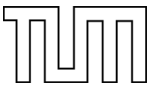


Technische Universität München

Department of Mathematics

Bachelor's Thesis

# Analyzing Deep Convolutional Networks as Gaussian Processes

Analyse tiefer Faltungsnetzwerke als Gaußprozesse

Cedrik Laue

Supervisor: Prof. Dr. Hans-Joachim Bungartz

Advisor: Dr. Felix Dietrich

Submission Date: 7/18/2022

I assure the single handed composition of this bachelor's thesis only supported by declared resources.

Garching, 7/18/2022

# Zusammenfassung

Im Laufe der letzten Jahre wurden zahlreiche Verbindungen zwischen Kernelmaschinen und neuralen Netzen entdeckt und untersucht. Vor allem die Beziehung zwischen unendlich breiten bayessche neuronalen Netzen und einer bestimmten Form von Gaußprozesses war Quelle einer Vielzahl von wissenschaftlichen Beiträgen. Diese Korrespondenz macht es möglich neuronale Netze mittels des entsprechenden Gaußprozesses zu untersuchen und besser zu verstehen.
In dieser Arbeit werden von tiefen Faltungsnetzwerken abgeleitete Gaußprozesse beschrieben und im Speziellen auf das Konvergenzverhalten der entprechenden Kernelfunktion bezüglich der Anzahl von Netzwerkschichten untersucht. Um das zu bewerkstelligen, werden oberer und untere Schranken der Kernelfunktion ermittelt. Während der Untersuchung dieser Schranken wird die Beobachtung gemacht, dass die Kernelfunktion für eine große Anzahl von Schichten gegen die obere Schranke zu konvergieren scheint. Diese Beobachtung bildet die Grundlage für eine Erklärung des Phänomens der abnehmenden Genauigkeit des ConvNet-GP-$L$-Prädiktors.

# Abstract

In the last few years, several connections between kernel machines and neural networks have been established. Specifically, the relationship between infinitely wide Bayesian neural networks and special Gaussian processes has been a source of many research contributions. This connection can be used to better understand deep neural networks through the lens of the corresponding Gaussian process.

In this thesis, we describe Gaussian processes derived from convolutional networks and specifically investigate the scaling behavior of their kernel function with respect to the number of layers. In doing so, we will derive a lower and an upper bound of the kernel function. Interestingly, we make the observation that the true kernel function converges against the derived upper bound of the kernel for large a number of layers. We use this fact to propose that the reason for the deteriorating accuracy of a standard ConvNet-GP-$L$ predictor with many layers lies in this convergence behavior.

# Contents

# 1 Introduction

In recent years, machine learning has become an increasingly popular tool to aid and create modern software systems. Neural networks have been at the forefront of this development by being able to solve highly complex problems. The use cases for these modern algorithms range from autonomous driving systems to protein folding applications [1].
Neural networks come in many flavors: There are fully connected networks (FCNs), recurrent neural networks (RNNs), convolutional neural networks (CNNs), and many more special architectures created for specific use cases [2]. Especially convolutional networks have become one of the most widely used algorithms in the field of deep learning. Computer vision and natural language processing are just two examples of the many areas of application [3]. Although there has been a lot of progress in the field of deep learning, the inner workings of these neural network-based systems are still not fully understood [4].
Fortunately, there are approaches to equate neural networks to other machine learning methods with a much broader mathematical foundation. One of these methods is the so-called Gaussian process (GP). GPs are used in GP predictors that are a type of kernel machine, i.e., they compare new data to stored information from old data using a similarity measure called the kernel. These predictors naturally provide an uncertainty estimate, which allows for the implementation of a Bayesian framework [5, 6].
It is possible to derive Gaussian processes from infinitely wide Bayesian fully connected networks [7, 8]. This principle can be extended to infinitely wide CNNs and RNNs [9, 10]. Analyzing Gaussian processes derived from neural networks might therefore open a new understanding of how varying specific hyperparameters changes the behavior of the system.
In this thesis, we want to explore the change of certain system properties with respect to the number of layers. In section 2, the current state of the art is presented. In the first block, we give an introduction to deep convolutional neural networks and describe a mathematical formalism to efficiently use these types of networks. In the second block, we introduce Gaussian processes in general, show how they can be used for supervised learning and derive the connection to neural networks. Our main contribution can be found in section 3. Firstly, we derive bounds for the kernel of a Gaussian process derived from a standard CNN with and without a ReLU activation function. We test the validity of these results in multiple experiments. Secondly, we relate these scaling results to the performance of the underlying Gaussian process. In the last section, we formulate a conclusion and present possible future work.

# 2 State of the Art

## 2.1 Convolutional Neural Networks

This thesis focuses on Gaussian processes derived from convolutional neural networks (CNNGPs). To understand these special Gaussian processes it is important to fully comprehend CNNs first.

Neural networks are functions dependent on a large number of parameters. By effectively adapting and tuning those parameters using gradient-based optimization methods like stochastic gradient descent or Adam [11], neural networks are able to represent complicated functions. CNNs are a special type of neural network. Convolution-based networks are usually built by stacking multiple layers of different types on top of each other. In this thesis, we will focus on fully connected layers as well as convolutional layers. Other frequently used layer types like residual layers [12], as well as pooling layers [3], will not play any role in this work.

This section will introduce neural networks superficially and describe a consistent formalism for the rest of this thesis. For a deeper introduction readers are referred to the literature cited in this section.

### 2.1.1 Fully Connected Layers

In fully connected layers, the neurons of the current layer are connected with all neurons/components of the preceding layer. The $i$-th output component $z_i^{(l)}$ of a single fully connected layer $l \in \mathbb{N}_{\geqslant 1}$ can be written for $i \in \{1, ..., N_{l+1}\}$ as

$$z_i^{(l)}(\mathbf{x}) = b_i^{(l)} + \sum_{j=1}^{N_l} W_{ij}^{(l)} x_j^{(l)}(\mathbf{x}), \quad x_j^{(l)}(\mathbf{x}) = \phi(z_j^{(l-1)}(\mathbf{x})), \tag{1}$$

where $\phi$ is the activation function, $W_{ij}^{(l)}$ the weights, $b_i^{(l)}$ the biases of each component and $N_l$ the number of outputs of the previous layer [8]. We write the post-activation $x_j^{(l)}(\mathbf{x})$ and the output component $z_i^{(l)}(\mathbf{x})$ explicitly as a function of a single input sample $\mathbf{x} = (x_1, x_2, ..., x_{N_0})^\top$ of the full network which will become relevant later.

Using this recursive formula with the base case $z_i^{(0)}(\mathbf{x}) = b_i^{(0)} + \sum_{j=1}^{N_0} W_{ij}^{(0)} x_j$, we can define a fully connected neural network recursively [8]. The activation function should be a nonlinear function otherwise the result in the last layer would only represent a linear combination of the input. Well-known nonlinearities like the ReLU or the Sigmoid activation function might be suitable depending on the use case. In literature, the activation function often represents a separate layer. In contrast to that, we will include the activation function always in a layer.

If we write the post-activation function for a single input as vector $\mathbf{x}^{(l)}(\mathbf{x}) = (x_1^{(l)}(\mathbf{x}), x_2^{(l)}(\mathbf{x}), ..., x_{N_l}^{(l)}(\mathbf{x}))^\top$, the sum (1) can be simplified as a simple matrix multiplication,

$$\mathbf{z}^{(l)}(\mathbf{x}) = \mathbf{b}^{(l)} + \mathbf{W}^{(l)}\mathbf{x}^{(l)}(\mathbf{x}), \quad \mathbf{x}^{(l)}(\mathbf{x}) = \phi\left(\mathbf{z}^{(l-1)}(\mathbf{x})\right), \tag{2}$$

with $\mathbf{W}^{(l)} = (W_{ij}^{(l)})_{i=1,...,N_{l+1},j=1,...,N_l}$ and $\mathbf{b}^{(l)} = (b_1^{(l)}, ..., b_{N_{l+1}}^{(l)})^\top$ [8]. In fully connected layers all weights and biases are not directly related to each other or fixed to a certain value. This differentiates fully connected layers from convolutional layers as we will see next.

### 2.1.2 Convolutional Layers

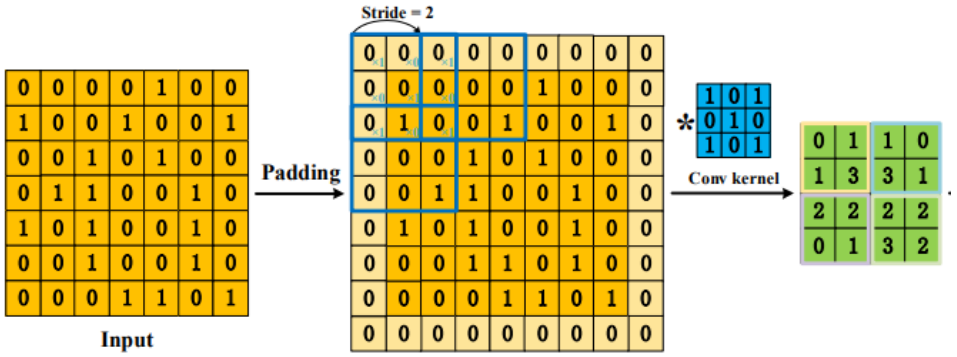



Figure 1: Procedure of two-dimensional convolution with zero-padding using a $3 \times 3$ kernel with stride 2 [3].

When working with pictures, audio data, or similar data forms, it is often the case that neighboring elements of a single data point are correlated.
Take for example a natural picture. In this picture, we will find hierarchical structures, e.g., in a picture of a house we will find edges, these edges form higher-order structures like rectangles and these rectangles form again higher-order objects [3]. These structures introduce correlation between neighboring pixels. This well-known property is exploited in several applications like compression or error correction [13]. A type of neural network that makes use of this property is a CNN.
Compared to fully connected layers, convolutional layers possess the following characteristics: 1) Local connections: Each neuron is only connected to a subset of neurons of the previous layer. 2) Weight sharing: Groups of connections share weights. Both these properties enable a large reduction of trainable parameters which simplifies the training procedure. Without correlation, this would not be a useful implementation [4].
Convolutional layers are implemented using a convolution operation [3]. In the case of two-dimensional convolution, an $n \times n$ convolutional filter window slides over the two-

dimensional object with stride size $s$. Each element of the 2D object is multiplied by the weights in the filter and summed up. Each shift of the filter window results in a new number in the output of the convolution operation. As a result, each convolutional filter can contain $n^2$ trainable parameters, where $n$ is the filter size. It is often the case that the two-dimensional objects are padded to achieve compatibility of the input size with the convolutional operation or to achieve a certain output size. The most common padding type is zero-padding although other methods are possible. In Figure 1, you can find an example of a convolution operation with $3 \times 3$ filter. This operation works in one or more than two dimensions analogously.

It is often the case that a data point has multiple channels or that multiple channels in hidden layers improve the performance. A channel might represent a color channel (RGB-Channel) in a picture or an audio frequency channel for audio data. To accommodate for multiple input channels, one applies the convolution operation for each input channel separately and sums up the corresponding results of the convolution operations [3]. Multiple output channels would then correspond to doing this for different filters with different weights independent of each other. Afterward, it is common to apply an activation function to introduce nonlinearity. One can summarize the filters of one layer within a 4-dimensional object $\mathbf{U} \in \mathbb{R}^{C^{(l+1)} \times C^{(l)} \times n \times n}$, where $C^{(l+1)}$ is the number of output channels and $C^{(l)}$ the number of input channels of the current layer.

We now want to formalize this mathematically. An image $\mathbf{X}_{\text{pic}}$ is a $C \times H \times D$ dimensional object, where $C$ is the number of channels, $H$ the height and $D$ the width. We now can write a single convolutional layer with activation function for each output channel $i \in \{1, ..., C^{(l+1)}\}$ as

$$\mathbf{Z}^{(l)}_{\text{pic},i}(\mathbf{X}_{\text{pic}}) = b^{(l)}_i \mathbb{I} + \sum_{j=1}^{C^{(l)}} \mathbf{U}^{(l)}_{i,j} * \mathbf{X}^{(l)}_{\text{pic},j}(\mathbf{X}_{\text{pic}}), \quad \mathbf{X}^{(l)}_{\text{pic}}(\mathbf{X}_{\text{pic}}) = \phi(\mathbf{Z}^{(l-1)}_{\text{pic}}(\mathbf{X}_{\text{pic}})), \tag{3}$$

where $*$ denotes the convolutional operation, $\mathbf{U}^{(l)}_{i,j}$ describes the used filter for the $j$-th input channel and $i$-th output channel, $\mathbf{Z}^{(l)}_{\text{pic},i}$ describes the output of the convolution operation of the $i$-th output channel of layer $l$ and $\mathbf{X}^{(l)}_{\text{pic},j}$ describes the $j$-th channel of the input of $l$-th layer. Using this we can implement a multi-layer system recursively with the base case $\mathbf{Z}^{(0)}_{\text{pic},i}(\mathbf{X}_{\text{pic}}) = b^{(0)}_i \mathbb{I} + \sum_{j=1}^{C^{(0)}} \mathbf{U}^{(0)}_{i,j} * \mathbf{X}_{\text{pic},j}$ [3, 4, 9].

The convolutional operation itself can also be implemented using standard matrix multiplication [9]. To do this, we have to reorder the input first. Here we transform $\mathbf{X}_{\text{pic}} \in \mathbb{R}^{C^{(0)} \times H^{(0)} \times D^{(0)}}$ to $\mathbf{X} \in \mathbb{R}^{(H^{(0)} D^{(0)}) \times C^{(0)}}$ by ordering each channel $i$ in row-major order into the $i$-th column $\mathbf{X}_i$ of $\mathbf{X}$. Analogously, we can proceed with the reordering process for all post-activations of all layers. In the next step, we can create a Toeplitz

matrix $\mathbf{W}_{i,j}$ from the filter $\mathbf{U}_{i,j}$ by ordering the weights of the filter into the corresponding positions. An example of this can be found in Figure 2. As a side note, in literature, the Toeplitz matrix is usually introduced for a standard convolutional operation. Although this operation is similar to the convolution in CNNs it is not the same [14]. After this is done, we can rewrite equation (3) for $i \in \{1, ..., C^{(l+1)}\}$ as

$$\mathbf{Z}_i^{(l)}(\mathbf{X}) = b_i^{(l)}\mathbf{1} + \sum_{j=1}^{C^{(l)}} \mathbf{W}_{i,j}^{(l)} \mathbf{X}_j^{(l)}(\mathbf{X}), \quad \mathbf{X}^{(l)}(\mathbf{X}) = \phi(\mathbf{Z}^{(l-1)}(\mathbf{X})). \tag{4}$$

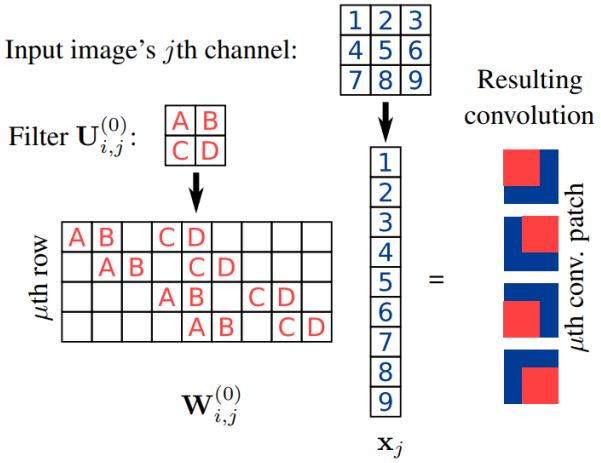



Figure 2: Creation of Toeplitz Matrix $\mathbf{W}_{i,j}^{(l)}$ from filter [9].

## 2.2 Neural Network Gaussian Processes

In this thesis, we will analyze Gaussian processes based on convolutional networks. Therefore it is necessary to gain a solid grasp of Gaussian processes first. In this section Gaussian processes are introduced and we describe how they can be utilized for prediction purposes. Afterward, we show that infinitely wide Bayesian neural networks are equivalent to a special case of Gaussian process. Based on this, we will extend this notion to convolutional networks which are the focus of this work.

### 2.2.1 Gaussian Processes

Gaussian processes (GPs) can be thought of as an extension of the Gaussian distribution to real functions. A good definition of GPs is given in definition 1 from [5].

**Definition 1.** *A Gaussian Process is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions.*

Like Gaussian distributions, Gaussian processes are fully determined by a mean function $m(x)$ and a co-variance or kernel function $k(x, x')$ [5, 15]. Since Gaussian processes are essentially distributions over functions, we will write

$$f \sim \mathcal{GP}(m, k), \tag{5}$$

denoting that the function $f$ is sampled from a GP with mean function $m$ and kernel function $k$.
Since infinite-dimensional vectors like functions are hard to work with and we are often only interested in output values of the function at a finite number $N$ of positions, we can use the fact that according to its definition any finite number of output values has a Gaussian distribution [15]. Given the finite number $N$ of x-values indexed by $i$ we get a corresponding mean vector $\mathbf{m}$ and kernel matrix $\mathbf{K}$ with entries

$$\begin{aligned} m_i &= m\left(x_i\right), \quad i = 1, \ldots, N \text{ and} \\ K_{ij} &= k\left(x_i, x_j\right), \quad i, j = 1, \ldots, N. \end{aligned} \tag{6}$$

Using $\mathbf{m}$ and $\mathbf{K}$ we can write the distribution of the function output at the $N$ x-values as follows:

$$\mathbf{f} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}), \tag{7}$$

where $\mathbf{f} = (f(x_1), ..., f(x_N))^\top$.
The kernel function and the mean function determine the shape and type of function generated from its GP. For regression tasks, the mean function is often set to zero while

the kernel function is the focus of the engineering task [5]. Probably the most well-known kernel function is the radial basis function kernel [16] given by

$$k(x, x') = \sigma_y^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right), \tag{8}$$

with parameters $\ell$ and $\sigma_y$. Other commonly used kernels are linear and periodic kernels. Another way to generate kernels is to derive them from neural networks which we focus on in section 2.2.3. The kernel matrix between two sets of samples will be written as follows:

$$\mathbf{K}_{\mathbf{x},\mathbf{x}'} = k(\mathbf{x}, \mathbf{x}') = \begin{bmatrix} k(x_1, x_1') & \dots & k(x_1, x_m') \\ \vdots & \ddots & \vdots \\ k(x_n, x_1') & \dots & k(x_n, x_m') \end{bmatrix}. \tag{9}$$

### 2.2.2 Gaussian Process Regression and Classification

Gaussian processes can be used for regression and classification tasks. The GP predictor is considered a non-parametric predictor, that is, it is not defined by a finite number of parameters $\theta$ but instead by an infinite-dimensional object, which we usually think of as functions [15]. A big advantage of the GP predictor is the fact that Gaussian processes deliver an uncertainty distribution instead of a single prediction value which makes the GPs particularly interesting for predictions within a Bayesian framework [15].
In the last section, we showed how a GP can be considered a distribution over functions. For prediction tasks, we will now use this Gaussian process as a Bayesian prior [15]. By definition, the prior itself does not depend on the training data but describes the properties of the class of possible functions. In a prediction task, we want to update this prior information based on training data.
Let $\mathbf{f}$ and $\mathbf{f}^*$ be the function values corresponding to the training data and test data respectively. Since both are generated from the same GP they are jointly Gaussian distributed, i.e.,

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m} \\ \mathbf{m}^* \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\mathbf{x},\mathbf{x}} & \mathbf{K}_{\mathbf{x},\mathbf{x}^*} \\ \mathbf{K}_{\mathbf{x}^*,\mathbf{x}} & \mathbf{K}_{\mathbf{x}^*,\mathbf{x}^*} \end{bmatrix}\right). \tag{10}$$

Because we want to predict the test function values based on the training data, it makes sense to derive a conditional distribution of $\mathbf{f}^*$ given $\mathbf{f}$,

$$\mathbf{f}^* \mid \mathbf{f} \sim \mathcal{N}\big(\mathbf{m}^* + \mathbf{K}_{\mathbf{x}^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}(\mathbf{f} - \mathbf{m}), \mathbf{K}_{\mathbf{x}^*,\mathbf{x}^*} - \mathbf{K}_{\mathbf{x}^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x},\mathbf{x}^*}\big), \tag{11}$$

which is the posterior distribution for the test set [5]. The corresponding posterior GP is given as follows:

$$\begin{aligned} f \mid \mathcal{D} &\sim \mathcal{GP}(m_{\mathcal{D}}, k_{\mathcal{D}}), \\ m_{\mathcal{D}}(x) &= m(x) + \mathbf{K}_{x,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}(\mathbf{f} - \mathbf{m}) \\ k_{\mathcal{D}}(x, x') &= k(x, x') - \mathbf{K}_{x,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x},x'}. \end{aligned} \tag{12}$$

A prediction for a single test case $x^*$ can be is given by a Gaussian distribution $\mathcal{N}(m_{\mathcal{D}}(x^*), k_{\mathcal{D}}(x^*, x^*))$. It is therefore possible to not only generate a single point output prediction but an uncertainty distribution over that output [5].

If prediction tasks are performed as above, the predictor would interpolate known test inputs. However, since observations are often noisy and to ensure proper convergence in the limit of a large data a noise term must be included. Under the assumption of i.i.d. Gaussian noise we can accommodate for this by adding variance to the kernel function resulting in

$$\begin{aligned} y(x) &= f(x) + \varepsilon, & \varepsilon &\sim \mathcal{N}\left(0, \sigma^2\right), \\ f &\sim \mathcal{GP}(m, k), & y &\sim \mathcal{GP}\left(m, k + \sigma^2\delta_{ii'}\right), \end{aligned} \tag{13}$$

where $\delta_{i,i'}$ is the Kronecker's delta [15]. The corresponding GP predictor can be evaluated via

$$\begin{aligned} \mathbf{f}^* \mid \mathbf{f} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \boldsymbol{\mu} &= \mathbf{m}^* + \mathbf{K}_{\mathbf{x}^*,\mathbf{x}}(\mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma\mathbb{I})^{-1}(\mathbf{f} - \mathbf{m}) \\ \boldsymbol{\Sigma} &= \mathbf{K}_{\mathbf{x}^*,\mathbf{x}^*} - \mathbf{K}_{\mathbf{x}^*,\mathbf{x}}(\mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma\mathbb{I})^{-1}\mathbf{K}_{\mathbf{x},\mathbf{x}^*} \end{aligned} \tag{14}$$

To improve the GP, we can choose kernel and mean function based on the data. The kernel function, as well as the mean function, are often parameterized. The optimal way to choose these hyper-parameters is to maximize the log marginal likelihood with respect to the hyper-parameters. Under the assumption of data sampled from a Gaussian distribution, the log marginal likelihood is given by

$$L = \log p(\mathbf{f} \mid \mathbf{X}, \theta) = -\frac{1}{2}\log|\mathbf{K}_{\mathbf{x},\mathbf{x}}| - \frac{1}{2}(\mathbf{f} - \mathbf{m})^\top \mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}(\mathbf{f} - \mathbf{m}) - \frac{n}{2}\log(2\pi), \tag{15}$$

where $\theta$ is the hyper-parameter vector [5]. For complicated kernel functions, cross-validation should be applied, otherwise, gradient-based methods can be used to maximize $L$ [5].

Because we will perform an image classification task, we will quickly introduce classification. In classification tasks, the likelihoods are not Gaussian and we also cannot just use a Softmax function as is commonly done for neural networks [5]. One can resort to approximations, the Laplace approximation and projections on the closest Gaussian are just two methods to be useful here [15].

In this thesis we will not use any of these techniques, instead, we perform the classification directly with exact regression as done in [9]. Here we encode the class labels as one-hot-negative vectors and perform a simple multi-class regression on $\mathbf{y}_i \in \{-1; +1\}^C$ where $C$ is the number of classes and where 1 corresponds to "is the class" and -1 corresponds to "not the class".

### 2.2.3 Neural Networks Gaussian Processes

Neural networks have become a very important tool in many fields of machine learning. In computer vision, speech recognition, and machine translation, they dominate the industry [17]. As a contrasting approach, GPs have been used as a traditional tool with a natural capability for Bayesian inference. In 1994, Neal derived in his seminal dissertation an equivalence between the two methods for the case of single-layer infinitely wide Bayesian neural networks [7]. This idea has been extended to deep fully connected neural networks by Lee et al. [8]. Later on, this equivalence has been shown for other neural network architectures like convolutional networks [9] and recurrent networks [10]. A huge advantage of this equivalence is that this might give a natural way of implementing Bayesian inference for neural networks, which has long been inaccurate and computationally demanding [18]. We will call Gaussian processes derived from neural networks: neural network gaussian processes (NNGPs).
We briefly introduce the correspondence for single-layer neural networks, move on to deep neural networks, and at last review the relationship between GPs and convolutional networks.

#### Single Layer Networks

We now want to derive this correspondence for the single-layer case following [7]. The $i$-th component of the single layer network output, $z_i^{(1)}$, is given using equation (1) for $l = 1$ as follows:

$$z_i^{(1)}(\mathbf{x}) = b_i^{(1)} + \sum_{j=1}^{N_1} W_{ij}^{(1)} x_j^{(1)}(\mathbf{x}), \quad x_j^{(1)}(\mathbf{x}) = \phi\left(b_i^{(0)} + \sum_{j=1}^{N_0} W_{jk}^{(0)} x_k\right), \tag{16}$$

where $W_{ij}^{(l)}$ and $b_j^{(l)}$ are i.i.d. drawn with variance $\sigma_\omega^2/N_l$ and $\sigma_b^2$ respectively. Because the parameters are i.i.d., $x_j^{(1)}$ and $x_{j'}^{(1)}$ are independent for $j \neq j'$. Therefore $z_i^{(1)}$ is the sum of i.i.d. terms and it follows by the central limit theorem that for $N_1 \to \infty$, $z_i^{(1)}$ assumes a Gaussian distribution. As a result, any finite collection of $\{z_i^{(1)}(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^{N_0}\}$ has a joint Gaussian distribution, which is exactly the definition of a Gaussian process [7].

In conclusion, we have $z_i^{(1)} \sim \mathcal{GP}(m, k^{(1)})$ with mean function $m(x) = \mathbb{E}[z_i^{(1)}(\mathbf{x})] = 0$ and kernel function

$$k^1(\mathbf{x}, \mathbf{x}') \equiv \mathbb{E}[z_i^{(1)}(\mathbf{x}) z_i^{(1)}(\mathbf{x}')] = \sigma_b^2 + \sigma_\omega^2 \mathbb{E}[x_i^{(1)}(\mathbf{x}) x_i^{(1)}(\mathbf{x}')], \tag{17}$$

independent of $i$ [8]. The expectation value $\mathbb{E}[x_i^{(1)}(\mathbf{x}) x_i^{(1)}(\mathbf{x}')]$ can be obtained by integrating over the distribution of $\mathbf{W}^{(0)}$ and $\mathbf{b}^{(0)}$. Note that, any outputs $z_i^{(1)}$, $z_{i'}^{(1)}$ with $i \neq i'$ are independent from each other, despite using the same features.

### Deep Fully Connected Networks

Now we want to extend this to deep fully connected networks by induction following [8]. Suppose that $z_i^{(l-1)}$ is an identical and independent GP for every $i$ and hence $x_i^{(l)}(\mathbf{x})$ is i.i.d.. To calculate $z_i^{(l-1)}$ we use equation (1). Similar to the case of a single layer network, $z_i^{(l-1)}$ is again a sum of i.i.d. terms. As before, for $N_l \to \infty$ by the central limit theorem any finite collection of $\{z_i^{(l)}(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^{N_0}\}$ has a Gaussian distribution, which means $z_i^{(l)} \sim \mathcal{GP}(0, k^{(l)})$ with kernel function

$$\begin{aligned} k^{(l)}(\mathbf{x}, \mathbf{x}') \equiv \mathbb{E}[z_i^{(l)}(\mathbf{x}) z_i^{(l)}(\mathbf{x}')] &= \sigma_b^2 + \sigma_\omega^2 \mathbb{E}_{z_i^{(l-1)} \sim \mathcal{G}(0, k^{(l-1)})}[\phi(z_i^{(l-1)}(\mathbf{x})) \phi(z_i^{(l-1)}(\mathbf{x}'))] \\ &= \sigma_b^2 + \sigma_\omega^2 F_\phi(k^{(l-1)}(\mathbf{x}, \mathbf{x}'), k^{(l-1)}(\mathbf{x}, \mathbf{x}), k^{(l-1)}(\mathbf{x}', \mathbf{x}')), \end{aligned} \tag{18}$$

where the form of function $F_\phi$ is completely determined by the activation function. This gives us an recursive formula to determine the kernel function. The base case $l = 0$ is determined by

$$k^{(0)}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[z_j^{(0)}(\mathbf{x}) z_j^{(0)}(\mathbf{x}')] = \sigma_b^2 + \sigma_\omega^2 \left( \frac{\mathbf{x} \cdot \mathbf{x}'}{d_{N_0}} \right). \tag{19}$$

At last, since this is a very important case, for the ReLU-activation function there is a close form representation of $F_\phi$ given in [8] as

$$\begin{aligned} &F_\phi(k^{(l-1)}(\mathbf{x},\mathbf{x}'), k^{(l-1)}(\mathbf{x}, \mathbf{x}), k^{(l-1)}(\mathbf{x}', \mathbf{x}')) = \\ &\quad = \frac{\sqrt{k^{(l-1)}(\mathbf{x}, \mathbf{x}) k^{(l-1)}(\mathbf{x}', \mathbf{x}')}}{2\pi} \left( \sin \theta_{\mathbf{x},\mathbf{x}'}^{(l-1)} + (\pi - \theta_{\mathbf{x},\mathbf{x}'}^{(l-1)}) \cos \theta_{\mathbf{x},\mathbf{x}'}^{(l-1)} \right), \end{aligned} \tag{20}$$

with

$$\theta_{\mathbf{x},\mathbf{x}'}^{(l)} = \cos^{-1} \left( \frac{k^{(l)}(\mathbf{x}, \mathbf{x}')}{\sqrt{k^{(l)}(\mathbf{x}, \mathbf{x}) k^{(l)}(\mathbf{x}', \mathbf{x}')}} \right). \tag{21}$$

### Convolutional Networks

Finally, we want to extend this to convolutional networks. Since the derivation is quite extensive and not of particular importance for this thesis, we refer to [9] for an in-depth proof and instead present only the important results.
We assume the weights and biases to be i.i.d. Gaussian, that is,

$$U_{i,j,x,y}^{(\ell)} \sim \mathcal{N}\left(0, \sigma_\omega^2 / C^{(\ell)}\right), \quad b_i^{(\ell)} \sim \mathcal{N}\left(0, \sigma_b^2\right), \tag{22}$$

with the corresponding Toeplitz matrix representation $W_{i,j,x,y}$. If we apply $C^{(l)} \to \infty$ iteratively for every layer we can show again by using the central limit theorem that the resulting output $\mathbf{Z}_i^{(l)}$ with channel $i$ is an independent GP with zero-mean, i.e., $\mathbf{Z}_i^{(l)} \sim \mathcal{GP}(0, k^{(l)})$. The corresponding kernel function in the $l$-th layer can be calculated recursively for $\mu \in \{1, ..., H^{(l+1)} D^{(l+1)}\}$ with

$$\begin{aligned} k_\mu^{(l)}(\mathbf{X}, \mathbf{X}') &= \mathrm{Cov}[\mathbf{Z}_{\mu,i}^{(l)}(\mathbf{X}), \mathbf{Z}_{\mu,i}^{(l)}(\mathbf{X}')] = \\ &= \sigma_b^2 + \sum_{j=1}^{C^{(l)}} \sum_{\nu=1}^{H^{(l)} D^{(l)}} \mathbb{E}[W_{i,j,\mu,\nu}^{(l)} W_{i,j,\mu,\nu}^{(l)}] \mathbb{E}[\phi(\mathbf{Z}_{\nu,i}^{(l-1)}(\mathbf{X})) \phi(\mathbf{Z}_{\nu,i}^{(l-1)}(\mathbf{X}'))] \\ &= \sigma_b^2 + \sigma_\omega^2 \sum_{\nu \in \mu\text{-th patch}} F_\phi(k_\nu^{(l-1)}(\mathbf{X}, \mathbf{X}'), k_\nu^{(l-1)}(\mathbf{X}, \mathbf{X}), k_\nu^{(l-1)}(\mathbf{X}', \mathbf{X}')), \end{aligned} \tag{23}$$

where the $\mu$-th patch corresponds to the set of non-zero positions of the $\mu$-th row of the Toeplitz matrix $W_{i,j}^{(l)}$. The base case $l = 0$ is simply given by

$$k_\mu^{(0)}(\mathbf{X}, \mathbf{X}') = \sigma_b^2 + \frac{\sigma_\omega^2}{C^{(0)}} \sum_{i=1}^{C^{(0)}} \sum_{\nu \in \mu\text{-th patch}} X_{\nu,i} X'_{\nu,i}. \tag{24}$$

# 3 Deep Convolutional Networks as Gaussian Processes

In the previous section, we introduced Gaussian processes derived from convolutional networks. We now want to specifically look at the behavior of GPs that are based on very deep convolutional networks. In the first part, we will present the main contribution of this thesis which can be summarized as bounds on the scaling of the kernel function with respect to the number of layers. We will prove these bounds, show their interesting properties, and test them experimentally. In doing so, we will introduce the MNIST data set and the ConvNet-GP-$L$ architecture used in our implementation. If not stated otherwise, all operations and comparisons except the multiplication "$\cdot$", which is calculated as always, are performed element-wise.

## 3.1 Linear Convolutional NNGP

In this section, we want to start the main part of this thesis by introducing the case of a linear convolutional network as GP, i.e., a Gaussian process derived from a convolutional network without activation functions. This will give us a better grasp of how we can easily transform CNN matrix representation into a corresponding matrix representation of the kernel function.

### 3.1.1 Kernel Functions in Matrix Representation

Equations (23) and (24) can be rewritten as a convolutional operation. In comparison to the convolution operation in a normal convolutional network, we use a constant single-channel filter of size $n_l \times n_l$ given by

$$\widetilde{\mathbf{U}}^{(l)} = \sigma_\omega^2 \cdot \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix} \in \mathbb{R}^{n_l \times n_l}. \tag{25}$$

Since calculations with matrix multiplication are a lot easier to handle mathematically, we implement the convolution with the corresponding Toeplitz matrix $\widetilde{\mathbf{W}}^{(l)}$. Remember, in case of zero-padding the columns corresponding to the zeros are removed from $\widetilde{\mathbf{W}}^{(l)}$. As a result, we can re-represent (23) and (24) with a single operation as

$$\mathbf{k}^{(l)}(\mathbf{X}, \mathbf{X}') = \sigma_b^2 \mathbb{I} + \widetilde{\mathbf{U}} * \mathbf{F}_\phi(\mathbf{k}^{(l-1)}(\mathbf{X}, \mathbf{X}'), \mathbf{k}^{(l-1)}(\mathbf{X}, \mathbf{X}), \mathbf{k}^{(l-1)}(\mathbf{X}', \mathbf{X}')), \tag{26}$$

where $\mathbf{k}$ is the matrix representation with entries corresponding to the patches as is done for $\mathbf{F}$ and $*$ is the convolution operation. We translate this using the Toeplitz matrix as

follows:

$$\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X}') = \sigma_b^2 \mathbf{1} + \widetilde{\mathbf{W}}^{(l)} \mathbf{V}^{(l-1)}(\mathbf{X}, \mathbf{X}'), \tag{27}$$

where $\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X}') = \text{vec}(\mathbf{k}^{(l)}(\mathbf{X}, \mathbf{X}'))$ and
$\mathbf{V}^{(l-1)}(\mathbf{X}, \mathbf{X}')) = \text{vec}(\mathbf{F}_\phi(\mathbf{k}^{(l-1)}(\mathbf{X}, \mathbf{X}'), \mathbf{k}^{(l-1)}(\mathbf{X}, \mathbf{X}), \mathbf{k}^{(l-1)}(\mathbf{X}', \mathbf{X}')))$.
The full kernel can be reconstructed recursively with the base case

$$\mathbf{K}^{(0)}(\mathbf{X}, \mathbf{X}') = \sigma_b^2 \mathbf{1} + \frac{1}{C^{(0)}} \widetilde{\mathbf{W}}^{(0)} \sum_{i=1}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i, \tag{28}$$

where $\odot$ is the Hadamard-product.

### 3.1.2 Linear Convolutional Kernel

Based on the representation derived in the previous section, we can simplify the kernel function for a CNNGP based on a network with only convolutional layers and no activation functions, i.e., $\mathbf{V}^{(l)} = \mathbf{K}^{(l)}$. The results are given in theorem 1.

**Theorem 1.** *Given a linear L-layer CNNGP with kernel function* $\mathbf{K}^{(L)}$ *and* $\widetilde{\mathbf{W}}^{(l)}$ *for* $l \in \{0, 1, ..., L\}$ *Toeplitz transformation matrices, then* $\mathbf{K}^{(L)}$ *can be written for all* $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{(H^{(0)} D^{(0)}) \times C^{(0)}}$ *as*

$$\mathbf{K}^{(L)}(\mathbf{X}, \mathbf{X}') = \sigma_b^2 \cdot \left( \sum_{k=0}^{L} \prod_{i=0}^{k-1} \widetilde{\mathbf{W}}^{(L-i)} \right) \mathbf{1} + \frac{1}{C^{(0)}} \left( \prod_{k=0}^{L} \widetilde{\mathbf{W}}^{(L-k)} \right) \left( \sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i \right).$$

*Proof.* The result can be easily shown by induction.
*Base case:* $L = 1$

$$\begin{aligned}
\mathbf{K}^{(1)}(\mathbf{X}, \mathbf{X}') &= \sigma_b^2 \mathbf{1} + \widetilde{\mathbf{W}}^{(1)} \mathbf{K}^{(0)}(\mathbf{X}, \mathbf{X}') \\
&= \sigma_b^2 \mathbf{1} + \sigma_b^2 \widetilde{\mathbf{W}}^{(1)} \mathbf{1} + \frac{1}{C^{(0)}} \widetilde{\mathbf{W}}^{(1)} \widetilde{\mathbf{W}}^{(0)} \left( \sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i \right) \\
&= \sigma_b^2 \cdot \left( \sum_{k=0}^{1} \prod_{i=0}^{k-1} \widetilde{\mathbf{W}}^{(1-i)} \right) \mathbf{1} + \frac{1}{C^{(0)}} \left( \prod_{k=0}^{1} \widetilde{\mathbf{W}}^{(1-k)} \right) \left( \sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i \right)
\end{aligned}$$

*Induction step:* Suppose the induction hypothesis is true for a single step $L$. It follows:

$$
\begin{aligned}
&\mathbf{K}^{(L+1)}(\mathbf{X},\mathbf{X}') = \\
&= \sigma_b^2\mathbf{1} + \widetilde{\mathbf{W}}^{(L+1)}\mathbf{K}^{(L)}(\mathbf{X},\mathbf{X}') \\
&= \sigma_b^2\mathbf{1} + \widetilde{\mathbf{W}}^{(L+1)}\left(\sigma_b^2\cdot\left(\sum_{k=0}^{L}\prod_{i=0}^{k-1}\widetilde{\mathbf{W}}^{(L-i)}\right)\mathbf{1} + \frac{1}{C^{(0)}}\left(\prod_{k=0}^{L}\widetilde{\mathbf{W}}^{(L-k)}\right)\left(\sum_{i=0}^{C^{(0)}}\mathbf{X}_i\odot\mathbf{X}'_i\right)\right) \\
&= \sigma_b^2\mathbf{1} + \sigma_b^2\left(\sum_{k=1}^{L+1}\prod_{i=0}^{k-1}\widetilde{\mathbf{W}}^{(L+1-i)}\right)\mathbf{1} + \frac{1}{C^{(0)}}\left(\prod_{k=0}^{L+1}\widetilde{\mathbf{W}}^{(L+1-k)}\right)\left(\sum_{i=0}^{C^{(0)}}\mathbf{X}_i\odot\mathbf{X}'_i\right) \\
&= \sigma_b^2\cdot\left(\sum_{k=0}^{L+1}\prod_{i=0}^{k-1}\widetilde{\mathbf{W}}^{(L+1-i)}\right)\mathbf{1} + \frac{1}{C^{(0)}}\left(\prod_{k=0}^{L+1}\widetilde{\mathbf{W}}^{(L+1-k)}\right)\left(\sum_{i=0}^{C^{(0)}}\mathbf{X}_i\odot\mathbf{X}'_i\right),
\end{aligned}
$$

that is, the hypothesis holds true for $L+1$, establishing the induction step. Therefore by induction, the theorem holds. □

This theorem just illustrates what was to be suspected, i.e., that the kernel of a linear convolutional NNGP is essentially an affine transformation of the Hadamard product of the input data.

## 3.2 ReLU Convolutional NNGP

### 3.2.1 Preliminary Work

We now want to move on to nonlinear CNNGPs and specifically look at GPs based on convolutional networks with a ReLU-activation applied in each layer. In this case, $\mathbf{V}^{(l)}$ is given following equation (20) by

$$
\mathbf{V}^{(l)}(\mathbf{X},\mathbf{X}')) = \frac{\sqrt{\mathbf{K}^{(l)}(\mathbf{X},\mathbf{X})\odot\mathbf{K}^{(l)}(\mathbf{X}',\mathbf{X}')}}{2\pi}\odot\left(\sin\theta^{(l)} + (\pi-\theta^{(l)})\odot\cos\theta^{(l)}\right), \tag{29}
$$

with

$$
\theta^{(l)} = \cos^{-1}\left(\frac{\mathbf{K}^{(l)}(\mathbf{X},\mathbf{X}')}{\sqrt{\mathbf{K}^{(l)}(\mathbf{X},\mathbf{X})\odot\mathbf{K}^{(l)}(\mathbf{X}',\mathbf{X}')}}\right), \tag{30}
$$

where all operations are applied element-wise. To simplify this equation we introduce the Person correlation between two samples

$$
\rho^{(l)}(\mathbf{X},\mathbf{X}') = \frac{\mathbf{K}^{(l)}(\mathbf{X},\mathbf{X}')}{\sqrt{\mathbf{K}^{(l)}(\mathbf{X},\mathbf{X})\odot\mathbf{K}^{(l)}(\mathbf{X}',\mathbf{X}')}}, \tag{31}
$$

where $\rho_i \in [-1, 1]$.
With the identity $\sin(\cos^{-1}(x)) = \sqrt{1 - x^2}$ for $x \in [-1, 1]$ equation (29) simplifies to

$$\begin{aligned}\mathbf{V}^{(l)}(\mathbf{X}, \mathbf{X}') =& \frac{\sqrt{\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X}) \odot \mathbf{K}^{(l)}(\mathbf{X}', \mathbf{X}')}}{2\pi} \\ &\odot \underbrace{\left(\sqrt{1 - \rho^{(l)}(\mathbf{X}, \mathbf{X}')^2} + \left(\pi - \cos^{-1}\left(\rho^{(l)}(\mathbf{X}, \mathbf{X}')\right)\right) \odot \rho^{(l)}(\mathbf{X}, \mathbf{X}')\right)}_{f(\rho^{(l)}(\mathbf{X}, \mathbf{X}'))}.\end{aligned} \tag{32}$$

In case of $\mathbf{X} = \mathbf{X}'$, i.e., $\rho^{(l)}(\mathbf{X}, \mathbf{X}) = \mathbf{1}$, (32) reduces to

$$\mathbf{V}^{(l)}(\mathbf{X}, \mathbf{X}) = \frac{\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X})}{2}. \tag{33}$$

Since the bias-variance introduces a lot of extra complexity, we will set it to zero for the time being. We will justify this in section 3.4.1. If we have a zero bias-variance ($\sigma_b^2 = 0$), the kernel function for self-interaction is just a linear transformation of the squared input data. Following theorem 1, the self-interaction kernel function is given by

$$\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X}) = \frac{1}{2^l \cdot C^{(0)}} \left(\prod_{k=0}^{l} \widetilde{\mathbf{W}}^{(l-k)}\right) \left(\sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i\right). \tag{34}$$

Leading up to the main results of the thesis we will define the $l$-level transformed covariance in definition 2.

**Definition 2.** *The $l$-level transformed covariance $P^{(l)}$ for all $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{(H^{(0)} D^{(0)}) \times C^{(0)}}$ is defined as*

$$\begin{aligned}\mathbf{P}^{(l)}(\mathbf{X}, \mathbf{X}') &:= 2^l \cdot \sqrt{\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X}) \odot \mathbf{K}^{(l)}(\mathbf{X}', \mathbf{X}')} \\ &= \frac{1}{C^{(0)}} \sqrt{\left(\left(\prod_{k=0}^{l} \widetilde{\mathbf{W}}^{(l-k)}\right)\left(\sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}_i\right)\right) \odot \left(\left(\prod_{k=0}^{l} \widetilde{\mathbf{W}}^{(l-k)}\right)\left(\sum_{i=0}^{C^{(0)}} \mathbf{X}'_i \odot \mathbf{X}'_i\right)\right)}.\end{aligned}$$

This definition will become relevant later. Expressing a ReLU network without recursion to directly analyze the scaling of the kernel function is either impossible or too challenging for this thesis. Instead, we have to determine a lower and upper bound of the kernel function to investigate its scaling behavior. We first formulate the lower bound and re-express it in a usable way. Afterward, we move on to the upper bound.

### 3.2.2 Lower Bound

A lower bound of the kernel function $\mathbf{K}^{(L)}(\mathbf{X}, \mathbf{X}')$ of a convolutional NNGP without bias-variance is given in theorem 2. Figure 3 illustrates the approximations used in both lower and upper bound derivations.

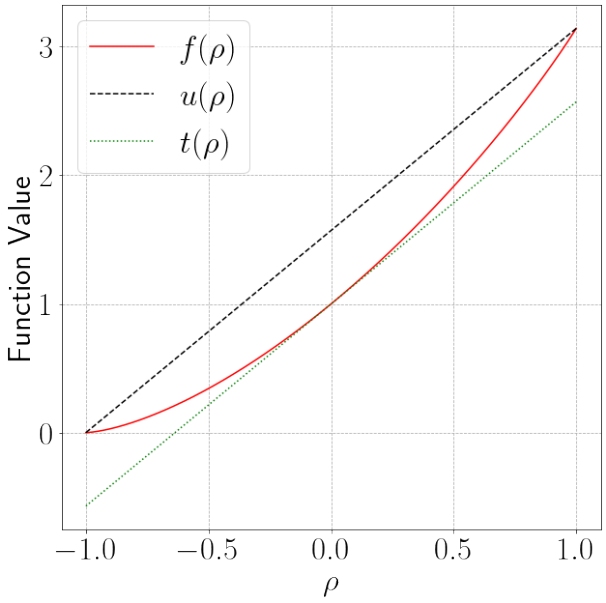



Figure 3: The read straight line represents the function $f$ as determined in equation (32) defined on the interval $[-1, 1]$. The dotted green line serves as a representation of the function $t$ as used in the proof of theorem 2. The dashed black line shows the function $u$ as used in the proof of the theorem 4.

**Theorem 2.** *Given a L-layer ReLU CNNGP without bias-variance, i.e., $\sigma_b^2 = 0$, a lower bound of the corresponding kernel function $\mathbf{K}^{(L)}(\mathbf{X}, \mathbf{X}')$ for all $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{(H^{(0)}D^{(0)})\times C^{(0)}}$ is given by*

$$\mathbf{K}_{low}^{(L)}(\mathbf{X}, \mathbf{X}') = \widetilde{\mathbf{W}}^{(L)}\Big(\frac{1}{2\pi}\frac{1}{2^{L-1}}\mathbf{P}^{(L-1)}(\mathbf{X}, \mathbf{X}') + \frac{1}{4}\mathbf{K}_{low}^{(L-1)}(\mathbf{X}, \mathbf{X}')\Big),$$

*with base case $\mathbf{K}_{low}^{(0)}(\mathbf{X}, \mathbf{X}') = \mathbf{K}^{(0)}(\mathbf{X}, \mathbf{X}')$.*

*Proof.* The kernel function for a $L$-layer ReLU convolutional NNGP without bias-variance is given by

$$\mathbf{K}^{(L)}(\mathbf{X}, \mathbf{X}') = \widetilde{\mathbf{W}}^{(L)} \underbrace{\frac{\sqrt{\mathbf{K}^{(L-1)}(\mathbf{X}, \mathbf{X}) \odot \mathbf{K}^{(L-1)}(\mathbf{X}', \mathbf{X}')}}{2\pi}}_{\geq 0} \odot f(\rho^{(L-1)}(\mathbf{X}, \mathbf{X}')).$$

Since $f$ is an element-wise monotone increasing convex function for all $\rho_i \in [-1, 1]$, we can lower bound it with its first order Taylor approximation at position $\rho = 0$ given by

$t(\rho) = \mathbf{1} + \frac{\pi}{2}\rho$, i.e., $t(\rho) \leq f(\rho)$ for all $\rho_i \in [-1, 1]$.
Therefore a first lower bound is given by

$$\begin{aligned}\mathbf{K}^{(L)}_{\text{low,inter}}(\mathbf{X}, \mathbf{X}') &= \widetilde{\mathbf{W}}^{(L)} \frac{\sqrt{\mathbf{K}^{(L-1)}(\mathbf{X}, \mathbf{X}) \odot \mathbf{K}^{(L-1)}(\mathbf{X}', \mathbf{X}')}}{2\pi} \odot t(\rho^{(L-1)}(\mathbf{X}, \mathbf{X}')) \\ &= \widetilde{\mathbf{W}}^{(L)} \Big( \frac{1}{2\pi} \frac{1}{2^{L-1}} \mathbf{P}^{(L-1)}(\mathbf{X}, \mathbf{X}') + \frac{1}{4} \mathbf{K}^{(L-1)}(\mathbf{X}, \mathbf{X}') \Big).\end{aligned}$$

We can easily show by induction that $\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X}') \geq \mathbf{K}^{(l)}_{\text{low,inter}}(\mathbf{X}, \mathbf{X}')$ for all $l \leq L$. Since $\mathbf{K}^{(L)}_{\text{low,inter}}$ is element-wise monotone increasing in $\mathbf{K}^{(L-1)}$ it follows $\mathbf{K}^{(L)}_{\text{low}} \leq \mathbf{K}^{(L)}_{\text{low,inter}}$, which proofs the result. □

Unfortunately, this lower bound is still only in its recursive form. To analyze the scaling behavior properly we must rewrite it as a sum expression. The next theorem delivers this result.

**Theorem 3.** *Given a L-layer ReLU CNNGP without bias-variance, i.e., $\sigma_b^2 = 0$, and the lower bound $\mathbf{K}^{(L)}_{low}$ of its kernel function, then for all $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{(H^{(0)} D^{(0)}) \times C^{(0)}}$*

$$\begin{aligned}\mathbf{K}^{(L)}_{low}(\mathbf{X}, \mathbf{X}') = \frac{2}{\pi} \frac{1}{2^L} \sum_{k=1}^{L} \frac{1}{2^k} \left( \prod_{i=0}^{k-1} \widetilde{\mathbf{W}}^{(L-i)} \right) \mathbf{P}^{(L-k)}(\mathbf{X}, \mathbf{X}') \\ + \frac{1}{C^{(0)} 4^L} \left( \prod_{k=0}^{L} \widetilde{\mathbf{W}}^{(L-k)} \right) \left( \sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i \right).\end{aligned}$$

*Proof.* The result can be easily shown by induction.
*Base case:* $L = 1$

$$\begin{aligned}\mathbf{K}^{(1)}_{\text{low}}(\mathbf{X}, \mathbf{X}') &= \widetilde{\mathbf{W}}^{(1)} \Big( \frac{1}{2\pi} \frac{1}{2^{1-1}} \mathbf{P}^{(1-1)}(\mathbf{X}, \mathbf{X}') + \frac{1}{4} \mathbf{K}^{(1-1)}_{\text{low}}(\mathbf{X}, \mathbf{X}') \Big) \\ &= \frac{1}{2\pi} \frac{1}{2^{1-1}} \widetilde{\mathbf{W}}^{(1)} \mathbf{P}^{(1-1)}(\mathbf{X}, \mathbf{X}') + \frac{1}{4C^{(0)}} \widetilde{\mathbf{W}}^{(1)} \widetilde{\mathbf{W}}^{(0)} \left( \sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i \right) \\ &= \frac{2}{\pi} \frac{1}{2^1} \sum_{k=1}^{1} \frac{1}{2^k} \left( \prod_{i=0}^{k-1} \widetilde{\mathbf{W}}^{(1-i)} \right) \mathbf{P}^{(1-k)}(\mathbf{X}, \mathbf{X}') \\ &+ \frac{1}{C^{(0)} 4^1} \left( \prod_{k=0}^{1} \widetilde{\mathbf{W}}^{(1-k)} \right) \left( \sum_{i=0}^{C^{(0)}} \mathbf{X}_i \odot \mathbf{X}'_i \right).\end{aligned}$$

*Induction step:* Suppose the induction hypothesis is true for a single step $L$. It follows

that:

$$
\begin{aligned}
\mathbf{K}_{\text{low}}^{(L+1)}(\mathbf{X},\mathbf{X}') &= \widetilde{\mathbf{W}}^{(L+1)}\Big(\frac{1}{2\pi}\frac{1}{2^{L+1-1}}\mathbf{P}^{(L+1-1)}(\mathbf{X},\mathbf{X}') + \frac{1}{4}\mathbf{K}_{\text{low}}^{(L+1-1)}(\mathbf{X},\mathbf{X}')\Big) \\
&= \frac{2}{\pi}\frac{1}{2^{L+1}\cdot 2}\widetilde{\mathbf{W}}^{(L+1)}\mathbf{P}^{(L)}(\mathbf{X},\mathbf{X}') \\
&+ \frac{2}{\pi}\frac{1}{2^{L+1}\cdot 2}\widetilde{\mathbf{W}}^{(L+1)}\sum_{k=1}^{L}\frac{1}{2^k}\left(\prod_{i=0}^{k-1}\widetilde{\mathbf{W}}^{(L-i)}\right)\mathbf{P}^{(L-k)}(\mathbf{X},\mathbf{X}') \\
&+ \frac{1}{C(0)4^{L+1}}\left(\prod_{k=0}^{L+1}\widetilde{\mathbf{W}}^{(L+1-k)}\right)\left(\sum_{i=0}^{C^{(0)}}\mathbf{X}_i\odot\mathbf{X}'_i\right) \\
&= \frac{2}{\pi}\frac{1}{2^{L+1}}\frac{1}{2}\widetilde{\mathbf{W}}^{(L+1)}\mathbf{P}^{(L+1-1)}(\mathbf{X},\mathbf{X}') \\
&+ \frac{2}{\pi}\frac{1}{2^{L+1}}\sum_{k=2}^{L+1}\frac{1}{2^k}\left(\prod_{i=0}^{k-1}\widetilde{\mathbf{W}}^{(L+1-i)}\right)\mathbf{P}^{(L+1-k)}(\mathbf{X},\mathbf{X}') \\
&+ \frac{1}{C(0)4^{L+1}}\left(\prod_{k=0}^{L+1}\widetilde{\mathbf{W}}^{(L+1-k)}\right)\left(\sum_{i=0}^{C^{(0)}}\mathbf{X}_i\odot\mathbf{X}'_i\right) \\
&= \frac{2}{\pi}\frac{1}{2^{L+1}}\sum_{k=1}^{L+1}\frac{1}{2^k}\left(\prod_{i=0}^{k-1}\widetilde{\mathbf{W}}^{(L+1-i)}\right)\mathbf{P}^{(L+1-k)}(\mathbf{X},\mathbf{X}') \\
&+ \frac{1}{C^{(0)}4^{L+1}}\left(\prod_{k=0}^{L+1}\widetilde{\mathbf{W}}^{(L+1-k)}\right)\left(\sum_{i=0}^{C^{(0)}}\mathbf{X}_i\odot\mathbf{X}'_i\right),
\end{aligned}
$$

that is, the hypothesis holds true for $L+1$, establishing the induction step. Therefore by induction, the theorem holds.

□

### 3.2.3 Upper Bound

Now we want to derive an upper bound of the kernel function, which, as you will see later, is the more important bound. Theorem 4 describes the recursive formula of the upper bound.

**Theorem 4.** *Given a L-layer ReLU CNNGP without bias-variance, i.e., $\sigma_b^2 = 0$, an upper bound of the corresponding kernel function $\mathbf{K}^{(L)}(\mathbf{X},\mathbf{X}')$ for all $\mathbf{X},\mathbf{X}' \in \mathbb{R}^{(H^{(0)}D^{(0)})\times C^{(0)}}$ is given by*

$$
\mathbf{K}_{up}^{(L)}(\mathbf{X},\mathbf{X}') = \widetilde{\mathbf{W}}^{(L)}\Big(\frac{1}{4}\frac{1}{2^{L-1}}\mathbf{P}^{(L-1)}(\mathbf{X},\mathbf{X}') + \frac{1}{4}\mathbf{K}_{low}^{(L-1)}(\mathbf{X},\mathbf{X}')\Big),
$$

*with base case $\mathbf{K}_{up}^{(0)}(\mathbf{X},\mathbf{X}') = \mathbf{K}^{(0)}(\mathbf{X},\mathbf{X}')$.*

*Proof.* Since $f$ is an element-wise monotone increasing convex function for all $\rho_i \in [-1, 1]$ we can upper bound it with the straight line $u$ from $(-1, 0)$ to $(1, \pi)$, that is, $u(\rho) = \frac{\pi}{2}\mathbf{1} + \frac{\pi}{2}\rho$ for $\rho_i \in [-1, 1]$.
Therefore a first upper bound is given by

$$\mathbf{K}^{(L)}_{\text{up,inter}}(\mathbf{X}, \mathbf{X}') = \widetilde{\mathbf{W}}^{(L)}\Big(\frac{1}{4}\frac{1}{2^{L-1}}\mathbf{P}^{(L-1)}(\mathbf{X}, \mathbf{X}') + \frac{1}{4}\mathbf{K}^{(L-1)}(\mathbf{X}, \mathbf{X}')\Big).$$

We can easily show by induction that $\mathbf{K}^{(l)}(\mathbf{X}, \mathbf{X}') \leq \mathbf{K}^{(l)}_{\text{up,inter}}(\mathbf{X}, \mathbf{X}')$ for all $l \leq L$. Since $\mathbf{K}^{(L)}_{\text{up,inter}}$ is element-wise monotone increasing in $\mathbf{K}^{(L-1)}$ it follows $\mathbf{K}^{(L)}_{\text{up}} \geq \mathbf{K}^{(L)}_{\text{up,inter}}$, which proofs the result. □

Again, we can again rewrite this recursive formula as an iterative sum.

**Theorem 5.** *Given a L-layer ReLU CNNGP without bias-variance, i.e., $\sigma_b^2 = 0$, and the upper bound $\mathbf{K}^{(L)}_{up}$ of its kernel function, then for all $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{(H^{(0)}D^{(0)}) \times C^{(0)}}$*

$$\begin{aligned}\mathbf{K}^{(L)}_{up}(\mathbf{X}, \mathbf{X}') = \frac{1}{2^L}\sum_{k=1}^{L}\frac{1}{2^k}\left(\prod_{i=0}^{k-1}\widetilde{\mathbf{W}}^{(L-i)}\right)\mathbf{P}^{(L-k)}(\mathbf{X}, \mathbf{X}') \\ + \frac{1}{C^{(0)}4^L}\left(\prod_{k=0}^{L}\widetilde{\mathbf{W}}^{(L-k)}\right)\left(\sum_{i=0}^{C^{(0)}}\mathbf{X}_i \odot \mathbf{X'}_i\right).\end{aligned}$$

*Proof.* The result can be easily shown by induction.
The proof can be performed analogously to theorem 3. □

An interesting observation we can make is that the second term of the upper bound is the same as the second term of the lower bound. Furthermore, the first term is also very similar only a factor of $\frac{2}{\pi}$ distinguishes both. As a result, the type of scaling of the kernel function as determined via the upper and lower bound for a large number of layers can be roughly determined by using just one of these bounds.

## 3.3 Dataset and Methods

We want to verify the result of the last two sections. To do that, we first introduce the type of convolutional GP and the data set used in the experimental process.

### 3.3.1 MNIST Dataset

The dataset used for our experiments is the MNIST data set [19]. It is a set of $28 \times 28$ gray-scale images of handwritten digits. These images are labeled with the corresponding

digit represented in the image. Since it is grey-scaled we have therefore 784 input features per prediction and a single output corresponding to the digit represented in the picture. The total data set consists of about 70.000 samples. These samples are usually split up into roughly 60.000 samples for training and 10.000 samples for testing. Most of our calculations will require a lot of training time since the calculation of GP kernel matrices requires extensive computational resources. Because of that, we will reduce the size of the training set substantially depending on the task at hand. The only preprocessing we will apply is scaling the images originally ranging from 0 to 255 to the range $[0, 1]$. Transformations like centering will not be applied. In Figure 4, you will find a few samples of the used data set.

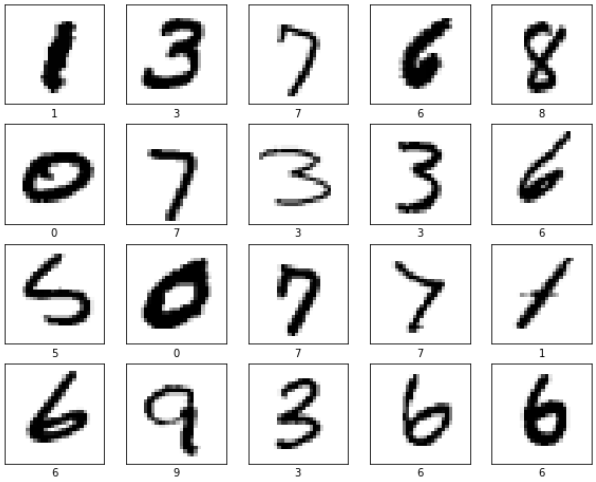


Figure 4: Sample of the MNIST data set [19].

### 3.3.2 ConvNet-GP-$L$

In this section, we will introduce the type of CNNGP used in this thesis. The architecture of the underlying CNNs is very simple and based on the ConvNet-GP architecture introduced in [9].

The CNN architecture described in [9] has seven convolution layers, where each convolutional operation uses a $7 \times 7$ filter. The input data of each layer is padded with zeros such that the output dimension is equal to the input dimension, i.e., zero-padding is applied such that the input and output dimension is $28 \times 28$. Each of the convolutions is followed by a ReLU nonlinearity. The final layer is a fully-connected layer with no nonlinearity following. To implement this we use a $28 \times 28$ filter with no padding. The GP based on this architecture is called ConvNet-GP. For classification, the labels are one-hot-negative encoded, that is, $\mathbf{y}_i \in \{-1, 1\}^{10}$. No transformations like a Softmax-activation function are used. Instead, we perform just a multi-class regression as described in 2.2.2.

Since we want to vary the number of layers used, we call a convolutional NNGP with $L$

layers which has a ConvNet-GP-based architecture ConvNet-GP-$L$, e.g., ConvNet-GP-7 is equivalent to ConvNet-GP.

We now want to derive some properties for a ConvNet-GP-$L$. First of all, since all layers except the last perform the same operation, the first $L$ Toeplitz matrices with removed padding columns are equal and we write $\widetilde{\mathbf{W}}$ for the first $L$ matrices, that is, $\widetilde{\mathbf{W}}^{(i)} = \widetilde{\mathbf{W}}$ for all $i \in \{0, 1, ..., L-1\}$. The last layer is a fully-connected layer with a Toeplitz matrix that is just a row vector with only $\sigma_\omega^2$ entries denoted by $\widetilde{\mathbf{W}}^{(L)}$. Furthermore, since the MNIST Dataset has only one channel, $C^{(0)} = 1$. Based on this, we can simplify the bound equations. First, we rewrite the $l$-level transformed covariance as

$$\mathbf{P}^{(l)}(\mathbf{X}, \mathbf{X}') = \sqrt{\Big(\widetilde{\mathbf{W}}^{l+1}\Big(\mathbf{X} \odot \mathbf{X}\Big)\Big) \odot \Big(\widetilde{\mathbf{W}}^{l+1}\Big(\mathbf{X}' \odot \mathbf{X}'\Big)\Big)}, \tag{35}$$

where the matrix power is not applied element-wise. For the lower bound, we can now write

$$\begin{aligned}\mathbf{K}_{\text{low}}^{(L)}(\mathbf{X}, \mathbf{X}') = \frac{2}{\pi}\frac{1}{2^L}\sum_{k=1}^{L}\frac{1}{2^k}\widetilde{\mathbf{W}}^{(L)}\widetilde{\mathbf{W}}^{k-1}\mathbf{P}^{(L-k)}(\mathbf{X}, \mathbf{X}') \\ + \frac{1}{4^L}\widetilde{\mathbf{W}}^{(L)}\widetilde{\mathbf{W}}^{L}\Big(\mathbf{X} \odot \mathbf{X}'\Big),\end{aligned} \tag{36}$$

and for the upper bound

$$\begin{aligned}\mathbf{K}_{\text{up}}^{(L)}(\mathbf{X}, \mathbf{X}') = \frac{1}{2^L}\sum_{k=1}^{L}\frac{1}{2^k}\widetilde{\mathbf{W}}^{(L)}\widetilde{\mathbf{W}}^{k-1}\mathbf{P}^{(L-k)}(\mathbf{X}, \mathbf{X}') \\ + \frac{1}{4^L}\widetilde{\mathbf{W}}^{(L)}\widetilde{\mathbf{W}}^{L}\Big(\mathbf{X} \odot \mathbf{X}'\Big).\end{aligned} \tag{37}$$

An interesting observation we can make is that each sum element of the first term, as well as the second term, contributes roughly $\widetilde{\mathbf{W}}^L$, although the transformations applied in $\mathbf{P}^{(l)}$ skew that a bit. Furthermore, the resulting bounds are scalar values since $\widetilde{\mathbf{W}}^{(L)}$ is a row vector and $\mathbf{P}^{(l)}$ a column vector.

$\widetilde{\mathbf{W}}$ is a positive real quadratic symmetric matrix that results in two interesting properties: First, there is a real orthogonal matrix $\mathbf{Q}$ and a real diagonal matrix $\mathbf{\Lambda}$ such that $\widetilde{\mathbf{W}} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$. And second, $\widetilde{\mathbf{W}}^k = \mathbf{Q}\mathbf{\Lambda}^k\mathbf{Q}^\top$.

The scaling of $\widetilde{\mathbf{W}}^L$ with respect to $L$ is therefore mostly determined by its spectral radius $\rho(\widetilde{\mathbf{W}}) = \max\{|\lambda| : \lambda \text{ is eigenvalue of } \widetilde{\mathbf{W}}\}$, which is equal to the highest eigenvalue $\lambda_{\max} \approx 46.9\sigma_\omega^2$ for matrices with only positive entries. In case of $\widetilde{\mathbf{W}}$, the eigenspace corresponding to the eigenvalue $\lambda_{\max}$ is one dimensional. As a result, for high values of $L$, $\widetilde{\mathbf{W}}$ is approximately

$$\widetilde{\mathbf{W}}^L \approx \lambda_{\max}^L \mathbf{q}_1\mathbf{q}_1^\top, \tag{38}$$

where $\mathbf{q}_1$ is the normalized eigenvector corresponding to $\lambda_{\max}$.
Based on the results of the last paragraph, we can roughly approximate the upper bound for very high values of L as follows:

$$\begin{aligned}
\mathbf{K}_{\text{up}}^{(L)}(\mathbf{X}, \mathbf{X}') &\approx \frac{\lambda_{\max}^L}{2^L} \sum_{k=1}^{L} \frac{1}{2^k} \widetilde{\mathbf{W}}^{(L)} \mathbf{q}_1 \sqrt{\mathbf{q}_1^\top \Big(\mathbf{X} \odot \mathbf{X}\Big) \cdot \mathbf{q}_1^\top \Big(\mathbf{X}' \odot \mathbf{X}'\Big)} \\
&+ \frac{\lambda_{\max}^L}{4^L} \widetilde{\mathbf{W}}^{(L)} \mathbf{q}_1 \mathbf{q}_1^\top \Big(\mathbf{X} \odot \mathbf{X}'\Big) \\
&= \frac{\lambda_{\max}^L}{2^L} (1 - \frac{1}{2^L}) \widetilde{\mathbf{W}}^{(L)} \mathbf{q}_1 \sqrt{\mathbf{q}_1^\top \Big(\mathbf{X} \odot \mathbf{X}\Big) \cdot \mathbf{q}_1^\top \Big(\mathbf{X}' \odot \mathbf{X}'\Big)} \\
&+ \frac{\lambda_{\max}^L}{4^L} \widetilde{\mathbf{W}}^{(L)} \mathbf{q}_1 \mathbf{q}_1^\top \Big(\mathbf{X} \odot \mathbf{X}'\Big) \\
&\approx \frac{\lambda_{\max}^L}{2^L} \widetilde{\mathbf{W}}^{(L)} \mathbf{q}_1 \sqrt{\mathbf{q}_1^\top \Big(\mathbf{X} \odot \mathbf{X}\Big) \cdot \mathbf{q}_1^\top \Big(\mathbf{X}' \odot \mathbf{X}'\Big)}.
\end{aligned} \tag{39}$$

Essentially, the upper bound degenerates to a very simple expression only dependent on the first eigenvector of the Toeplitz matrix.

## 3.4 Computational Results

In this section, we want to test the derivations made in the previous segments of this thesis. All computations will be performed using the ConvNet-GP-$L$ architectures on an i7-6700k Intel CPU. Since the calculations are usually made for only small amounts of data, the used hardware does not play any limiting role in calculating the kernel matrix. We will first justify why the derivation was made with a zero bias-variance. Secondly, we will show the correctness of the derived bounds. After that, we will investigate the convergence of the kernel function and determine the usefulness of the approximations made in equation (39). At last, we want to determine how these results can be used to predict the accuracy of ConvNet-GP-$L$ with a large number of layers.

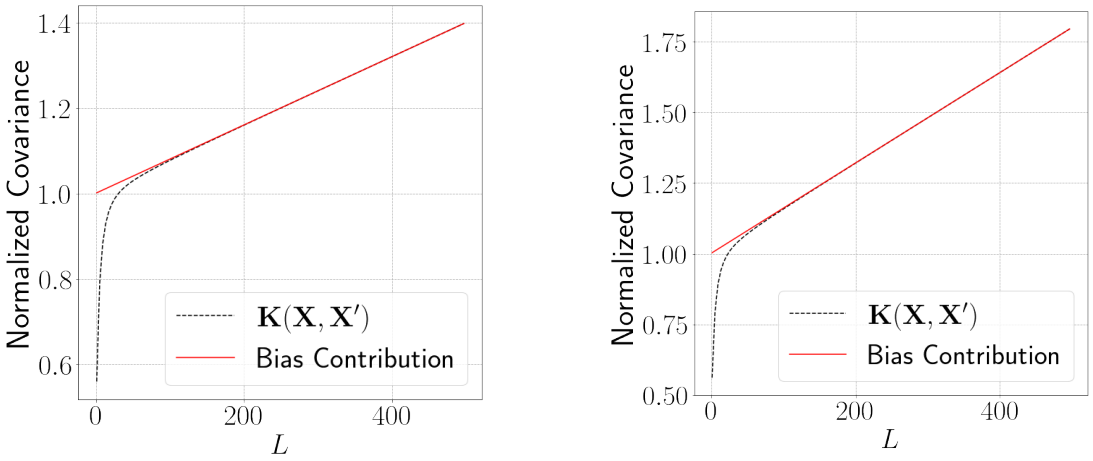
### 3.4.1 Bias Contribution

In the previous section, derivations were made without the use of a bias-variance. In this section, we want to justify why these results are also useful for biased ConvNet-GP-$L$ architectures.
Interestingly, we can observe that a bias-variance introduces a linear component to the scaling behavior of the kernel function as visible in Figure 5. In case weight-variance and bias-variance are of similar magnitude, this linear contribution only plays a minor role in the scaling behavior of the kernel function. However, if $\sigma_\omega^2 \ll \sigma_b^2$ the contribution is clearly visible. To make it more noticeable, $\sigma_\omega^2$ with some abuse of notation is chosen to be

$\sigma_\omega^2 = \frac{2}{\lambda_{\max}(\widetilde{\mathbf{W}}(\sigma_\omega^2=1))}$. As we will see later, in this case, a zero bias-variance system converges against a constant value. Furthermore, we normalize the covariance with respect to this constant such that the bias contribution becomes more clear.

The approximately linear bias contribution is a function described by roughly $b(L) = a \cdot \sigma_b^2 \cdot L$, where $a$ is dependent on most of the parameters determining the ConvNet-GP-$L$ architecture except $L$.

In conclusion, if weight-variance and bias-variance are of similar magnitude we can mostly ignore the bias contribution. Choosing weight-variance and bias-variance to be of similar magnitude is a common choice performed also in [9]. Moreover, if $\sigma_\omega^2 \ll \sigma_b^2$, the bias contribution can simply be modeled by a linear contribution $b(L)$. It can be therefore justified to only look at the scaling behavior of a zero bias-variance system.




(a) Bias-variance $\sigma_b^2 = 20$

(b) Bias-variance $\sigma_b^2 = 40$

Figure 5: The dotted black line represents normalized covariance of ConvNet-GP-$L$ for some $\mathbf{X} \neq \mathbf{X}'$ with weight-variance $\sigma_\omega^2 \approx \frac{2}{46.9}$ and with respective bias-variance with respect to number of layers $L$. The red line represents the linear contribution of the bias-variance determined via linear regression of covariance values for $L > 200$.

### 3.4.2 Validity of Bounds

Now we want to verify the correctness of both bound theorems 3 and 5. In Figure 6, one can see the bounds on the kernel function $\mathbf{K}^{(L)}(\mathbf{X}, \mathbf{X}')$ for some $\mathbf{X} \neq \mathbf{X}'$ as a fraction of the upper bound. After a short convergence period, the normalized lower bound converges to $\frac{2}{\pi}$ which is exactly what was to be expected, i.e., the first term of both bounds in their series representation dominates for large $L$, which differ by a factor of $\frac{2}{\pi}$. Furthermore, the kernel function stays within the determined bounds of the system. Interestingly, the kernel function also seems to be converging against the upper bound. Similar observations can be made for a non-zero bias-variance ConvNet-GP-$L$ system shown in Figure 7.

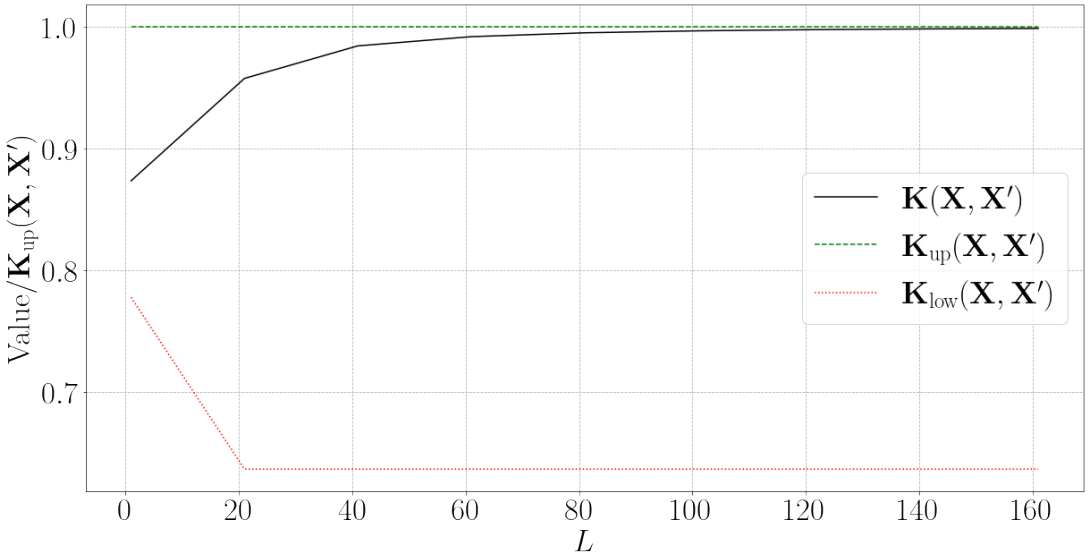



Figure 6: The straight black line shows the covariance of ConvNet-GP-$L$ for some $\mathbf{X} \neq \mathbf{X}'$ with weight-variance $\sigma_\omega^2 = 0.25$ and with zero bias-variance divided by the upper bound with increasing $L$. The dotted red line represents the lower bound as a fraction of the upper bound. The green dashed line shows the upper bound divided by the upper bound itself which is therefore always 1.

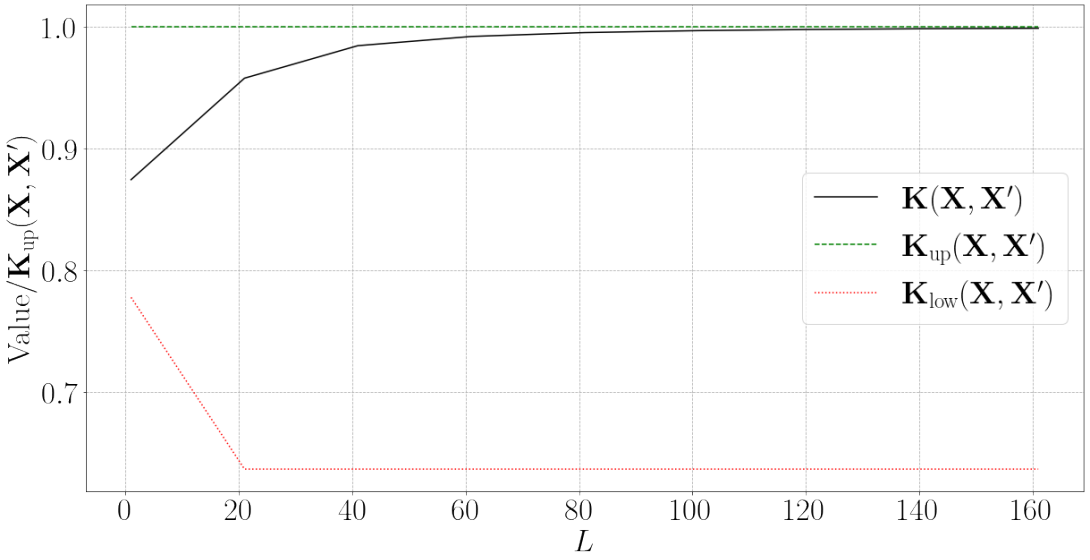



Figure 7: The straight black line shows the covariance of ConvNet-GP-$L$ for some $\mathbf{X} \neq \mathbf{X}'$ with weight-variance $\sigma_\omega^2 = 0.25$ and with $\sigma_b^2 = 20$ divided by the upper bound with increasing $L$. The dotted red line represents the lower bound as a fraction of the upper bound. The green dashed line shows the upper bound divided by the upper bound itself which is therefore always 1.

### 3.4.3 Convergence Behavior of the Kernel Function

We now want to study the convergence behavior of the kernel function in more detail. As we have seen in the last section, the kernel function seems to converge against the upper bound stated in theorem 4. As visible in Figure 8, for $\sigma_\omega^2 = \frac{2}{\lambda_{\max}(\widetilde{\mathbf{W}}(\sigma_\omega^2=1))}$ the upper bound resembles - after a short convergence phase - a constant function. This constant is approached by the kernel function for large values of $L$.
A question that comes to mind: Is the GP predictor based on calculations of the kernel matrix via the upper bound delivering similar results as the corresponding ConvNet-GP-$L$ predictor for large $L$? Because it is surely possible that small differences in the values of the kernel matrix result in large differences in predictions, this should be investigated.
To better analyze this, we developed GP predictors with kernel functions calculated via the upper bound and compared the predictions with the respective ConvNet-GP-$L$ predictor. Clearly, as shown in Figure 9, the convergence for increasing $L$ holds equally true for the corresponding predictions.
To conclude, for large $L$ the kernel function as well as the predictor seem to converge against its equivalent calculated via the upper bound.

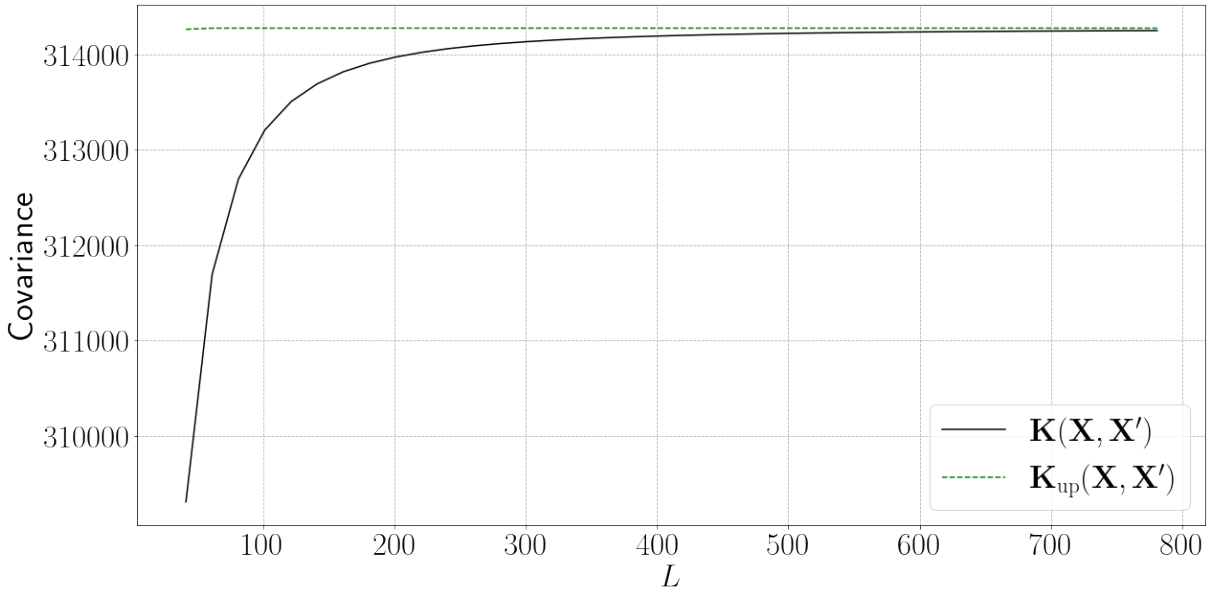
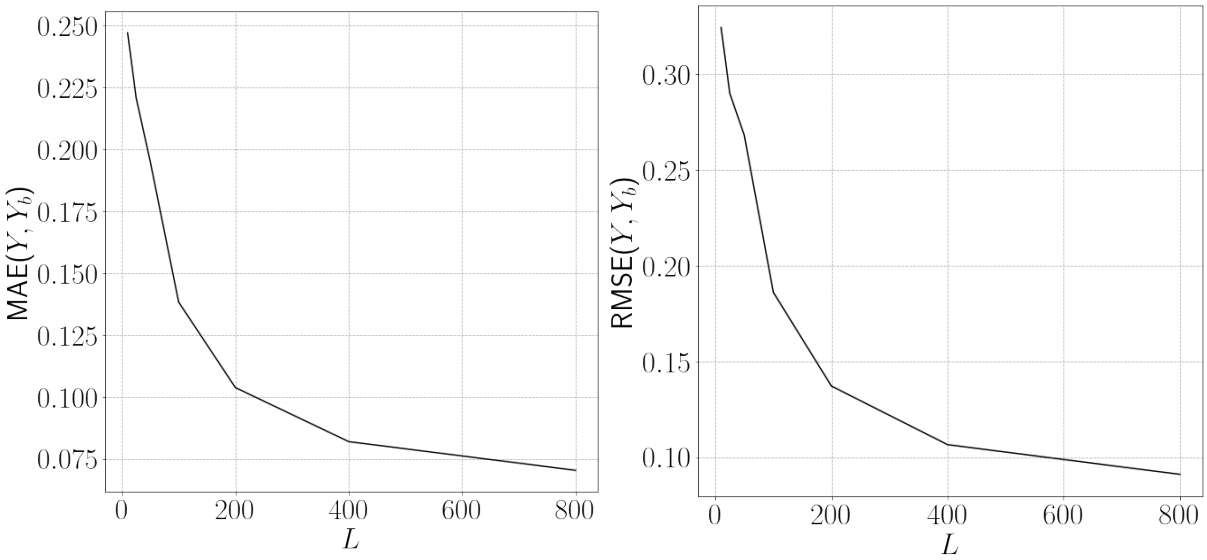



Figure 8: The straight black line shows the covariance of ConvNet-GP-$L$ for some $\mathbf{X} \neq \mathbf{X}'$ with weight-variance $\sigma_\omega^2 = \frac{2}{\lambda_{\max}(\widetilde{\mathbf{W}}(\sigma_\omega^2=1))}$ and zero bias-variance with increasing $L$. The the dashed green line represents the corresponding upper bound.




(a) Mean Absolute Error

(b) Root Mean Squared Error

Figure 9: (a) MAE between the prediction $Y$ of using the ConvNet-GP-$L$ predictor and the prediction $Y_b$ of using the GP predictor based on the upper bound. (b)RMSE between the prediction $Y$ of using the ConvNet-GP-$L$ predictor and the prediction $Y_b$ of using the GP predictor based on the upper bound.

### 3.4.4 Convergence Limit

As we have seen in the last section, the kernel function seems to converge against the upper bound as derived in theorem 4. Although the validity of this convergence should be mathematically proven, we will assume that the convergence holds true to at least some extent as demonstrated by the experimental data. But what are the conclusions we can formulate from this?

As we have derived in equation (39), the upper bound can be approximated for large values of $L$ as follows:

$$\mathbf{K}_{\text{up}}^{(L)}(\mathbf{X}, \mathbf{X}') \approx \mathbf{K}_{\text{approx,up}}^{(L)}(\mathbf{X}, \mathbf{X}') = \frac{\lambda_{\max}^{L}}{2^L} \widetilde{\mathbf{W}}^{(L)} \mathbf{q}_1 \sqrt{\mathbf{q}_1^\top \Big( \mathbf{X} \odot \mathbf{X} \Big) \cdot \mathbf{q}_1^\top \Big( \mathbf{X}' \odot \mathbf{X}' \Big)}. \tag{40}$$

As visible in Figure 10, the upper bound converges rather quickly against this approximation from below. For large $L$, we can therefore use this approximation to describe the upper bound. This translates into a good convergence of the true kernel function against the approximation of the upper bound as shown in Figure 11.

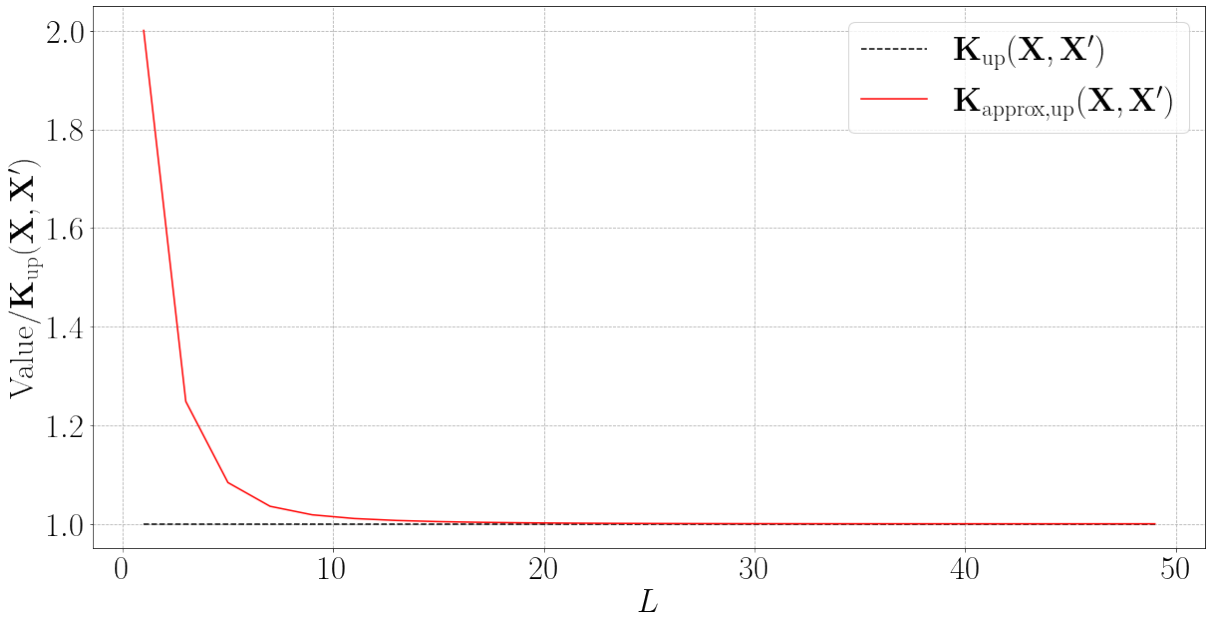



Figure 10: Dotted black line describes the upper bound divided by itself for $\sigma_\omega^2 = \frac{2}{\lambda_{\max}(\widetilde{\mathbf{W}}(\sigma_\omega^2=1))}$. The straight red line corresponds to the approximation divided by the upper bound. The approximation converges to the upper bound of the kernel function from above.

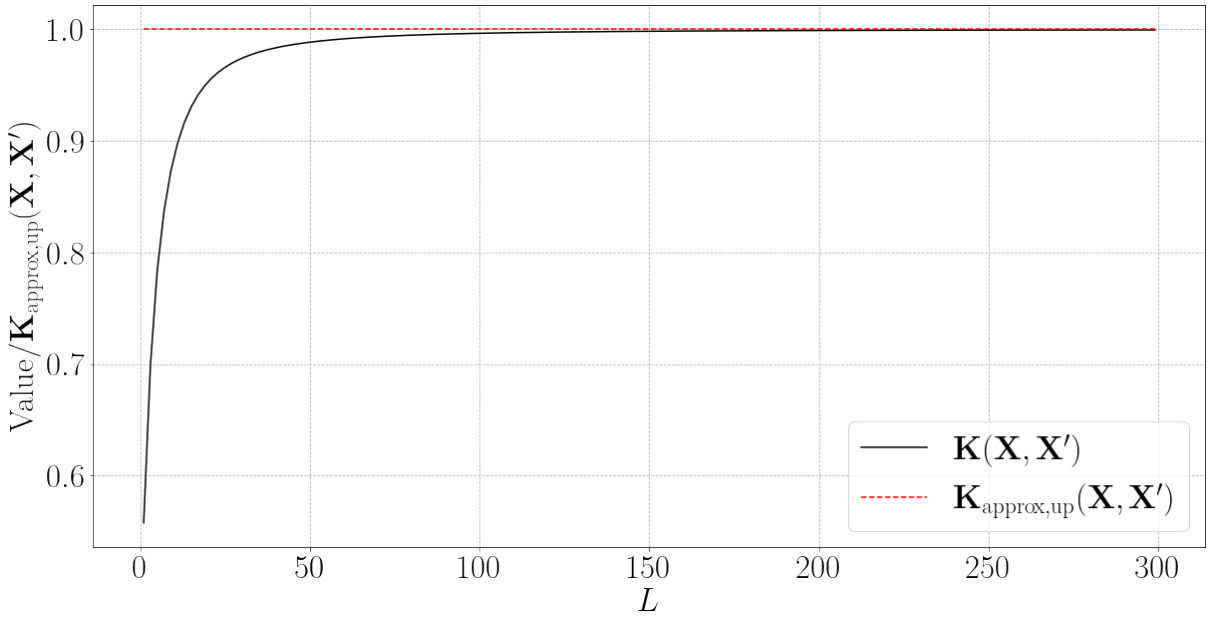



Figure 11: Black line describes the upper bound divided by the approximation for $\mathbf{X} \neq \mathbf{X}'$ and $\sigma_\omega^2 = \frac{2}{\lambda_{\max}(\widetilde{\mathbf{W}}(\sigma_\omega^2=1))}$. The dashed red line corresponds to the approximation divided by itself.

So we can assume with some certainty that the kernel function converges against the approximation (39). The first observation we can make is that only the prefactor of (39) is dependent on $L$. As has been stated earlier, if $\sigma_\omega^2 = \frac{2}{\lambda_{\max}(\widetilde{\mathbf{W}}(\sigma_\omega^2=1))}$ the approximation becomes a constant independent of $L$.
Furthermore, we also know that the predictor itself is independent of any prefactors even if the weight-variance is not chosen such that the prefactor is independent of $L$. Therefore, the relevant part of the approximation reduces to

$$\mathbf{K}^{(L)}_{\text{approx, rel}}(\mathbf{X}, \mathbf{X}') = \sqrt{\mathbf{q}_1^\top \Big(\mathbf{X} \odot \mathbf{X}\Big) \cdot \mathbf{q}_1^\top \Big(\mathbf{X}' \odot \mathbf{X}'\Big)}. \tag{41}$$

Since this is the relevant part sufficient for describing the predictor for large values of $L$, it might be beneficial to examine this formula a little more. First, we can see that under the square root two inner product operations are performed, i.e., we take the standard inner product of the square of $\mathbf{X}$ with $\mathbf{q}_1$ and we take the standard inner product of the square of $\mathbf{X}'$ with $\mathbf{q}_1$. The inner product calculation essentially behaves as a similarity measure, that is, the more similar the squared vectors are to $\mathbf{q}_1$ the higher the corresponding values of the inner product. After the two inner products are calculated, the geometric mean of the two is determined resulting in $\mathbf{K}^{(L)}_{\text{approx, rel}}(\mathbf{X}, \mathbf{X}')$. So to summarize, we compare both input images with the image shown in Figure 12 and take their geometric mean.
As visible in Figure 12, the eigenvectors measure very different properties of the image. The 2D representation of $\mathbf{q}_1$ is a nearly radial symmetric image with its highest point in the middle and decreasing with distance to this center. The other lower-level eigenvectors do not play any role for large values of $L$. As a result, this lower level, finer-grained information is lost and values in the center of the input images are just promoted via $\mathbf{q}_1$.

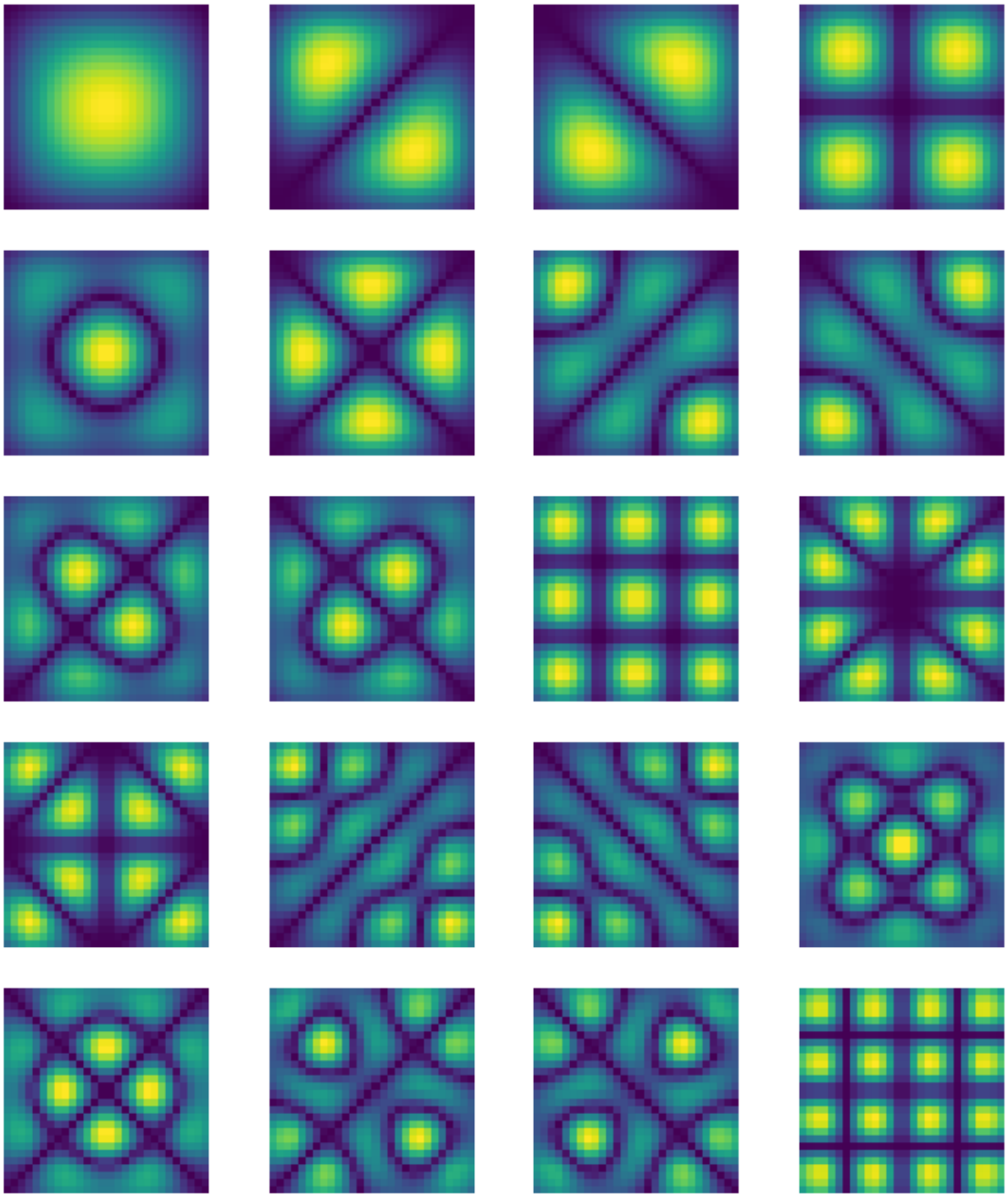


Figure 12: Representation of the first 20 normalized eigenvectors corresponding to the largest absolute eigenvalues of $\widetilde{\mathbf{W}}$ reshaped as a $28 \times 28$ 2D representation ordered by decreasing absolute eigenvalue in row-major order starting with $\mathbf{q}_1$ in the left upper corner.

### 3.4.5 Accuracy

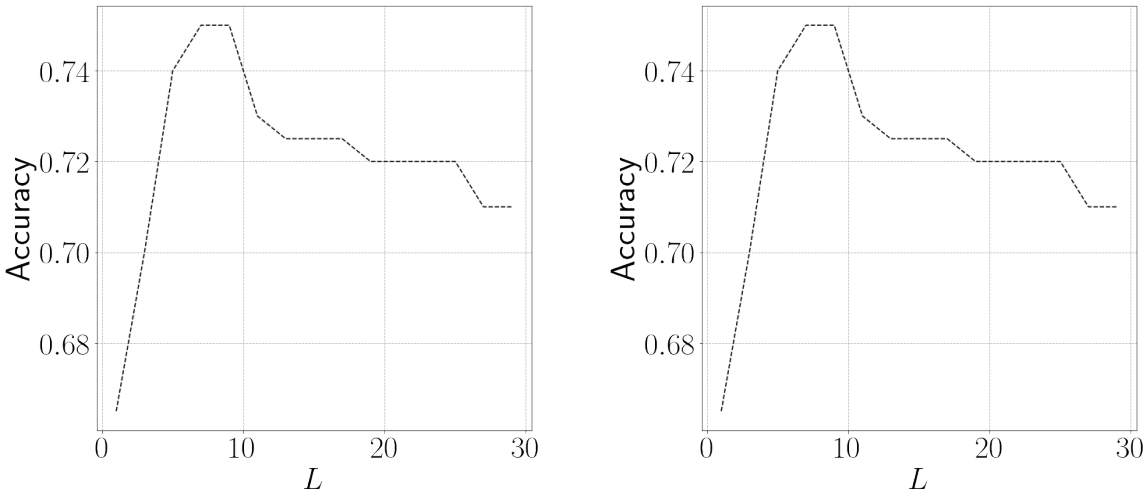
At last, we want to analyze how this convergence behavior translates into accuracy change. As visible in Figure 13, the accuracy first increases until 7 layers and after that peak is achieved decreases with an increasing number of layers in both cases. Again we can see that the bias-variance has close to no influence on the GP prediction as seen by the full overlap of both graphs.

The first increase in accuracy can probably be explained by the fact that too shallow convolutional systems are not able to capture features at various levels of abstraction [20]. At the peak, the number of layers seems to be enough to allow the ConvNet-GP-7 to understand these features. However, we see a decrease in accuracy after this peak has been reached. The convergence limit of the ConvNet-GP-$L$ system explored in the last sections might be an explanation for this.

As we have seen in the last section, the true kernel function of a zero bias-variance system seems to converge against $\mathbf{K}^{(L)}_{\text{approx,up}}(\mathbf{X}, \mathbf{X}')$. Since $\sigma_\omega^2$ and $\sigma_b^2$ are of similar magnitude, this property can be adopted also for the system with non-zero bias-variance.

Incidentally, $\mathbf{K}^{(L)}_{\text{approx,up}}(\mathbf{X}, \mathbf{X}')$ is a very low information kernel function only capable of capturing similarity features of the inputs using $\mathbf{q}_1$. This might be the reason for the poor performance of the corresponding ConvNet-GP-$L$ for large $L$.

To summarize, a large number of layers with convolution operations seems to blur low-level interaction information between $\mathbf{X}$ and $\mathbf{X}'$ which then deteriorates the performance of the kernel machine.




(a) $\sigma_\omega^2 = 2.79$ and $\sigma_b^2 = 7.86$ (b) $\sigma_\omega^2 = 2.79$ and $\sigma_b^2 = 0$

Figure 13: Accuracy of ConvNet-GP-$L$ with $\sigma_\omega^2 = 2.79$ and $\sigma_b^2 = 7.86$ as proposed by [9] train on 100 MNIST images with increasing number of layers $L$.

# 4 Conclusion and Outlook

In this thesis, we have investigated Gaussian processes derived from Bayesian infinite channel convolutional networks. We first introduced the convolutional networks as well as Gaussian processes and presented how convolutional neural network Gaussian processes (CNNGP) can be derived from infinitely wide Bayesian convolutional networks in section 2. In the main part of the thesis, the focus was specifically on the scaling behavior of kernel functions of CNNGPs. In section 3.1 and 3.2, we first introduced a formalism to more efficiently describe the kernel function of CNNGPs using constant Toeplitz matrices. Using this formalism, upper and lower bounds on the kernel function for zero bias-variance systems were derived and proofed resulting in a simple, easy-to-handle sum representation of the bounds. In the computational section 3.4, we tested the derived theorems and analyzed the convergence behavior of the kernel function with respect to the number of layers experimentally.

As a starting point, we first tried to justify the neglect of bias-variance in our derivations. By plotting the kernel function of a ConvNet-GP-$L$ architecture with and without bias-variance, we saw that non-zero bias-variance contributed only a small linear contribution added to the kernel function. This contribution was neglectable for bias- and weight-variance with the same order of magnitude. Based on this, we verified the correctness of both bounds for the case of the ConvNet-GP-$L$ architecture. To do this, we analyzed the scaling of the kernel function ConvNet-GP-$L$ against the scaling of both upper and lower bounds. During this process, we made the interesting observation that the kernel function seems to converge against the derived upper bound for a large number of layers $L$. This fact influenced us to examine the upper bound more closely. We showed that for the ConvNet-GP-$L$ architecture, the upper bound itself degenerates to a very simple expression for large $L$ and essentially matches this expression already at about 20 layers ($L = 20$). This simple expression is determined by the geometric mean of the inner product of the normalized eigenvector of the largest eigenvalue of the corresponding Toeplitz matrix with the squared input image $\mathbf{X}$ and the squared input image $\mathbf{X}'$. This degenerate kernel function promotes high values for images with a strong central focus and is not sufficiently useful as a similarity measure between the two images. At last, we tried to translate this convergence behavior into predictions about the accuracy of the ConvNet-GP-$L$ architecture for large values of $L$. Incidentally, the accuracy of a ConvNet-GP-$L$ predictor degenerates for a large number of layers. We proposed that the convergence of the kernel function against the degenerate form of the upper bound can be used as an explanation for this phenomenon. Although, further research is needed to validate this claim.

There is still a lot of research to be done. First of all, even though we somewhat justified the zero bias-variance as a condition for the validity of the bounds, it would still be interesting to extend these bounds to CNNGPs with non-zero bias-variance rigorously. Furthermore, since the witnessed bias contributions were minor at best, the actual effect of bias-variance would be an interesting question to analyze. In addition to this, other types of layers should also be evaluated and their contributions should be investigated. A residual or a batch normalization layer might be able to avoid some convergence effects witnessed in this thesis. Furthermore, experiments should also be extended to other convolution-based architectures different from ConvNet-GP-$L$ to analyze whether convergence and accuracy results can be generalized. Last and most importantly, it should be analyzed how the convergence results for the CNNGP can be used for a better understanding of the underlying convolutional networks.

# References

[1] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, et al. Alphafold 2, 2020.

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

[3] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[4] Wei Wang, Yujing Yang, Xin Wang, Weizheng Wang, and Ji Li. Development of convolutional neural network and its application in image classification: a survey. *Optical Engineering*, 58(4):040901, 2019.

[5] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[6] Mark Ebden. Gaussian processes: A quick introduction. *arXiv preprint arXiv:1505.02965*, 2015.

[7] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[8] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[9] Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes. *arXiv preprint arXiv:1808.05587*, 2018.

[10] Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *Advances in Neural Information Processing Systems*, 32, 2019.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Michael W Marcellin, Michael J Gormish, Ali Bilgin, and Martin P Boliek. An overview of jpeg-2000. In *Proceedings DCC 2000. Data Compression Conference*, pages 523–541. IEEE, 2000.

[14] Ali Salehi. 2d convolution as matrix multiplication using toeplitz matrices. 2018.

[15] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[16] Michael Mongillo et al. Choosing basis functions and shape parameters for radial basis function methods. *SIAM undergraduate research online*, 4(190-209):2–6, 2011.

[17] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[18] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

[19] THE MNIST DATABASE handwritten digits. `http://yann.lecun.com/exdb/mnist/`. Accessed: 2022-05-23.

[20] Ivars Namatēvs. Deep convolutional neural networks: Structure, feature extraction and training. *Information Technology and Management Science*, 20(1):40–47, 2017.

# List of Figures