

Daniel Lukic, BSc

# **Convolutional Neural Networks as Kinetic Energy Predictors in Orbital-free Density Functional Theory**

## **MASTER'S THESIS**

to achieve the university degree of  
Diplom-Ingenieur

Master's degree programme:  
Technical Physics

submitted to  
**Graz University of Technology**

### **Supervisor**

Assoc.Prof. Mag.phil. Dipl.-Ing. Dr.phil. Dr.techn. Andreas Hauser



# Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

04.04.2022

---

Date

---

Signature



# Acknowledgments

I would like to thank Prof. Martin Schulze, Prof. Andreas W. Hauser and Ralf Mayer for giving me the opportunity to work on this thesis in this stimulating environment, at the Institute of Experimental Physics.

My deepest appreciation goes to Prof. Andreas W. Hauser for his devoted supervision during this thesis, which also worked excellently during the Covid-19 pandemic. I am also grateful to have had the opportunity to give a talk at the joint annual meeting of the ÖPG and the SPS in 2021.

I would also like to thank smaXtec animal care GmbH for kindly providing additional computational infrastructure for some tasks regarding the training of the machine learning models.

Furthermore, I am very grateful to my parents for giving me the opportunity to study and for their unconditional support. I would also like to thank Katharina Kirchsteiger for her support, motivation and encouragement during this thesis.

# Abstract

An essential ingredient of orbital-free density functional theory is the kinetic energy functional. In 2012, Snyder et al. [26] introduced a machine learning approximation for the kinetic energy functional on a one-dimensional model system and reported a poor performance of their approach with respect to the quality of the functional derivative, resulting in the need computationally expensive workarounds to achieve chemical accuracy. In a recent work [20] we therefore included the functional derivative into the training of various machine learning approaches, which allowed for a substantial improvement on the error of the functional derivative. The quality of the latter is highly important since the usual task in DFT calculations is the minimization of the energy by a variation of the electronic density.

Among the methods we tested, the usage of convolutional neural networks (CNNs), an approach borrowed from image recognition, appeared as particularly promising. However, a remaining limitation here was that the CNN performed a discrete convolution on continuous data, which left room for further improvement. In an ongoing extension of this work we attempt to replace the discrete convolution by a continuous implementation to improve the accuracy. Several kinds of continuous functions are currently evaluated, which contain an intrinsic damping behaviour to compensate unphysical oscillations of the functional derivative near its spatial boundaries.



# Contents

<b>1</b>	<b>Background and Methology</b>	<b>1</b>
1.1	Density Functional Theory . . . . .	1
1.1.1	Hohenberg-Kohn theorems . . . . .	1
1.1.2	Kohn-Sham equations . . . . .	2
1.1.3	Orbital-free Density Functional Theory (OF-DFT) . . .	3
1.2	Convolutional Neural Networks . . . . .	5
1.2.1	The Convolution Operation . . . . .	5
1.2.2	Properties of CNNs . . . . .	6
1.2.3	Pooling . . . . .	8
1.2.4	Types of Convolution . . . . .	9
1.2.5	Dealing with Vanishing/Exploding Gradient Problem	9
1.2.6	Optimizers . . . . .	11
1.3	GPU vs CPU in Machine Learning . . . . .	12
1.4	Description of needed tools . . . . .	13
1.4.1	TensorFlow, CUDA and GPU support . . . . .	13
1.4.2	Setup . . . . .	14
<b>2</b>	<b>Neural Networks for DFT</b>	<b>15</b>
2.1	Data Generation . . . . .	15
2.2	Continuous Convolutional Neural Network - CCNN . . . . .	16
2.2.1	CCNN - Version 1 . . . . .	17
2.2.2	Dense CCNN - Version 1 . . . . .	17
2.2.3	CCNN - Version 2 . . . . .	17
2.3	Hyperparameters and Configurations . . . . .	18
2.3.1	Hyperparameter . . . . .	18
2.3.2	Configurations . . . . .	19
2.3.3	Hyperparameter Search . . . . .	19
2.4	UML Diagrams of Custom Code with TensorFlow . . . . .	21
<b>3</b>	<b>Results</b>	<b>29</b>
3.1	Results CCNN - Version 1 . . . . .	29
3.2	Results CCNN - Version 2 . . . . .	35
3.3	Results ResNet - Version 1 . . . . .	36
3.4	Results ResNet - Version 2 . . . . .	41
3.5	Results for Dense CCNN . . . . .	43



## Contents

---

<b>4 Conclusion</b>	<b>46</b>
4.1 Outlook . . . . .	47
<b>Bibliography</b>	<b>48</b>

# List of Figures

1.1	Representation of sparse connectivity. In the CNN ( <i>top</i> ) a convolution with a kernel size of 3 is performed and only 3 outputs are effected by $x_3$ , where in the traditional neural network ( <i>bottom</i> ) all outputs $s_{1,...,5}$ are effected by the matrix multiplication [14]. . . . .	7
1.2	Representation of parameter sharing. The use of a parameter in two different models is represented by a black arrow. The black arrow ( <i>Top</i> ) indicates that the single parameter is used in all input locations. The centered black arrow ( <i>Bottom</i> ) indicates the element of a fully connected model where the parameter is used only once [14]. . . . .	8
1.3	A block in a Residual Neural Network [10]. . . . .	10
1.4	Schematic chip architectures for a CPU and GPU [15] . . . . .	13
2.1	Kernel values of first layer with a size of 100 and a count of 32 kernes which are Glorot Normal distributed. . . . .	16
2.2	Plot of the kinetic energy derivative after the density calculated with the Numerov's method . . . . .	18
2.3	UML Diagram of Kernel_INITIALIZER classes for version 1 and 2 . . . . .	22
2.4	UML Diagram of Kernel_INITIALIZER and it's Inheritances . . . . .	22
2.5	UML Diagram of Costum Layer Classes for CCNN version 1 and 2 . . . . .	23
2.6	UML Diagram of Costum Layer Classes for CCNN version 1 . . . . .	24
2.7	UML Diagram of Costum Layer Classes for CCNN version 2 . . . . .	24
2.8	UML Diagram of CCNN Models Version 1 and 2 . . . . .	25
2.9	UML Diagram of CCNN Dense Models Version 1 . . . . .	26
2.10	UML Diagram of CCNN Models with combination of continuous and non-continuous layers Version 1 and 2 . . . . .	26
2.11	UML Diagram of ResNet Models Version 1 and 2 . . . . .	27
2.12	UML Diagram of CCNN Models Version 1 . . . . .	28
3.1	Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions. Left is before the training and right after the training for the runs in table 3.1. . . . .	33
a	Gaussian Kernel before training . . . . .	33
b	Gaussian Kernel after training . . . . .	33

## List of Figures

---

c	Harmonic Kernel before training . . . . .	33
d	Harmonic Kernel after training . . . . .	33
e	Trigonometric Kernel before training . . . . .	33
f	Trigonometric Kernel after training . . . . .	33
g	Glorot Uniform Kernel before training . . . . .	33
h	Glorot Uniform Kernel after training . . . . .	33
3.2	Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions for the runs in table 3.2 . .	34
a	Gaussian Kernel before training . . . . .	34
b	Gaussian Kernel after training for CCNNAlter . . . . .	34
c	Gaussian Kernel after training for CCNNBeginHalf . .	34
d	Gaussian Kernel after training for CCNNOnlyBegin .	34
e	Glorot Uniform Kernel before training . . . . .	34
f	Glorot Uniform Kernel after training for CCNNEndHalf	34
g	Glorot Uniform Kernel after training for CCNNOnlyEnd	34
3.3	Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions. Left is before the training and right after the training for the runs in table 3.6. . . . .	39
a	Gaussian kernel before training . . . . .	39
b	Gaussian kernel after training . . . . .	39
c	Trigonometric kernel before training . . . . .	39
d	Trigonometric kernel after training . . . . .	39
e	Glorot uniform kernel before training . . . . .	39
f	Glorot uniform kernel after training . . . . .	39
3.4	Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions for the runs in table 3.2. . .	40
a	Gaussian Kernel before training . . . . .	40
b	Gaussian Kernel after training for ResNetAlter . . . .	40
c	Gaussian Kernel after training for ResNetBeginHalf .	40
d	Gaussian Kernel after training for ResNetOnlyBegin .	40
e	Glorot Uniform Kernel before training . . . . .	40
f	Glorot Uniform Kernel after training for ResNetEndHalf	40
g	Glorot Uniform Kernel after training for ResNetOnlyEnd	40
3.5	Kernel plot of the first layer before training for the runs in table 3.9 . . . . .	42
a	Gaussian Kernel before training . . . . .	42
b	Gaussian Kernel after training . . . . .	42
3.6	Kernel plot of the first layer before and after training for the harmonic run in table 3.11 . . . . .	43
a	Harmonic Kernel before training . . . . .	43

## List of Figures

---

	b	Harmonic Kernel after training with a softplus activation function . . . . .	43
3.7		Kernel plot of the first layer before and after training for the Gaussian run in table 3.11 . . . . .	44
	a	Gaussian Kernel before training . . . . .	44
	b	Gaussian Kernel after training with a softplus activation function . . . . .	44
	c	Gaussian Kernel after training with a linear activation function . . . . .	44
3.8		Kernel plot of the first layer before and after training for the trigonometric run in table 3.11 . . . . .	45
	a	Trigonometric Kernel before training . . . . .	45
	b	Trigonometric Kernel after training with a softplus activation function . . . . .	45
	c	Trigonometric Kernel after training with a linear activation function . . . . .	45



# List of Tables

1.1	Setup of smaXtec and acluster system . . . . .	14
2.1	Summary of the hyperparameters used for training the machine learning models CCNN and ResNet. With $\lambda$ the L2 regularization, fl the Filter Size, kl the Kernel Size, ll the layer length, and lr the Learning Rate . . . . .	19
2.2	Description of combinations of continuous and non-continuous layers for CCNN and ResNet . . . . .	19
2.3	Hyperparameters Grid Search for CCNN. . . . .	20
2.4	Hyperparameters Grid Search for ResNet version 1. . . . .	20
2.5	Hyperparameters Grid Search for ResNet version 2 with AdaDelta Optimizer 1.2.6.4. . . . .	20
2.6	Hyperparameters Grid Search for ResNet version 2 with AdaGrad Optimizer 1.2.6.3. . . . .	20
2.7	Hyperparameters Grid Search for ResNet version 2 with Adam and Nadam Optimizer 1.2.6.1 1.2.6.2. . . . .	20
3.1	Mean squared error (mse), mean absolute error (mae), and the maximum of mae (max) in kcal/mol for CCNN - Version 1 with Gaussian, harmonic, trigonometric, and Glorot uniform kernel functions. With $\Delta T$ the kinetic energy and $\Delta \frac{\delta T}{\delta n}$ it's kinetic energy derivative. A list of hyperparameters in Table 2.1. The hyperparameters for the model <i>CCNNV1GaussOpt</i> and <i>CCNNV1HarmOpt</i> are listed in the Tables 3.3 and 3.4. . . . .	30
3.2	Mean squared error (mse), mean absolute error (mae), and the maximum of mae (max) in kcal/mol for CCNN - Version 1 with combination of continuous and non-continuous layers, which are described in Table 2.2. With $\Delta T$ the kinetic energy and $\Delta \frac{\delta T}{\delta n}$ it's kinetic energy derivative. The hyperparameter configuration is provided in Table 2.1 . . . . .	31
3.3	Mean squared error (mse), mean absolute error (mae) in kcal/mol CCNN - Version 1 hyperparameter grid search with a harmonic kernel function with the hyperparameters from Table 2.1. Only runs with a mae for $\Delta \frac{\delta T}{\delta n}$ below 50 kcal/mol are shown for better visibility. . . . .	31

3.4	Mean squared error (mse), mean absolute error (mae) in kcal/mol CCNN - Version 1 hyperparameter grid search with a Gaussian kernel function with the hyperparameters from Table 2.1. Only runs with a mae for $\Delta \frac{\delta T}{\delta n}$ below 50 kcal/mol are shown for better visibility. . . . .	32
3.5	Mean squared error (mse), mean absolute error (mae), and the maximum of mae (max) in kcal/mol for CCNN - Version 2 with Gaussian, Chauchy, Lorentz. With $\Delta T$ the kinetic energy and $\Delta \frac{\delta T}{\delta n}$ it's kinetic energy derivative. The hyperparameter configuration is provided in Table 2.1. . . . .	35
3.6	Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) in kcal/mol for ResNet - Version 1 with Gaussian, Lorentz, Cauchy, and Glorot uniform kernel functions. With $\Delta T$ the kinetic energy and $\Delta \frac{\delta T}{\delta n}$ its kinetic energy derivative. The hyperparameter configuration is listed in table 2.1. . . . .	37
3.7	Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) for ResNet - Version 1 with combination of continuous and non-continuous layers. With $\Delta T$ the kinetic energy and $\Delta \frac{\delta T}{\delta n}$ it's kinetic energy derivative. The hyperparameter configuration are listed in Table 2.1 and the description of the combination in Table 2.2. . . . .	37
3.8	Mean Squared Error (mse) and Mean Absolute Error (mae) ResNet - Version 1 hyperparameter grid search. Only runs with a mae for $\Delta \frac{\delta T}{\delta n}$ below 60 kcal/mol are shown for better visibility. . . . .	38
3.9	Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) for ResNet - Version 2 with Gaussian, Lorentz, and Cauchy kernel functions (in kcal/mol). With $\Delta T$ the kinetic energy and $\Delta \frac{\delta T}{\delta n}$ its kinetic energy derivative. The corresponding hyperparameter configuration is provided in Table 2.1. . . . .	41
3.10	Mean squared error (mse), mean absolute error (mae) ResNet - version 2 hyperparameter grid search with a Gaussian kernel function. Only runs with a mae for $\Delta \frac{\delta T}{\delta n}$ below 1000 kcal/mol are shown for better visibility. . . . .	42
3.11	Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) for Dense CCNN - Version 1 with Gaussian, Harmonic, Trigonometric and Glorot uniform kernel functions (in kcal/mol). With $\Delta T$ the kinetic energy and $\Delta \frac{\delta T}{\delta n}$ it's kinetic energy derivative. The corresponding hyperparameter configuration is provided in Table 2.1. . . . .	43

# 1 Background and Methology

## 1.1 Density Functional Theory

The basic idea behind density functional theory is to write the ground state electronic energy as a functional of the density instead of using a wavefunction based formalism. The basis for this approach is the proof of Hohenberg and Kohn.

### 1.1.1 Hohenberg-Kohn theorems

As is known, the external potential determines all the properties of the system, and by solving the Schrödinger equation of the system we obtain the standard approach to quantum mechanical problems.

The importance of the Hohenberg-Kohn theorems can be demonstrated by the following comparison: An  $N$  electron system contains  $4N$  variables for each wave function, one for the spin and 3 for the spatial coordinates, with the computational cost increasing exponentially. On the other hand, if the electronic energy in the ground state is written as a density functional, the problem becomes independent of the number of electrons and therefore the number of variables remains the same for a growing system. [16]

The first Hohenberg-Kohn theorem may be stated as: *the potential is uniquely determined by the ground state density,  $n(\mathbf{r})$  and the proof runs as follows for densities with non-degenerate states.*

If a constant is added to the potential, the wave functions and thus the charge density are not affected by it [5]. We are assuming now that there are two external potentials  $\hat{V}_{\text{ext}}$  and  $\hat{V}'_{\text{ext}}$  differing by more than a constant, yet resulting in the same density. These two external potentials are part of two Hamiltonians  $\hat{H} = \hat{T} + \hat{V}_{\text{ee}} + \hat{V}_{\text{ext}}$  and  $\hat{H}' = \hat{T} + \hat{V}_{\text{ee}} + \hat{V}'_{\text{ext}}$  with two different ground state functions  $\Psi_2$  and  $\Psi_1$ [16]. Applying the variational principle,

$$\langle \Psi_2 | \hat{T} + \hat{V}_{\text{ee}} + \hat{V}_{\text{ext}} | \Psi_2 \rangle \geq \langle \Psi_1 | \hat{T} + \hat{V}_{\text{ee}} + \hat{V}_{\text{ext}} | \Psi_1 \rangle \quad (1.1)$$

both wave functions yield the same density and this implies

$$\langle \Psi_2 | \hat{T} + \hat{V}_{\text{ee}} | \Psi_2 \rangle \geq \langle \Psi_1 | \hat{T} + \hat{V}_{\text{ee}} | \Psi_1 \rangle. \quad (1.2)$$



This leads to the conclusion that there are no two different external potentials  $\hat{V}_{\text{ext}}$  leading to the same electron density for the ground state, and that the electron density uniquely describes the external potential.

If we define a functional

$$F[n] = \min_{\Psi \rightarrow n} \langle \Psi | \hat{T} + \hat{V}_{\text{ee}} | \Psi \rangle, \quad (1.3)$$

where the search is over all asymmetric wavefunctions  $\Psi$  yielding into  $n(r)$ , then for any  $\Psi$  minimizing  $\hat{T} + \hat{V}_{\text{ee}}$ ,  $\Psi$  is a ground-state wavefunction with a ground state energy of

$$E = \min_n \left( F[n] + \int d^3r v_{\text{ext}}(\mathbf{r}) n(\mathbf{r}) \right). \quad (1.4)$$

The second Hohenberg-Kohn theorem therefore states that there is a universal functional  $F[n]$ , which is the same for all electronic structure problems [5].

### 1.1.2 Kohn-Sham equations

The Kohn-Sham equations are producing the exact kinetic energy for non-interacting electrons with the same density as the physical system, which is close to the true kinetic energy. The orbitals which are defined via

$$\left\{ -\frac{1}{2} \nabla^2 + v_{\text{S}}(\mathbf{r}) \right\} \phi_i(\mathbf{r}) = \epsilon_i \phi_i(\mathbf{r}), \quad (1.5)$$

with  $v_{\text{S}}$  denoting some potential chosen somehow to mimic the true electron system and  $\epsilon_i$  as corresponding orbital energy, yield a total electron density

$$n(\mathbf{r}) = \sum_{i=1}^N |\phi_i(\mathbf{r})|^2, \quad (1.6)$$

with the equivalent Euler equations

$$\frac{\delta T_{\text{S}}}{\delta n(\mathbf{r})} + v_{\text{S}}(\mathbf{r}) = \mu, \quad (1.7)$$

where

$$T_{\text{S}}[n] = \min_{\Phi \rightarrow n} \langle \Phi | \hat{T} | \Phi \rangle, \quad (1.8)$$

with  $T_{\text{S}}[n]$  denoting the kinetic energy of the non-interacting electron system [5]. The  $s$  stands for single-electron equations.

By minimizing the expectation value of the kinetic energy in Equation 1.8 we get the definition:

The Kohn-Sham wavefunction of density  $n(r)$  is the wave function that yields  $n(r)$  and has the lowest kinetic energy. (Burke, 2007, p. 66)[5]. With this, the ground state functional  $F[n]$  (of an interacting system) can be formalized in terms of the non-interacting kinetic energy [5]

$$F[n] = T_s[n] + U[n] + E_{xc}[n], \quad (1.9)$$

with the functional  $U[n]$  denoting the static self-repulsion of the charge density  $n$  and  $E_{xc}$  the exchange-correlation energy. If we put 1.9 into the Euler-Lagrange equation

$$\frac{\delta F}{\delta n(\mathbf{r})} + v_{\text{ext}}(\mathbf{r}) = \mu, \quad (1.10)$$

which is obtained by minimizing the functional for the ground state energy  $E[n]$  for a given external potential  $v_{\text{ext}}$ , and compare it with 1.7, we see that

$$v_S(\mathbf{r}) = v_{\text{ext}}(\mathbf{r}) + \int d^3r' \frac{n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} + v_{xc}[n](\mathbf{r}) \quad v_{xc}(\mathbf{r}) = \frac{\delta E_{xc}}{\delta n(\mathbf{r})} \quad (1.11)$$

and thereof that the potential felt by non-interacting electrons can be extracted from  $F[n]$ , which is the first important relation of the exact density functional theory.

### 1.1.3 Orbital-free Density Functional Theory (OF-DFT)

*In modern materials science and engineering modeling, first-principles quantum mechanical methods are widely used because they can offer reliable results and predictions at the atomic scale, explaining many interesting phenomena that classical theories cannot. (Carter and Xia, p.1) [34]*

One of the most popular methods is the Kohn-Sham density functional theory (KSDFT) because of its balance of efficiency and accuracy. In the ansatz of KSDFT the exact kinetic energy of an interacting electron system is replaced with an approximate, non-interacting, single determinantal wavefunction which leads to the same density. This approach involves the introduction of one-electron orbitals and leads to a kinetic energy for a system of non-interacting electrons  $T_{KS}[\phi_1, \dots, \phi_N]$  [31], [34]. If the  $\phi_s$  are orthonormal,  $T_{KS}$  can be written as

$$T_{KS} = \frac{1}{2} \sum_{i=1}^N \phi_i^*(r) \nabla^2 \phi_i(r). \quad (1.12)$$

The computational effort of this orthogonalization, respectively matrix diagonalization, of the one-electron orbitals scales cubically with the basis size.

In OF-DFT, the universal energy functional can be expressed in terms of the electronic density alone. The accuracy, which can be achieved in the OF-DFT, strongly depends on the quality of approximated orbital-free kinetic energy functionals.

If one looks at the historical development of the kinetic energy functional, there was first the Thomas-Fermi functional (local density approximation) contribution, given by [31]

$$T_{\text{TF}} = \frac{3}{10} (3\pi^2)^{2/3} \int d\mathbf{r} [n(\mathbf{r})]^{5/3}, \quad (1.13)$$

and then the von Weizsäcker improvement with a gradient based correction [30]

$$T_{\text{vW}} = \frac{1}{8} \int \frac{|\nabla n(\mathbf{r})|^2}{n(\mathbf{r})} d\mathbf{r}. \quad (1.14)$$

With this improvement, the approximation was not accurate enough to be used in computational chemistry calculations, and even corrections involving higher order derivatives for the Thomas-Fermi functional 1.13 have been shown to diverge in the long term limit [12]. Therefore, most research at the time is spent on the non-local functional [27], which is expressed by

$$T^{\text{NL}}[n] = C \iint n(\mathbf{r})^\alpha \omega[n](\mathbf{r}, \mathbf{r}') n(\mathbf{r}')^\beta d\mathbf{r} d\mathbf{r}'. \quad (1.15)$$

These three terms form the kinetic energy density functionals (KEDFs), which are less accurate than KSDFT, but they provide an efficient approach to compute the energy of large samples with many thousands of atoms. [34]

For problems with highly localized electrons, such as semiconductors, transition metals, or molecules that deviate from the scenario of a uniform electron gas, KEDFs are numerically and physically unsound. However, KEDFs can model nearly-free-electron-like systems, such as metals and main-group alloys, with accuracy comparable to KSDFT [34].

Huang-Carter (HC) proposed the (HC) KEDF in 2010, which took into account the linear response of semiconductors, resulting in improved accuracy for Si and III-V semiconductors and for covalently bonded molecules. However, with the HC KEDF, the accuracy of the properties of metallic SI phases was not sufficient, it underestimated the electron density bonding regions of Si and III-V semiconductors [13], [27].

In 2012, Carter-Xia proposed a density decomposition scheme using a Wang-Govind-Carter (WGC) [33], [32], which is based on the KEDF to simulate covalently bound molecules and materials within OFDFT more accurately and efficiently by using a local density-dependent scale function. The new approach predicted reasonable equilibrium volumes, bulk moduli, and phase order energies for semiconductors compared to KSDFT. But with the WGCDF model the density in the bonding region and the vacancy formation energy is underestimated [34].

## 1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are specialized neural networks with a convolutional operation to process data with a grid-like topology. CNNs are most often used in 2D or 3D form for image processing and pattern recognition, whereas a 1D CNN is used for processing time series data.

### 1.2.1 The Convolution Operation

The following chapter is closely following Ref. [14]. A convolutional operation is a mathematical operation of two functions  $f$  and  $g$  with a real-valued input, yielding a third function  $f * g$ . The convolution can be interpreted in such a way that, for  $f * g$ , each value of  $f$  is replaced by the weighted mean  $g$  of its surrounding values, which can be used for example to compute a weighted average. For continuous values a convolutional operation of  $f * g$ , where  $f, g : \mathbb{R}^n \rightarrow \mathbb{C}$ , is defined as:

$$(f * g)(x) := \int_{\mathbb{R}^n} f(\tau)g(x - \tau)d\tau \quad (1.16)$$

In terms of a CNN, the first argument in 1.16 is referred to as the input, the second as the kernel, and the output sometimes as the feature map. Since we are dealing with discrete rather than continuous data in real-world problems, the definition for a convolution changes as follows in a discrete representation:

$$(f * g)(x) := \sum f(\tau)g(x - \tau), \quad (1.17)$$

where  $f, g : D \rightarrow \mathbb{C}$  are discrete functions with a discrete domain  $D \subseteq \mathbb{Z}$ . Most machine learning libraries implement a cross-correlation function instead of a convolutional operation. Cross-correlation is not commutative, but the kernel is able to learn the commutative behavior.

The discrete convolution for CNN is defined as follows:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m]. \quad (1.18)$$

The cross-correlation can be written as

$$(f \star g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[m + n]. \quad (1.19)$$

In a one-dimensional CNN with one kernel and one layer, the convolution is given by

$$z[n] = f \left( b + \sum_{m=1}^{\sigma_w} n[m]w[m + n] \right), \quad (1.20)$$

where  $n$  is the input data,  $w$  the kernel,  $b$  the bias, and  $f$  the activation function. A part of the incoming data in the size of the kernel is multiplied by the kernel. This is repeated and summed up until all parts of the data are processed. After this, the bias is added and the result is processed by an activation function. The bias gives the neural network the ability to shift the activation function, which then performs the final operation by converting the input and forwarding it to the next layer.

### 1.2.2 Properties of CNNs

Compared to a traditional neural network where each output unit interacts with each input unit, convolutional networks have **sparse interactions**. Sparse interactions are achieved by making the kernel smaller than the input. This has the advantage that small features can be detected, less memory is used (fewer parameters are stored), and less computation is required (fewer operations are needed to compute the output).

These advantages are evident when, for example, an image with millions of pixels is processed and small important features are supposed to be detected with a kernel that processes only hundreds of pixels.

A matrix multiplication in a traditional neural network has a runtime of  $O(m \times n)$ , while in a CNN the runtime is  $O(k \times n)$ , where  $k$  is the size of the kernel and is smaller than  $m$ . If  $k$  is chosen to be several orders of magnitude smaller than  $m$ , good performance in terms of quality and computation time can be achieved for many practical applications. A graphical interpretation of sparse interaction is visible in 1.1.

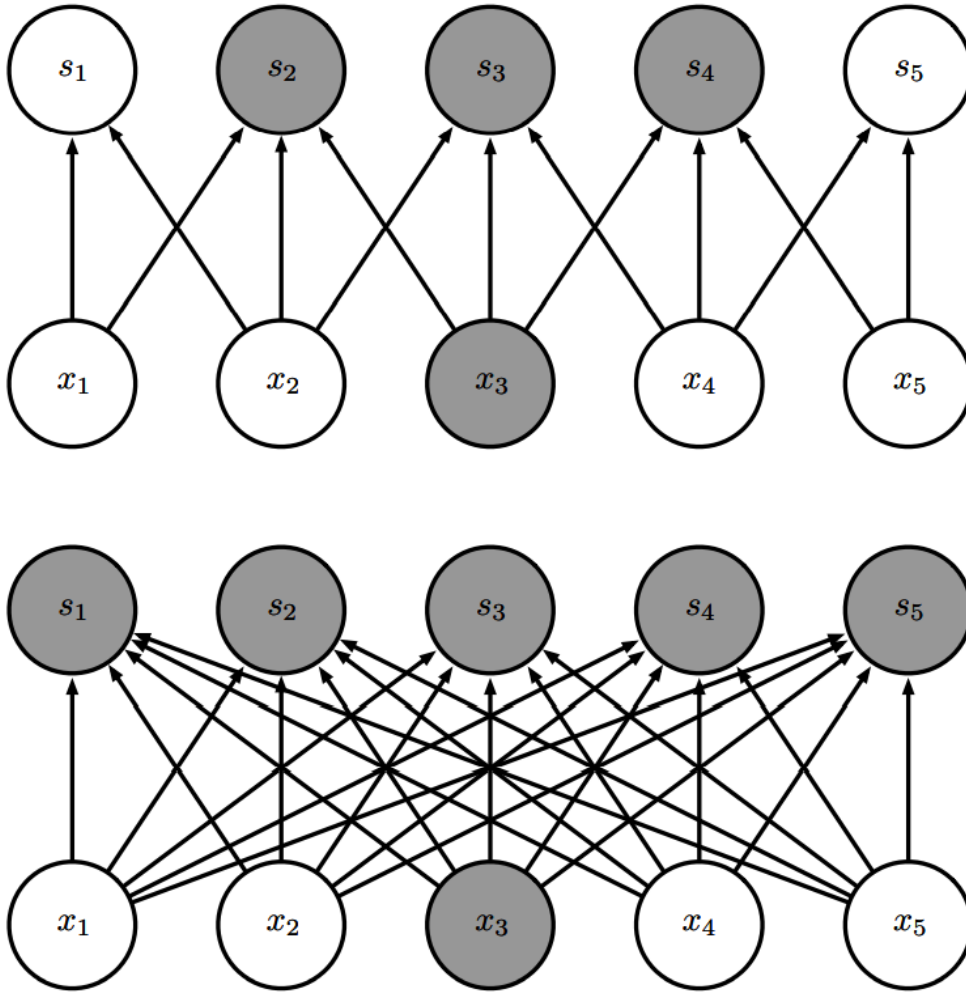


Figure 1.1: Representation of sparse connectivity. In the CNN (*top*) a convolution with a kernel size of 3 is performed and only 3 outputs are effected by  $x_3$ , where in the traditional neural network (*bottom*) all outputs  $s_{1,...,5}$  are effected by the matrix multiplication [14].

In a CNN, each element of the kernel is used at each position of the input, leading to the effect of **parameter sharing**. This means that compared to a traditional neural network, where each element of the weight matrix is used only once and therefore a separate set of parameters is learned for each position, a CNN learns the set of parameters only once. The running time  $O(k \times n)$  is not affected by this, but the storage of the parameters is reduced. In figure 1.2 the graphical representation between parameter sharing and no parameter sharing is visible.

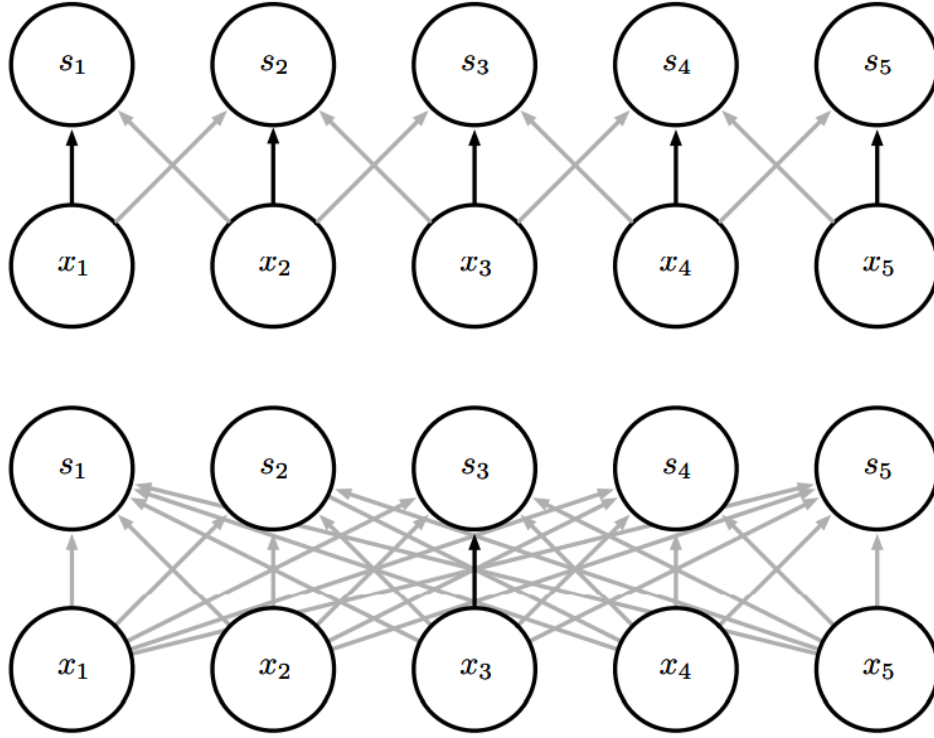


Figure 1.2: Representation of parameter sharing. The use of a parameter in two different models is represented by a black arrow. The black arrow (*Top*) indicates that the single parameter is used in all input locations. The centered black arrow (*Bottom*) indicates the element of a fully connected model where the parameter is used only once [14].

The parameter sharing property also leads to the **equivariance** property in the case of CNN.

### 1.2.3 Pooling

The structure of a layer in a CNN consists of three stages, where in the first stage the layer performs several parallel convolutions to produce a set of linear activation. In the second stage, also called the detector stage, these linear activations are processed by a nonlinear activation function. The last stage in a layer of a CNN is **pooling**. Pooling means that the output of a network is replaced by a statistical summary at a certain location with the neighboring outputs and makes the representation approximately invariant to small transitions. Invariance is useful in cases where the detection of a feature is more important than the exact position. This means that the pooling function learns the layer to be invariant to small transitions by adding an infinitely strong prior.

An example of a pooling function is max pooling. The max pooling operation

returns the maximum output within a rectangular neighborhood. Another pooling function is the  $L^2$  norm in a rectangular neighborhood.

### 1.2.4 Types of Convolution

An important feature in convolutional network implementations is to implicitly pad zeros to the input to make it wider. This feature is important because with each layer the width  $l$  of the representation shrinks by  $l = k - 1$ , with  $k$  being the kernel length. This would lead either to shrinking the spatial extent of the network rapidly or to choosing a small kernel, which would reduce the quality of feature detection.

One type of zero padding is the **valid** convolution. Here, no zero padding is applied, and the kernel is only allowed to visit positions where the kernel is contained entirely within the image.

Another way of zero padding is to add so many zeros that the output size matches the input size. This is often called **same** convolution. With this type, the network can contain as many layers as the hardware can support. The disadvantage of this is that the input pixels near the edge affect fewer output pixels than the input pixels in the center, which can make the pixels at the edge underrepresented.

One type of zero padding that does not have this disadvantage is the so-called **full** convolution. Here, so many zeros are added that each pixel can occur  $k$  times in each direction, resulting in an output with a width of  $m + k - 1$ . The output pixels are therefore a function of fewer pixels than the output pixels near the center, which makes it difficult for a single kernel to perform well at all positions.

### 1.2.5 Dealing with Vanishing/Exploding Gradient Problem

If deep convolutional neural networks have a lot of stacked layers, the **vanishing gradient problem** can occur during training.

The **vanishing gradient problem** occurs as more layers with a given activation function are added to a neural network, the gradient of the loss function approaches zero, and the neural network becomes untrainable. This occurs because certain activation functions, like sigmoid etc., are mapping a large input space into a small input space between 0 and 1 [8].

One approach to overcome this problem is the implementation of a Residual Neural Network - (ResNet).



### 1.2.5.1 Residual Learning

If  $H(x)$  is an underlying mapping with  $x$  as the input and multiple nonlinear layers can hypothetically fit this mapping, then it should be also hypothetically possible to fit the residual function  $F(x) := H(x) - x$ . The original function  $H(x)$  becomes  $F(x) + x$ . Thus, added layers in a ResNet can be constructed as identity mappings, and the training error of a deeper network should not be greater than a shallower neural network. The formulation of  $F(x) + x$  can be realized by shortcut connections within feed-forward neural networks as can be seen in figure 1.3. Here shortcut connections (skipping one or more layers) perform identity mapping, and the outputs are added to the outputs of the stacked layer. In neural networks with residual learning, if identity mapping is optimal, the solver should be able to drive the weights of the multiple nonlinear layers to zero and approach identity mappings [10] [4].

In other words, the residual connection does not pass the activation function, and therefore the input is not “squeezed”, which is conducive to the prevention of a vanishing gradient.

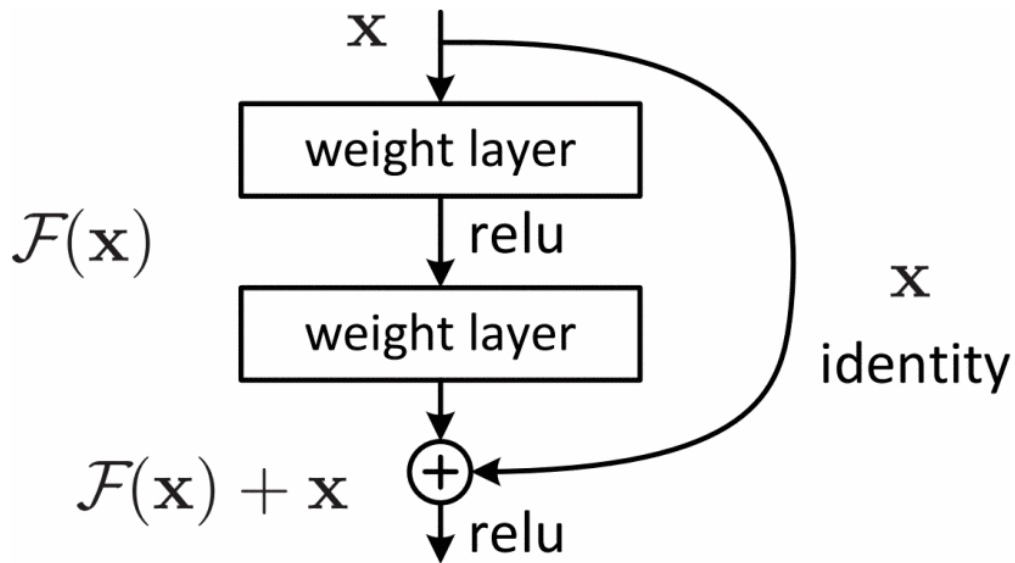


Figure 1.3: A block in a Residual Neural Network [10].

Another way to not only deal with vanishing gradient problem but also exploding gradient is gradient clipping.

### 1.2.5.2 Gradient Clipping

One option is to clip the norm  $\|\mathbf{g}\|$  of the gradient  $\mathbf{g}$  before the parameter update:

$$\begin{aligned} &\text{if } \|\mathbf{g}\| > v \\ &\quad \mathbf{g} \leftarrow \frac{\mathbf{g}v}{\|\mathbf{g}\|}, \end{aligned} \tag{1.21}$$

where  $v$  is the norm threshold. This option uses a single scaling factor. Another option is to clip the gradient in a minibatch element wise manner, which guarantees that each step is in gradient direction. Both options mentioned show a similar performance.

## 1.2.6 Optimizers

### 1.2.6.1 Adam Optimizer

The advantage of the Adaptive Moment Estimation (Adam) [17] is that this algorithm computes an adaptive learning rate for each parameter and keeps an exponential decaying average of previous gradients. So, in comparison with the gradient descent optimizer, for every weight in the neural net a learning rate is calculated and updated [24].

### 1.2.6.2 Nadam

Nadam (Nesterov-accelerated Adaptive Moment Estimation) [23] that combines the momentum of Adam and Nesterov Accelerated Gradient [11].

### 1.2.6.3 AdaGrad

AdaGrad is a stochastic optimizer with an adaptive learning rate for the parameters, making smaller updates for parameters with frequently occurring features and larger updates for parameters with rarely occurring features [3].

### 1.2.6.4 AdaDelta

AdaDelta is a stochastic optimizer that performs a per-dimension learning rate method for SGD, an extension of AdaGrad that aims to reduce its strongly monotonically decreasing learning rate. [35].

**input** : Learning Rate  $\eta$   
**input** : Exponential Decay Momenta  $\beta_1$  and  $\beta_2$   
**input** : A small constant for numerical stability  $\epsilon$   
 initialization of 1st and 2nd momentum vector  $\mathbf{m}_0 \leftarrow 0, \mathbf{v}_0 \leftarrow 0$ ;  
 initialization of weights  $\mathbf{w}$ ;  
 initialization of time step  $\mathbf{t} \leftarrow 0$ ;  
**while** *stop criterion is not reached* **do**  
     Take a minibatch of  $m$  from Training set ( $\mathbf{x}$  input and  $\mathbf{y}$  target);  
     Evaluate gradient:  $\mathbf{g} \leftarrow \nabla \sum_i L(f(\mathbf{x}^{(i)}, \mathbf{w}), \mathbf{y}^{(i)})$ ;  
      $t \leftarrow t + 1$ ;  
     Update biased first moment:  $\mathbf{m}_t \leftarrow \beta_1 \cdot \mathbf{m}_t + (1 - \beta_1) \cdot \mathbf{g}$ ;  
     Update biased second moment:  $\mathbf{v}_t \leftarrow \beta_2 \cdot \mathbf{v}_t + (1 - \beta_2) \cdot \mathbf{g} \odot \mathbf{g}$ ;  
     Correct bias in first moment:  $\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1 - \beta_1^t}$ ;  
     Correct bias in second moment:  $\hat{\mathbf{v}}_t \leftarrow \frac{\mathbf{v}_t}{1 - \beta_2^t}$ ;  
     Compute update:  $\Delta \mathbf{w} = \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}}$ ;  
     Apply Update:  $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$   
**end**

**Algorithm 1:** Pseudocode Algorithm of Adam Optimizer. The symbol  $\odot$  refers to the Hadamard product (component-wise multiplication for matrices)

### 1.3 GPU vs CPU in Machine Learning

A CPU (Central Processing Unit) may contain several computing cores and executes various commands and processes for the operating system. In comparison to a GPU, the advantage of the CPU is that it computes individual tasks more quickly with a smaller amount of cores. Thus, CPUs are usually better suited for a single, more complex serial task [15].

A GPU (Graphic Processing Unit) is a single chip processor specialized to perform floating point operations with a dedicated memory, which is required for rendering graphics. GPUs consist of smaller computing cores in comparison to a CPU, which makes a GPU better for parallel computing. This is because GPUs have more transistors dedicated to arithmetic logic units and fewer to cache and flow control.

A dedicated video RAM (VRAM), which is included in a GPU, is able to provide a large memory bandwidth to accommodate large datasets [15] [6].

In Figure 1.4, a schematic diagram is presented, showing the differences of

CPU and a GPU architecture. As can seen the CPU is capable of executing a sequence of threads as fast as possible, but only a few of these threads in parallel, whereas a GPU is designed to execute thousands of threads in parallel.

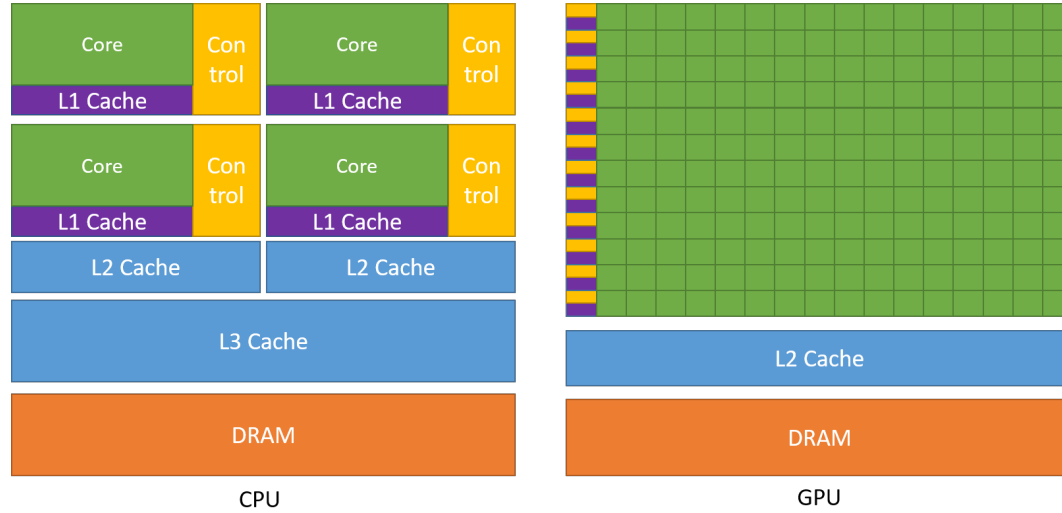


Figure 1.4: Schematic chip architectures for a CPU and GPU [15]

## 1.4 Description of needed tools

The programming language Python [22] was used in this master thesis. The most convenient way to install Python, which also includes the most commonly used libraries, is through the Anaconda package manager [1] [2].

### 1.4.1 TensorFlow, CUDA and GPU support

To achieve a better performance during the training of a machine learning model with TensorFlow on a GPU, the installation of CUDA and GPU supported TensorFlow is highly recommended. For the installation, the documentation of TensorFlow can be followed [29].

#### 1.4.1.1 TensorFlow

TensorFlow is an open source software library (released under the Apache License 2.0 in 2015 [19]) for numerical computation using data flow graphs and differentiable programming. The library TensorFlow is used for machine learning and artificial intelligence and was developed by Google Brain [9].

#### 1.4.1.2 CUDA - Compute Unified Device Architecture

CUDA is a programming interface (API) and parallel computing platform developed by Nvidia [7] and used for general purpose computing on graphics processing units (GPGPU). With the CUDA platform (a software layer) it is possible to get direct access to the GPU's virtual instruction set and parallel computing elements.

#### 1.4.2 Setup

The machine learning models in this master thesis were trained on two different systems. The first system was provided by **smaXtec animal care GmbH** [25]. The second system is called **acluster** and was provided by the **IT Services of Graz University of Technology** [28]

System	GPU	CUDA-V	TensorFlow-V	Python-V
smaXtec	GeForce Ti GTX 1080	11.1	2.4.1	3.8
acluster	NVIDIA Tesla T4	11.1	2.5 -dev (nightly)	3.9

Table 1.1: Setup of smaXtec and acluster system

## 2 Neural Networks for DFT

### 2.1 Data Generation

For training the machine learning models the same dataset as in [20] was used to compare previous models to a new implementation featuring a *continuous convolution*. A one-dimensional model system of noninteracting spinless fermions with one particle in a hard wall box and a potential described by a linear combination of three Gaussians has been investigated. With the Numerov's method [21] the 1D Schrödinger equation for these potentials is solved on a grid of  $G = 500$  points. The solutions are then used to compute the data for the training: The density

$$n_j(x) = \sum_k^N \left( \psi_j^k(x) \right)^2, \quad (2.1)$$

the kinetic energy density

$$\tau_j(x) = \frac{1}{2} \sum_{k=1}^N \left( \nabla \psi_j^k(x) \right)^2, \quad (2.2)$$

the kinetic energy

$$T_j = \int_0^1 \tau_j(x) dx, \quad (2.3)$$

and the kinetic energy functional derivative

$$\frac{\delta T [n_j]}{\delta n_j(x)} = \mu_i - V_j(x). \quad (2.4)$$

A length of 100 data points is used for training and 1000 for testing.

## 2.2 Continuous Convolutional Neural Network - CCNN

The values of the kernel in the TensorFlow library are randomly distributed with a lot of noise, as can be seen in Figure 2.1.

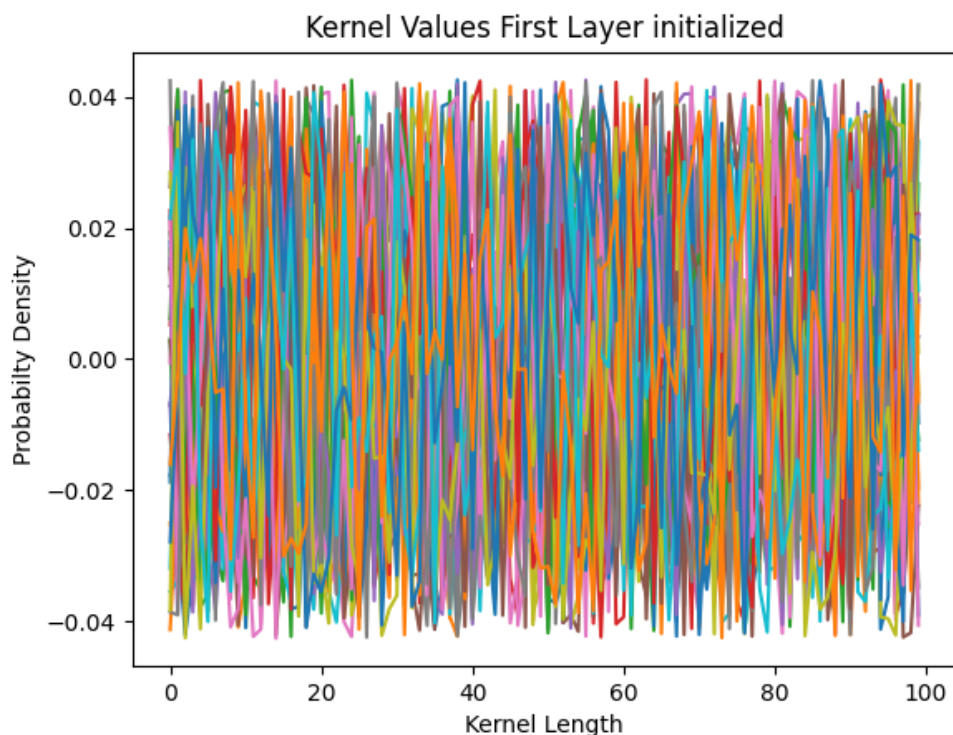


Figure 2.1: Kernel values of first layer with a size of 100 and a count of 32 kernes which are Glorot Normal distributed.

The working question of this thesis is that

- smooth kernel functions *should* translate to smooth functional derivatives and
- that kernel functions are independent of the grid. The trained NNs can be applied to arbitrary grids. This also results in translational invariance for the uneven grid spacing typically as it is common in computational chemistry (finer grid near the nuclei).

To *simulate* a continuous convolution with discrete data, the kernel values are replaced by smooth distributions vanishing on the left and right sides. To achieve this, two versions are implemented with the TensorFlow code.

### 2.2.1 CCNN - Version 1

The first approach is to replace the weights to be trained (kernel values) with the following three custom implemented functions:

- Gaussian Distribution

$$f(x) = \frac{A}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad (2.5)$$

- Solutions of Quantum Harmonic Oscillator
- Trigonometric Functions

$$f(x) = A_1 \cdot \sin(\omega_1 \cdot x + \phi_1) + A_2 \cdot \cos(\omega_2 \cdot x + \phi_2) \quad (2.6)$$

The parameters of the distribution and functions are initialized randomly. In version 1, the weights are trained as in the TensorFlow implementation. The optimizer updates each value of the kernel after each epoch of training. The goal is to have a smooth kernel with values vanishing on the left and right side. Furthermore, these properties should be preserved during the training. By eliminating the noise and keeping the distribution vanishing, the convolution should be quasi-continuous.

### 2.2.2 Dense CCNN - Version 1

While in most applications convolutional layers are combined with dense layers, the architecture of CCNN - Version 1 and Version 2 consists solely of convolutional layers. By using just a single convolutional layer followed by several dense layers, it is investigated if similar accuracy can be achieved. This corresponds to interpreting the output of the convolution as descriptors of the local electron density, and is tried for continuous convolutions by using a single convolutional layer with a linear and a softplus activation function.

### 2.2.3 CCNN - Version 2

In Version 2, the values of the kernel are also replaced, but the weights to be trained are the parameters of the distributions

- Gaussian (see equation 2.6),
- Lorentz

$$f(x) = \frac{1}{\pi\gamma} \left[ \frac{\gamma^2}{(x-x_0)^2 + \gamma^2} \right], \quad (2.7)$$



- Cauchy

$$f(x) = \frac{1}{(\omega^2 - \omega_0^2)^2 + \gamma^2 \omega_0^2}, \quad (2.8)$$

In this version, after each epoch, each weight (parameter of distribution) is trained, and before the convolution is performed, the kernels are calculated with the updated parameter. With the similar shape and smooth values of the kernel and only updating the parameters of the distributions, the NN *should* reduce the noise and therefore, also the error at the end of the training.

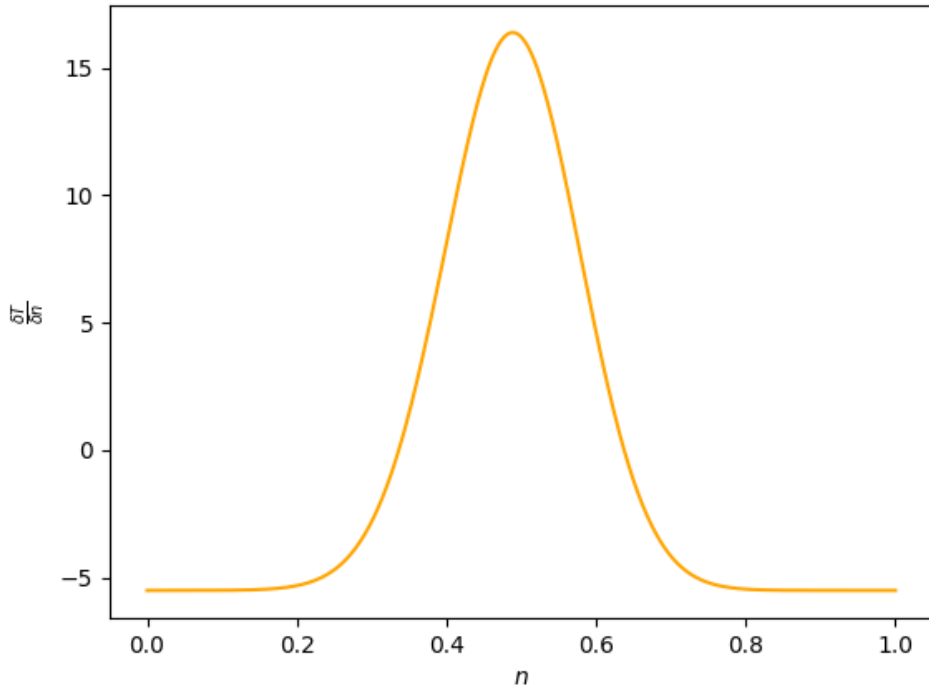


Figure 2.2: Plot of the kinetic energy derivative after the density calculated with the Numerov's method

## 2.3 Hyperparameters and Configurations

### 2.3.1 Hyperparameter

To compare the effect of continuous convolution on machine learning models, the same hyperparameters are used for the CNN and ResNet as in [20] and are summarized in table 2.1. A softplus activation function was used

and a warm-up exponential learning rate with  $N_c$  and  $N_d$  for decreasing the learning rate during the training. The kinetic energy flows with a magnitude of 0.2 in the training and its functional derivative with 1.0. As not otherwise stated, the hyperparameters remain the same for all machine learning models.

Also, a batch size of 100 is used during the training.

Table 2.1: Summary of the hyperparameters used for training the machine learning models CCNN and ResNet. With  $\lambda$  the L2 regularization, fl the Filter Size, kl the Kernel Size, ll the layer length, and lr the Learning Rate

model	N	epochs	$\lambda$	$N_c$	$N_d$	fl	kl	ll	lr
CCNN	1	100000	0.00025	21800	1000	32	100	6	0.0001
ResNet	1	100000	0.00025	21800	1000	32	100	3	0.0001

### 2.3.2 Configurations

Both versions 2.2.1 and 2.2.3 were trained as ResNet and non ResNet CNN machine learning models. Several combinations, see Table 2.2, of continuous and non-continuous layers are used to see if a combination of the two can reduce noise in the area where the values are denser. These combinations also attempt to combine the continuous effect and the initial randomness of the non-continuous layers. For the ResNet implementation, the same configuration is used as in Table 2.2.

Table 2.2: Description of combinations of continuous and non-continuous layers for CCNN and ResNet

Name	Description	ll
BeginHalf	First half of layers are continuous, second half is non-continuous	8
EndHalf	First half of layers are non-continuous, second half is continuous	8
OnlyBegin	Only first layer is continuous	6
OnlyEnd	Only last layer is continuous	6
Alternating	Alternating continuous and non-continuous starting with continuous	6

### 2.3.3 Hyperparameter Search

The hyperparameters in table 2.3 are optimized for the machine learning models in [20] and not primarily for machine learning models with continuous convolution. Therefore, a hyperparameter grid search was performed

for the continuous machine learning models ResNet version 1 and 2 and CCNN version 1. In Table 2.3 and 2.4 the hyperparameters, on which the grid search was performed, are visible.

Table 2.3: Hyperparameters Grid Search for CCNN.

l2 reg	0.0001	0.00001	-
filter size	16	32	64
activation function	softplus	sigmoid	exponential
kernel size	50	100	125

Table 2.4: Hyperparameters Grid Search for ResNet version 1.

l2 reg	0.0001	0.00001	0.000001	-
filter size	8	16	32	64
activation function	relu	softplus	sigmoid	exponential
kernel size	50	100	125	-

Table 2.5: Hyperparameters Grid Search for ResNet version 2 with AdaDelta Optimizer 1.2.6.4.

learning rate	0.0001	0.001	0.01	0.1	1.0
$\rho$	0.0009	0.009	0.09	0.95	-

Table 2.6: Hyperparameters Grid Search for ResNet version 2 with AdaGrad Optimizer 1.2.6.3.

learning rate	0.0001	0.001	0.01	0.1	1.0
$\rho$	0.0001	0.001	0.01	0.1	-

Table 2.7: Hyperparameters Grid Search for ResNet version 2 with Adam and Nadam Optimizer 1.2.6.1 1.2.6.2.

learning rate	0.0001	0.001	0.01	0.1	1.0
$\beta_1$	0.009	0.09	0.1	0.9	-
$\beta_1$	0.0099	0.099	0.1	0.999	-

## 2.4 UML Diagrams of Custom Code with TensorFlow

A UML (Unified Modeling Language) diagram describes the attributes/parameter, method/functions of a class/object, and the relation between classes/objects. In the following figures, custom classes, which were inherited from TensorFlow classes, are described with UML diagrams. Bellow, custom classes and their realations are described:

- Kernel\_INITIALIZER
  - Kernel1DV2
  - ContinuousConvKernel1DV1
- Costum Layers
  - ContinuousConv1D
  - ContinuousConv1DV2
- The Models
  - ClassicCNN
    - \* CustomCNNV1Model
      - CNNV1Beginhalf
      - CNNV1OnlyBegin
      - CNNV1Endhalf
      - CNNV1OnlyEnd
      - CNNV1Alter
    - \* CustomCNNV2Model
  - ResNetConv1DModel
    - \* ResNetCostumLayer1DModel
    - \* ResNetBeginLayer1DModel
    - \* ResNetBeginHalfLayer1DModel
    - \* ResNetEndLayer1DModel
    - \* ResNetEndHalfLayer1DModel
    - \* ResNetAlterLayer1DModel
    - \* ResNetContConv1DV2Model

## 2 Neural Networks for DFT

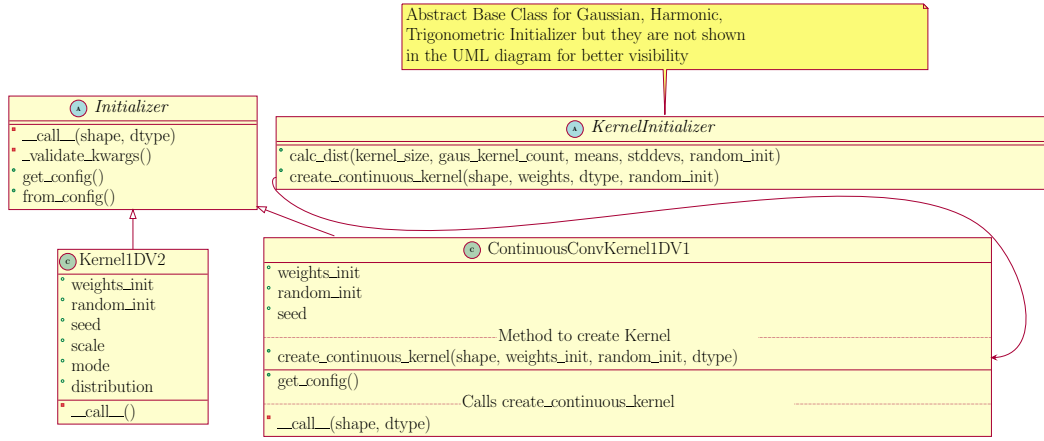


Figure 2.3: UML Diagram of Kernel Initializer classes for version 1 and 2

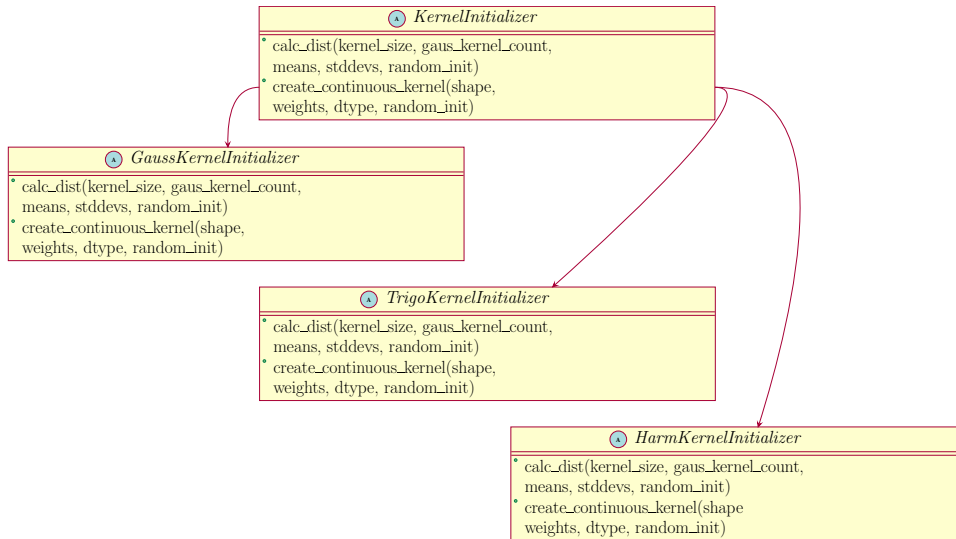


Figure 2.4: UML Diagram of Kernel Initializer and its Inheritances

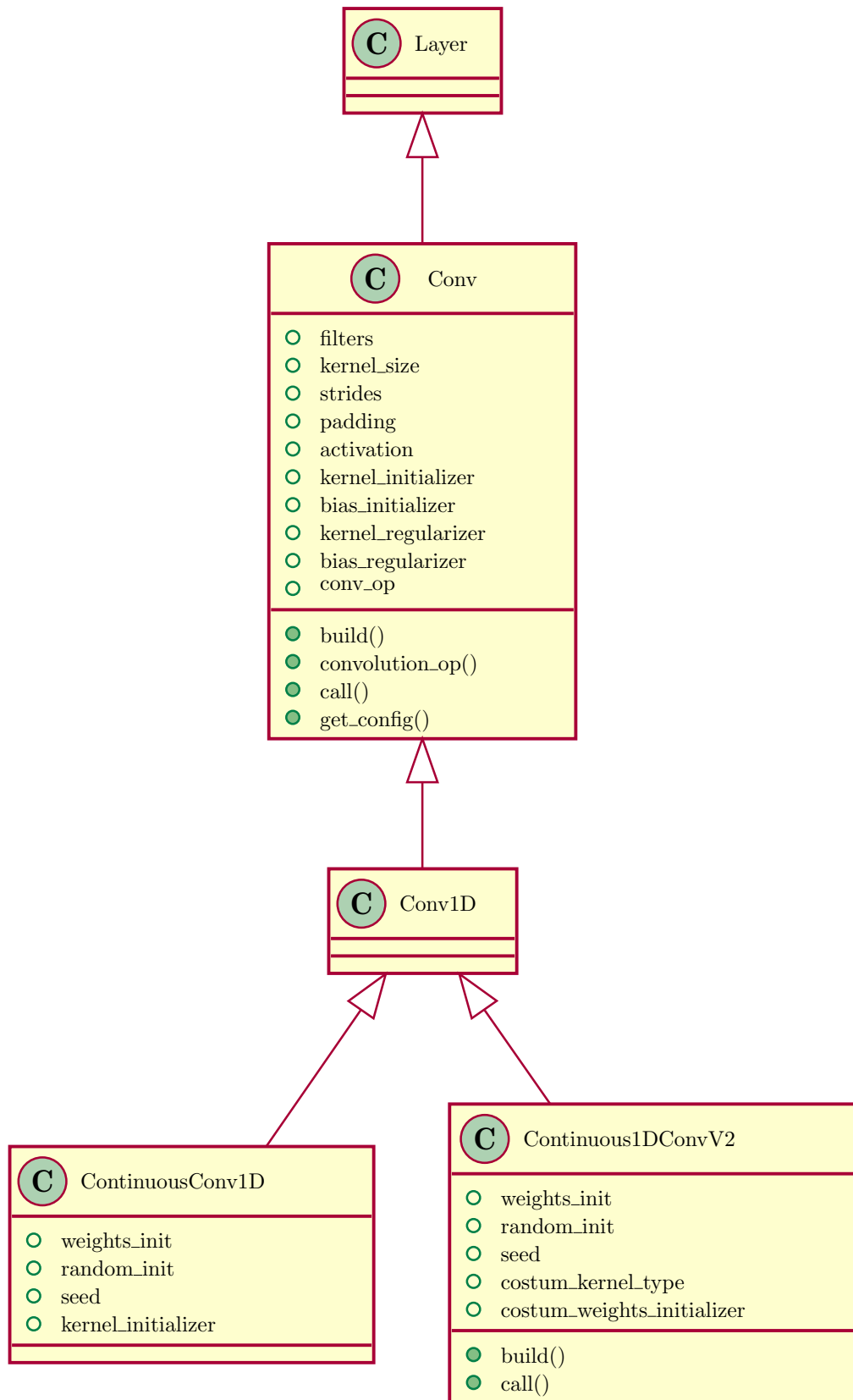


Figure 2.5: UML Diagram of Costum Layer Classes for CCNN version 1 and 2

## 2 Neural Networks for DFT

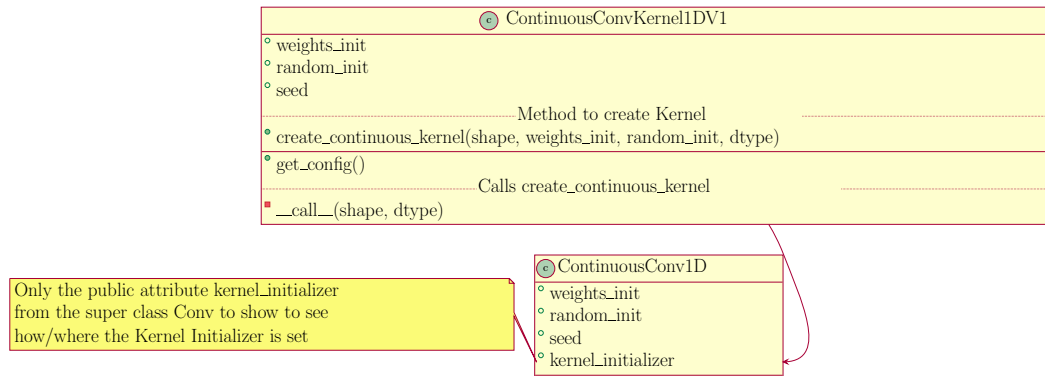


Figure 2.6: UML Diagram of Custom Layer Classes for CCNN version 1

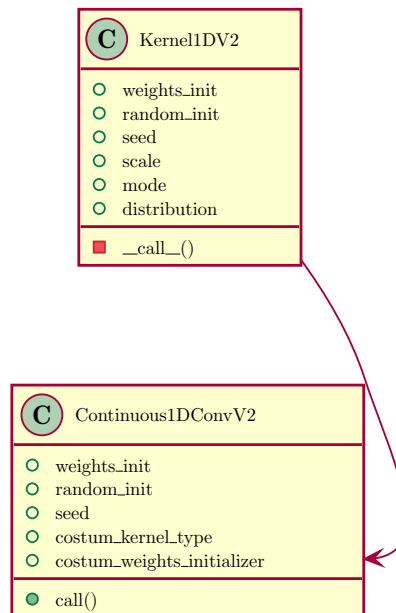


Figure 2.7: UML Diagram of Custom Layer Classes for CCNN version 2

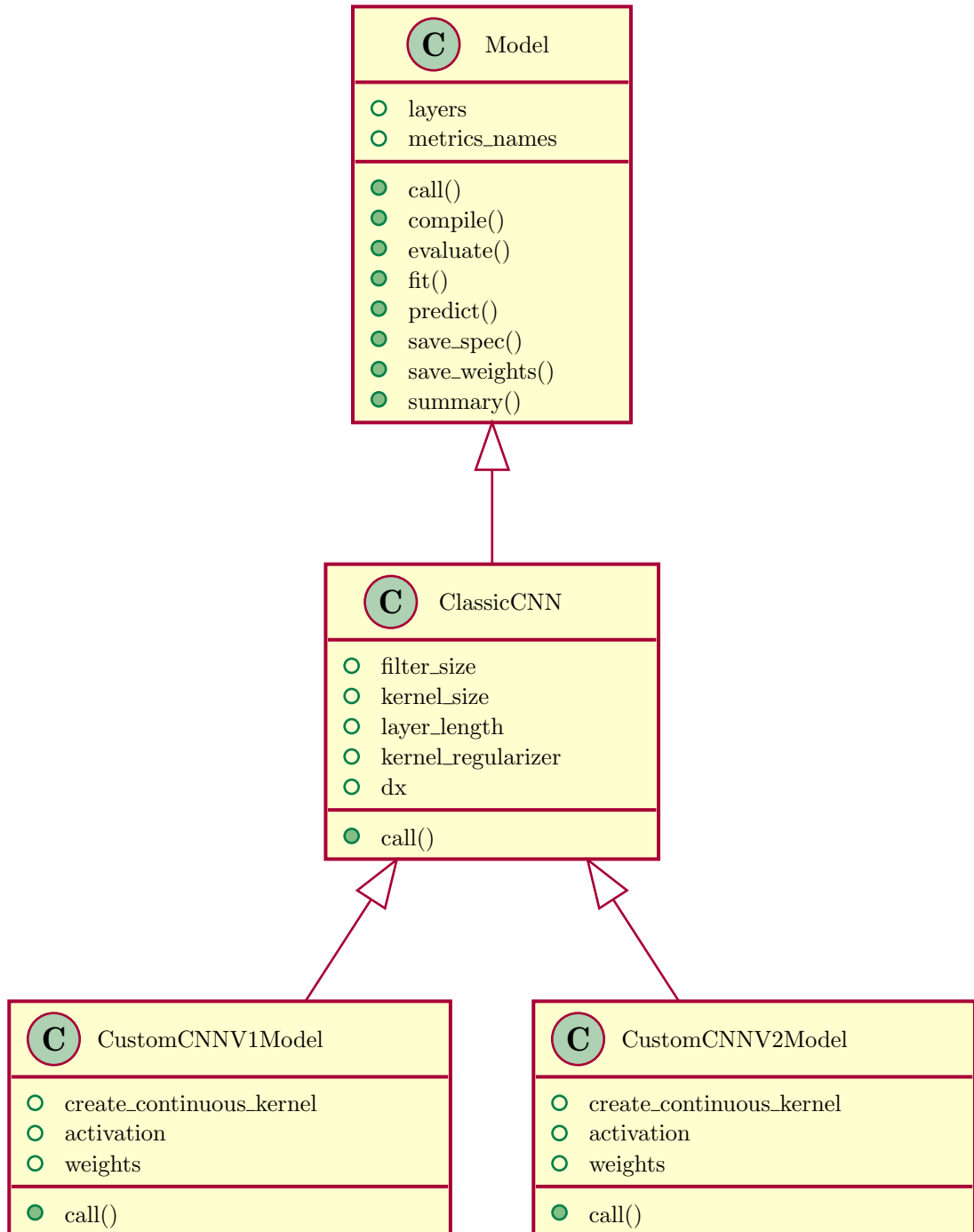


Figure 2.8: UML Diagram of CCNN Models Version 1 and 2



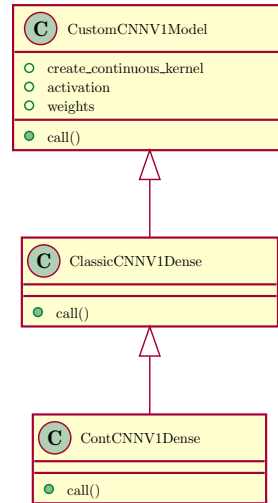


Figure 2.9: UML Diagram of CCNN Dense Models Version 1

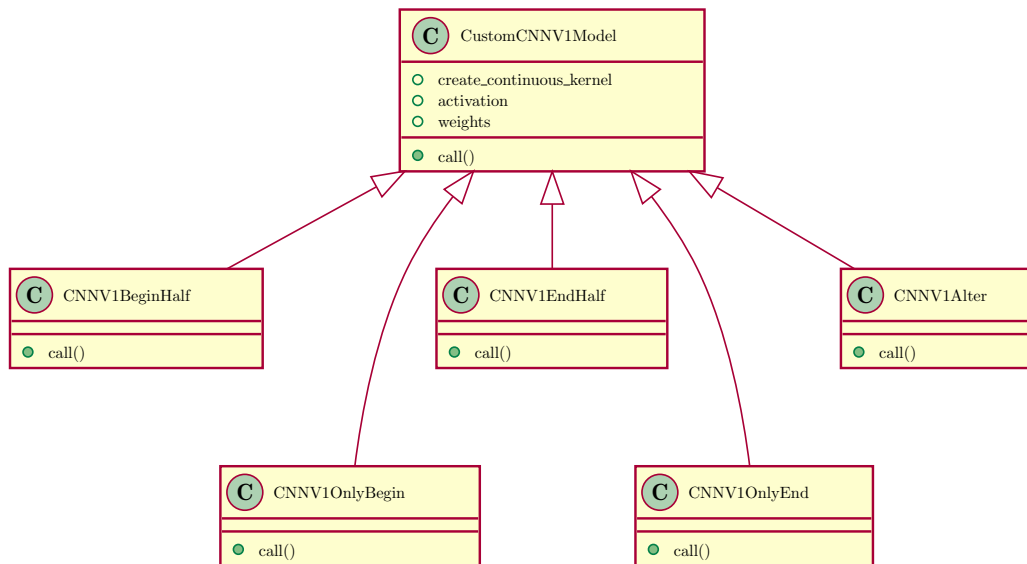


Figure 2.10: UML Diagram of CCNN Models with combination of continuous and non-continuous layers Version 1 and 2

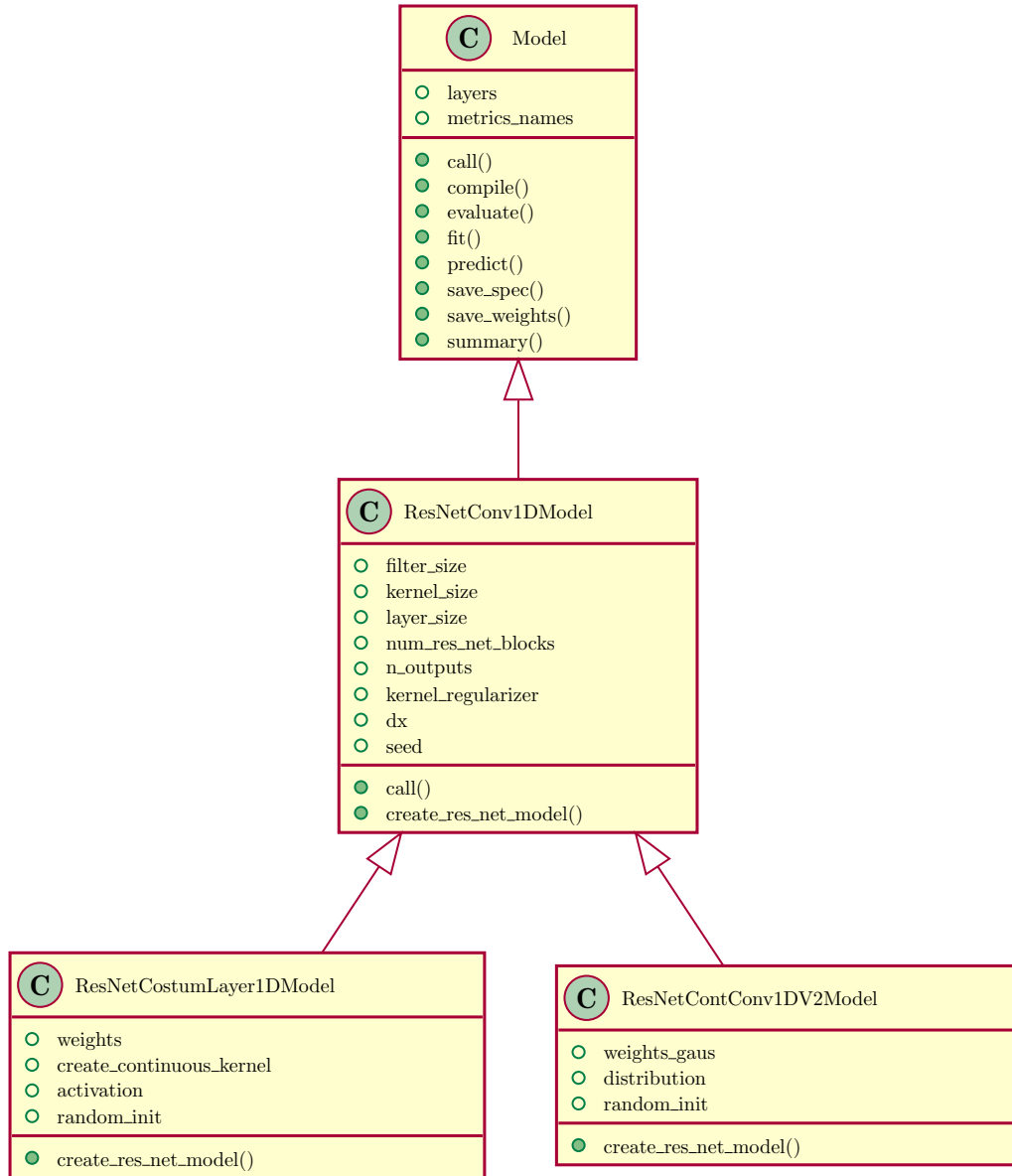


Figure 2.11: UML Diagram of ResNet Models Version 1 and 2

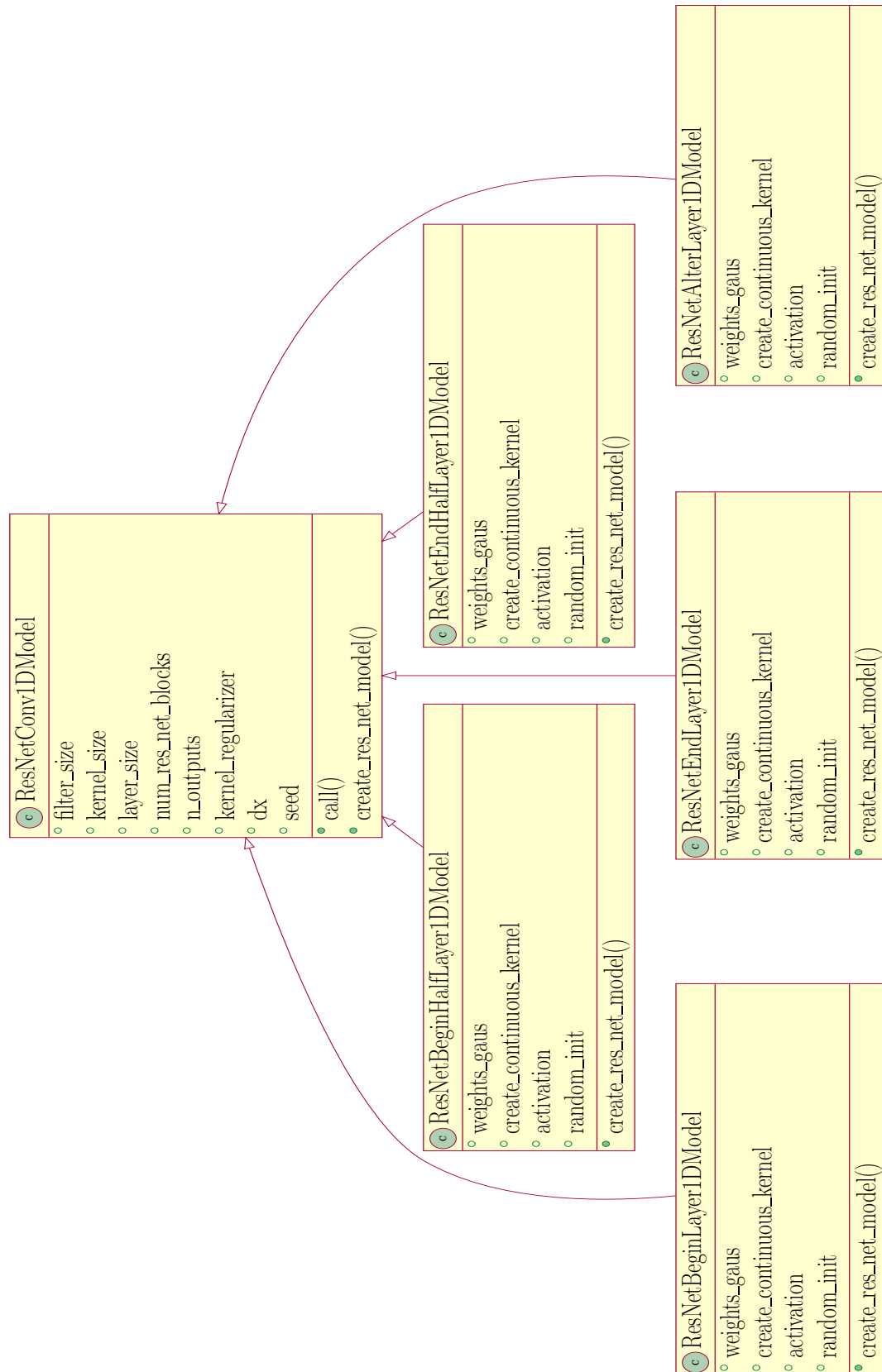


Figure 2.12: UML Diagram of CCNN Models Version 1

## 3 Results

### 3.1 Results CCNN - Version 1

The results in Table 3.1 show that for the implementation of the continuous layer, the mean square error (mse) and the mean absolute error (mae) for the kinetic energy  $T$  and its derivative  $\frac{\delta T}{\delta n}$  are of the same order of magnitude as for the reference model *CNN*, with the Glorot uniform distribution as the kernel function. It can also be seen that the model with the harmonic kernel (*CCNNV1Harm*) function outperforms the reference by 1kcal/mol, which is not significantly better.

For all three continuous kernel functions a grid search on the hyperparameters from table 2.1 was performed. The result can be seen in Tables 3.3 and 3.4, where the hyperparameter configurations which gave the lowest mse and mae for  $T$  and  $\frac{\delta T}{\delta n}$  were used to train the models *CCNNV1GaussOpt* and *CCNNV1HarmOpt*. The hyperparameter grid search was unsuccessful for the trigonometric kernel function because the hyperparameters in table 2.1 ran either into the exploding gradient or the vanishing gradient problem. The mse and mae improved for the model *CCNNV1GaussOpt*, whereas the model *CCNNV1HarmOpt* failed to deliver better results.

In Table 3.2 the results of various combinations of continuous and non-continuous layers can be seen, where they attempted to combine the advantages of random start from the non-continuous Glorot uniform distribution and continuous distributions. The results show that the mse and the mae for the  $T$  and  $\frac{\delta T}{\delta n}$  are of the same order of magnitude, but are not better than the reference for any combination.

Figure 3.1 shows one kernel of the first layer before training and the same kernel of after training to demonstrate the effect of the training on several kernel functions. A kernel consists of 32 kernel functions. The figures were also created to see how and if the optimizer trains the custom kernel functions. In the case of continuous kernel functions (Gaussian, harmonic, trigonometric), it can be seen that the kernels remain continuous and can be trained even if they do not start at a general position, as the models with the Glorot uniform kernel function do. The effect of the optimizer and the training for models with only continuous layers is most visible for the model

*CCNNV1Gauss*. The smallest change compared to before and after training is visible for the kernel with the harmonic kernel functions.

In Figure 3.2 it can be seen that for the model *CCNNV1EndHalf*, which started with a Glorot uniform kernel function, the optimizer is trying to train the weights the same way as for the layers with continuous kernels compared to the *CNN* reference model, where the kernel function remains non-continuous and noisy after the training as it can be seen in Figure 3.1. For the *CCNNV1OnlyEnd* model, the optimizer trained the noise non-continuous Glorot uniform kernel to a continuous and smooth kernel. For the models *CCNNV1Alter*, *CCNNV1BeginHalf*, and *CCNNV1OnlyBeginn* there is a step in the middle of the kernel functions, where the kinetic energy derivative has its peak as can be seen in Figure 2.2.

Using the kernel function that gave the best result in table 3.1, a model of a combination with the Glorot uniform function, where half of the functions in a kernel are harmonic, and the other half are Glorot uniform, was trained. The result can be seen in Table 3.1 under the model name *CCNNV1Mixed*. It shows that the model outperforms the reference by about 2kcal/mol, which is also not significantly better.

Table 3.1: Mean squared error (mse), mean absolute error (mae), and the maximum of mae (max) in kcal/mol for CCNN - Version 1 with Gaussian, harmonic, trigonometric, and Glorot uniform kernel functions. With  $\Delta T$  the kinetic energy and  $\Delta \frac{\delta T}{\delta n}$  it's kinetic energy derivative. A list of hyperparameters in Table 2.1. The hyperparameters for the model *CCNNV1GaussOpt* and *CCNNV1HarmOpt* are listed in the Tables 3.3 and 3.4.

model	$ \Delta T $			$ \Delta \frac{\delta T}{\delta n} $		
	mse	mea	max	mse	mea	max
CCNNV1Gauss	0.0003	0.2943	2166.4549	3.6461	26.2798	2166.4549
CCNNV1GaussOpt	0.0002	0.3081	3407.8603	0.9772	15.6202	2188.8060
CCNNV1Harm	0.0002	0.2758	2302.0575	0.3180	8.4085	2302.0575
CCNNV1HarmOpt	0.0002	0.2873	1017.2919	0.9237	15.1215	2241.9463
CCNNV1HarmMixed	0.0002	0.2732	1432.7363	0.4620	7.1195	2167.7309
CCNNV1Trigo	0.0002	0.2796	3407.8609	0.6299	10.7188	2188.8348
CNN	0.0002	0.2688	2128.3019	0.5064	9.4125	2128.3019

### 3 Results

Table 3.2: Mean squared error (mse), mean absolute error (mae), and the maximum of mae (max) in kcal/mol for CCNN - Version 1 with combination of continuous and non-continuous layers, which are described in Table 2.2. With  $\Delta T$  the kinetic energy and  $\Delta \frac{\delta T}{\delta n}$  it's kinetic energy derivative. The hyperparameter configuration is provided in Table 2.1

model	$ \Delta T $			$ \Delta \frac{\delta T}{\delta n} $		
	mse	mea	max	mse	mea	max
CCNNV1BeginHalf	0.0004	0.2846	2188.8348	4.2358	27.3365	2188.8348
CCNNV1OnlyBegin	0.0002	0.3003	2188.8348	0.6010	12.2129	2188.8348
CCNNV1EndHalf	0.0002	0.3062	0.0000	1.0880	14.6321	0.0000
CCNNV1OnlyEnd	0.0002	0.3062	2114.5265	1.0880	14.6321	2114.5265
CCNNV1Alter	0.0002	0.2794	0.0000	1.3178	15.7503	0.0000

Table 3.3: Mean squared error (mse), mean absolute error (mae) in kcal/mol CCNN - Version 1 hyperparameter grid search with a harmonic kernel function with the hyperparameters from Table 2.1. Only runs with a mae for  $\Delta \frac{\delta T}{\delta n}$  below 50 kcal/mol are shown for better visibility.

fl	actfun	kl	l2reg	$ \Delta T $		$ \Delta \frac{\delta T}{\delta n} $	
				mse	mea	mse	mea
32.0	softplus	100.0	0.0025	0.0004	1.4046	0.4526	18.3664
64.0	softplus	50.0	0.0025	0.0007	2.864	0.5552	24.5639
16.0	softplus	50.0	0.0025	0.0006	2.8387	0.4468	24.8076
64.0	softplus	125.0	0.0025	0.0005	3.2656	0.313	26.6331

Table 3.4: Mean squared error (mse), mean absolute error (mae) in kcal/mol CCNN - Version 1 hyperparameter grid search with a Gaussian kernel function with the hyperparameters from Table 2.1. Only runs with a mae for  $\Delta \frac{\delta T}{\delta n}$  below 50 kcal/mol are shown for better visibility.

fl	actfun	kl	l2reg	$ \Delta T $		$ \Delta \frac{\delta T}{\delta n} $	
				mse	mea	mse	mea
64.0	softplus	50.0	0.0001	0.0003	0.7104	0.3665	12.4053
64.0	exponential	50.0	0.0001	0.0003	0.6412	0.2944	11.9056
64.0	softplus	50.0	0.0	0.0006	0.9699	0.5545	14.5317
32.0	softplus	50.0	0.0	0.0007	1.1231	0.5643	16.3775
32.0	exponential	50.0	0.0001	0.0006	1.2397	0.5408	17.1276
16.0	softplus	50.0	0.0001	0.0006	2.6195	0.473	24.1931
32.0	exponential	50.0	0.0	0.0006	2.7385	0.5252	25.1448
32.0	softplus	50.0	0.0001	0.0005	2.8697	0.4158	26.3697
16.0	softplus	50.0	0.0	0.0006	3.6918	0.5018	26.8585
64.0	softplus	125.0	0.0001	0.0002	2.8669	0.292	27.0016
16.0	softplus	100.0	0.0	0.0003	3.1088	0.2903	28.6019
16.0	softplus	100.0	0.0001	0.0004	3.6345	0.3128	29.774
32.0	softplus	100.0	0.0001	0.0007	3.4361	0.5786	29.9962
64.0	softplus	125.0	0.0	0.0004	5.9632	0.3336	34.7492
32.0	softplus	100.0	0.0	0.0005	4.9082	0.3018	35.1827
32.0	softplus	125.0	0.0	0.0004	6.4416	0.3251	39.4163
16.0	softplus	125.0	0.0	0.0006	9.252	0.5048	48.5123

### 3 Results

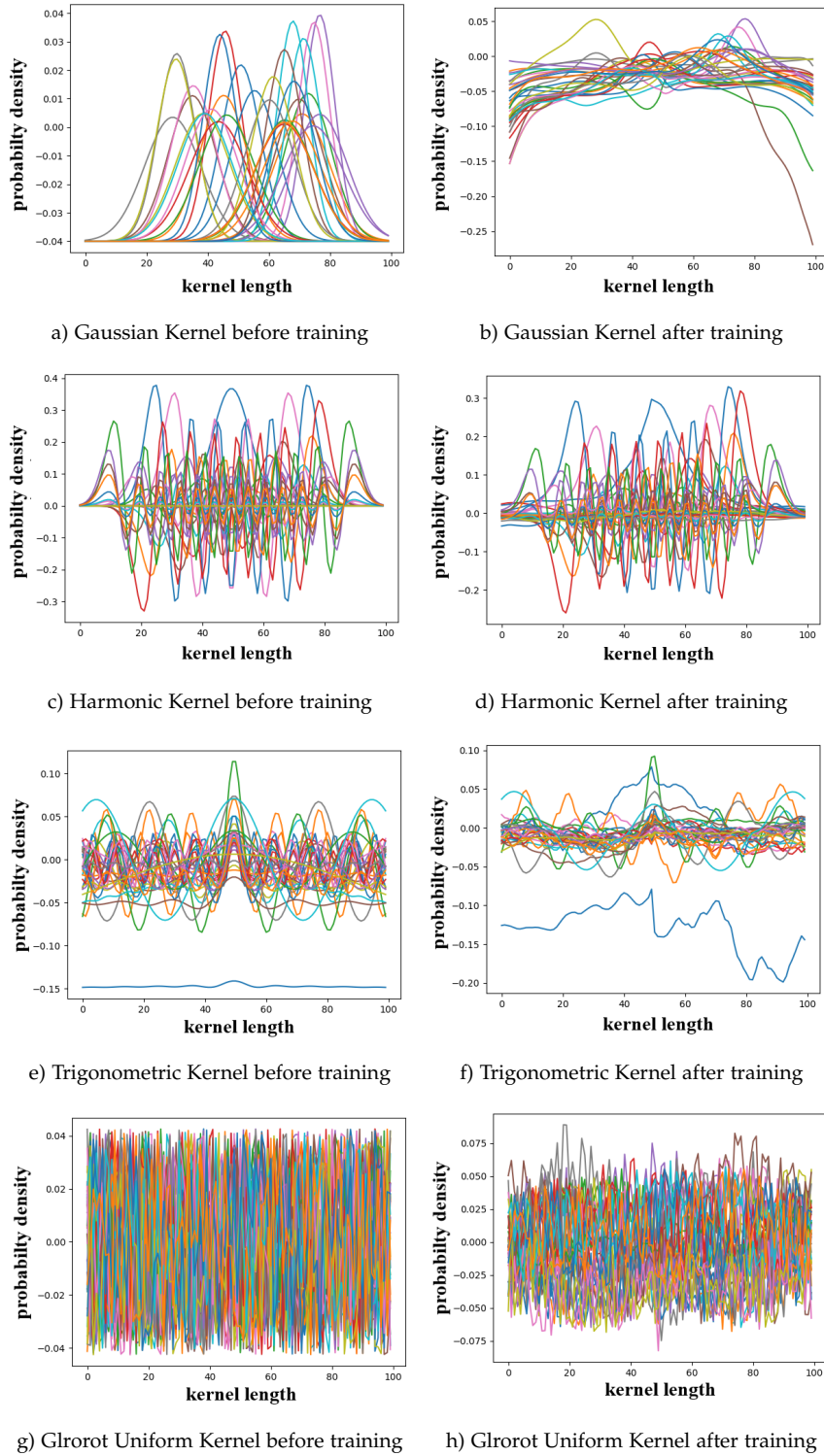


Figure 3.1: Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions. Left is before the training and right after the training for the runs in table 3.1.



### 3 Results

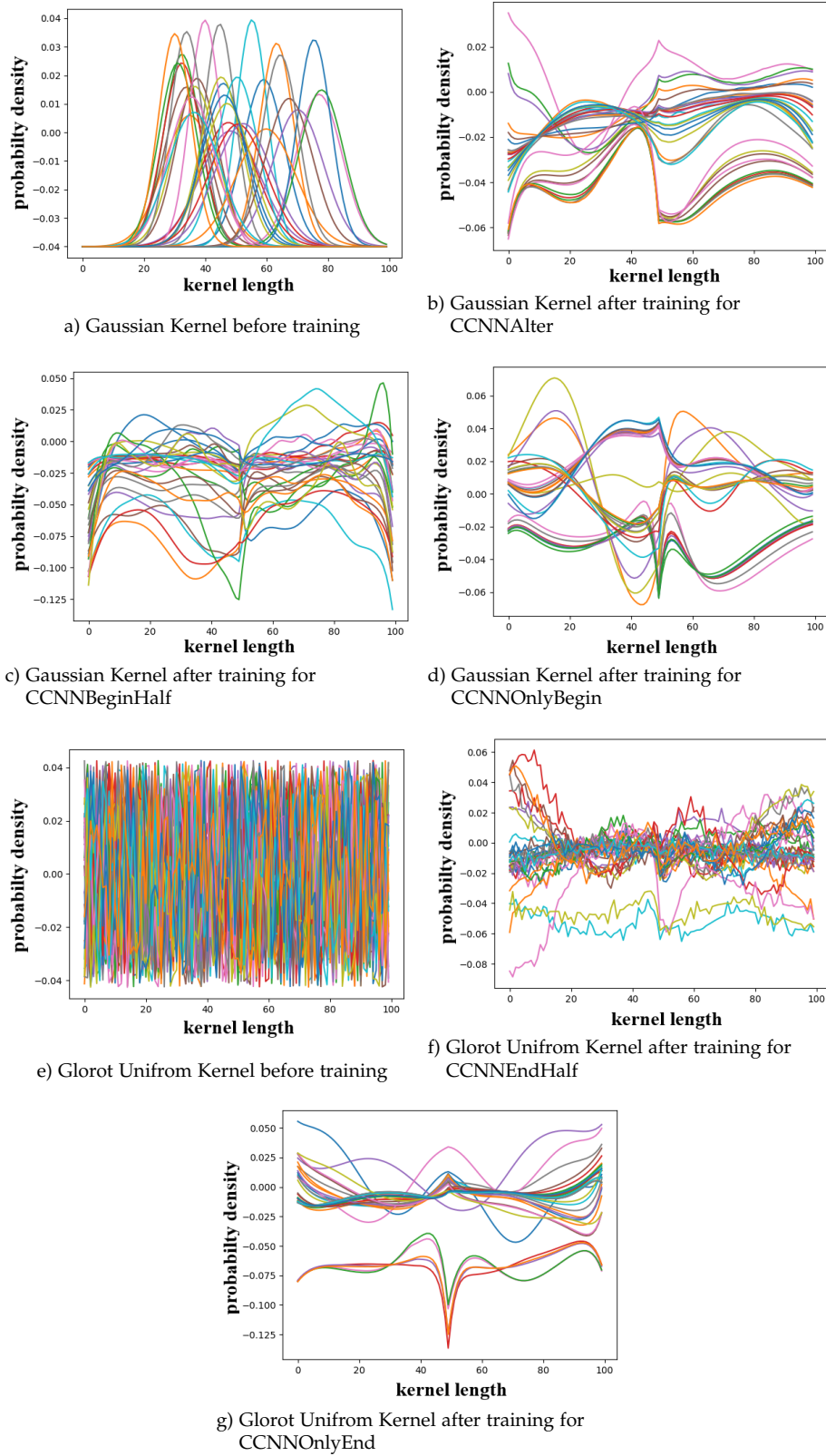


Figure 3.2: Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions for the runs in table 3.2

## 3.2 Results CCNN - Version 2

In the second version of CCNN, only the parameters of the Gaussian, Cauchy, and Lorentz distributions were trained, which are then used to compute the kernel functions. During the training of the models in table 3.5 the problem of vanishing and exploding gradient (1.2.5) appeared.

The vanishing gradient is noticeable in that during training the errors (mse and mae) stagnate and the model is no longer able to reduce the error, as was the case with *CCNNV2Gaussian*. The exploding gradient, on the other hand, is noticeable in the *CCNNV2Cauchy* and *CCNNV2Lorentz* models, where the optimizer acts too strongly on the weights and the computed distribution for the kernel is then no longer within in a valid range for the kernel. The result of this effect is that the mse and mae are *NaN* values. For multiple runs with the same configuration, all three models delivered the same result.

Table 3.5: Mean squared error (mse), mean absolute error (mae), and the maximum of mae (max) in kcal/mol for CCNN - Version 2 with Gaussian, Chauchy, Lorentz. With  $\Delta T$  the kinetic energy and  $\Delta \frac{\delta T}{\delta n}$  it's kinetic energy derivative. The hyperparameter configuration is provided in Table 2.1.

model	$ \Delta T $			$ \Delta \frac{\delta T}{\delta n} $		
	mse	mae	max	loss	mean	max
CCNNV2Gaussian	14658	3032	747850	3706	1236	586340
CCNNV2Cauchy	<i>NaN</i>	<i>NaN</i>	995080	<i>NaN</i>	<i>NaN</i>	826343
CCNNV2Lorentz	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>

### 3.3 Results ResNet - Version 1

The results in table 3.6 show that for the implementation of the continuous layer, the mse and the mae for the kinetic energy  $T$  and its derivative  $\frac{\delta T}{\delta n}$  are in the same order of magnitude as the reference model *Resnet* is for the CCNN-version 1 (2.2.1). In comparison to the results for the CCNN - version 1 (see Table 3.1) no model outperforms the reference implementation. Also, the harmonic distribution function model was not trainable because of the exploding gradient problem (1.2.5), while the *CNNV1Harm* model could be trained successfully as can be seen in Table 3.1.

Since the trigonometric kernel function gave the best result for the mse and mea of  $T$  and  $\frac{\delta T}{\delta n}$  in Table 3.6 it was used to create a model for the model *CCNNV1HarmMixed* (see table 3.1), where the continuous and non-continuous kernel functions are half/half mixed in one kernel. The result can be seen in table 3.6 under the model name *ResNetTrigoMixed*, and it shows that the mse and the mae for  $T$  and  $\frac{\delta T}{\delta n}$  improved in comparison to the model *ResNetTrigo*. Still, it does not outperform the reference model *ResNet*.

A hyperparameter search was only successful for the model *ResNetV1Gauss* with the Gaussian kernel function and improved the mse and mae for  $\frac{\delta T}{\delta n}$ , but failed to improve it for  $T$ . The hyperparameters which were chosen for the model *ResNetGuassOpt* can be seen in the table 3.8 and were chosen to give the lowest mse and mea for the  $T$  and  $\frac{\delta T}{\delta n}$ .

Figure 3.3 shows like for the plots for CCNN - V1 3.1, one kernel of the first layer before training and the same kernel of after training to demonstrate the effect of the training on several kernel functions. Similar to the results for CCNN - V1 3.1, it can be seen that the Gaussian kernel function shows the largest effect of training and that the change is smallest for the trigonometric kernel functions. Both custom kernel functions stay continuous after the training.

### 3 Results

Table 3.6: Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) in kcal/mol for ResNet - Version 1 with Gaussian, Lorentz, Cauchy, and Glorot uniform kernel functions. With  $\Delta T$  the kinetic energy and  $\Delta \frac{\delta T}{\delta n}$  its kinetic energy derivative. The hyperparameter configuration is listed in table 2.1.

model	$ \Delta T $			$ \Delta \frac{\delta T}{\delta n} $		
	mse	mae	max	mse	mae	max
ResNetV1Gauss	0.0027	0.3340	43583.7247	8.3878	33.2968	50863.1463
ResNetV1GaussOpt	0.0003	0.3596	7438.9101	2.1824	22.1267	10763.0969
ResNetV1Harm	NaN	NaN	NaN	NaN	NaN	NaN
ResNetV1Trigo	0.0002	0.2957	3395.9470	0.6498	13.0167	2188.8347
ResNetV1TrigoMixed	0.0002	0.2853	58165.1983	0.5596	11.1948	89539.6972
Resnet	0.0002	0.2905	2876.9487	0.5194	10.8088	2112.2659

Similar for the CCNV1 - version 1, several models with a combination of continuous and non-continuous layers were trained to see if the advantages of both layers can be combined. As can be seen in Table 3.7, no combination performed better than the reference in Table 3.6, but the mse and the mae are in the same order of magnitude. In addition, the kernel functions for the combinations of the continuous ResNet models also show the jump in the middle, as can be seen in Figure 3.4. The kernel functions for the *ResNetV1EndHalf* and *ResNetV1OnlyEnd* models do not exhibit the same behavior as the *CCNNV1EndHalf* and *CCNV1OnlyEnd* models to smoothe the Glorot uniform kernel functions.

Table 3.7: Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) for ResNet - Version 1 with combination of continuous and non-continuous layers. With  $\Delta T$  the kinetic energy and  $\Delta \frac{\delta T}{\delta n}$  it's kinetic energy derivative. The hyperparameter configuration are listed in Table 2.1 and the description of the combination in Table 2.2.

model	$ \Delta T $			$ \Delta \frac{\delta T}{\delta n} $		
	loss	mean	max	loss	mean	max
ResNetV1Alter	0.0002	0.2826	65.4583	0.5866	12.3649	466.3916
ResNetV1BeginHalf	0.0002	0.2876	183.1679	1.4588	15.1504	426.7068
ResNetV1End	0.0010	0.3228	3390.6946	6.4972	33.7884	2114.5265
ResNetV1EndHalf	0.0002	0.2764	3390.8882	1.9758	20.5435	2114.5265
ResNetV1Begin	0.0002	0.2883	314.2425	0.7036	12.5735	575.6891

### 3 Results

Table 3.8: Mean Squared Error (mse) and Mean Absolute Error (mae) ResNet - Version 1 hyperparameter grid search. Only runs with a mae for  $\Delta \frac{\delta T}{\delta n}$  below 60 kcal/mol are shown for better visibility.

fl	actfun	kl	l2reg	$ \Delta T $		$ \Delta \frac{\delta T}{\delta n} $	
				mse	mea	mse	mea
64	softplus	50	0	0.0015	0.6852	4.5418	32.1709
32	sigmoid	50	0	0.0605	6.1326	4.2662	31.1984
32	sigmoid	50	0	0.0012	0.735	4.4673	32.9463
16	softplus	50	0	0.0116	2.6483	5.2093	37.2702
32	sigmoid	100	0	0.0043	1.4729	5.3486	38.3794
32	sigmoid	50	0	0.0225	3.7082	8.4317	40.7178
16	softplus	50	0	0.022	3.6987	6.2427	41.9366
16	softplus	100	0	0.0511	5.5899	8.5412	45.8214
16	softplus	50	0	0.0593	6.0243	9.0184	46.9465
16	sigmoid	50	0	0.1151	8.3998	8.5753	48.4906
64	sigmoid	125	0	0.0523	5.6331	9.115	49.3164
32	sigmoid	100	0	0.0055	1.7408	12.2599	50.8923
16	sigmoid	50	0	0.012	2.1531	11.0191	57.6687
32	softplus	50	0	0.1167	8.5425	11.0361	57.9628

### 3 Results

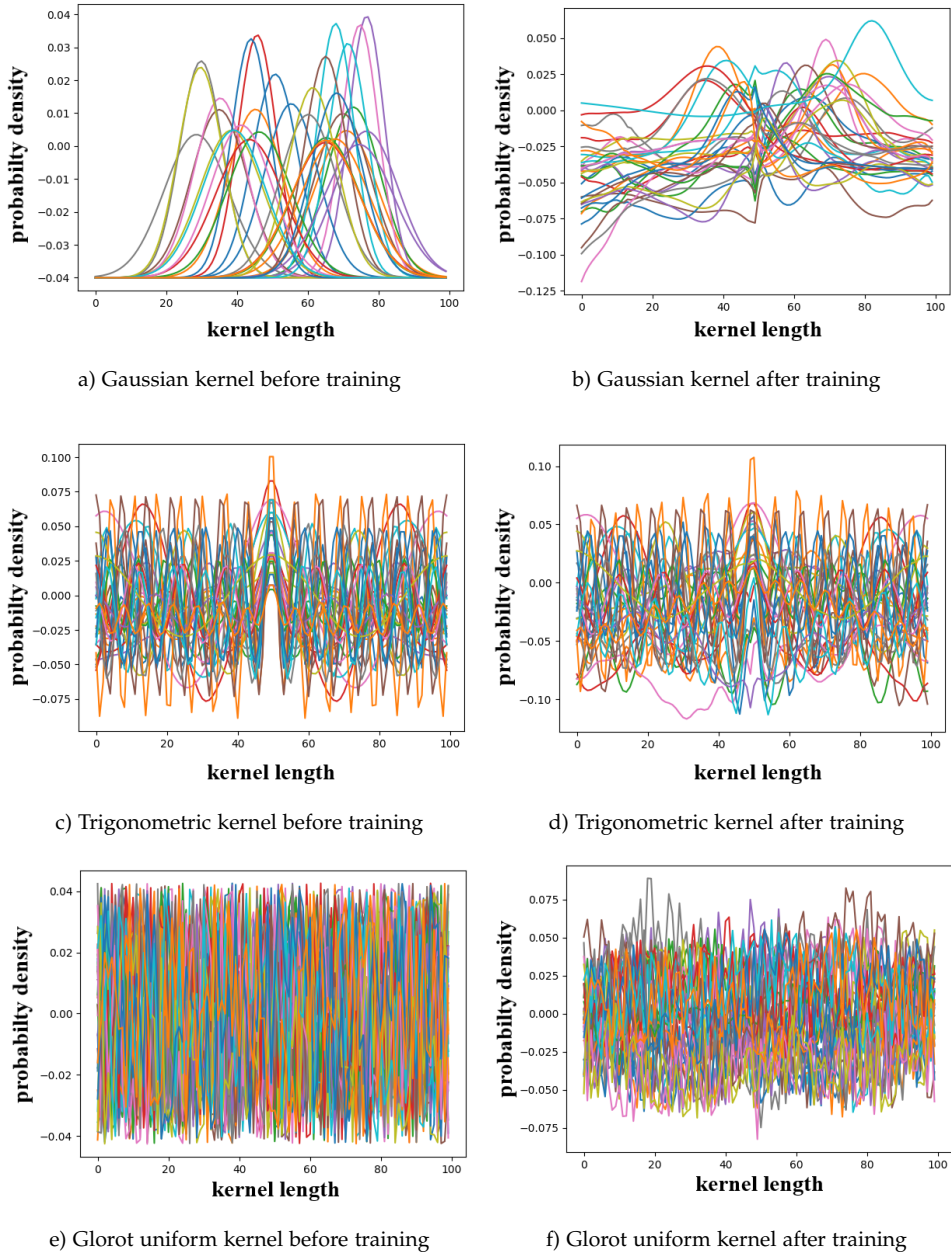


Figure 3.3: Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions. Left is before the training and right after the training for the runs in table 3.6.

### 3 Results

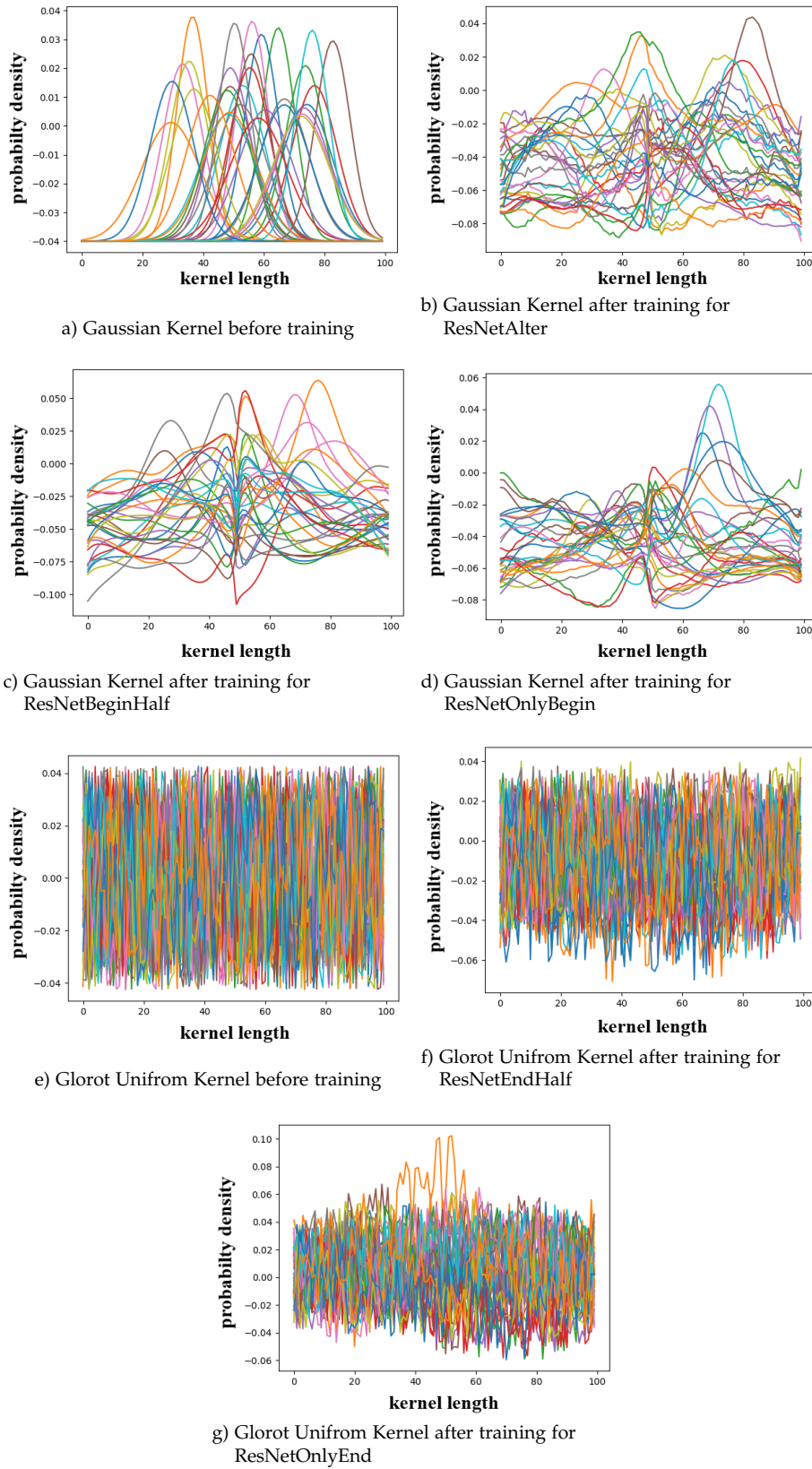


Figure 3.4: Plot for 32 kernel functions of one kernel of the first layer for several kernel functions to see if and how the optimization trains the custom kernel functions for the runs in table 3.2.

### 3.4 Results ResNet - Version 2

The second version for the ResNet performed better than the second version for the CCNN model with respect to the exploding gradient problem, as can be seen in Table 3.9 and 3.5. However, the vanishing gradient problem also occurred to the models with Gaussian and Cauchy kernel functions. A grid search on the hyperparameters from the Tables 2.5 to 2.7 was performed for all three kernel functions to review if other optimizers and their parameters are able to circumvent the vanishing gradient problem, and are better suited for weight reduction.

Only the model with the Gaussian kernel function gave results that are not *NaN* for the grid search. The result of the model with the hyperparameter configuration in Table 3.10, which gave the lowest mse and mae for the kinetic energy and its derivative, can be seen in table 3.9 under the name *ResNetV2GaussianOpt*. The optimized hyperparameter configuration for the model *ResNetV2GaussianOpt* was able to reduce the error, but the model still had the vanishing gradient problem.

In Figure 3.5 for the model *ResNetV2GaussianOpt*, it can be seen that for some kernel functions the sign of the weight representing the amplitude is reversed, and that the weights representing the mean and standard deviation for the Gaussian kernel function change only slightly for 100000 epochs of training.

Table 3.9: Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) for ResNet - Version 2 with Gaussian, Lorentz, and Cauchy kernel functions (in kcal/mol). With  $\Delta T$  the kinetic energy and  $\Delta \frac{\delta T}{\delta n}$  its kinetic energy derivative. The corresponding hyperparameter configuration is provided in Table 2.1.

Model	$ \Delta T $			$ \Delta \frac{\delta T}{\delta n} $		
	mse	mea	max	mse	mea	max
ResNetV2Cauchy	$455 \cdot 10^1$	$168 \cdot 10^1$	$136 \cdot 10^4$	$157 \cdot 10^8$	$211 \cdot 10^1$	$189 \cdot 10^4$
ResNetV2Gaussian	48.8416	139.8193	$957 \cdot 10^3$	$157 \cdot 10^2$	$211 \cdot 10^1$	$116 \cdot 10^4$
ResNetV2GaussianOpt	0.8966	21.0541	$957 \cdot 10^3$	$201 \cdot 10^1$	578.3750	$116 \cdot 10^4$
ResNetV2Lorentz	<i>nan</i>	<i>nan</i>	$514 \cdot 10^16$	<i>nan</i>	<i>nan</i>	$438 \cdot 10^16$



### 3 Results

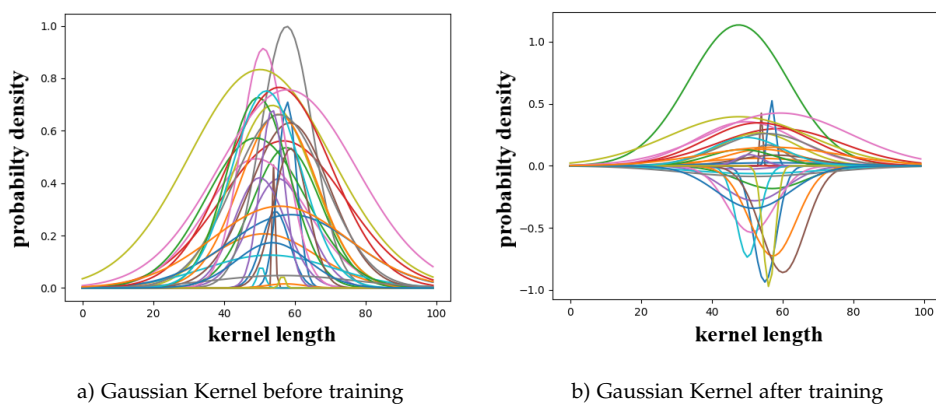


Figure 3.5: Kernel plot of the first layer before training for the runs in table 3.9

Table 3.10: Mean squared error (mse), mean absolute error (mae) ResNet - version 2 hyper-parameter grid search with a Gaussian kernel function. Only runs with a mae for  $\Delta \frac{\delta T}{\delta n}$  below 1000 kcal/mol are shown for better visibility.

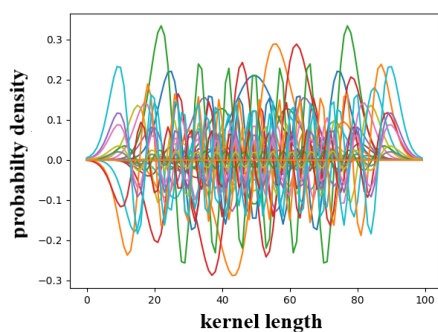
lr	rho	optimizer	$\beta_1$	$\beta_2$	$ \Delta T $		$ \Delta \frac{\delta T}{\delta n} $	
					mse	mea	mse	mea
0.001	-	nadam	0.09	0.0999	0.5316	1273	16.1791	676
0.001	-	adam	0.1	0.0999	7.0591	1171	63.804	670
0.001	-	adam	0.09	0.0999	2.7774	1336	39.5401	691
0.001	-	nadam	0.009	0.0999	8.676	1486	71.3246	741
0.001	-	adam	0.1	0.1	0.7295	1581	14.055	789
0.001	-	nadam	0.1	0.0999	5.2519	1570	53.8401	792
0.001	-	adam	0.009	0.0999	0.8298	1643	19.4703	794
0.001	-	adam	0.009	0.1	19.2641	1693	108.3537	797
0.001	-	nadam	0.1	0.1	9.5619	1808	73.0695	837
0.001	-	adam	0.1	0.01	4.1421	1844	46.7154	849
0.0001	-	adam	0.09	0.1	1878.4877	1967	1085.488	876
0.01	-	adam	0.9	0.999	1.2981	1953	19.7757	885
0.001	-	nadam	0.09	0.01	33.7671	2099	141.5056	896
0.001	-	nadam	0.09	0.1	8.7877	2029	67.7983	898
0.001	-	adam	0.09	0.1	7.9225	20393	62.4776	902
0.001	-	nadam	0.1	0.01	15.3692	2044	95.5648	910
0.001	-	adam	0.09	0.01	6.4945	2167	58.4546	936
1.0	0.95	adadelta	-	-	191.1371	2167	345.4543	939

### 3.5 Results for Dense CCNN

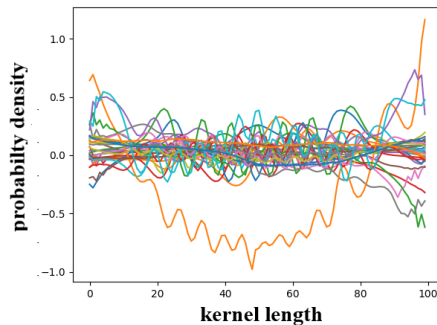
For the combination of a CCNN and a DNN (Dense Neural Network), the models are trained with a linear and a softplus activation for the convolutional layer. In the Table, it can be seen that for no Dense CCNN model, the errors were as low as for the previous models of CCNNV<sub>1</sub> or ResNetV<sub>1</sub>. In the Figures from 3.7 to 3.6 it can be seen that the training has a high effect on the continuous kernel functions.

Table 3.11: Mean Squared Error (mse), Mean Absolute Error (mae), and the maximum of mae (max) for Dense CCNN - Version 1 with Gaussian, Harmonic, Trigonometric and Glorot uniform kernel functions (in kcal/mol). With  $\Delta T$  the kinetic energy and  $\Delta \frac{\delta T}{\delta n}$  it's kinetic energy derivative. The corresponding hyperparameter configuration is provided in Table 2.1.

Model	activation	$ \Delta T $			$ \Delta \frac{\delta T}{\delta n} $		
		mse	mea	max	mse	mea	max
Gaussian	softplus	0.0023	0.7791	2738	46.5289	101.8658	2188
Gaussian	linear	0.0006	0.4980	2810	47.4775	83.9335	2200
Trigonometric	softplus	0.0850	7.2804	2736	33.1676	84.3073	2188
Trigonometric	linear	0.0008	0.5243	2748	17.8016	56.6206	2197
Harmonic	softplus	0.0042	0.9406	2715	66.2126	117.9190	2182
Harmonic	linear	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>	<i>NaN</i>



a) Harmonic Kernel before training



b) Harmonic Kernel after training with a softplus activation function

Figure 3.6: Kernel plot of the first layer before and after training for the harmonic run in table 3.11

### 3 Results

---

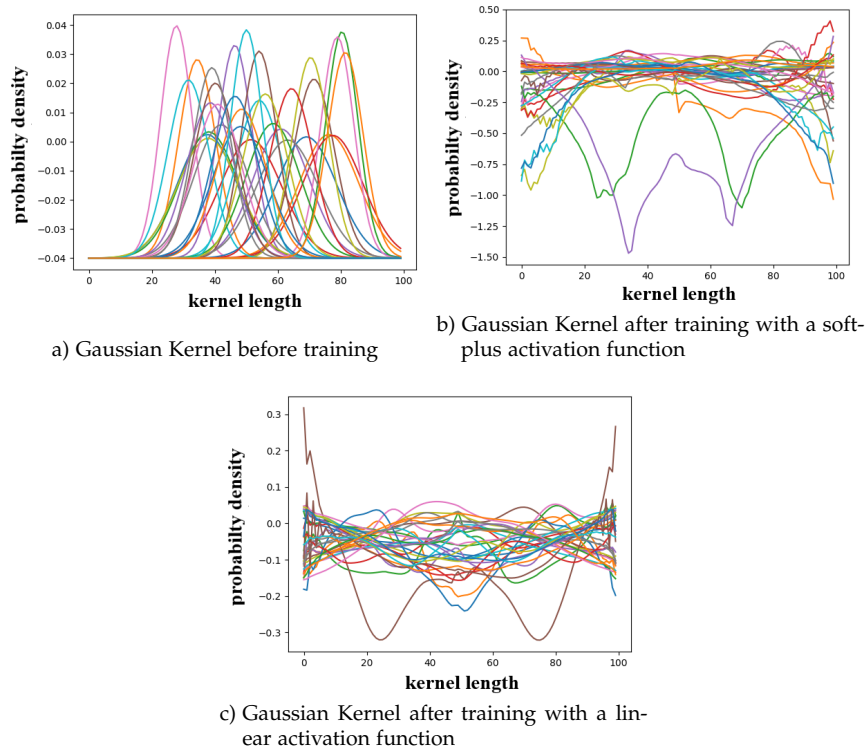


Figure 3.7: Kernel plot of the first layer before and after training for the Gaussian run in table 3.11

### 3 Results

---

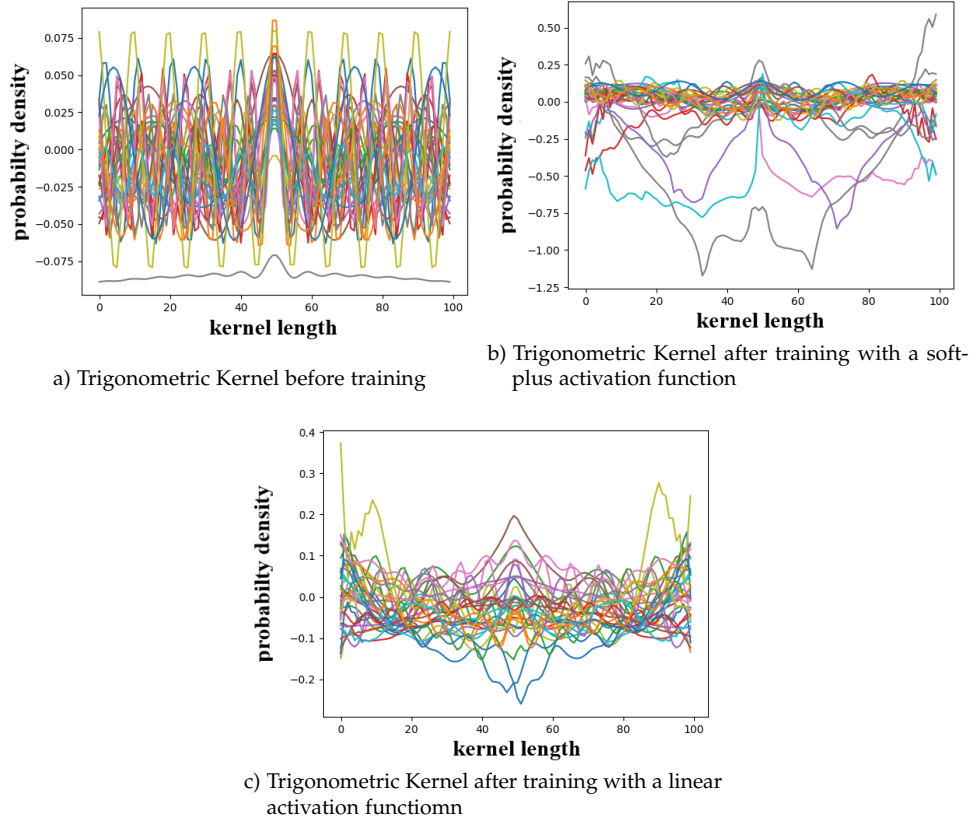


Figure 3.8: Kernel plot of the first layer before and after training for the trigonometric run in table 3.11

## 4 Conclusion

The results for the CCNNV1 3.1 show that replacing the random noisy kernel of a layer with a continuous kernel does not significantly reduce the mean squared error and mean absolute error for the kinetic energy derivative, as can be seen in Table 3.1. Moreover, the result for the *CCNNV1HarmMixed* model in Table 3.1 shows that a combination of continuous and non-continuous kernel functions in one layer cannot combine the advantage of a random noisy kernel function and a continuous one, and therefore cannot significantly reduce the mean squared error and the mean absolute error for the kinetic energy derivative compared to the reference *CNN* model. From the results shown in table 3.2, it can be seen that also the combination of continuous and non-continuous layers cannot combine both advantages to reduce the mean squared error and mean absolute error for the kinetic energy derivative.

Figures 3.1 and 3.2 illustrated that the continuous kernel functions are trainable and remain continuous after training, but do not help to reduce the mean squared error and mean absolute error for the kinetic energy derivative. The same issues and implications of the CCNNV1 are seen for the ResNetV1 3.3 in the results of the Tables 3.9 and 3.10, except that for ResNetV1 no model performs better than the reference. Just as the CCNNV1, the figures for the ResNetV1 3.3 also show continuous behavior.

The conclusion that can be drawn from this for the CCNNV1 and ResNetV1 results is that by replacing the random and noisy kernel values with continuous and smooth values, a discrete convolution cannot be simulated and thus a smooth functional derivative, and transition invariance cannot be achieved. As a consequence, for CCNN layers and a DNN as described in 2.2.2 it can be seen in the Table 3.11 that chemical accuracy still remains out of reach.

For the second version of the CCNN and ResNet, the results in Tables 3.5 and 3.9 show that the models encounter the vanishing and exploding gradient problem when training only the parameters of the distributions 2.2.3, and are therefore not trainable. This happens because of the parameter reduction from 100 weights per kernel to only three trainable weights.

## 4.1 Outlook

A CNN can best be trained with Euclidean data in Euclidean space, while Graph Neural Networks (GNN) [36] and Graph Convolutional Networks (GCN) [18] can be trained successfully with non-Euclidean data. Therefore, the next step would be, to see if the problem with the discrete convolution with non-equidistant data can be overcome via a GCN implementation.

# Bibliography

- [1] Anaconda. *Anaconda*. 2021. URL: <https://www.anaconda.com/> (visited on 09/11/2021) (cit. on p. 13).
- [2] Anaconda. *Documentation for installing Anaconda*. 2021. URL: <https://docs.anaconda.com/anaconda/install/index.html> (visited on 09/11/2021) (cit. on p. 13).
- [3] Rohan Anil et al. "Memory-Efficient Adaptive Optimization for Large-Scale Learning." In: *CoRR* abs/1901.11150 (2019). arXiv: 1901.11150. URL: <http://arxiv.org/abs/1901.11150> (cit. on p. 11).
- [4] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1996. ISBN: 0198538499 (cit. on p. 10).
- [5] K Burke. "The ABC of DFT." In: (Apr. 2007), p. 92697 (cit. on pp. 1–3).
- [6] NVIDIA Corporation. *CUDA C++ Programming Guide*. 2021. URL: [https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#from-graphics-processing-to-general-purpose-parallel-computing\\_\\_gpu-devotes-more-transistors-to-data-processing](https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#from-graphics-processing-to-general-purpose-parallel-computing__gpu-devotes-more-transistors-to-data-processing) (visited on 05/14/2021) (cit. on p. 12).
- [7] NVIDIA Corporation. *NVIDIA*. 2021. URL: <https://www.nvidia.com> (visited on 09/11/2021) (cit. on p. 14).
- [8] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterton. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: <http://proceedings.mlr.press/v9/glorot10a.html> (cit. on p. 9).
- [9] Google. *Google Brain*. 2021. URL: <https://research.google/teams/brain/> (visited on 09/11/2021) (cit. on p. 13).
- [10] Kaiming He et al. "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on p. 10).
- [11] Tong He et al. "Bag of Tricks for Image Classification with Convolutional Neural Networks." In: *CoRR* abs/1812.01187 (2018). arXiv: 1812.01187. URL: <http://arxiv.org/abs/1812.01187> (cit. on p. 11).

- [12] C. H. Hodges. “Quantum Corrections to the Thomas–Fermi Approximation—The Kirzhnits Method.” In: *Canadian Journal of Physics* 51.13 (1973), pp. 1428–1437. DOI: 10.1139/p73-189. eprint: <https://doi.org/10.1139/p73-189>. URL: <https://doi.org/10.1139/p73-189> (cit. on p. 4).
- [13] Chen Huang and Emily A. Carter. “Nonlocal orbital-free kinetic energy density functional for semiconductors.” In: *Phys. Rev. B* 81 (4 Jan. 2010), p. 045206. DOI: 10.1103/PhysRevB.81.045206. URL: <https://link.aps.org/doi/10.1103/PhysRevB.81.045206> (cit. on p. 4).
- [14] Ian Goodfello and Yoshua Bengio and Aaron Courville. <http://www.deeplearningbook.org>. Cambridge, MA, USA: MIT Press, 2016 (cit. on pp. 5, 7, 8).
- [15] Intel. *CPU vs. GPU: Making the Most of Both*. 2021. URL: <https://www.intel.com/content/www/us/en/products/docs/processors/cpu-vs-gpu.html> (visited on 05/13/2021) (cit. on pp. 12, 13).
- [16] Frank Jensen. *Introduction to Computational Chemistry*. Hoboken, NJ, USA: John Wiley Sons, Inc., 2006. ISBN: 0470011874 (cit. on p. 1).
- [17] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: CoRR abs/1412.6980 (2014). arXiv: 1412.6980. URL: <http://arxiv.org/abs/1412.6980> (cit. on p. 11).
- [18] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2016. arXiv: 1609.02907 [cs.LG] (cit. on p. 47).
- [19] Cade Metz. *Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine*. 2021. URL: <https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/> (visited on 09/11/2021) (cit. on p. 13).
- [20] Ralf Meyer, Manuel Weichselbaum, and Andreas W. Hauser. “Machine Learning Approaches toward Orbital-free Density Functional Theory: Simultaneous Training on the Kinetic Energy Density Functional and Its Functional Derivative.” In: *Journal of Chemical Theory and Computation* 16.9 (2020). PMID: 32786898, pp. 5685–5694. DOI: 10.1021/acs.jctc.0c00580. eprint: <https://doi.org/10.1021/acs.jctc.0c00580>. URL: <https://doi.org/10.1021/acs.jctc.0c00580> (cit. on pp. vi, 15, 18, 19).
- [21] W.H. Press et al. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. Cambridge University Press, 2007. ISBN: 9780521880688. URL: <http://nr.com/> (cit. on p. 15).



- [22] Python. *Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine*. 2021. URL: <https://www.python.org/> (visited on 09/11/2021) (cit. on p. 13).
- [23] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond." In: *CoRR* abs/1904.09237 (2019). arXiv: 1904.09237. URL: <http://arxiv.org/abs/1904.09237> (cit. on p. 11).
- [24] Sebastian Ruder. "An overview of gradient descent optimization algorithms." In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747 (cit. on p. 11).
- [25] smaXtec. *smaXtec animal care GmbH*. 2021. URL: <https://smaxtec.com/en/> (visited on 09/11/2021) (cit. on p. 14).
- [26] John C. Snyder et al. "Finding Density Functionals with Machine Learning." In: *Phys. Rev. Lett.* 108 (25 June 2012), p. 253002. DOI: 10.1103/PhysRevLett.108.253002. URL: <https://link.aps.org/doi/10.1103/PhysRevLett.108.253002> (cit. on p. vi).
- [27] John C. Snyder et al. "Orbital-free bond breaking via machine learning." In: *The Journal of Chemical Physics* 139.22 (Dec. 2013), p. 224104. ISSN: 1089-7690. DOI: 10.1063/1.4834075. URL: <http://dx.doi.org/10.1063/1.4834075> (cit. on p. 4).
- [28] Graz University of Technology. *IT Services of Graz University of Technology*. 2021. URL: <https://www.tugraz.at/en/tu-graz/organisational-structure/service-departments-and-staff-units/it-services/> (visited on 09/11/2021) (cit. on p. 14).
- [29] TensorFlow. *GPU support*. 2021. URL: <https://www.tensorflow.org/install/gpu> (visited on 09/11/2021) (cit. on p. 13).
- [30] Yatendra Varshni. "von Weizsäcker, Carl Friedrich." In: Jan. 2014, pp. 2265–2267. DOI: 10.1007/978-1-4419-9917-7\_1455 (cit. on p. 4).
- [31] Sidney Yip Wanda Andreoni. *Handbook of Materials Modeling*. Springer International Publishing. ISBN: 978-3-319-44676-9 (cit. on pp. 3, 4).
- [32] Yan Alexander Wang, Niranjana Govind, and Emily A. Carter. "Orbital-free kinetic-energy density functionals with a density-dependent kernel." In: *Phys. Rev. B* 60 (24 Dec. 1999), pp. 16350–16358. DOI: 10.1103/PhysRevB.60.16350. URL: <https://link.aps.org/doi/10.1103/PhysRevB.60.16350> (cit. on p. 5).
- [33] Yan Alexander Wang, Niranjana Govind, and Emily A. Carter. "Orbital-free kinetic-energy functionals for the nearly free electron gas." In: *Phys. Rev. B* 58 (20 Nov. 1998), pp. 13465–13471. DOI: 10.1103/PhysRevB.58.13465. URL: <https://link.aps.org/doi/10.1103/PhysRevB.58.13465> (cit. on p. 5).

- [34] Junchao Xia and Emily A. Carter. "Density-decomposed orbital-free density functional theory for covalently bonded molecules and materials." In: *Phys. Rev. B* 86 (23 Dec. 2012), p. 235109. DOI: 10.1103/PhysRevB.86.235109. URL: <https://link.aps.org/doi/10.1103/PhysRevB.86.235109> (cit. on pp. 3–5).
- [35] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method." In: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701. URL: <http://arxiv.org/abs/1212.5701> (cit. on p. 11).
- [36] Jie Zhou et al. "Graph neural networks: A review of methods and applications." In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: <https://doi.org/10.1016/j.aiopen.2021.01.001>. URL: <https://www.sciencedirect.com/science/article/pii/S2666651021000012> (cit. on p. 47).