i-1}$, we see that entries $\mathcal{F}^{-1}_{x,y} = -1$ if $(y, x) \in \mathcal{E}$. The last case of equation (5.9) explains the highly repetitive pattern of values within one layer in $\mathcal{F}$. It shows that $\mathcal{F}_{x,y}$ only depends on y and the direct causes of x, namely $\{z \mid (z, x) \in \mathcal{E}\}$.

Linear layer Any neuron $x \in \mathcal{V}_i$, with i being a linear layer, has incoming edges from all neurons in $\mathcal{V}_{i-1}$. As a consequence all $x \in \mathcal{V}_i$ have the same set of direct causes, i.e. $\{z \mid (z, x) \in \mathcal{E}\}$ is the same set for any $x \in \mathcal{V}$.

Convolutional layer By definition of the convolutional layer i, any neuron x at position $[c, h, w]$ in the activation pattern has the same set of direct causes as neuron x' at position $[c', h, w]$, where c and c' are any two output channels and h and w are fixed. Formally, $\{z \mid (z, x) \in \mathcal{E}\} = \{z \mid (z, x') \in \mathcal{E}\}$.

Therefore, it is sufficient to compute the entries of F for one row of a lin-

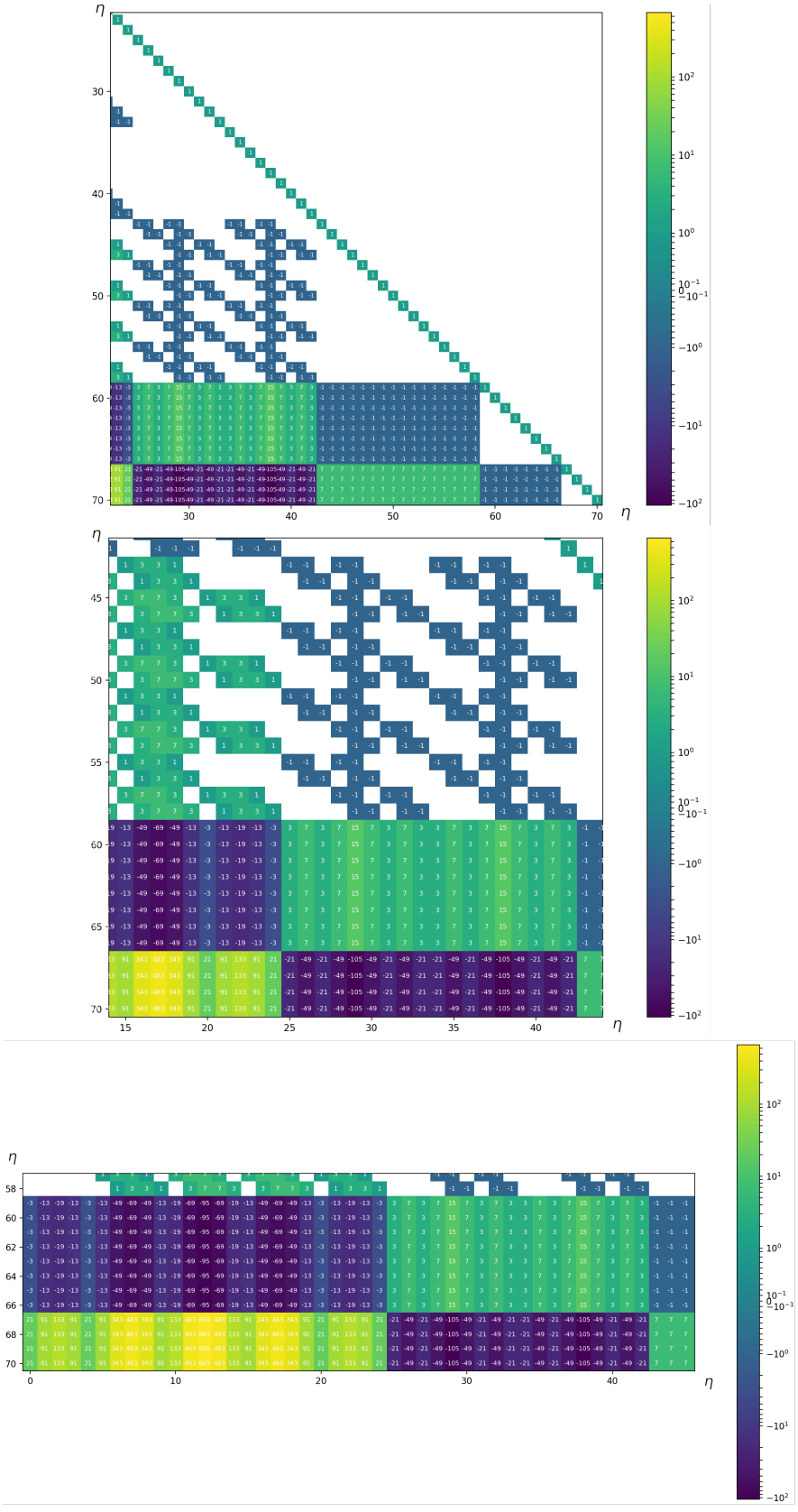


Figure 5.4: Repetitive patterns in $\mathcal{F}$

ear layer to infer the entries of all other rows associated with this layer i. Likewise, we can compute the rows associated with one channel of a convolutional layer to infer the entries of all other channels.

Figure 5.4 shows that there are in fact many more repetitive patterns found in the Fourier Transform of a CNN. Such patterns can be exploited to decrease runtimes.

5.3 Signals s and Fourier Coefficients $\widehat{s}$

CNNs trained on image data have the nice property that their features maps are well-visualizable. In this section, we look at the signals s and their coefficients $\widehat{s}$. Figures 5.5 and 5.6 show sample activation patterns of Network 3 and 7, respectively. The corresponding Fourier coefficients are shown in Figures 5.7 and 5.8.

5.3.1 Signal s

The activation patterns generated by the Networks are typical. Especially in the initial layers, we see that the feature maps learned to detect very simple, low-level characteristics of the input image such as lines and edges. The consecutive layers build on top of these simple features and construct more complicated ones. The deeper we go, with respect to layers, the more abstract the features get.

In Figure 5.5, the first two channels (top left) of Layer 1 clearly learned to detect horizontal edges while channels 3 and 4 (top right) detect vertical edges. In Figure 5.6 the entirely white channels correspond to so-called dead neurons, which we addressed in Section 4.1.2. If all incoming edges of a neuron are dead, i.e. their weight is close to 0, the neuron does not receive any input and is never activated. The large number of dead channels in Figure 5.6 attributes to the excess amount of channels. In the third layer of Figure 5.6, we see that some channels, retrieve the plaid texture of the shirt while channels 1, 10 and 12 predominantly extract vertical lines. Channel 15 in layer 4 seems to mask the object of the input image.

Towards the end of neural networks feature detectors become very complicated and non-human interpretable. While the Network in Figure 5.5 correctly classifies the input images in the output layer as *1*, *4*, and *7*, it is difficult to visually understand how it obtains this information, given layer 5.

5.3.2 Fourier Coefficients $\widehat{s}$

In contrast to the signal, we immediately observe that the channels within one layer in the Fourier-domain look extremely similar.

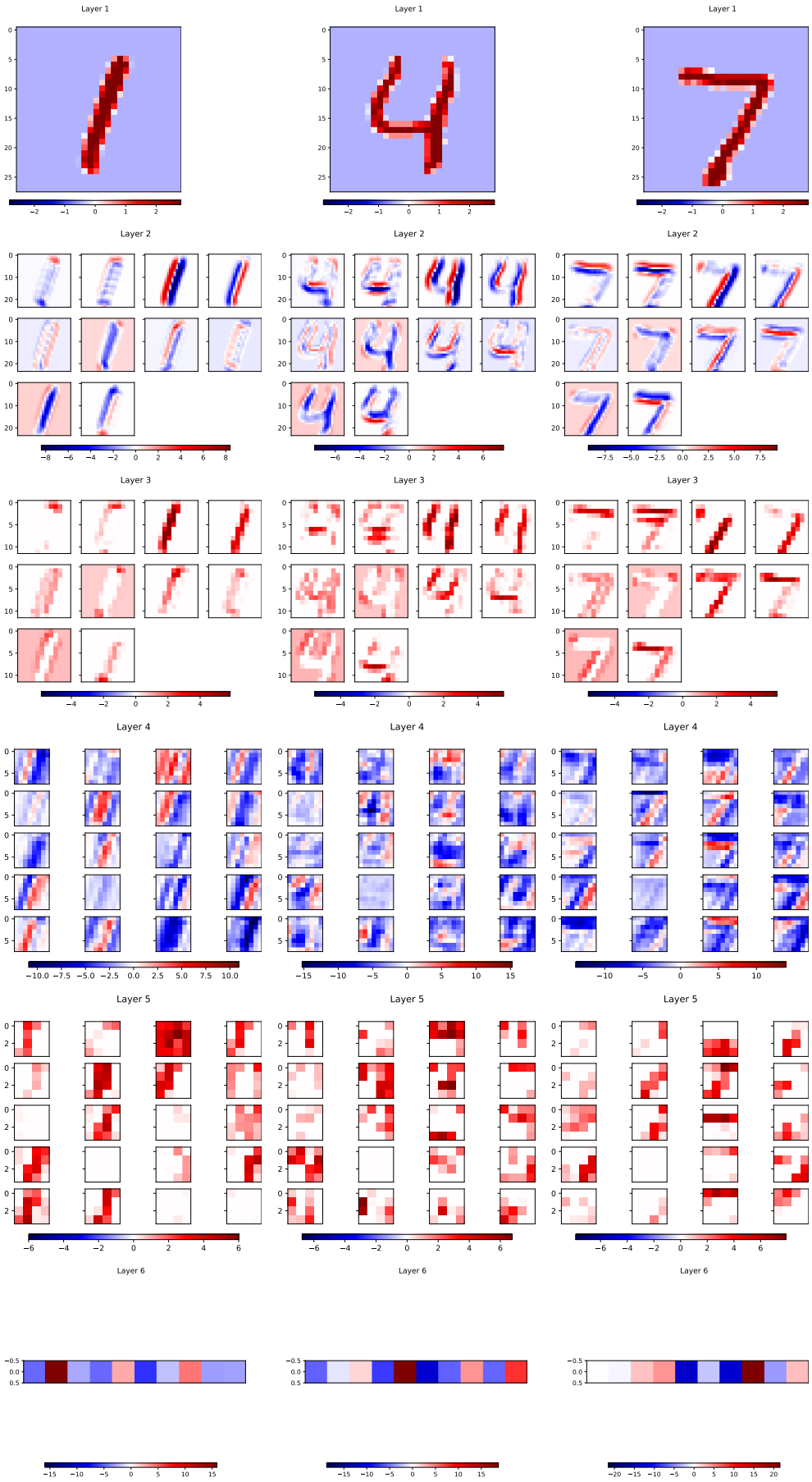


Figure 5.5: Three activation patterns on Network 3. The classification given by the Network is the class of the maximal value in layer 6.

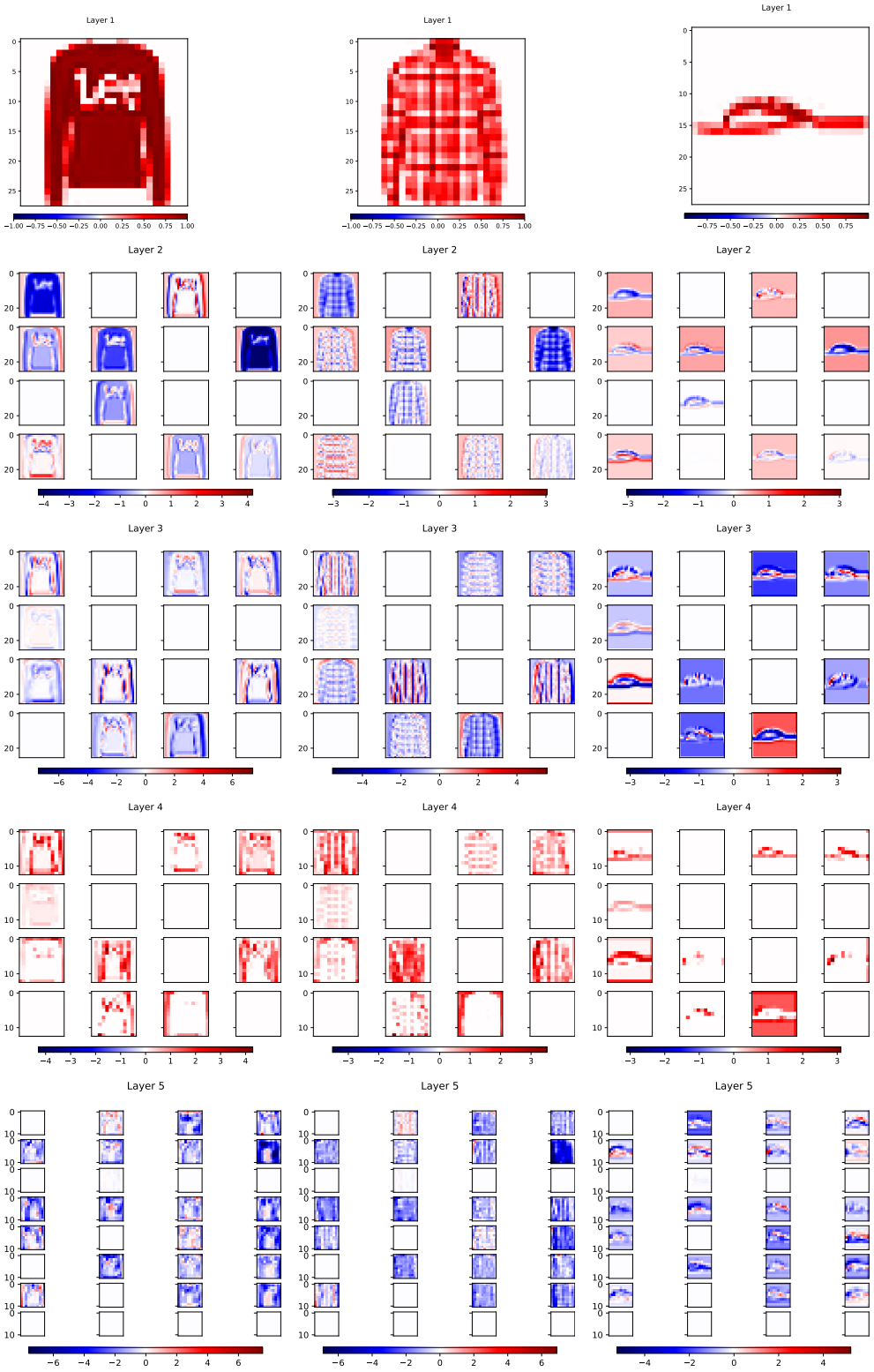


Figure 5.6: Three activation patterns on Network 7

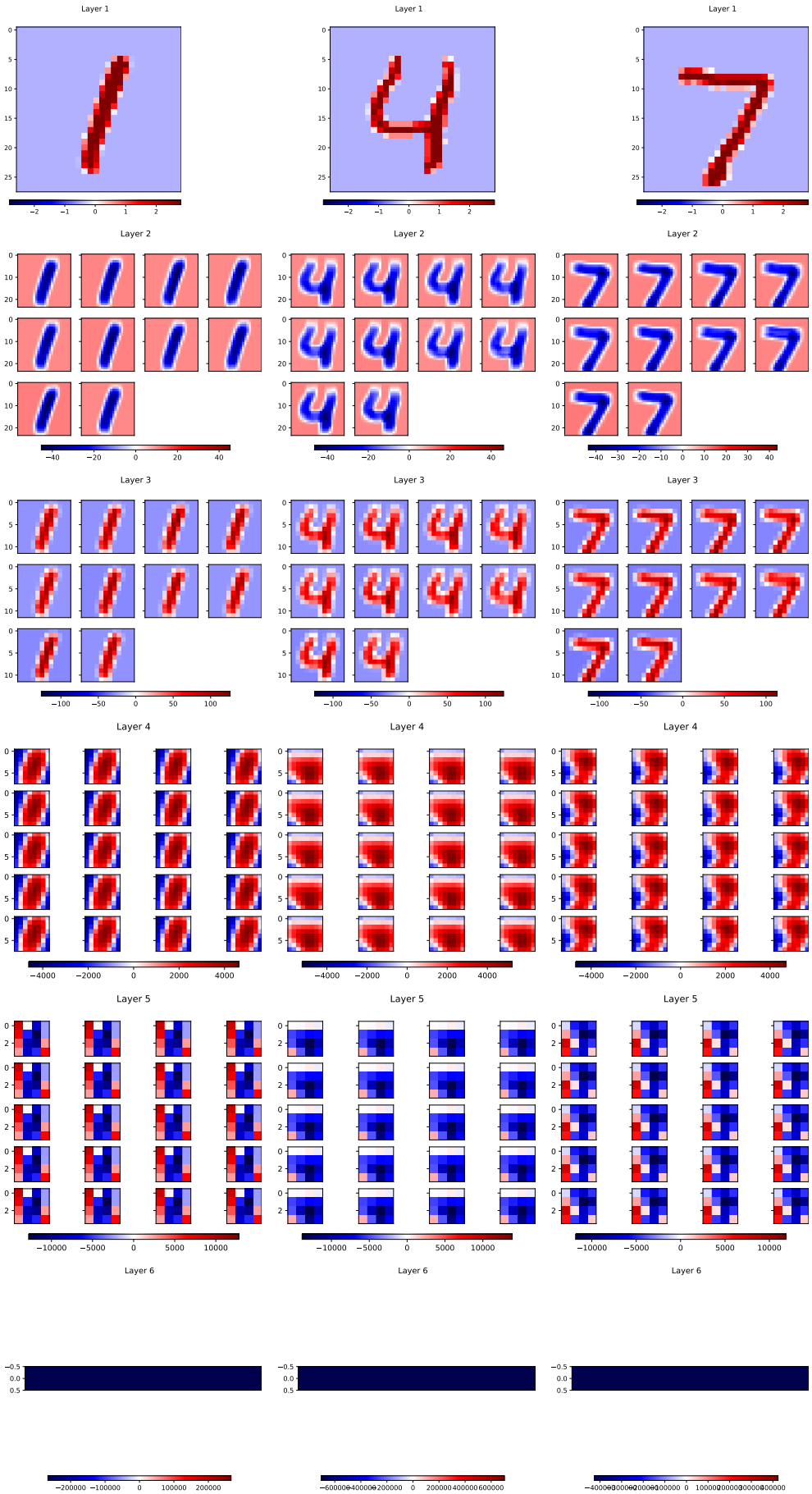


Figure 5.7: Fourier signals $\widehat{s}$ corresponding to signals s in Figure 5.5.

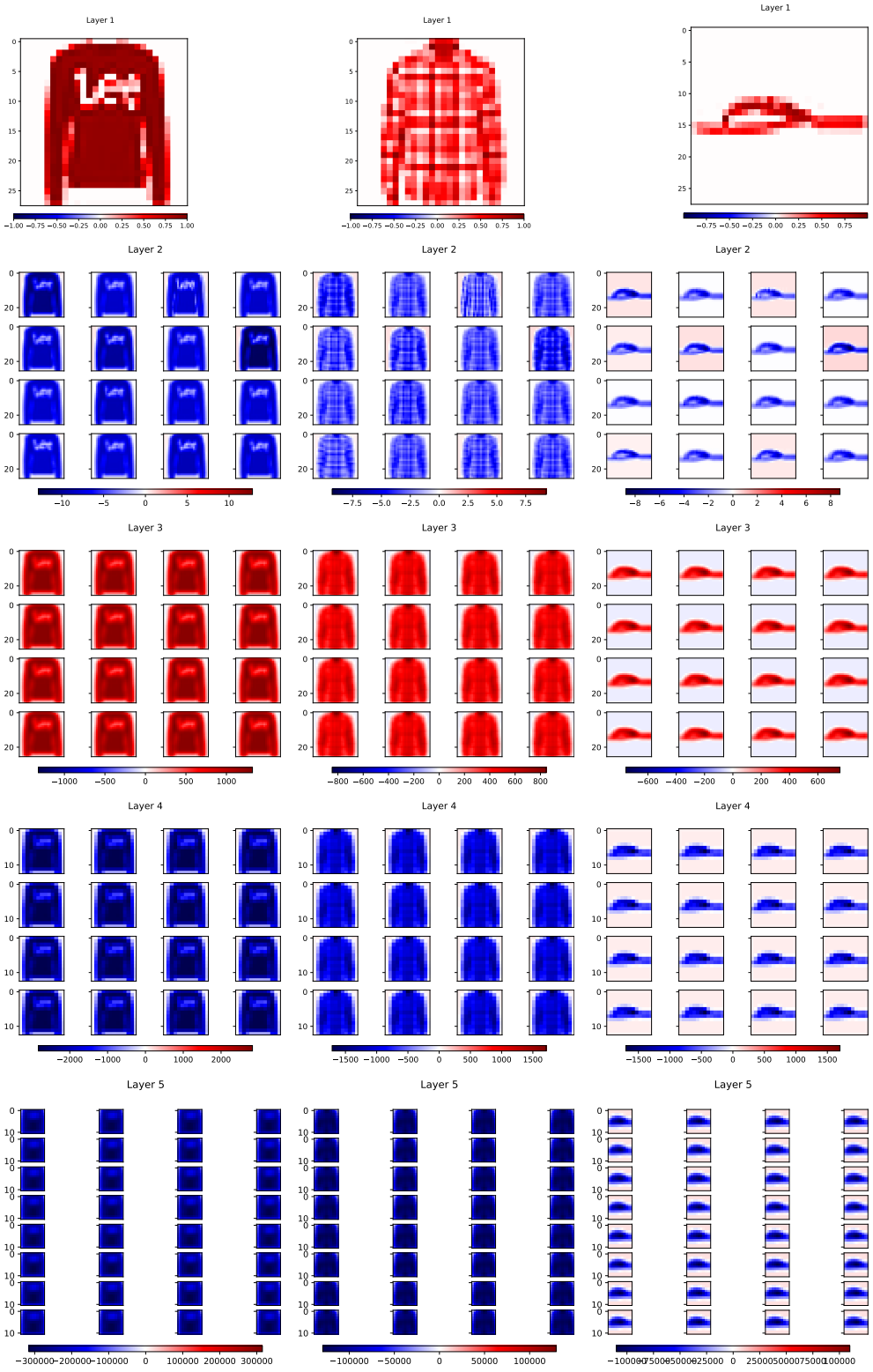


Figure 5.8: Fourier signals $\widehat{s}$ corresponding to signals s in Figure 5.6. Layers 6-8 not shown.

All input neurons, which are found in layer 1, are unchanged after the transformation. This is as expected. In the poset, the inputs are the root causes of all other events, hence their signal values are not dependent on others. Formally, we get:

$$\begin{aligned}
\widehat{s}_x &= \mathcal{F}_{x,:} s && (5.10)\\
&= \sum_{y \leq x,\ y \in \mathcal{V}} \mu(y,x) s_y && (5.11)\\
&= \sum_{y < x,\ y \in \mathcal{V}} \mu(y,x) s_y + \mu(x,x) s_x && (5.12)\\
&= s_x && (5.13)
\end{aligned}$$

Subsequent layers look like blurred versions of their previous layer transferred to a different scale. While we have a slight perceptual channel difference in low levels[1], deeper down the layers all channels appear the same. The same applies to all coefficients in the linear layers. We justify this strong similarity between the channels as follows:

$$\begin{aligned}
\widehat{s}_x &= \mathcal{F}_{x,:} \cdot s && (5.14)\\
&= \sum_{y \leq x,\ y \in \mathcal{V}} \mu(y,x) \cdot s_y && (5.15)\\
&= \sum_{y < x,\ y \in \mathcal{V}} \mu(y,x) \cdot s_y + \mu(x,x) \cdot s_x && (5.16)\\
&= \underbrace{\sum_{y < x,\ y \in \mathcal{V}} \left(- \sum_{y \leq z < x} \mu(z,y) \right) \cdot s_y}_{(5.17.1)} + \underbrace{s_x}_{(5.17.2)} && (5.17)
\end{aligned}$$

We see that equation (5.17.1) only depends on causes of x, not x itself. Hence, if causes of poset elements x and x' majorly overlap and the value of (5.17.1) $\gg$ (5.17.2) then $s_x \approx s_{x'}$.

In section 5.2 we addressed the overlap of causes in convolutional and linear layers. For convolutional layer i this suggests that the subtraction of two channels $[c,:,:]$ and $[c',:,:]$ in the Fourier-domain yields the same result as their subtraction in the poset-domain as equation (5.17.1) cancels out:

$$\widehat{s}_{[c,:,:]} - \widehat{s}_{[c',:,:]} = s_{[c,:,:]} - s_{[c',:,:]} \tag{5.18}$$

Indeed we can verify equation (5.18) empirically, as seen in Figure 5.9. Analogously, this applies to any two neurons in a linear layer.

[1] or none if you are reading this on paper

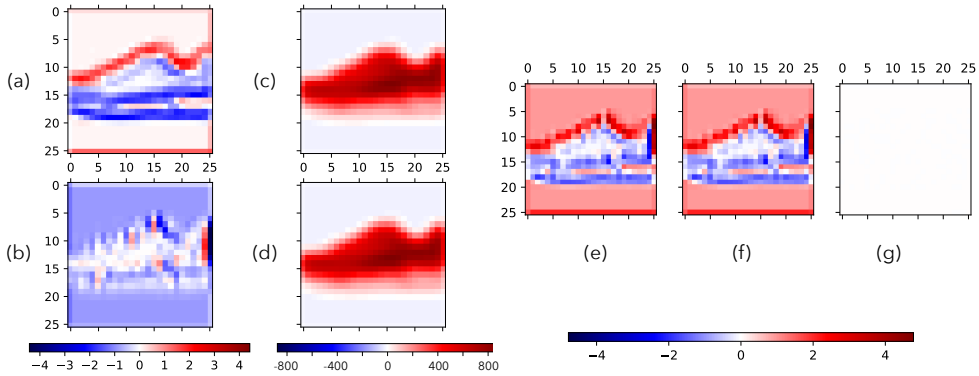


Figure 5.9: Visualization of equation (5.18) for the convolutional layer $i = 3$ in Network 6. (a) and (b) are two channels in s, (c) and (d) correspond to the same two channels in $\widehat{s}$. (e) = (a) - (b) and (f) = (c) - (d). The subtraction of (g) = (e) - (f) has entries 0, i.e. (e) is equal to (f).

5.4 Clustering

We want to find out whether the Fourier coefficients $\widehat{s}$ belonging to images of the same class resemble each other. Is there a pattern in the coefficients such that we can identify the class of the input image?

In order to find out, we cluster the data using the unsupervised learning technique K-means clustering. K-means partitions a sample set of N discrete oberservations $x \in \mathbb{R}^d$ into K groups, so-called clusters. Formally:

Definition 5.4 (K-Means) *Assume data points $x \in \mathbb{R}^d$ are in the Euclidean space. The clusters are represented as centers $\sigma_k \in \mathbb{R}^d$. Each data point x is assigned to the closest center $\sigma_k \in \{\sigma_1, \ldots, \sigma_K\}$. The goal is to pick centers $\widetilde{\sigma}$ that minimize the cost function $\mathcal{R}$ calculating the squared distance of x to its assigned center:*

$$\mathcal{R}(\sigma) = \mathcal{R}(\sigma_1, \ldots, \sigma_K) = \sum_{i=1}^{N} \min_{k \in K\{1,\ldots,K\}} \|x_i - \sigma_k\|_2^2 \tag{5.19}$$

$$\widetilde{\sigma} = \arg\min_{\sigma} \mathcal{R}(\sigma) \tag{5.20}$$

[9]

Given data with labels, one can assess how well the clusters represent the data classes using the V-Measure score introduced in [19].

Definition 5.5 (V-Measure) *The V-Measure is defined as*

$$\mathbf{v} := \frac{2 \times \textit{homogeneity} \times \textit{completeness}}{\textit{homogeneity} + \textit{completeness}} \tag{5.21}$$

where

> ***homogeneity*** $\in [0,1]$ *scores 1 if each cluster only contains data points of a single class and*
>
> ***completeness*** $\in [0,1]$ *scores 1 if all data points of a given class are assigned to the same cluster [15].*

The experiment setup is as follows: We choose $K = 10$, hoping for the data to be partitioned into the 10 classes of our image datasets. We then take the Fourier coefficients $\widehat{s}$ of $N = 1000$ evenly distributed samples and cluster them in the following trials: We

- cluster all coefficients $\widehat{s} = x \in \mathbb{R}^n$.
- cluster all coefficients except the ones belonging to the output layer $\widehat{s}_{[1:\, n-n_l]} = x \in \mathbb{R}^{n-n_l}$.
- cluster all coefficients belonging to exactly one layer $x \in \mathbb{R}^{n_i}$, $i \in \{1, \ldots, l\}$.

To obtain a bench mark, we perform the same clustering on the signal s and $\widehat{s}_{lap}$ and compare the performances.

By $\widehat{s}_{lap}$ we denote the Fourier coefficients of a different graph Fourier transform proposed in [20]. It uses the graph Laplacian matrix Ł as graph shift and its eigenvectors as Fourier basis. In the appendix A, we shortly summarize the transform defined in [20]. It makes use of the *undirected* Graph representation. In our case this translates to $\mathcal{G}_{lap} = (\mathcal{V}, \mathcal{E}_{lap})$ with $\mathcal{E}_{lap} = \{\{u, v\} |\ (u,\ v) \in \mathcal{E}\}$.

Evaluation

Clustering the data for various CNNs we received similar results. In Figure 5.10 we plot the results of Network 4.

First, we observe that the clustering of all activations is at least as precise as the clustering of all activations excluding the last layer. This is the case for s, $\widehat{s}$, and $\widehat{s}_{lap}$. The difference of a data point $x \in \mathbb{R}^n$ and $x' \in \mathbb{R}^{n-n_\ell}$ is that the latter does not contain the values associated with the classification computed by the network.

The clustering of an entire signal s scores lower than the clustering of its last layer 8. The unsupervised clustering algorithm at hand is unable to assign the centers such that the data points $s \in \mathbb{R}^n$ are separated according to the output class found in the last n_ℓ values of s. In other words, the centers minimizing the mean squared distances to data points s lie somewhere else.

The input layer 1 of $\widehat{s}$ performs equally well as the input layer 1 of s. This should come as no surprise as we have seen in (5.13) that their values are

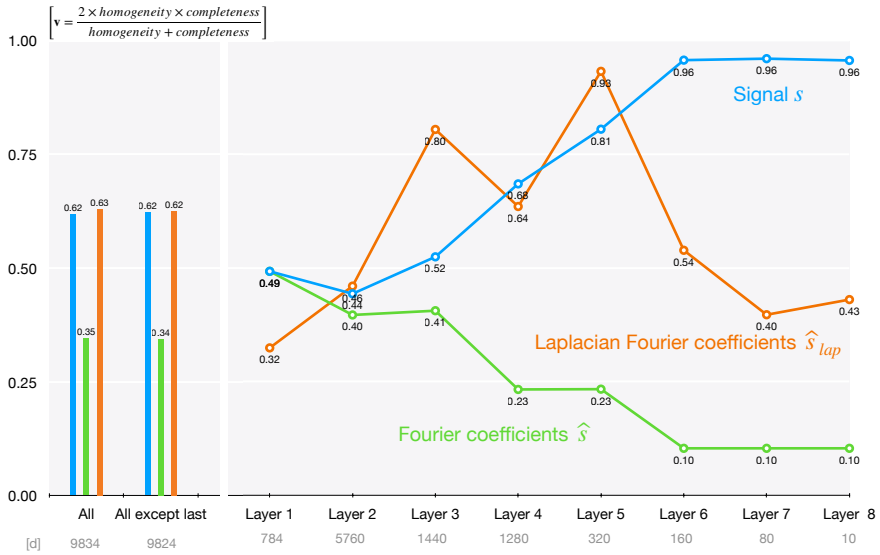


Figure 5.10: K-means Clustering on Network 4. K=10, N=1000

equal. On the signal s, the **v**-measure gradually increases going deeper into the network. In the last three layers 6-8 the clustering classifies the data as good as the neural network itself. $\widehat{s}_{lap}$ yields varying performances along the layers but outperforms s in layers 3 and 5. Meanwhile, the **v**-measure decreases for Fourier coefficients $\widehat{s}$. The drop in the performance of $\widehat{s}$ in linear layers 6-8 is significant. A score of $\mathbf{v} = 0.1$ on 10 classes is not better than random clustering. To find an explanation for this observation we investigate the behaviour formally:

Let i be a linear layer. By definition of a linear layer all its neurons have the exact same set of causes. Therefore, we can apply what we derived in equation (5.17). (5.17.1) is equal for all neurons $v \in \mathcal{V}_i$. Let $c_{s,i}$ denote the variable (5.17.1). We get:

$$\widehat{s}_v = c_{s,i} + s_v \quad \forall v \in \mathcal{V}_i \tag{5.22}$$

This constant $c_{s,i}$ dependens on layer i and s. For a fixed layer i, $c_{s,i}$ is input-specific, meaning that a different input image results in a different constant $c_{s,i}$. Moreover, the magnitude of $c_{s,i}$ can get large, especially towards deeper layers. In the linear layers of Network 4 we observed $c_{s,i} \approx 10^8$. The drop in the performance of the clustering indicates that the additional information $c_{s,i}$ causes more distortion than separation of the classes in the Euclidean space. In other words two images of a digit 5 could have very different values for $c_{s,i}$. Likewise, the numerical imprecision of the data can distort the value of $c_{s,i}$ and hence the clustering.

For the remaining section we denote the last n_ℓ values of a signal s and $\widehat{s}$ by x and $\widehat{x}$, respectively.

We just referred to $c_{s,i}$ as *additional* information for a particular reason. The predicted class for an image in our CNNs is equal to the class associated with the maximum value in x. Therefore, the following set of centers perform as good in clustering data points x as the networks prediction itself, and not any better:

$$\widetilde{\sigma}_\ell = (\sigma_1, \dots, \sigma_{n_\ell}) = (z \cdot \mathbf{e}_1, \dots, z \cdot \mathbf{e}_{n_\ell}) \,, \tag{5.23}$$

where $\mathbf{e}_i \in \mathbb{R}^n$ denotes the ith eigenvector of the standard basis and z a large constant.

Assume we center the values $\widehat{x}$ for every of the N samples *separately*. Then, the clustering algorithm, operating in the euclidean space, should perform equally well as for x:

Let $\omega_{\widehat{x}}$ denote the mean of one sample $\widehat{x}$.

$$\omega_{\widehat{x}} := \frac{\sum_{i=1}^{n_\ell} \widehat{x}_i}{n_\ell} \overset{(5.22)}{=} \frac{\sum_{i=1}^{n_\ell} (c_{s,\ell} + x_i)}{n_\ell} = c_{s,\ell} + \frac{\sum_{i=1}^{n_\ell} x_i}{n_\ell} = c_{s,\ell} + \omega_x \tag{5.24}$$

Centering this sample yields:

$$\widehat{x} - \omega_{\widehat{x}} = (c_{s,\ell} + x) - (c_{s,\ell} + \omega_x) = x - \omega_x \tag{5.25}$$

By definition of ω_x it is not larger than any x_i with $i \in \{1, \dots, n_\ell\}$. We conclude that centers $\widetilde{\sigma}_\ell$ perform as good on the sample-wise centered Fourier coefficients $\widehat{x}$ as on x.

Likewise, we can argue that a clustering algorithm, that clusters by difference in the values $\widehat{x}$ within one sample should yield the same performance as x. Similarly, this holds for a difference in channels if x is a convolutional layer. We studied the justification for this statement in section 5.3.2, in particular, equation (5.18) and Figure 5.9.

Chapter 6

Conclusion

In relation to signal processing on classical domains, graphs are often referred to as irregular data-domains [20]. In CSP, neural networks exhibit highly repetitive patterns. The, yet 'irregular data-domain', possesses many recurring structures in the causal relationships between the layered neurons.

Numerical inaccuracies have complicated the analysis of the data. The transform matrices of CNNs are ill-conditioned. By normalizing the Fourier transform we were able to lift the numerical precision. Nevertheless, a small disturbance in the activations still leads to large perturbations of the Fourier coefficients.

We suspect that no particularly pleasant structures such as sparsity, a small group of distinct coefficients, or other forms of regularity can be found in the activation patterns by CSP. In the event that such patterns do exists they are difficult to detect by the experiments conducted since the sensitivity of the Fourier coefficients to a small noise in the activations is substantial. We might require a different approach to uncover them.

Chapter 7

Acknowledgments

I thank Prof. Püschel and Chris Wendler for proposing the topic of the Bachelor thesis and supervising my work.

I thank Chris Wendler for numerous discussions on several aspects of the topic.

I thank Tommaso Pegolotti for discussions on numerical issues and providing his fpft-library [16].

I thank Vinitra Swamy for comments on earlier versions of the manuscript.

I thank Vincent Schwarz for shearing the cat on page 13.

Appendix A

Laplacian Graph Fourier Transform

We give a short summery of the Fourier transform on graphs elaborated in [20]. First we define the graph spectral representation [20]. Its eigenvectors provide the Fourier basis for the Fourier transform.

Definition A.1 (Non-Normalized Graph Laplacian) *The **non-normalized Graph Laplacian** $Ł \in \mathbb{R}^{n \times n}$ is defined as*

$$Ł := D - W \tag{A.1}$$

*where $D \in \mathbb{N}^{n \times n}$ is the diagonal **degree matrix** of an undirected graph and $W \in \mathbb{R}^{n \times n}$ its symmetric **weighted adjacency matrix**. In our directed graph representation this corresponds to*

$$D_{(v,v)} := deg^-(v) + deg^+(v) \tag{A.2}$$

$$W_{(u,v)} := \begin{cases} 1 & \text{if } (u,v) \in \mathcal{E} \text{ or } (v,u) \in \mathcal{E}, \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

for all $u, v \in \mathcal{V}$.

Since Ł is a real symmetric matrix, it has a complete set of n orthonormal eigenvectors. Let $U \in \mathbb{R}^{n \times n}$ be the matrix containing the eigenvectors as columns, i.e

$$Ł = U \Lambda U^{-1} \overset{\text{orth.}}{=} U \Lambda U^T \tag{A.4}$$

Definition A.2 (Fourier transform) *The **Laplacian Fourier transform** is now defined as as the expansion of a signal s in terms of U^T*

$$\hat{s}_{lap} := U^T s$$

*and its corresponding **inverse Laplacian Fourier transform** as*

$$s := U \hat{s}_{lap}$$

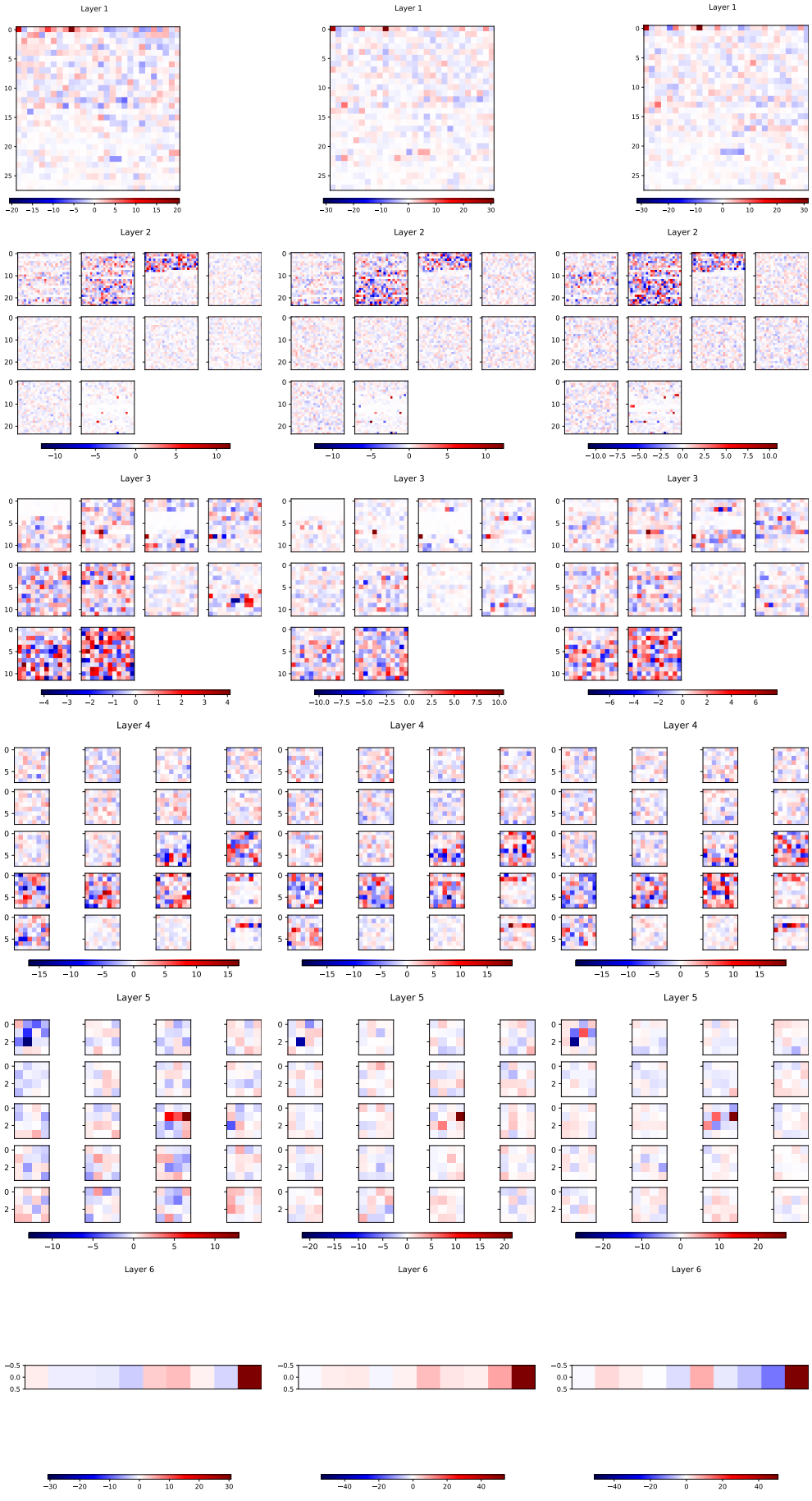


Figure A.1: Fourier signals $\widehat{s}_{lap}$ corresponding to signals s in Figure 5.5

Appendix B

Network Specifications

Network	Dataset	Nodes	Edges	Layers	Layer	Description
1	MNIST	3,098	80,640	3	1	Input
					2	Conv2d(1, 4, kernel_size=(5, 5), stride=(1, 1))
					3	Linear(in_features=2304, out_features=10, bias=True)
2	MNIST	8,506	237,600	4	1	Input
					2	Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
					3	Conv2d(8, 4, kernel_size=(3, 3), stride=(1, 1))
					4	Linear(in_features=2304, out_features=10, bias=True)
3	MNIST	9,594	550,400	6	1	Input
					2	Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
					3	MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
					4	Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
					5	MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
					6	Linear(in_features=320, out_features=10, bias=True)
4	MNIST	9,834	612,000	8	1	Input
					2	Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
					3	MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
					4	Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
					5	MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
					6	Linear(in_features=320, out_features=160, bias=True)
					7	Linear(in_features=160, out_features=80, bias=True)
					8	Linear(in_features=80, out_features=10, bias=True)
5	CIFAR	11,166	705,544	7	1	Input
					2	Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
					3	MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
					4	Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
					5	Linear(in_features=400, out_features=120, bias=True)
					6	Linear(in_features=120, out_features=84, bias=True)
					7	Linear(in_features=84, out_features=10, bias=True)
6	FashionMNIST	23,338	1,922,432	8	1	Input
					2	Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
					3	AvgPool2d(kernel_size=2, stride=2, padding=0)
					4	Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
					5	Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
					6	MaxPool2d(kernel_size=2, stride=2, padding=1, dilation=1, ceil_mode=False)
					7	Linear(in_features=1152, out_features=128, bias=True)
					8	Linear(in_features=128, out_features=10, bias=True)
7	FashionMNIST	30,282	2,579,264	8	1	Input
					2	Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
					3	Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
					4	MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
					5	Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
					6	MaxPool2d(kernel_size=2, stride=2, padding=1, dilation=1, ceil_mode=False)
					7	Linear(in_features=1152, out_features=128, bias=True)
					8	Linear(in_features=128, out_features=10, bias=True)

Table B.1: Network Specifications

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[2] Muhammad Aurangzeb Ahmad, Carly Eckert, and Ankur Teredesai. Interpretable machine learning in healthcare. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '18, page 559–560, New York, NY, USA, 2018. Association for Computing Machinery.

[3] Anonymous. A causal shift and fourier transform on directed acyclic graphs. *Submitted for publication*.

[4] Katja Grace, John Salvatier, Allan Dafoe, Baobao Zhang, and Owain Evans. When will ai exceed human performance? evidence from ai experts. *Journal of Artificial Intelligence Research*, 62:729–754, 2018.

[5] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[6] Ralf Hiptmair. Numerical methods for computational science and engineering. ETH Lecture Notes 401-0663-00L Numerical Methods for CSE, Autumn 2019.

[7] IEEE Signal Processing Society contributors. Ieee signal processing society, 2021. [Online; accessed 24-September-2021].

[8] Jean-Marie John-Mathews. Some critical and ethical perspectives on the empirical turn of ai interpretability. *Technological Forecasting and Social Change*, 174:121209, 2022.

[9] Andreas Krause. Introduction to machine learning. ETH Lecture Slides 252-0220-00L Introduction to Machine Learning, Spring 2020.

[10] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):91–100, 2016.

[11] Valerio Mante, Matthew Cook, Benjamin Grewe, Giacomo Indiveri, Daniel Kiper, and Wolfger von der Behrens. Introduction to neuroinformatics. ETH Lecture Slides 227-1037-00 S Introduction to Neuroinformatics, Autumn 2020.

[12] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[14] Judea Pearl. *Causality*. Cambridge university press, 2009.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] Tommaso Pegolotti. Fast moebius and zeta transforms. ETH Master Thesis, 2021.

[17] Markus Puschel, Bastian Seifert, and Chris Wendler. Discrete signal processing on meet/join lattices. *IEEE Transactions on Signal Processing*, 2021.

[18] Markus Puschel, Peter Widmayer, and David Steurer. Algorithmen und datenstrukturen. ETH Lecture Script 252-0026-00L Algorithmen und Datenstrukturen, Autumn 2017.

[19] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and*

computational natural language learning (EMNLP-CoNLL), pages 410–420, 2007.

[20] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[22] Ashu MG Solo. Multidimensional matrix mathematics: Notation, representation, and simplification, part 1 of 6. In *Proceedings of the world congress on engineering*, volume 3, pages 1824–1828, 2010.

[23] Angelika Steger and Emo Welzl. Algorithmen und wahrscheinlichkeit. ETH Lecture Script 252-0030-00L Algorithmen und Wahrscheinlichkeit, Spring 2018.

[24] Longwei Wang and Peijie Chen. Neurons activation visualization and information theoretic analysis. *arXiv preprint arXiv:1905.08618*, 2019.

[25] Chris Wendler, Dan Alistarh, and Markus Püschel. Powerset convolutional neural networks. *arXiv preprint arXiv:1909.02253*, 2019.

[26] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization, 2015.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.