HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

Optimizing a Neural Network Analysis Approach for Simulated Neutron Measurement Data of Polycrystalline Samples

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von:	Tim Schuster
geboren am:	09.07.2000
geboren in:	Elsterwerda
Gutachter/innen:	Prof. Dr. Lars Grunske Dr. Jens-Uwe Hoffmann

eingereicht am: ………………… verteidigt am: …………………

Abstract Physicists studying Quantum Materials are searching for useful novel quantum magnets as well as materials with strong correlated electrons. However the current methods to determine the describing parameters of these materials are lacking. They require a lot of manual work from the scientist and they are susceptible to human error, which is why more efficient approaches are needed. Machine Learning approaches offer a promising solution for this problem. They can reduce the amount of manual work needed significantly while also eliminating human error. In this bachelor thesis we will optimize a Neural Network approach that predicts the describing parameters of a magnetic material from simulated data.

Contents

1 Introduction **6**

2 Fundamentals **9**
- 2.1 Neural Networks . . . 9
- 2.2 Convolutional Neural Networks . . . 12

3 Related Work **15**

4 Data **16**
- 4.1 Experimental data . . . 16
- 4.2 Creating a dataset . . . 18
  - 4.2.1 Create enough data . . . 18
- 4.3 Mimic Experimental Data . . . 19
  - 4.3.1 Noise . . . 20
  - 4.3.2 Constant background . . . 21
  - 4.3.3 Scaling . . . 21
  - 4.3.4 Peaks . . . 21
  - 4.3.5 Masks . . . 21
  - 4.3.6 Final image . . . 22

5 Experiments **24**
- 5.1 Experimental Setup . . . 24
  - 5.1.1 Hyperparameter Optimization . . . 24
  - 5.1.2 Evaluating model Performance . . . 24
- 5.2 Baseline Approach . . . 25
  - 5.2.1 Pre-Processing . . . 25
  - 5.2.2 Neural Network Architecture . . . 25
  - 5.2.3 Training . . . 28
  - 5.2.4 Evaluation . . . 28
- 5.3 Improving Pre-Processing . . . 30
  - 5.3.1 Training . . . 31
  - 5.3.2 Evaluation . . . 31
- 5.4 Applying Batch Normalization and Dropout . . . 33
  - 5.4.1 Training . . . 34
  - 5.4.2 Evaluation . . . 37
- 5.5 Improving the Baseline Neural Network further . . . 39
  - 5.5.1 Training . . . 40
  - 5.5.2 Evaluation . . . 41
- 5.6 Convolutional Neural Networks . . . 41
  - 5.6.1 Training . . . 41
  - 5.6.2 Evaluation . . . 41

5.7 Optimizing the CNN Approach 46
5.7.1 Training 46
5.7.2 Evaluation 46

6 Results **48**

7 Conclusion **51**

1 Introduction

In order to extract the describing properties of a magnetic material from simulated or experimental data, we first need to understand how we can describe these properties in a theoretical manner. The Spin, a type of angular momentum, is a physical property that elementary particles such as electrons possess. A Spin can have different orientations and it can interact with with adjacent Spins. These two properties determine the magnetic properties of a material. A Model to describe these two properties is called the Hamiltonian. The goal of scientists is to obtain the parameters of the Hamiltonian, however when performing experiments such as Inelastic Neutron Scattering experiments on magnetic compounds they only obtain a magnetic excitation spectrum as shown in Figure 1(a).

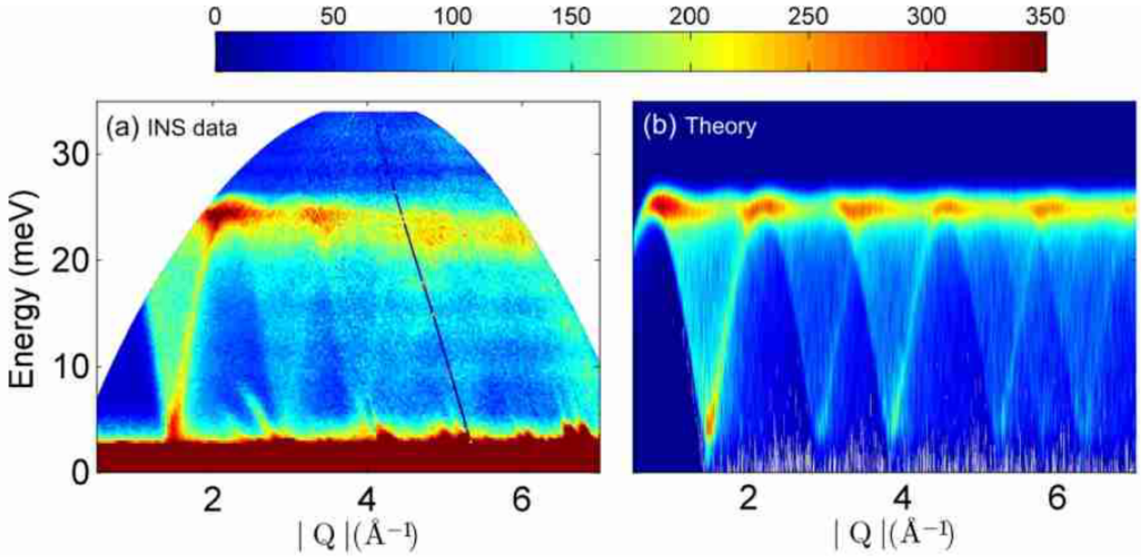


Figure 1: (a) INS data measured on a powder sample at T=5K using the Merlin spectrometer with an incident energy of 35 meV. (b) Spin-wave simulations of the powder excitation spectrum of $BaNi_2V_2O_8$ performed for the Hamiltonian Equation 5 with the parameters J_n = 12.3 meV, J_{nn} = 1.25 meV, J_{nnn} = 0.2 meV, J_{out} = -0.00045 meV, D_{EP} = 0.0695 meV, D_{EA} = -0.0009 meV. [19]

In order to obtain the parameters, scientists compare the experimental magnetic excitation spectrum with theoretical spectra that can be obtained using Spin-Wave-Theory. A common package used is the Matlab Library SpinW [41], which can be used to perform spin-wave simulations on a given Hamiltonian. Before theoretical spectra can be generated, the magnetic ground state of the spins has to be determined. "There is an extended literature on the determination of the classical magnetic ground state either using the Luttinger-Tisza method[27, 17] or Monte-Carlo simulations[23]"[41]. For $BaNi_2V_2O_8$, which will be the magnetic compound we focus on, the magnetic ground state is already solved.

Theoretical simulations of different Hamiltonians are then compared against the experimental data until a good fit is found. This method is time consuming and susceptible to human error, however other methods such as the χ^2 approach, which minimizes the squared distance between the experimental and theoretical spectra, are not reliable due to the artifacts found in experimental data. In addition "χ^2 is both noisy and effectively flat around its minimum, such that many distinct model Hamiltonians could achieve similarly small values of the χ^2 error measure"[35]. As the main bottleneck for understanding these materials better, is the extraction of the Hamiltonian in a reliable way [10], scientists are currently searching for more effective approaches, which are easier to use and require less manual work. Machine Learning methods are currently emerging as a promising solution and have already been successfully applied to related problems such as using Autoencoders to remove artifacts from experimental data [35]. For this reason the Helmholtz-Zentrum Berlin has created a Proof-of-Concept Neural Network approach to show that the direct prediction of the Hamiltonian from simulated Data using Neural Networks is potentially possible. The Proof-of-Concept approach is unoptimized and is currently not able to predict all the parameters of the Hamiltonian to an acceptable degree. This thesis will be written in cooperation with the Helmholtz-Zentrum Berlin and its goal is to improve the Proof-of-Concept approach as much as possible and answer the following research question: "Can Neural Networks be used to perform a reverse transformation of Spin-Wave-Theory using distorted simulated data." In order to achieve this goal we will optimize the Proof-of-Concept approach by investigating each step of the Machine Learning Pipeline e.g. Pre-Processing, the Neural Network Structure and Hyperparameters.

The Structure of this thesis is as follows: We will first provide the fundamentals necessary to understand the techniques used in this thesis in section section 2.

In section 3 we will provide an overview of the current applications of Machine Learning in related problems. As publications dealing with the predictions of the Hamiltonian using simulated data are sparse, we will put an emphasis on the few publications that deal with this subject, however we will also look at related problems.

Then in section 4 we will explain how the data set used for model training is created and how the artifacts are added to the simulated data. As the starting point of the data set is based on the findings of physicists, we will first summarize their findings in this section.

section 5 includes all experiments conducted in order to improve the baseline approach. This includes the evaluation of different Pre-Processing approaches as well as different Neural Network Architectures and finding the optimal set of Hyperparameters. All experiments will be compared against the previous experiment in order to show the direct improvement of each experiment.

In section 6 we will summarize our results and compare the findings of our experiments against the baseline approach to show the improvements we have achieved.

In section 7 we will give an overview of what was accomplished in this thesis and we will provide an answer to the research question and look at possible problems regarding this thesis.

2 Fundamentals

Artificial Inteligence (AI), Machine Learning (ML) and Deep Learning (DL) are terms that are often used synonymously. However in reality they form a hierarchy as shown in Figure 2.

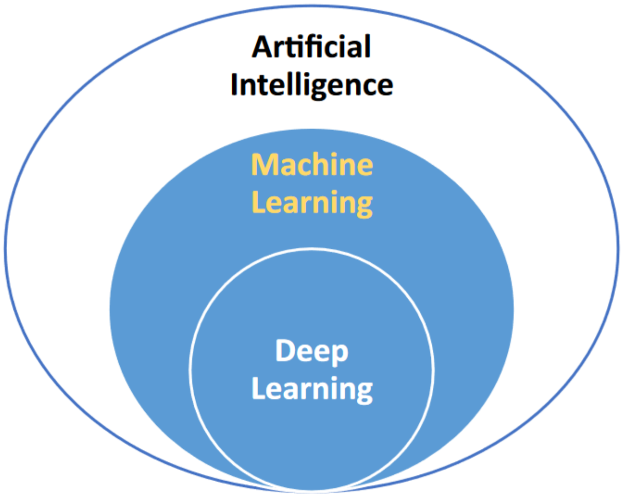


Figure 2: Hierarchy of Artificial Intelligence, Machine Learning and Deep Learning [9]

AI deals with using computers to solve tasks that normally require human intelligence. Machine Learning, a subset of AI, deals with "methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty"[28]. Machine Learning methods allow us to solve tasks that are normally either very hard to solve or cannot be solved at all using conventional computer programs. One class of ML algorithms is called Neural Networks (NNs). Deep Learning, a subset of ML, specializes on deep NNs that have three or more layers.

2.1 Neural Networks

The most basic unit of NNs is called a Neuron. It gets some input vector x comprised of numerical values and outputs a value according to the following equation:

$$y = f(x \cdot w + b) \tag{1}$$

where:

y = Output
x = Vector of inputs
w = Vector of weights
b = Bias term
f = Activation function

These Neurons can be combined together to form a Neural Network as shown in Figure 3. Multiple Neurons can form a layer and these layers can be stacked forming the hidden layers section of the Neural Network. If signals can only flow in one direction as indicated by the arrows in Figure 3 we call the NN a feedforward NN. In this case each layers input is the output of the previous layer.

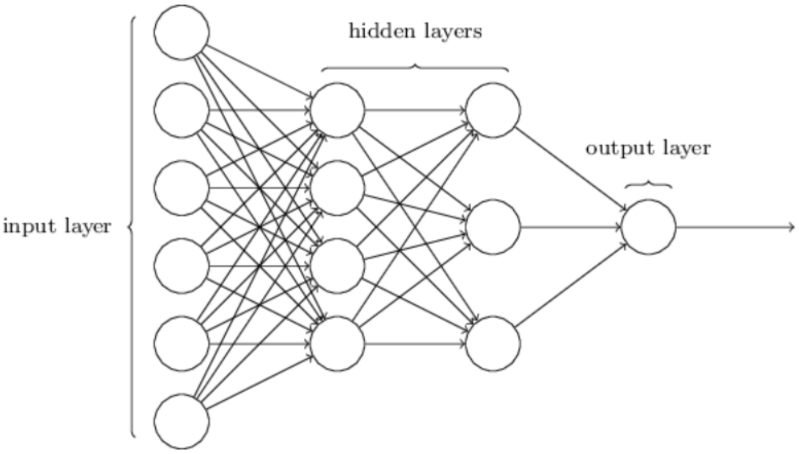


Figure 3: Example of a Neural Network with four layers [29]

The goal of a feedforward network is to approximate some function f^*. For example, for a regressor, $y = f^*(x)$ maps an input x to a numerical value y . A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.. If the Neural Network has multiple layers as the one shown in Figure 3, f(x) is composed of multiple functions that are chained together e.g. $f(x) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(x))))$. We call $f^{(1)}$ the first layer, $f^{(2)}$ the second layer and so on. The number of layers that a network has is called its depth.[12].

In order for the Neural Network to learn we need a measure that calculates the deviations between the predicted value of the NN and the true value. This function is called the loss or cost function. For a regression task the Mean Squared Error is a common loss function, which is defined as:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (f(x_i; \theta) - y_i)^2 \tag{2}$$

where:

N = Number of samples
y_i = true value
$f(x_i;\theta)$ = predicted value of the NN based on the input vector x_i and parameters θ

As the name suggests the MSE calculates the mean of the squared deviations between the NN prediction and the true value. Finding the set of parameters θ that minimize the MSE is the goal of the learning process. Essentially NN training can be defined as a minimization problem:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} (f(x_i;\theta) - y_i)^2 \tag{3}$$

However finding θ^* is difficult, because in most cases the surface of the loss function in regards to θ is non-convex as shown in Figure 4. The loss function can have many local minima, which makes finding the global minimum hard, however according to Choromanska et al.[7] in deeper NNs many local minima are of high quality and the search for the global minimum is irrelevant in practice, because the global minimum often leads to overfitting.

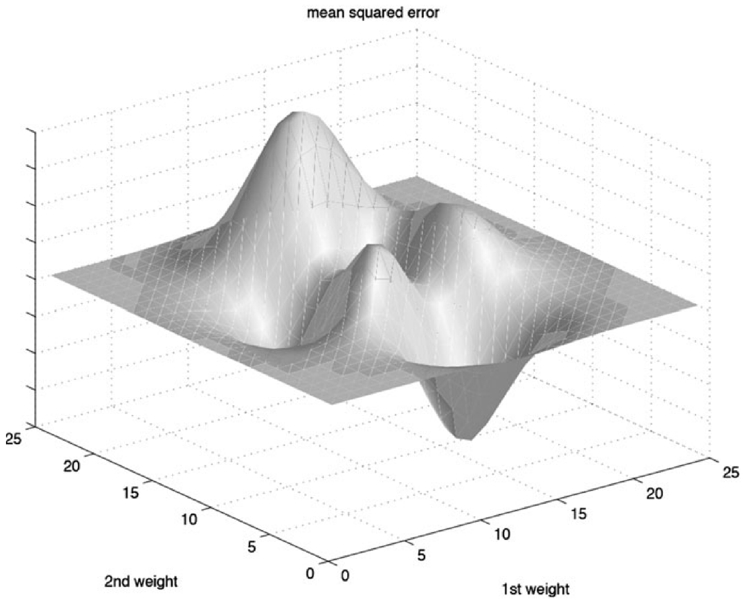
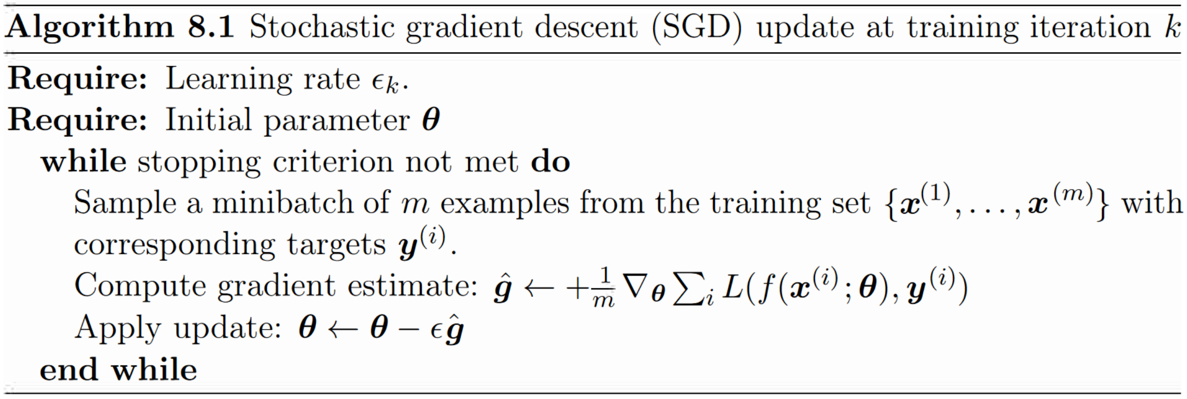


Figure 4: For a given example, each set of weights implies a certain mean squared error. The surface of the loss function is non-convex. [22]

In practice NNs are often trained by randomly initializing θ and then optimizing these parameters by using the Stochastic Gradient Descent (SGD) algorithm. The algorithm is shown in Figure 5. SGD first samples a minibatch (a subset of the training set) and computes the loss for each sample of the minibatch. The loss between the prediction of the NN and the true value for sample i is denoted by L(f($x^{(i)};\theta$),$y^{(i)}$). Then the gradient estimate is calculated using Back-propagation. The gradient contains the partial derivatives for each parameter in θ. θ is then updated by moving θ in the inverse direction of the gradient estimate multiplied with a step size ϵ which is also called the learning rate. The learning rate is necessary, because using minibatches instead of the whole training set introduces noise into the gradient, meaning that even when a local minimum is reached the gradient estimate does not have to be zero.



```
Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration $k$
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while
```

Figure 5: SGD algorithm written in pseudo-code. [12]

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), are a specialized kind of NN for processing data that has a known, grid-like topology, such as images. As the name suggests they make use of a mathematical operation called convolution. [12]

For a two-dimensional input I and kernel K, convolution is defined as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n) \tag{4}$$

As the optimal kernels differ depending on the task the CNN has to solve, they have to be learned during the training procedure. An illustration of how convolution works is shown in Figure 6. As shown in the figure the convolution operation reduces the size of layer q+1 in comparison to layer q. This behaviour is not desirable in general, because it tends to lose some information along the borders of the image [3]. Using Padding during the convolution operation can resolve this issue. Padding adds pixels to the borders of the input to maintain the size of the input. The pixels added due to padding are set to zero to ensure that they do not contribute to the dot product calculated during convolution. An illustration of padding is shown in Figure 7.

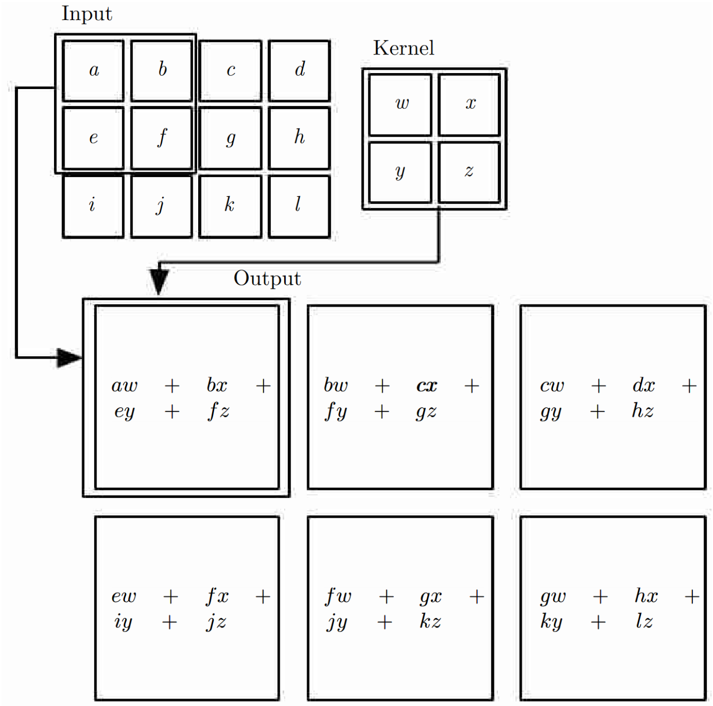
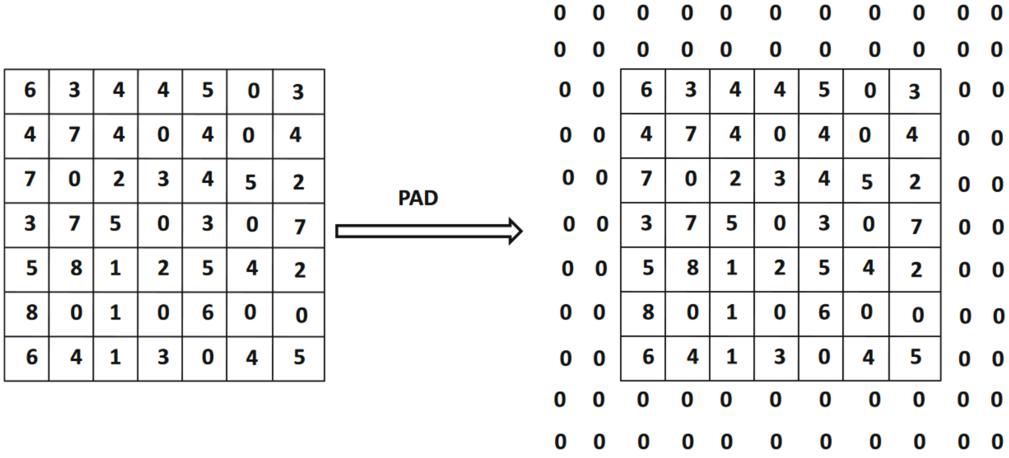


Figure 6: An example of 2-D convolution. We restrict the output to only positions where the kernel lies entirely within the image. [12]

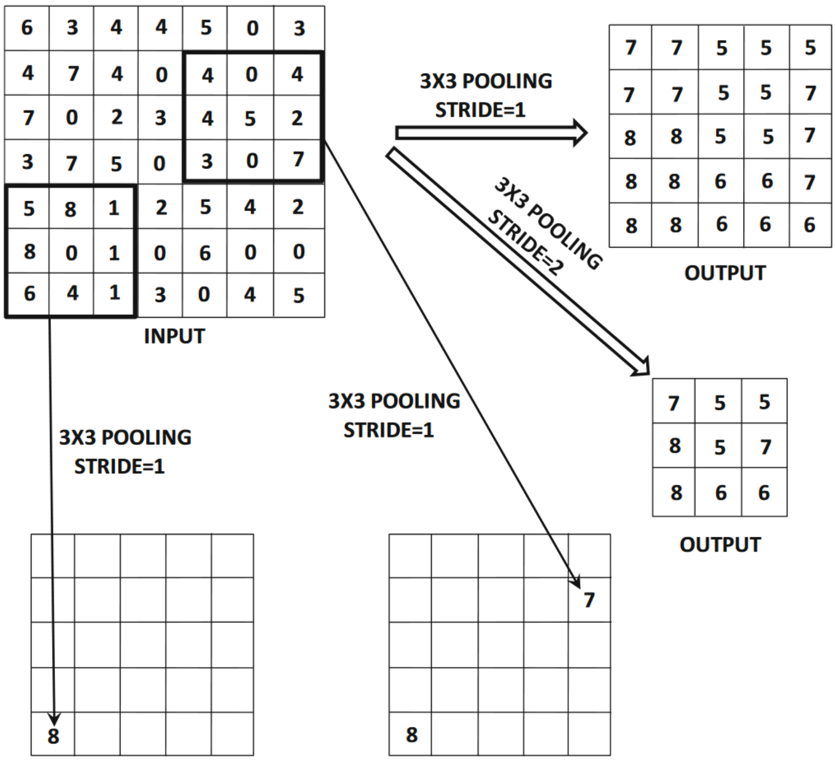


6	3	4	4	5	0	3
4	7	4	0	4	0	4
7	0	2	3	4	5	2
3	7	5	0	3	0	7
5	8	1	2	5	4	2
8	0	1	0	6	0	0
6	4	1	3	0	4	5

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	6	3	4	4	5	0	3	0	0
0	0	4	7	4	0	4	0	4	0	0
0	0	7	0	2	3	4	5	2	0	0
0	0	3	7	5	0	3	0	7	0	0
0	0	5	8	1	2	5	4	2	0	0
0	0	8	0	1	0	6	0	0	0	0
0	0	6	4	1	3	0	4	5	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure 7: An example of padding [3]

The layer that performs the convolution operation is called the convolutional layer. Another layer often found in CNNs is called the max-pooling layer, which performs the max-pooling operation. The max-pooling operation devides the input into grids of fixed size and simply returns the max value of each grid. An illustration of max-pooling is shown in Figure 8



INPUT

6	3	4	4	5	0	3
4	7	4	0	4	0	4
7	0	2	3	4	5	2
3	7	5	0	3	0	7
5	8	1	2	5	4	2
8	0	1	0	6	0	0
6	4	1	3	0	4	5

OUTPUT

7	7	5	5	5
7	7	5	5	7
8	8	5	5	7
8	8	6	6	7
8	8	6	6	6

OUTPUT

7	5	5
8	5	7
8	6	6

Figure 8: An example of a max-pooling of one activation map of size 7×7 with strides of 1 and 2. A stride of 1 creates a 5×5 activation map with heavily repeating elements because of maximization in overlapping regions. A stride of 2 creates a 3×3 activation map with less overlap. [3]

3 Related Work

At the time of writing this thesis the use of ML to predict the Hamiltonian of magnetic compounds with experimental or simulated data is rare however there are two publications that have dealt with this problem:

1. Hey et al.[13] have investigated the use of CNNs for predicting the first- and second-nearest neighbour coupling strengths J_n and J_{nn} in the near-ideal two-dimensional, spin 5/2 Heisenberg antiferromagnet Rb_2MnF_4. The CNN was trained on simulated data and achieved a mean absolute error of 0.0055 meV for J_n and a mean absolute error of 0.0036 meV for J_{nn} on unseen simulated data. The dataset however contains no artifacts except for masks that were added to simulate the parts on experimental data that are not recorded due to the detector geometry. Our dataset contains more artifacts like random noise or constant background which will make it more difficult for the NN to extract the necessary features from the images to predict the parameters of the Hamiltonian.

2. Samarakoon et al.[35] combine the use of ML and a more robust χ^2 method to extract optimal model Hamiltonians from the spin ice $Dy_2Ti_2O_7$. First an Autoencoder was trained on simulated data to extract the relevant characteristics of the input, while discarding irrelevant information such as noise or experimental artifacts. The output of the Autoencoder is then used as an input for a more robust version of the χ^2 measure which extracts optimal model Hamiltonians using a minimal squares approach. With this approach Smarakoon et al. are able to perform robust inference of the optimal model Hamiltonian. The use of ML algorithms is only used as an intermediate step to extract model Hamiltonians. We want to optimize a ML approach that performs a direct regression of the Hamiltonian from simulated data.

When generalizing the problem to the prediction of properties of materials using neutron scattering data Doucet et al.[10] provide an overview of related publications such as:

1. Liu et al.[26] trained a CNN to predict the most likely space groups of a structure given a simulated or measured atomic pair distribution.

2. Garcia-Cardona et al.[11] have trained a CNN on simulated powder diffraction data to predict crystallographic symmetry classes. A random forest regressor was then trained for each crystallographic symmetry class to predict the unit cell parameters based on the symmetry class.

4 Data

Data of Inelastic Neutron Scattering (INS) experiments is scarcely available in physics, due to the time and cost involved to create it. Machine learning algorithms on the other hand need large amounts of data [34]. This means that in order to train machine learning models we cannot solely rely on experimental data, but instead we need to artificially create a dataset. In this section we will describe how the dataset used for model training was created.

4.1 Experimental data

Since the dataset is based on the findings of Klyushina et al., 2017 [19], we will summarize their findings in this subsection. First a powder sample of the spin-1 honeycomb antiferromagnet $BaNi_2V_2O_8$ was grown at the "Core Lab for Quantum Materials at the Helmholtz Zentrum Berlin für Materialien und Energie (HZB)" in Germany. $BaNi_2V_2O_8$ is called a honeycomb antiferromagnet, because its structure looks like a honeycomb. The crystal and magnetic structure of $BaNi_2V_2O_8$ is depicted in Figure 9.

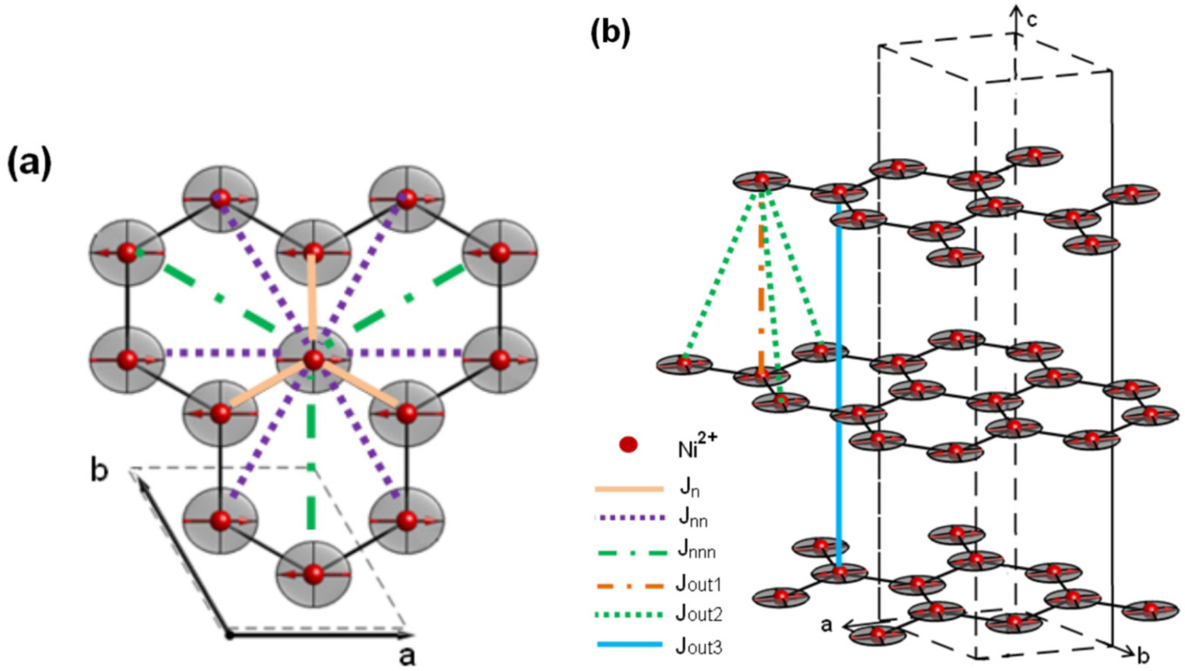


Figure 9: Crystal and magnetic structure (one twin) of $BaNi_2V_2O_8$ within (a) a honeycomb plane and (b) a single unit cell. Only the magnetic Ni^{2+} ions are shown for clarity. The magnetic structure is based on the results of powder neutron diffraction measurements [33]. The red arrows represent spin directions and the gray ellipsoids illustrate the easy-plane single-ion anisotropy.[19]

Then INS measurements were performed on the powder sample to explore the magnetic excitation spectrum of $BaNi_2V_2O_8$. The INS data was collected using the high count rate thermal time-of-flight spectrometer MERLIN [5] at ISIS, Rutherford Appleton Laboratory, U.K.. The results are plotted in Figure 1(a). The magnetic excitation spectrum was then analysed using the Matlab Library SpinW [41], which can be used to perform spin-wave simulations on a given Hamiltonian. The Hamiltonian assumed by Klyushina et al. is given by Equation 5. J_n, J_{nn} and J_{nnn} are the first-, second- and third-nearest-neighbor isotropic magnetic exchange interactions within the honeycomb-plane. J_n, J_{nn} and J_{nnn} describe how strong a spin interacts with its first- second- and third nearest neighbors. Isotropic means that the magnetic exchange interactions are independent of the spin direction. J_{out} describes the magnetic exchange interactions between the honeycomb-planes as shown in Figure 9. D_{EP} and D_{EA} describe the easy-plane and easy-axis anisotropies of the Ni^{2+} magnetic ions. The easy-axis anisotropy describes the preferred spin direction along the ab-plane and the easy-axis anisotropy describes the preferred spin direction along the c-plane. The three planes are depicted in Figure 9. Spin-wave-simulations were then performed iteratively, manually adjusting the six parameters J_n, J_{nn}, J_{nnn}, J_{out}, D_{EP} and D_{EA} until a set of parameters was found, that has a magnetic excitation spectrum similar to the captured INS data. Klyushina et al. have found the set of parameters in Table 1 to be in good agreement between the INS data and the spin-wave-simulations:

Parameter	Value in meV
J_n	12.3
J_{nn}	1.25
J_{nnn}	0.2
J_{out}	-0.00045
D_{EP}	0.0695
D_{EA}	-0.0009

Table 1: Set of parameters that is in good agreement with the magnetic excitation spectrum of $BaNi_2V_2O_8$

A comparison between the spin-wave-simulations of the magnetic excitation spectrum of this parameter set and the INS Data is shown in Figure 1. Since this set of parameters seems to be in good agreement with the INS data, it will be used as a starting point for the artificially created dataset. Further details regarding the creation of the powder sample and the determination of the parameter values, are found in the paper published by Klyushina et al.[19].

$$\begin{aligned} H = &\sum_{i>j} J_n * S_i * S_j + \sum_{i>j} J_{nn} * S_i * S_j + \sum_{i>j} J_{nnn} * S_i * S_j \\ &+ \sum_{i>j} J_{out} * S_i * S_j + \sum_{i>j} D_{EP} * S_i^{c^2} + \sum_{i>j} D_{EA} * S_i^{a^2} \end{aligned} \tag{5}$$

4.2 Creating a dataset

In this subsection we will explain how the dataset was created. When using the SpinW library to perform spin-wave-simulations on the magnetic excitation spectrum, an image like the one in Figure 1(b) is created. When comparing this image to INS data like the one in Figure 1(a) we can see that (a) has a lot more disturbing artifacts like the constant noise throughout the whole image or the white space at the sides. All these artifacts don't describe any magnetic property of the measured powder sample, but are simply measurement errors that originate from the experimental setup of INS experiments and thus cannot be prevented. The red beam at the bottom emanates from the sample itself, however it does not describe any magnetic properties. For this reason we will also consider it as an artifact. Spin-wave-simulations of the magnetic excitation spectrum on the other hand don't possess these disturbing artifacts, because they are solely created with spin-wave-theory. Since the goal for physicists is to later use neural networks to predict the Hamiltonian on real experimental data, we cannot train our neural networks on a dataset of "clean" spin-wave-simulations, but we need to add artifacts to the theory data in order to mimic real experimental data.

4.2.1 Create enough data

The starting point for this dataset was the set of parameters in Table 1. To create more samples for the dataset, 111397 different Hamiltonians were generated, where each of the six parameters was randomly sampled from the corresponding distribution in Table 2. Each Hamiltonian was generated as follows:

1. Randomly sample a value for each of the six parameters from the distributions in Table 2.

2. Check the set of parameters for the constraints imposed by spin-wave-theory (not every set of parameters is permitted by the model).

3. If the set of parameters is not allowed, discard the set and start again from step 1. If the set of parameters is allowed, the magnetic excitation spectrum is calculated via the SpinW library. The resulting image is saved as a (100x100) pixel image along with the set of parameters in a NeXus file (NeXus files [20] are a filetype commonly used for neutron and x-ray data).

This procedure was repeated until 111397 nexus files were generated. Each file has a size of 179 KB. The complete dataset has a size of 18.9 GB. A comparison of the image loaded from a nexus file and a spin-wave-simulation for the same parameters is shown in Figure 10 and the distribution of the six parameters used in the dataset is shown in Figure 11.

Parameter	Distribution
J_n	$\mathcal{N}(12.3, 1.2)$
J_{nn}	$\mathcal{N}(1.25, 0.3)$
J_{nnn}	$\mathcal{N}(0.2, 0.3)$
J_{out}	$\mathcal{N}(-0.00045, 0.002)$
D_{EP}	$\mathcal{N}(0.0695, 0.02)$
D_{EA}	$\mathcal{N}(-0.0009, 0.0004)$

Table 2: Distribution of each Parameter used for the generation of the Hamiltonians

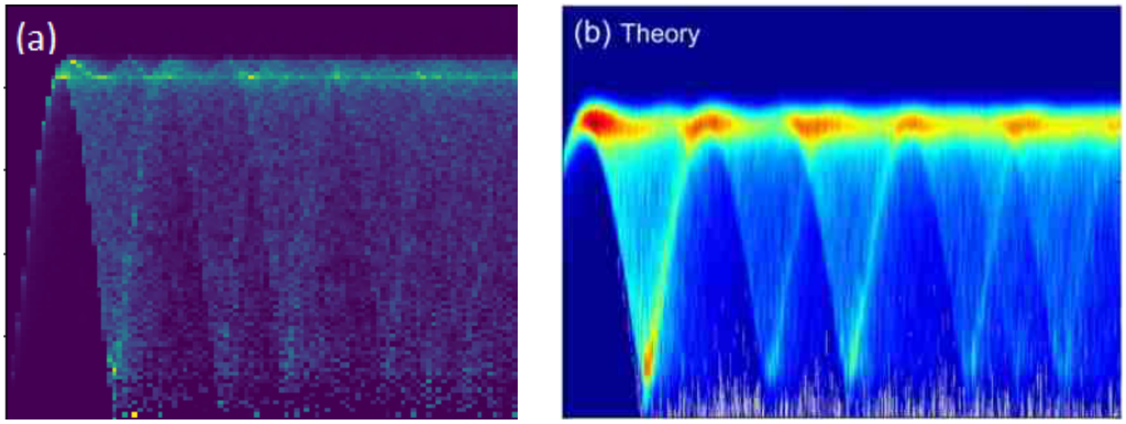


Figure 10: Comparison of the magnetic excitation spectrum image for the parameter set from Table 1 (a) loaded from a NeXus file. (b) spin-wave-simulation by SpinW (image is from [19]).

4.3 Mimic Experimental Data

The dataset now contains 111397 NeXus files with a (100x100) pixel image of the magnetic excitation spectrum and the corresponding interaction parameters in each of the files. Since the images of the magnetic excitation spectrum where simulated with spin-wave-theory, they are missing the typical artifacts observed in real experimental data that originate from the experimental setup of INS experiments. We will now explain how these artifacts where artificially added to each image. The following five artifact-types are typical for INS measurements and where applied to the dataset in the same order:

- Noise
- Constant background
- Scaling
- Peaks
- Masks

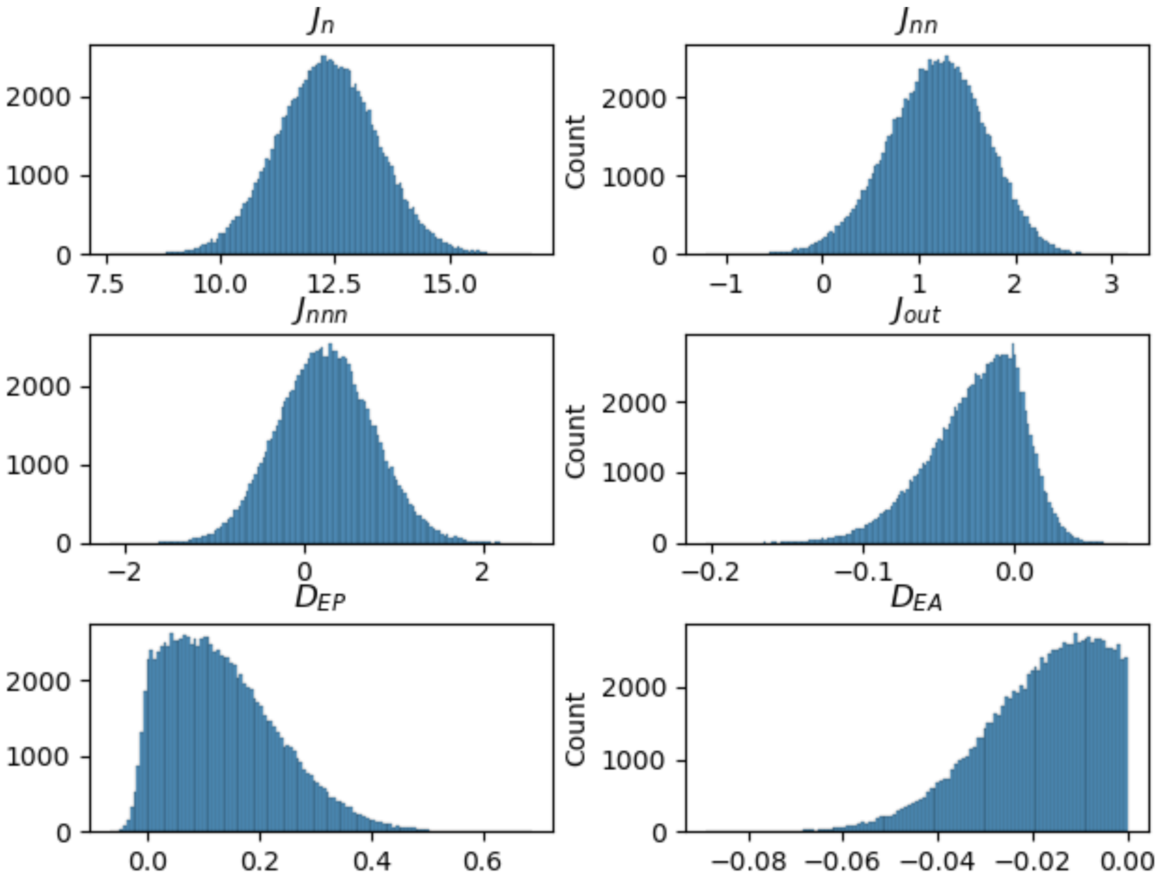


Figure 11: Distribution of the six parameter labels from the dataset.

Each image can be seen as a Matrix $\mathbf{D} = (d_{i,j}) \in \mathbb{R}^{100x100}$. We will use this property to demonstrate how all artifacts were added to the images. For the parameter labels.

4.3.1 Noise

Let $\mathbf{D} = (d_{\mathrm{i,j}})$ be the matrix of an image and $\mathbf{M} = (m_{i,j}) \in \mathbb{R}^{100x100}$ be a Matrix representing the noise in an image, whose elements are obtained by randomly sampling values from the random variable $X = 0.025 * \mathcal{N}(0, 0.4)$. Adding Noise to the image is achieved by adding $\mathbf{M}$ to $\mathbf{D}$, resulting in the matrix:

$$\mathbf{M} + \mathbf{D} = \begin{bmatrix} m_{1,1} + d_{1,1} & \dots & m_{1,100} + d_{1,100} \\ \vdots & \ddots & \vdots \\ m_{100,1} + d_{100,1} & \dots & m_{100,100} + d_{100,100} \end{bmatrix}$$

A different noise Matrix was sampled for each image.

4.3.2 Constant background

Let $\mathbf{D} = (d_{\mathrm{i,j}})$ be the matrix of an image. In order to add a constant background to this image we simply add a constant value c to each element of $\mathbf{D}$ which results in the following matrix:

$$c + \mathbf{D} = \begin{bmatrix} c + d_{1,1} & \dots & c + d_{1,100} \\ \vdots & \ddots & \vdots \\ c + d_{100,1} & \dots & c + d_{100,100} \end{bmatrix}$$

For the dataset, the value c is randomly sampled from the continuous uniform distribution $U(0, 0.025)$ for each image resulting in a different constant background for each image.

4.3.3 Scaling

Let $\mathbf{D} = (d_{\mathrm{i,j}})$ be the matrix of an image. In order to add scaling to this image we multiply $\mathbf{D}$ with a scalar α resulting in the matrix:

$$\alpha \cdot \mathbf{D} = \begin{bmatrix} \alpha \cdot d_{1,1} & \dots & \alpha \cdot d_{1,100} \\ \vdots & \ddots & \vdots \\ \alpha \cdot d_{100,1} & \dots & \alpha \cdot d_{100,100} \end{bmatrix}$$

For the dataset α is randomly sampled from the random variable $X = 0.8 \cdot U(0, 1) + 0.2$ for each image.

4.3.4 Peaks

Let $\mathbf{D} = (d_{\mathrm{i,j}})$ be the matrix of an image and $\mathbf{M} = (m_{i,j}) \in \mathbb{R}^{100x100}$ be a matrix representing a peak. The peak is applied to the image by adding $\mathbf{M}$ to $\mathbf{D}$ resulting in the following matrix:

$$\mathbf{M} + \mathbf{D} = \begin{bmatrix} m_{1,1} + d_{1,1} & \dots & m_{1,1} + d_{1,100} \\ \vdots & \ddots & \vdots \\ m_{1,1} + d_{100,1} & \dots & m_{1,1} + d_{100,100} \end{bmatrix}$$

For our dataset one peak is applied to each image. The peak is a random 2D Gaussian that gets placed at a random position of the image. A comparison of a clean image and one with a peak added is shown in Figure 12.

4.3.5 Masks

Let $\mathbf{D} = (d_{\mathrm{i,j}})$ be the matrix of an image and $\mathbf{M} = (m_{i,j}) \in \mathbb{R}^{100x100}$ be a matrix representing the mask caused by the experimental setup of INS experiments. The

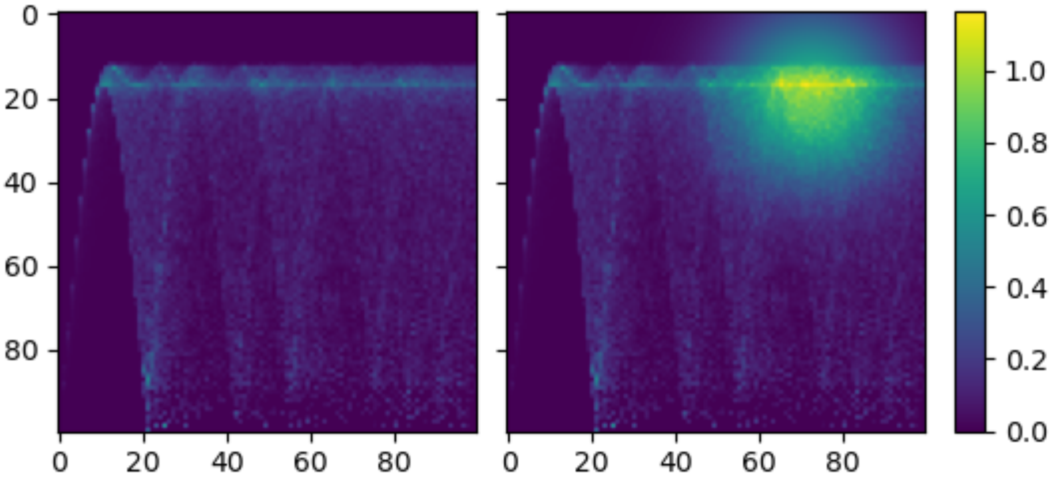


Figure 12: Impact of adding a peak to an image: (Left) Simulation with the parameter set Table 1, (Right) The same simulation with a peak.

mask can be added to the image by calculating the hadamard product resulting in the matrix:

$$\mathbf{M} \circ \mathbf{D} = \begin{bmatrix} m_{1,1} \cdot d_{1,1} & \dots & m_{1,1} \cdot d_{1,100} \\ \vdots & \ddots & \vdots \\ m_{1,1} \cdot d_{100,1} & \dots & m_{1,1} \cdot d_{100,100} \end{bmatrix}$$

Each element of $\mathbf{M}$ is either 0 or 1, resulting in information of $\mathbf{D}$ either being lost or retained depending on the value of $(m_{i,j})$ at each position. 100x100 pixel masks where provided by the Helmholtz-Zentrum Berlin and randomly applied to each image. An example of the impact of a mask in an image is shown in Figure 13

4.3.6 Final image

Let $\mathbf{D} = (d_{\mathrm{i,j}})$ be an image matrix, $\mathbf{N} = (n_{\mathrm{i,j}})$ the random noise, c the constant background, α the scaling, $\mathbf{P} = (p_{\mathrm{i,j}})$ the peak and $\mathbf{M} = (m_{\mathrm{i,j}})$ the mask. The final image is obtained by combining all these artifacts in the following manner:

$$\mathbf{M} \circ (\mathbf{P} + (\alpha \cdot (c + \mathbf{N} + \mathbf{D})))$$

An example image is shown in Figure 14. As we can see the artifacts drastically alter the raw image which will pose big a challenge for predicting the Hamiltonian in section 5 as these artifacts don't possess any information regarding the parameters of the Hamiltonian.

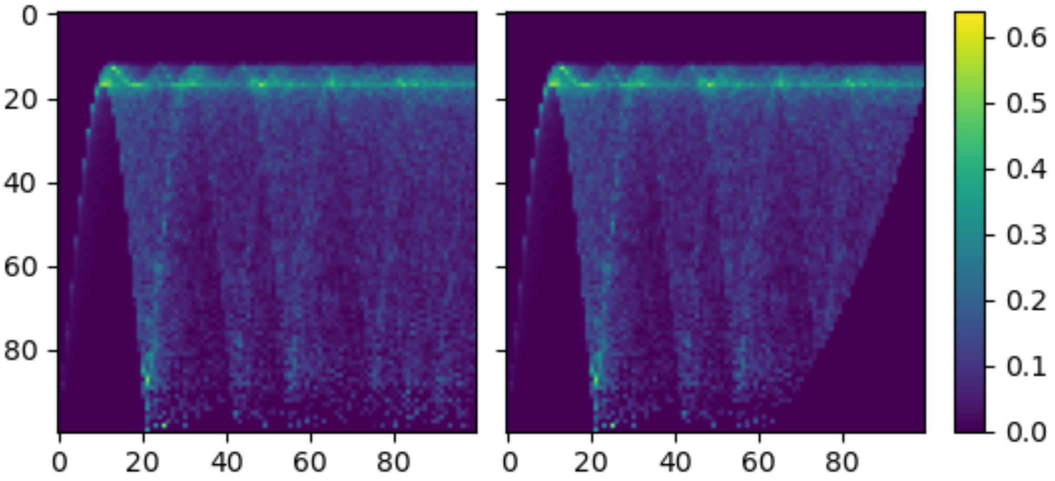


Figure 13: Impact of adding a mask to an image: (Left) Simulation with the parameter set Table 1, (Right) The same simulation with a mask. As shown in the right image the mask deletes the information contained at the bottom right of the image.

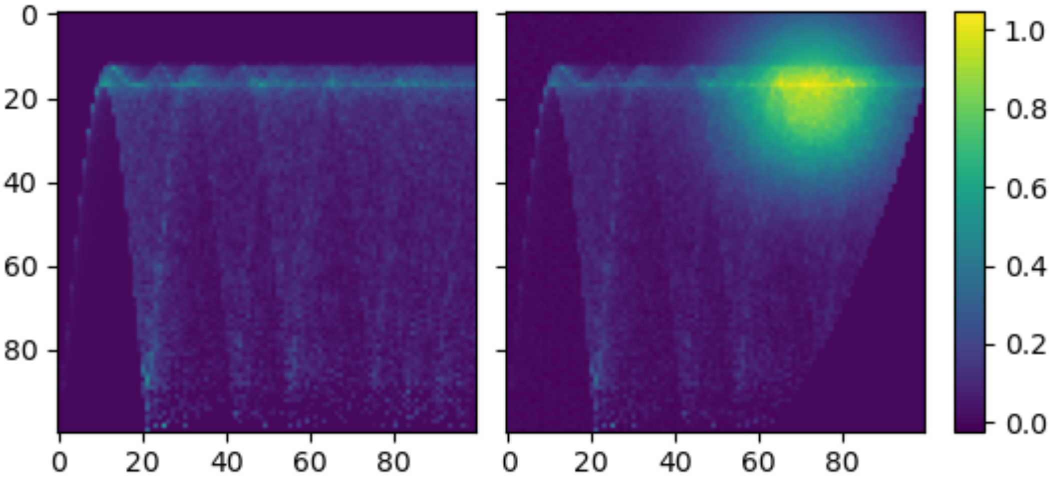


Figure 14: Impact of all five artifact-types applied to an image: (Left) Simulation with the parameter set Table 1, (Right) The same simulation with noise, constant background, scaling, peaks and a mask.

5 Experiments

5.1 Experimental Setup

5.1.1 Hyperparameter Optimization

Hyperparameters (HPs) are very important for the performance of a Neural Network and since they are sensitive to the values of the other hyperparameters they cannot be tuned individually. Finding the optimal set of HPs is the most basic task of automated machine learning (AutoML) [15], a research field which aims at automating machine learning tasks such as Hyperparameter Optimization (HPO) or Neural Architecture Search (NAS). HPO can essentially be seen as a minimization problem trying to obtain λ^* from the following equation:

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}} \mathcal{L}(A_\lambda, D_{train}, D_{validation}) \tag{6}$$

Where $\mathcal{L}$ is the loss of Model A with hyperparameters λ from the hyperparameter space Λ trained with train data D_{train} and validated on the validation data $D_{validation}$. Obtaining the optimal HPs λ^* can be difficult, because we have almost no knowledge about how different sets of parameters will influence the loss function. This forces us to use blackbox optimization methods such as grid search or random search which has been shown to be more effective in most cases than grid search by Bergstra and Bengio [4]. However there are also more novel approaches such as Hyperband [25] or Bayesian Optimization (BO), which has been used successfully for various applications [36]. One advantage of BO approaches is that they construct a probabilistic model and then exploit this model to make decisions about which HPs to try next [38] For this thesis we will use KerasTuner[30], a HPO Framework for Keras[6], and its implementation of BO.

5.1.2 Evaluating model Performance

In Order to compare different Pre-Processing schemes and different model architectures we have to use a standardized test setup which ensures comparability across the different approaches. This setup also has to align with the goals of physicists. One example would be that each of the six parameters is equally important, meaning that a model should aim at optimizing all parameters as much as possible. We propose the following approach: All models will be evaluated on min-max scaled label data using the minimum and maximum of each parameter individually as the scaling borders. This ensures that every parameter is weighted equally when evaluating model performance. Otherwise J_n would have the biggest impact on the MSE while D_{EA} would have the least impact. All models will be trained on a training dataset containing 80000 samples. Hyperparameter Optimization will be done on a validation dataset containing 20000 samples. A test dataset, containing the remaining 111397 samples, will be used to evaluate the final model performance. The test dataset is not used for model training and HPO, ensuring that model performance can be evaluated on unseen data. With this approach we can compare the true generalization performance of a model. For model Evaluation we will calculate the Mean Squared Error (MSE) for each model. As the MSE alone is not able to detect problems such as the NN only predicting the mean of a parameter we will also look at scatter plots for the tested models.

5.2 Baseline Approach

In this subsection we will explain how the complete ML pipeline of the baseline approach was set up starting at Pre-Processing the input data and labels and ending at predicting the six parameters of the Hamiltonian.

5.2.1 Pre-Processing

Both input images and labels where pre-processed prior to model training. The input images where clipped to the interval [0, 1], meaning that every pixel value outside the specified interval was instead replaced with 0 or 1 respectively. After clipping the input images they where flattened from the original (100,100) shape into a one dimensional vector of length 10000, which serves as the input for the neural network. The labels where "pseudo"-standardized by replacing each parameter value x with $(x-\mu)/\sigma$. μ and σ are different for each of the six parameters and are listed in Table 3. After transforming the labels they were also clipped to the interval [0, 1] like the input images.

Parameter	μ	σ
J_n	8.0	8.0
J_{nn}	-1.0	4.0
J_{nnn}	-2.0	4.0
J_{out}	-0.2	0.3
D_{EP}	-0.1	0.7
D_{EA}	-0.07	0.07

Table 3: μ and σ for each parameter used to normalize the labels.

5.2.2 Neural Network Architecture

The architecture of the baseline Neural Network is shown in Figure 15. It is a fully-connected Neural Network consisting of the input layer, 9 hidden layers and the output layer. The number of neurons in the hidden layers halves with each consecutive layer and starts at 4096. Each hidden layer uses the Rectified Linear Units (ReLU)[2] activation function, which is very popular for training NNs, because it is easy to train and it solves the vanishing gradient problem which other activation functions such as sigmoid or tanh face[32]. The ReLU activation function is defined as $f(x) = max(0, x)$ and is depicted in Figure 16.

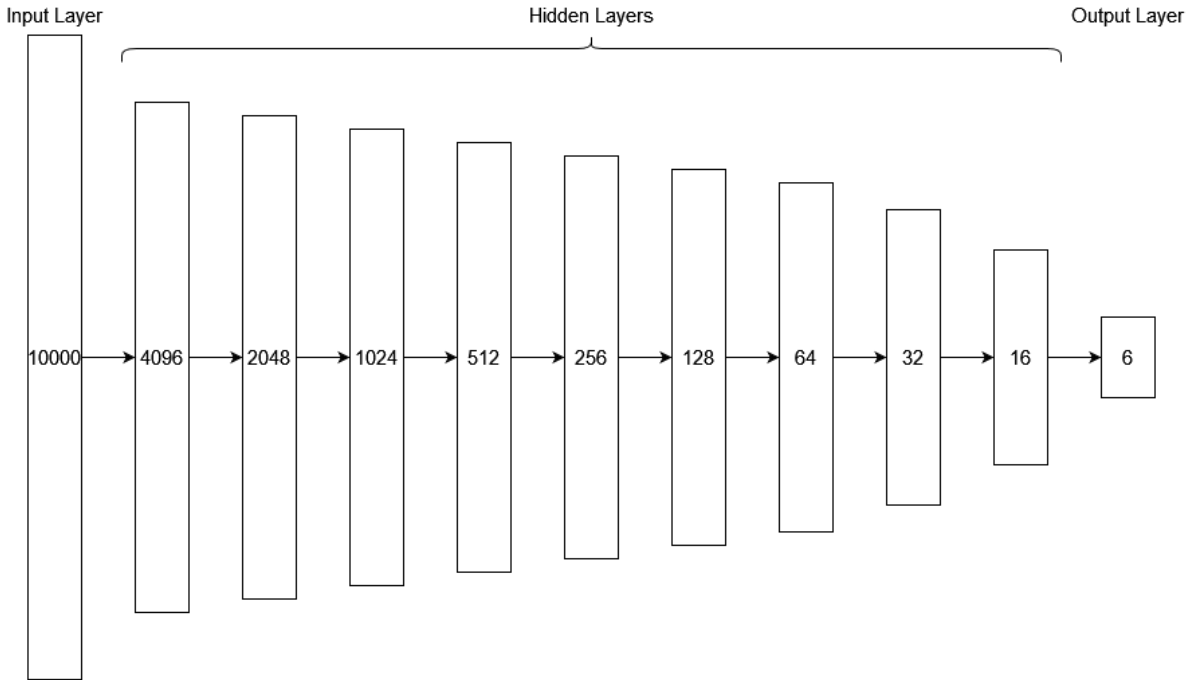


Figure 15: Architecture of the Baseline FC-NN. The input is the flattened (100,100) pixel image. The number of neurons in the hidden layers halves with each consecutive layer and starts at 4096. Each hidden layer uses the ReLU activation function. The output layer has 6 neuron in order to predict the six parameters of the Hamiltonian and uses the sigmoid activation function.

The output layer has six neurons, one for each parameter of the Hamiltonian and uses the sigmoid activation function, which is defined as: $\sigma(x) = \frac{1}{1+e^{-x}}$ and depicted in Figure 17. Since the sigmoid function can only output values in the interval (0,1), the labels have to be scaled to this interval.

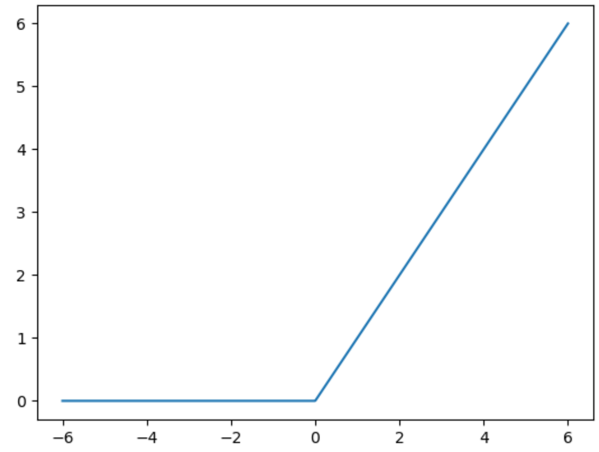


Figure 16: ReLU activation function $f(x) = max(0, x)$.

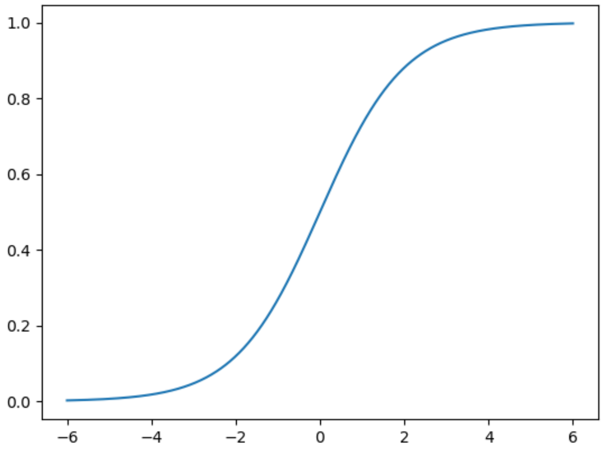


Figure 17: Sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$.

5.2.3 Training

The baseline NN (bNN) was trained with the Pre-Processing approach from subsubsection 5.2.1 for 50 Epochs and a batch size of 64. Binary Crossentropy (BCE) was the used loss function. The BCE loss function is normally used in binary classification tasks, but has also seen success in other areas such as denoising autoencoders[8]. The BCE loss is defined as:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right) \tag{7}$$

where:

N = Number of samples
y_i = true label
p_i = predicted label

The popular Adam optimizer[18] was used as the optimization algorithm, which is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement[18]. The standard parameters for Adam were used for model training, which are as follows:

$\alpha = 0.001$
$\beta_1 = 0.9$
$\beta_2 = 0.999$

α denotes the learning rate, whereas β_1 and β_2 denote the exponential decay rates for the first- and second moment estimates. The BCE and MSE for each epoch during training is depicted in Figure 18

5.2.4 Evaluation

The lowest validation MSE was found at epoch 28 which was then used to calculate the general performance of the baseline approach on the test data. Since the predictions of the baseline approach were scaled differently than the MinMax scaled test data, we rescaled the predictions, by first reverting the Pre-Processing technique used in subsubsection 5.2.1, which yields the true parameter prediction. These predictions were than MinMax scaled using the same minimum and maximum for each parameter as the ones used for MinMax scaling the test data. This ensures that the predictions and the test data are comparable. The MSE on the MinMax scaled test data is shown in Table 4.

The first thing we can observe is that the MSE differs drastically based on the parameter. D_{EA}, which describes the easy-axis anisotropy for example has a ~2.5x higher MSE than the best performing parameter J_{nnn}. Also when looking at the scatter plots in Figure 19 we can observe that the bNN has a "good" prediction of the

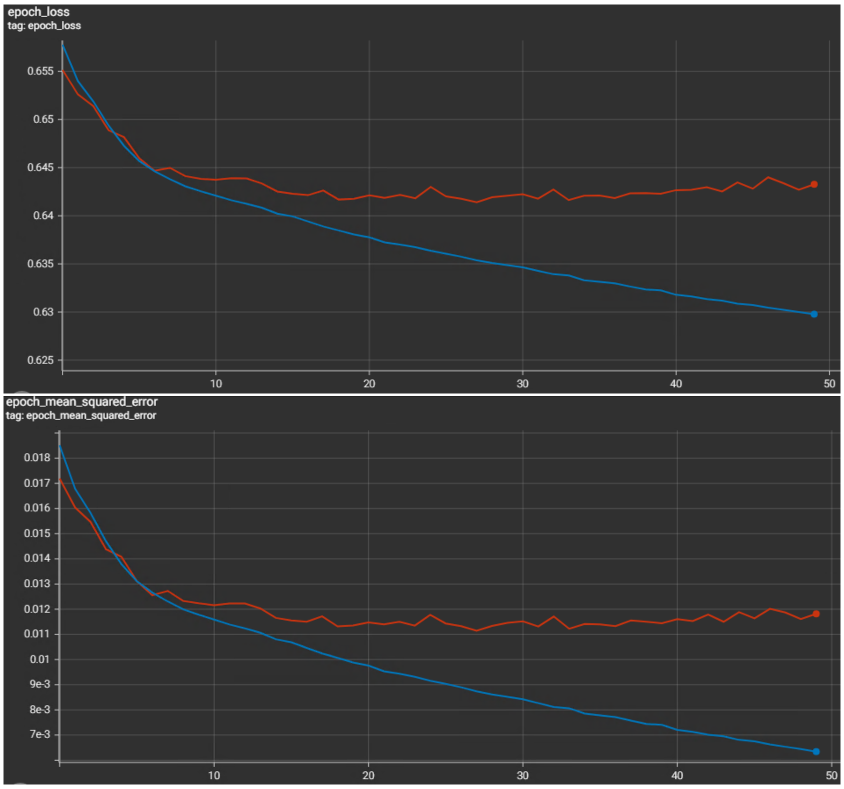


Figure 18: Training of the baseline model: BCE (top graph) and MSE (bottom graph) per epoch on training data (blue line) and validation data (red line).

MSE per Parameter						
J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877

Table 4: MSE of baseline NN on MinMax scaled test data for each parameter and overall

three magnetic exchange interaction parameters J_n, J_{nnn} and J_{nnn}, in contrast to the remaining three parameters. For J_{out} for example the bNN has learned to predict the mean, which means that the NN couldn't learn how to correctly predict this parameter. It is possible that the bNN doesn't have the capability to correctly predict J_{out} or the powder simulations with the added artifacts simply lack the necessary information to correctly predict J_{out}.

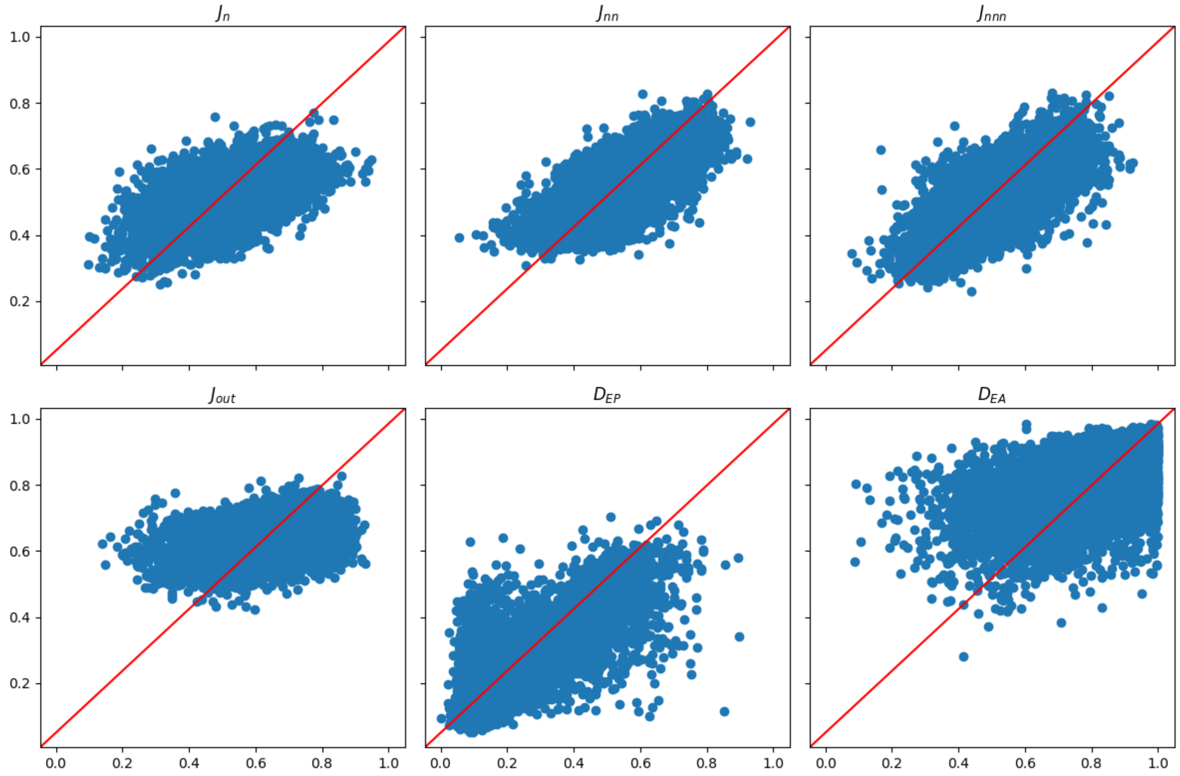


Figure 19: Scatter Plot of the baseline NN for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

5.3 Improving Pre-Processing

Pre-Processing (PP) can have a big impact on model performance. For this reason we will compare different PP strategies against the baseline PP approach described in subsubsection 5.2.1. The following PP strategies will be tested for the input images:

- Baseline Approach
- MinMax Scaling
- Standardization

The baseline approach for image Pre-Processing simply clips the pixel values of all images to the interval [0, 1].

MinMax Scaling takes the global minimum and maximum of all images and applies the following transformation to each pixel x of the input images: $(x-minimum)/(maximum-minimum)$.

The last Pre-Processing strategy we will test for the input images is Standardization which takes the global mean μ and standard deviation σ of the input images and applies the following transformation to each pixel x of every image: $(x - \mu)/\sigma$.

The baseline approach and MinMax-Scaling ensure that the global minimum and maximum of the input images are 0 and 1 respectively while Standardization ensures that the mean μ is 0 and the standard deviation σ is 1 for the pixel values. According to LeCun et al. [24] Standardization of the input images should lead to faster convergence of the NN, however since the training budget won't be a limiting factor in this thesis, we will not prioritize faster convergence of the model over model performance. However if the training budget would be more limited, one could prefer Standardization if the faster convergence is more important and the test MSE is as good as or slightly worse than the other PP methods.

For the labels we will test the following Pre-Processing methods:

- Baseline Approach
- MinMax Scaling

The baseline approach will transform the input labels as described in subsubsection 5.2.1.

For MinMax scaling each parameter will be scaled individually using the minimum and maximum for each parameter, which is important to ensure that all parameters are on the same scale.

All strategies will be tested on the bNN and a simple CNN. The archtitecture of the CNN is depicted in Figure 30. The reason we also test all strategies on a CNN is to avoid picking a strategy that performs well only on FC-NN's which could deteriorate the performance of all CNN models. We will test all image PP approaches with all label PP approaches, which adds up to six different PP strategies. Every strategy will be trained on the bNN and the CNN for 50 epochs which is enough to reach convergence. For each epoch the MSE on the validation set will be calculated and the epoch with the lowest validation MSE will be used to calculate the test MSE.

5.3.1 Training

Every Model was trained for 50 Epochs and a batch size of 64 was used. The MSE per epoch on the validation data for each PP approach trained on the bNN is shown in Figure 20. For the CNN it is shown in Figure 21.

5.3.2 Evaluation

As shown in Table 5 we can observe that for the bNN, MinMaxing the labels performs considerably better than the baseline method regardless of the image PP approach.

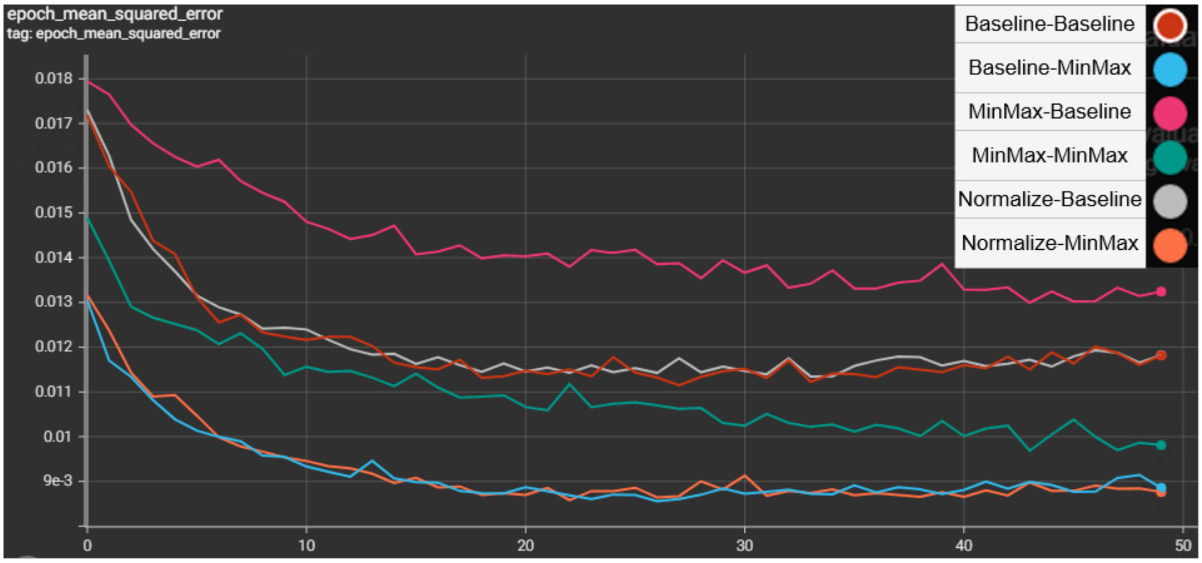


Figure 20: Training of baseline NN with different Pre-Processing approaches. The MSE per epoch on the validation data is shown.

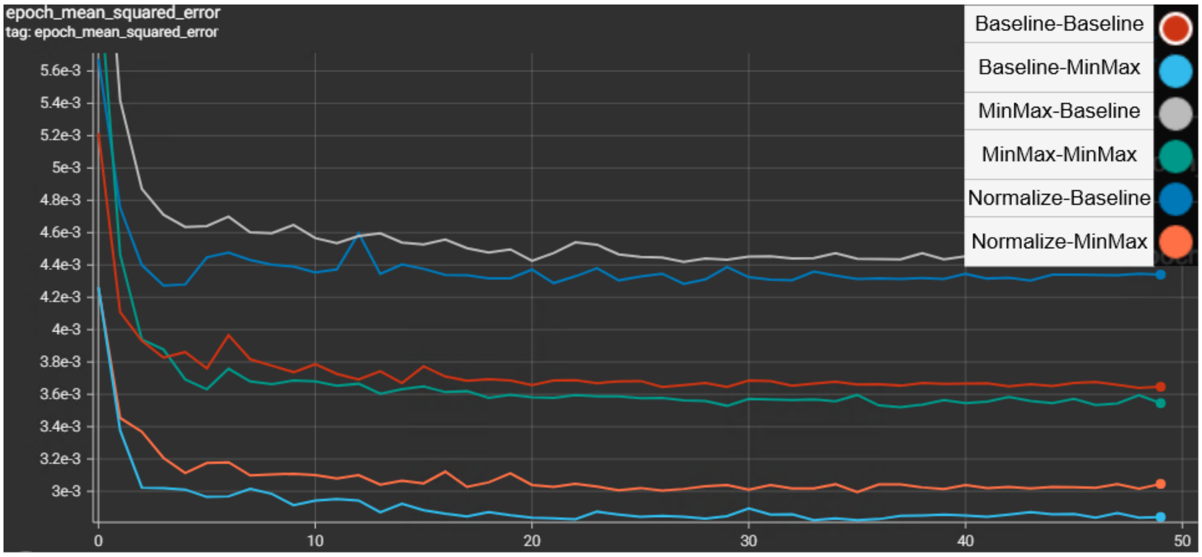


Figure 21: Training of a simple CNN with different Pre-Processing approaches. The MSE per epoch on the validation data is shown.

When looking at the results for the CNN in Table 6 we can observe the same, except for one instance, where the MSE is slightly higher for the MinMaxed labels. As MinMax scaling the labels performed better than the baseline approach except for one instance, we will chosoe it as our label PP approach for the following experiments.

When comparing the different image PP approaches we can observe that the baseline method always performed the best on the bNN as well as the CNN. For this reason we

Pre-Processing		MSE per Parameter						
Images	Labels	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
Baseline	Baseline	0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877
MinMax	Baseline	0.01118	0.00721	0.01025	0.01074	0.00685	0.01461	0.01014
Normalize	Baseline	0.00891	0.00615	0.00498	0.00996	0.00643	0.01629	0.00879
Baseline	MinMax	0.00911	0.00653	0.00593	0.00883	0.00652	0.01441	0.00855
MinMax	MinMax	0.00999	0.00698	0.00837	0.01114	0.00695	0.01493	0.00972
Normalize	MinMax	0.00889	0.00637	0.00527	0.00941	0.00631	0.01566	0.00865

Table 5: MSE of baseline NN for each Pre-Processing approach on MinMax scaled test data for each parameter and overall

Pre-Processing		MSE per Parameter						
Images	Labels	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
Baseline	Baseline	0.00263	0.00189	0.00174	0.00377	0.00108	0.00656	0.00294
MinMax	Baseline	0.00294	0.00227	0.00208	0.00434	0.00137	0.00755	0.00342
Normalize	Baseline	0.00298	0.00229	0.00215	0.00389	0.00140	0.00721	0.00332
Baseline	MinMax	0.00250	0.00186	0.00174	0.00337	0.00101	0.00660	0.00285
MinMax	MinMax	0.00293	0.00223	0.00212	0.00422	0.00137	0.00780	0.00344
Normalize	MinMax	0.00273	0.00200	0.00198	0.00351	0.00106	0.00691	0.00303

Table 6: MSE of CNN for each Pre-Processing approach on MinMax scaled test data for each parameter and overall

will use the baseline image PP for the following experiments. For the remainder of this thesis optimal Pre-Processing (OPP) will refer to MinMax scaled labels and baseline scaled images.

When comparing the scatter plots of the bNN and the bNN with OPP in Figure 22, we cannot observe any clear improvement over the bNN.

5.4 Applying Batch Normalization and Dropout

Since no HPO was performed on the bNN it should be fairly easy to improve its performance. Two simple and effective ways to improve the performance of NNs are Dropout [39] and Batch Normalization [16], thus we will first show how much these two concepts alone can improve the performance on the bNN. Batch Normalization (BN) is a technique that reduces the internal convariate shift, which is defined as "the change in the distribution of network activations due to the change in network parameters during training"[16]. It achieves this by normalizing the input of each layer[16]. For computational reasons the normalization is computed for each mini-batch individually. The algorithm is shown in Figure 23. ϵ is a constant added for numerical stability, while γ and β are learnable parameters that are necessary to "restore the representation power of the network" [16], which would otherwise get lost due to the normalization

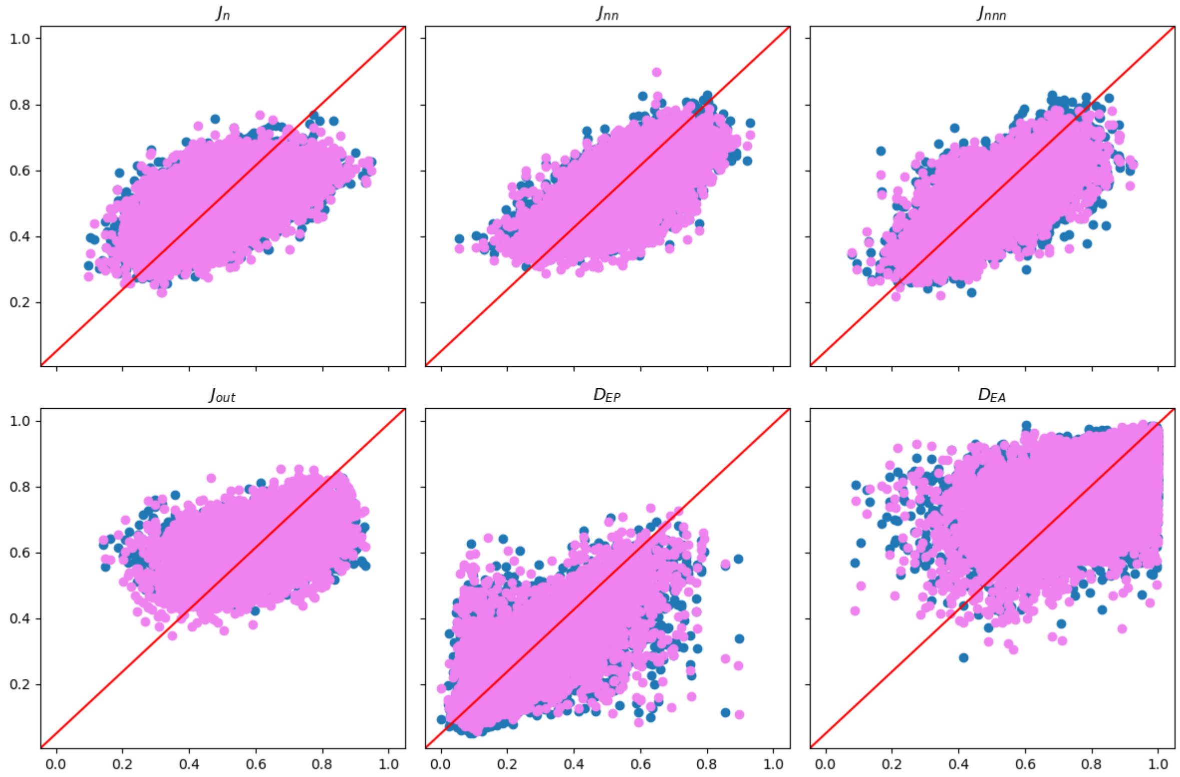


Figure 22: Scatter Plot of the baseline NN (blue) and baseline NN with OPP (pink) for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

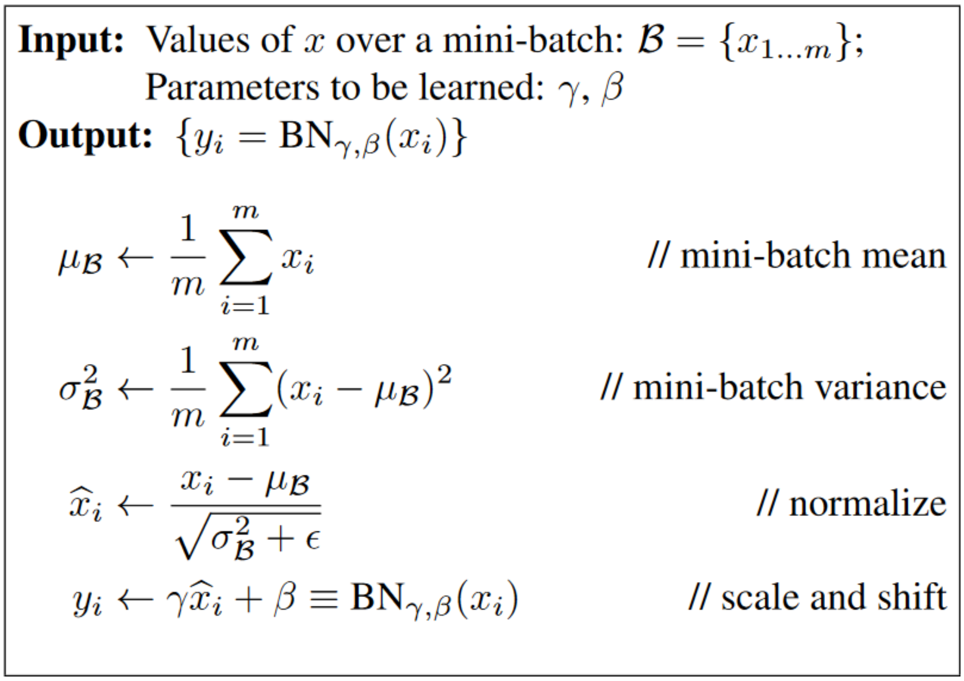
of the inputs. Since BN is computed for each mini-batch, different batch sizes should yield different results, which we will show later in this subsection.

The other method we want to apply in this subsection is called Dropout. Dropout "prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently" [39]. It achieves this by temporarily dropping out neurons along with their incoming and outgoing connections as shown in Figure 24.

The probability p to retain a neuron can be set to a different value for each layer and is a hyperparameter, meaning that the optimal value for each layer can be determined with HPO. It is important to note that dropping out neurons is only applied during training. When testing the NN every neuron is always present and its weights are multiplied by the probability p set for its layer as shown in Figure 25.

5.4.1 Training

All training in this subsection was performed on optimally Pre-Processed images and labels. We will first show the effect of adding BN and Dropout to the bNN individually and then both added at the same time. Whenever a model used BN it was added to



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

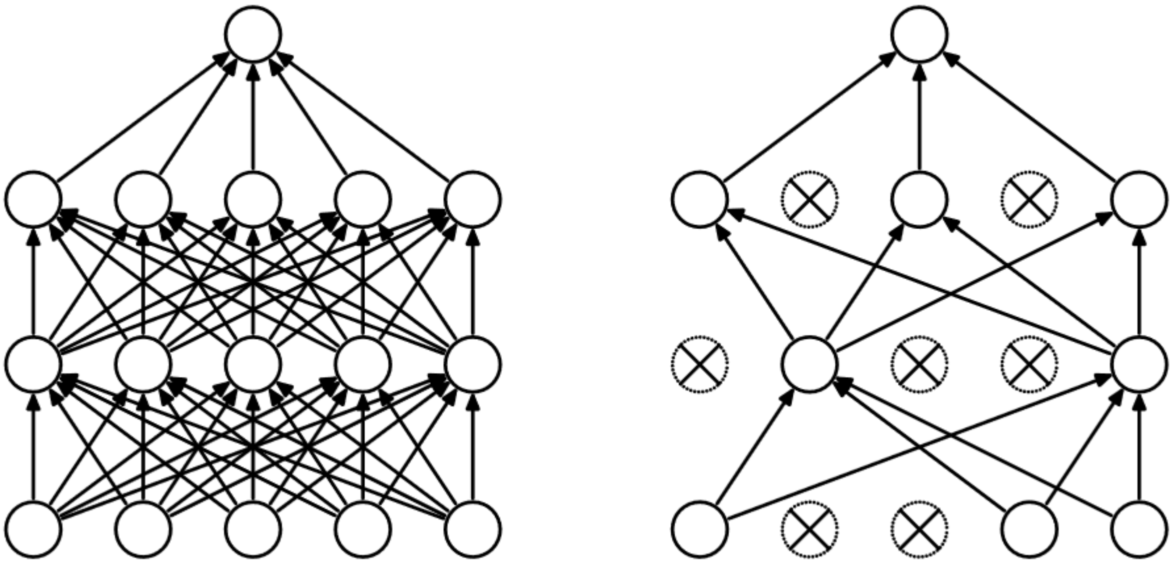
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 23: Batch Normalizing Transform, applied to activation x over a mini-batch. [16]



(a) Standard Neural Net

(b) After applying dropout.

Figure 24: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped [39].

every layer of the bNN and as Ioffe and Szegedy [16] suggest we will place the BN right before the activation function as opposed to in between each layer.

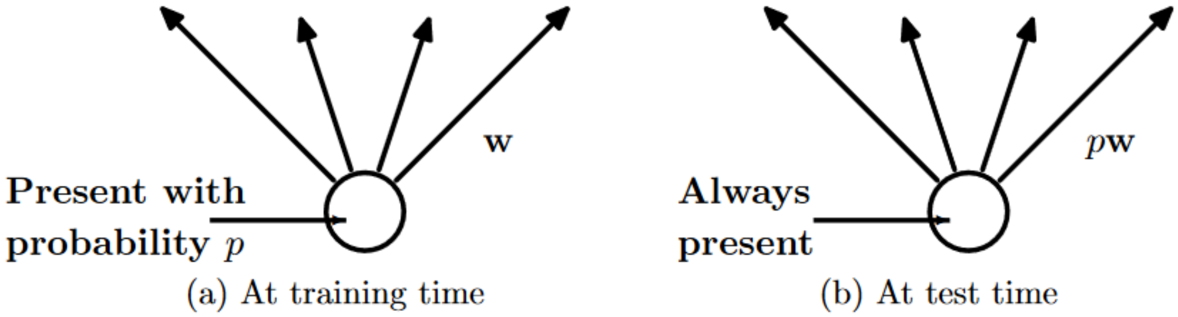


Figure 25: Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights w. Right: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time. [39].

The models mentioned in this section are labeled as follows:

1. bNN-x : bNN with baseline Pre-Processing for images and labels.
2. bNN+OPP-x : bNN with optimized Pre-Processing.
3. BN-x : bNN with OPP and BN applied to every layer.
4. Dropout-x : bNN with OPP and Dropout in every layer.
5. Dropout+BN-x : bNN with OPP and BN and Dropout in every layer.

The x behind "-" refers to the batch size used for model training.

To show the impact of the batch size when using BN we will train the BN-x model with different batch sizes starting at 16 and doubling until a batch size of 4096. Each of these models was trained for 50 Epochs. The training process for the different batch sizes on the BN-x model is shown in Figure 26.

For the Dropout-x and BN+Dropout-x models we performed HPO using Keras-Tuner to find the best HPs for Dropout. As HPO tends to take a large amount of time we will set the batch size to 256 and make use of a concept called Early Stopping [31] to save time on bad performing sets of HPs. As opposed to training each model for a fixed amount of epochs, Early Stopping simply stops the training when a stopping criterion is met. In our case we stopped training of the current model, when no improvement of 0.00005 on the validation MSE is achieved for 10 consecutive epochs. To denote the dropout values between each layer we will use the vector $X = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}]^T$, where x_1 denotes the dropout value between

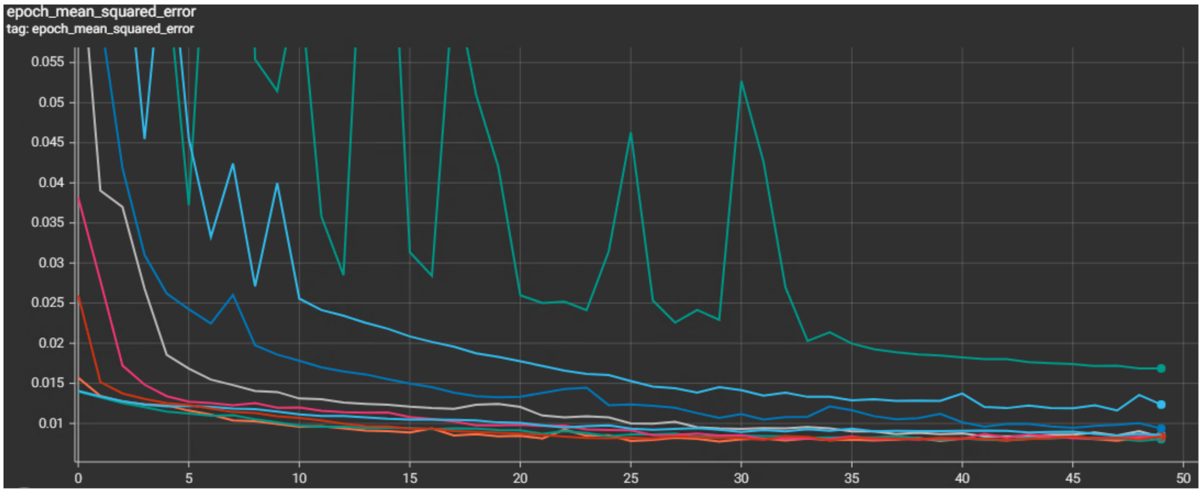


Figure 26: Training of the BN-x model with different batch sizes. The MSE per epoch on the validation data is shown. The worst performing batch sizes on the validation data are 4096, 2048 and 1024 in descending order.

the input and the first layer, x_2 denotes the dropout value between the first and second layer and so on. It should be noted that the value x_i denotes the percentage of a neuron being dropped out instead of being retained. This is because Tensorflow[1], the backend for Keras, implements the dropout layer in this way.

The best Dropout values for the Dropout-256 and Dropout+BN-256 models were found to be:

1. Dropout-256 $= [0, 0.3, 0, 0.4, 0, 0, 0, 0, 0, 0]^T$
2. Dropout+BN-256 $= [0, 0.2, 0, 0.7, 0.7, 0.7, 0.5, 0.15, 0, 0.5]^T$

5.4.2 Evaluation

When looking at the impact of different batch sizes to the BN-x Model as shown in Table 7 we can observe that the validation MSE gets increasingly worse for the larger batch sizes. This performance is probably caused by the "generalization gap", which states that "when using large batch sizes there is a persistent degradation in generalization performance"[14]. The batch sizes 32, 64 and 128 performed the best and they all have very similar performance. The smaller batch size 16 performs comparatively poor, This performance cannot be explained with the generalization gap and is probably caused, due to the small batch size inducing noise when calculating the mean and variance of the mini-batch during the Batch Normalization. As Table 7 shows, the addition of BN to the baseline NN already increases its performance significantly. The BN-32 Model which performed the best lowers the MSE to 0.00788 which corresponds to a ~7.8% lower MSE compared to the bNN+OPP-64 model. In total the use of an optimized Pre-Processing approach and the addition

	MSE per Parameter						
Model	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
bNN-64	0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877
bNN+OPP-64	0.00911	0.00653	0.00593	0.00883	0.00652	0.01441	0.00855
BN-16	0.00868	0.00625	0.00508	0.01067	0.00523	0.01505	0.00849
BN-32	0.00794	0.00575	0.00453	0.00861	0.00509	0.01540	0.00789
BN-64	0.00815	0.00585	0.00469	0.00883	0.00523	0.01454	0.00788
BN-128	0.00825	0.00586	0.00484	0.00883	0.00494	0.0146	0.00789
BN-256	0.00834	0.00608	0.00509	0.00867	0.00540	0.01463	0.00803
BN-512	0.00889	0.00618	0.00527	0.00927	0.00570	0.01504	0.00839
BN-1024	0.00933	0.00655	0.00607	0.01073	0.00803	0.01595	0.00944
BN-2048	0.01060	0.00708	0.00892	0.01209	0.01141	0.02004	0.01169
BN-4096	0.01290	0.01016	0.01157	0.01274	0.02228	0.03188	0.01692

Table 7: MSE of different models on MinMax scaled test data for each parameter and overall

	MSE per Parameter						
Model	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
bNN-64	0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877
bNN+OPP-64	0.00911	0.00653	0.00593	0.00883	0.00652	0.01441	0.00855
BN-256	0.00834	0.00608	0.00509	0.00867	0.00540	0.01463	0.00803
Dropout-256	0.00865	0.00609	0.00534	0.00843	0.00584	0.01323	0.00793
Dropout+BN-256	0.00810	0.00560	0.00464	0.00820	0.00474	0.01378	0.00750

Table 8: MSE of different models on MinMax scaled test data for each parameter and overall

of BN to the baseline NN lowers the MSE compared to the baseline approach by ~10.1%

When looking at the MSE for the Dropout-256 and Dropout+BN-256 model shown in Table 9 we can observe that the Dropout-256 model performs better than the BN-256 model. It is important to compare the models on the same batch size as otherwise we wouldn't isolate the effect of adding Dropout or BN to the model. Dropout alone is able to reduce the MSE in comparison to the bNN+OPP-64 by ~7.3%. The best result however was achieved when combining the use of Dropout and BN. The Dropout+BN-256 model achieves the lowest MSE so far with a MSE of 0.00750. This corresponds to a ~12.3% lower MSE compared to the bNN+OPP-64. Compared to the baseline approach the Dropout+BN-256 model achieves a ~14.5% lower MSE. When looking at the per parameter MSE the biggest improvement over bNN+OPP-64 can be observed for D_{EP}, which is ~27.3% lower than for bNN+OPP-64.

When looking at the scatter plots Figure 27 of the Dropout+BN-256 model, which is currently the best performing model we can observe that the deviations from the optimal line have gotten smaller for D_{EP} and the NN has started has stopped to simply predict values around the mean for J_{out}, however the predictions are still far from optimal. D_{EA} still is the worst performing parameter so far.

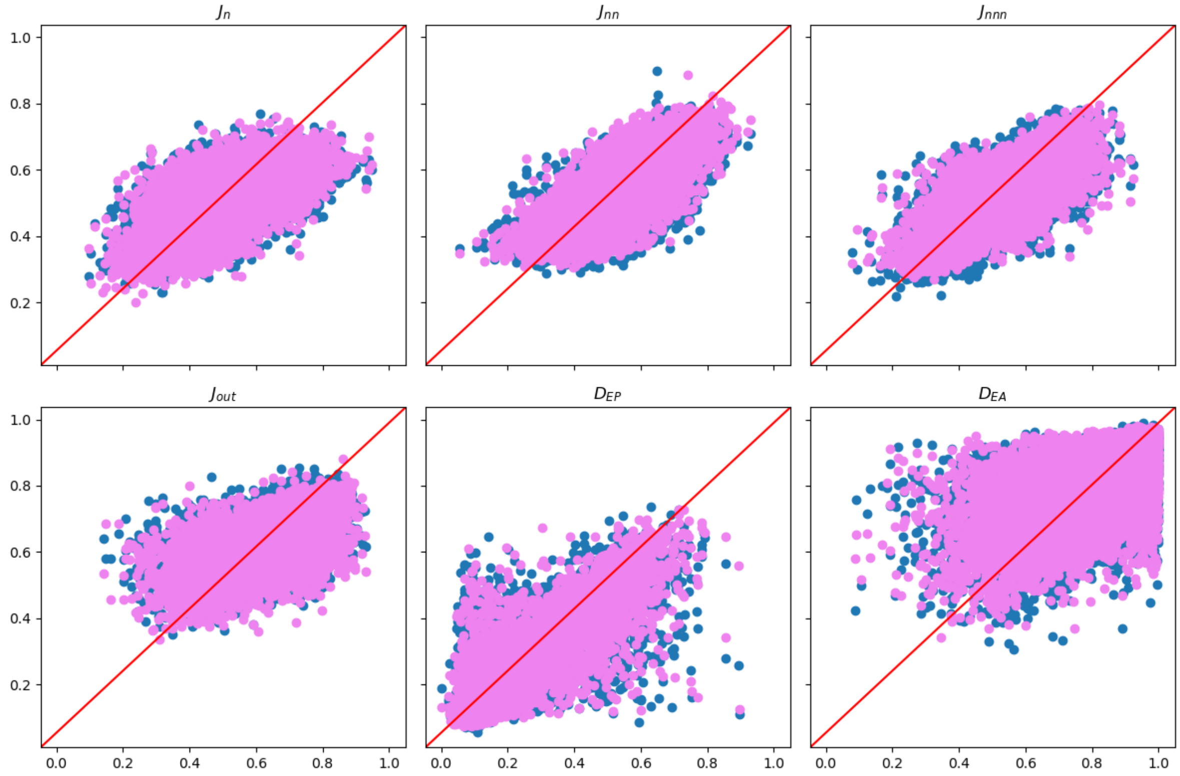


Figure 27: Scatter Plot of bNN+OPP-64 (blue) and Dropout+BN-256 (pink) for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

One insight we have gained from the experiments in this subsection is to use a batch size of 128 instead of 256 in future experiments were HPO is performed as the faster training time of the 256 batch size does not justify the significant deterioration of the NN performance.

5.5 Improving the Baseline Neural Network further

As we have seen the use of optimized Pre-Processing, Dropout and BN already increase the NN performance significantly, however there are still other HPs we haven't altered such as the Learning Rate, the number of neurons in each layer or the number of layers. In this subsection we want to achieve the best possible performance using a FC-NN approach.

5.5.1 Training

To find the best FC-NN Model we performed HPO using KerasTuner. The batch size was set to 128. To save training time on bad performing sets of HPs we used Early Stopping with the same stopping criterion as in 5.4. The HP search space included:

1. Number of Layers
2. Number of Neurons in each Layer
3. Loss function (either MSE or BCE)
4. Dropout Value for each Layer
5. Learning Rate for the Adam optimizer

The best model found by KerasTuner is depicted in Figure 28. The best Learning Rate was found at 0.0001 for the Adam optimizer and BCE seemed to outperform MSE as the loss function. In comparison to the baseline NN the number of layers was more than halved, however each layer contains much more neurons than the baseline NN. The optimal Learning Rate is also 10 times lower than the one used for the baseline approach. The optimized NN will be denoted as ONN-128.

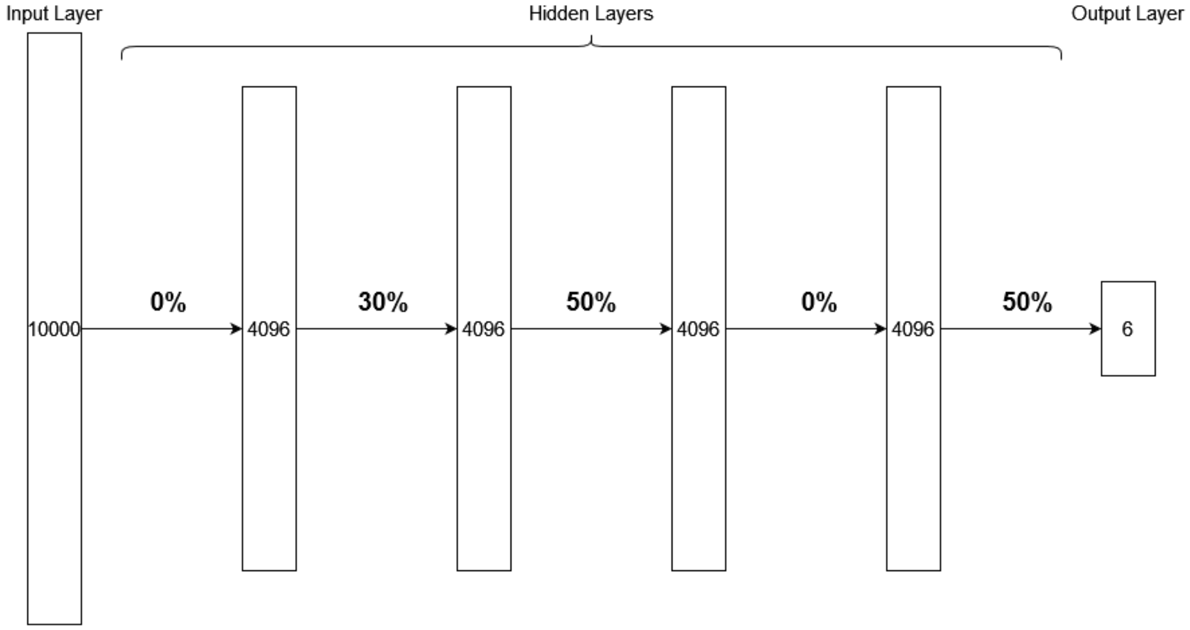


Figure 28: Architecture of the optimized NN: The model consists of 4 hidden Layers which all have 4096 Neurons. The output layer has six neurons. The Dropout value between each layer is indicated by the percentage between each layer. BN is applied to each Layer before the activation function, which was ReLU in all layers except the output Layer which used Sigmoid.

	MSE per Parameter						
Model	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
bNN-64	0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877
bNN+OPP-64	0.00911	0.00653	0.00593	0.00883	0.00652	0.01441	0.00855
Dropout+BN-256	0.00810	0.00560	0.00464	0.00820	0.00474	0.01378	0.00750
ONN-128	0.00761	0.00539	0.00457	0.00756	0.00477	0.01322	0.00719

Table 9: MSE of different models on MinMax scaled test data for each parameter and overall

5.5.2 Evaluation

When looking at Table 9 we can see that the ONN-128 model was able to decrease the MSE even further than the Dropout+BN-256 model. The MSE of ONN-128 is ∼4.1% lower than the Dropout+BN-256 Model. The per parameter MSE was also decreased for every parameter except for D_{EP} which is 0.00003 higher than the Dropout+BN-256 model. Compared to bNN+OPP-64 the optimized NN has a ∼15.9% lower MSE.

When comparing the scatter plots of ONN-128 and the Dropout+BN-256 Model shown in Figure 29 we cannot observe any significant improvement in prediction confidence, except for smaller deviations from the optimal line.

5.6 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have been shown to perform really well on image related Tasks such as image classification ([21, 37, 40]), thus we want to test if CNNs are also able to achieve good results on our simulated data. In theory CNNs can take advantage of the grid-like topology of 2D images [12] and since our dataset fundamentally consists of 2D images, CNNs should also perform well on our dataset. We propose a simple CNN, for which no HPO was performed. The architecture of the CNN is depicted in Figure 30.

5.6.1 Training

The proposed CNN was trained for 50 Epochs with the Adam optimizer and a batch size of 64. It will be denoted as UCNN-64. α, β_1 and β_2 needed for Adam were set to the same values used for training the baseline NN. We used MSE as the loss function. The CNN was trained on optimally Pre-Processed data. The MSE on the train and validation data during the training procedure is shown in Figure 31

5.6.2 Evaluation

The first thing we can observe when looking at Figure 31 is that the CNN already has a good performance on the validation data at the sixth epoch. There is only about a 5% difference in MSE between the sixth Epoch and the best epoch which was found at

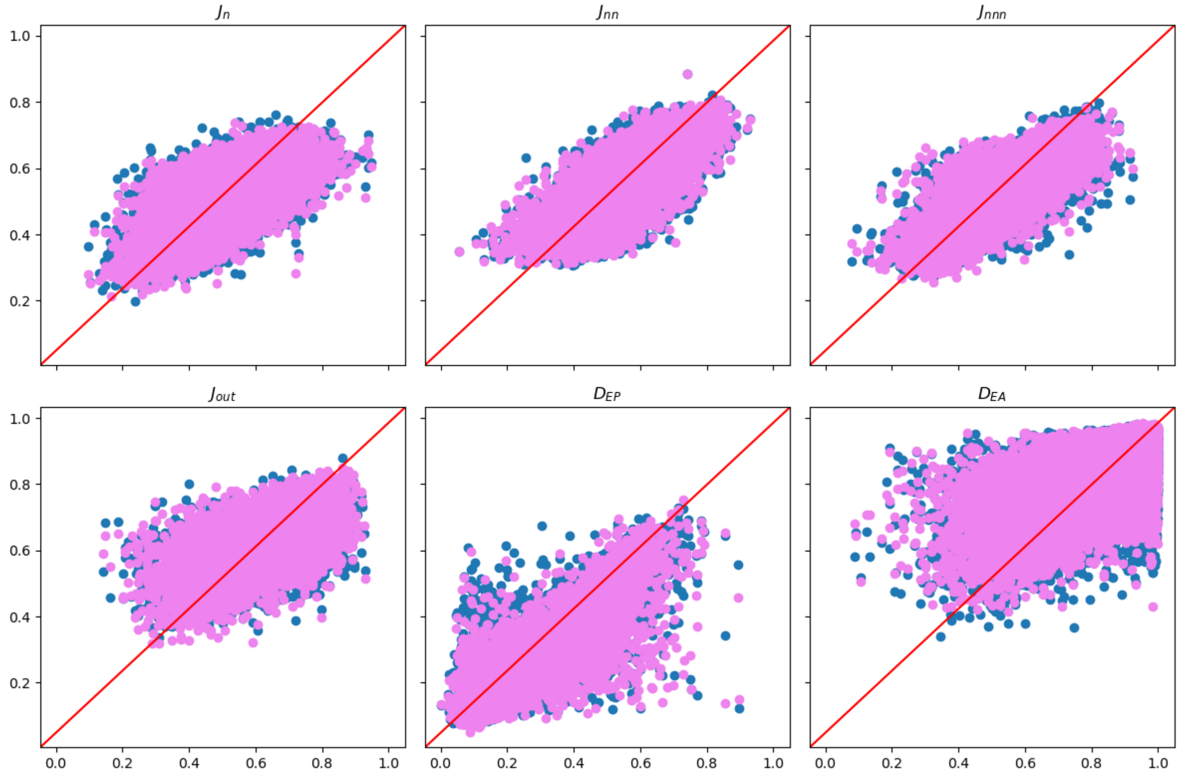


Figure 29: Scatter Plot of Dropout+BN-256 (blue) and ONN-128 (pink) for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

Epoch 36. When comparing this with the bNN-64 model, the same 5% difference in MSE from the best epoch would have been achieved at the 16th epoch meaning that the baseline NN needs more epochs to converge than the CNN.

When looking at the MSE on the Test Data in Table 12 we can observe that the unoptimized CNN already outperforms the ONN-128 model significantly. The MSE of UCNN-64 is ~60.4% lower than the MSE of ONN-128. When looking at the per parameter MSE the biggest improvement can be observed for D_{EP} which is ~78.8% lower than for ONN-128

The same big performance increase can be observed when looking at the scatter plots in Figure 32. All six parameters of UCNN-64 have smaller deviations from the optimal line in comparison to ONN-128. For J_n, J_{nn} and J_{nnn} the predictions of ONN-128 were already following the optimal line, these deviations have simply gotten smaller. The biggest improvements however can be observed for J_{out} and D_{EP}. The FC-NN approach had issues with correctly learning to predict J_{out}. This has changed for the unoptimized CNN, the predictions now follow the optimal line comparable to the first three interaction parameters, which is a huge improvement. The biggest improvement

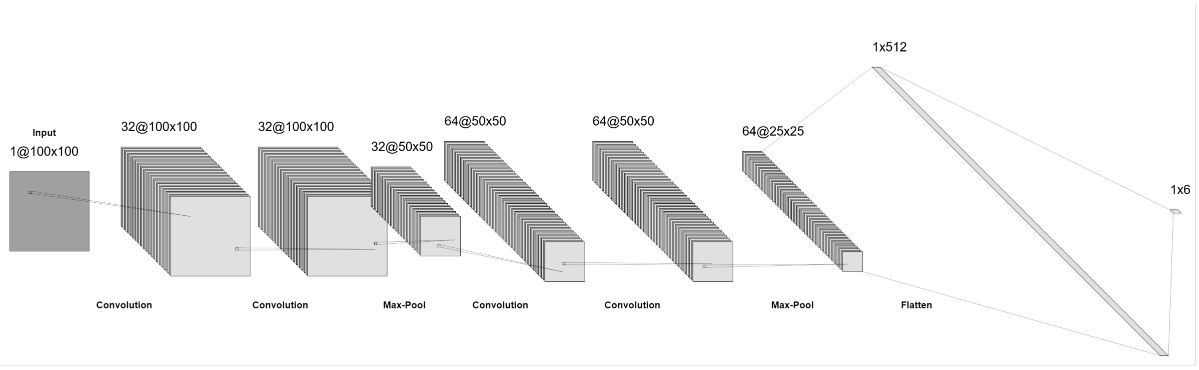


Figure 30: Architecture of the unoptimized CNN: The model consists of the input layer, four convolutional layers, one fully-connected layer and the output layer. A (2x2) Maxpooling Layer with a (1,1) stride was inserted after every two convolutional layers. The first two convolutional layers have 32 Filters, while the last two convolutional layers have 64 Filters. All convolutional layers have a (3x3) kernel with a (1,1) stride. The FC layer has 512 neurons and the output layer has 6 neurons, one for each parameter of the Hamiltonian. All convolutional layers and the FC layer use the ReLU activation function, while the output layer uses the Sigmoid activation function.

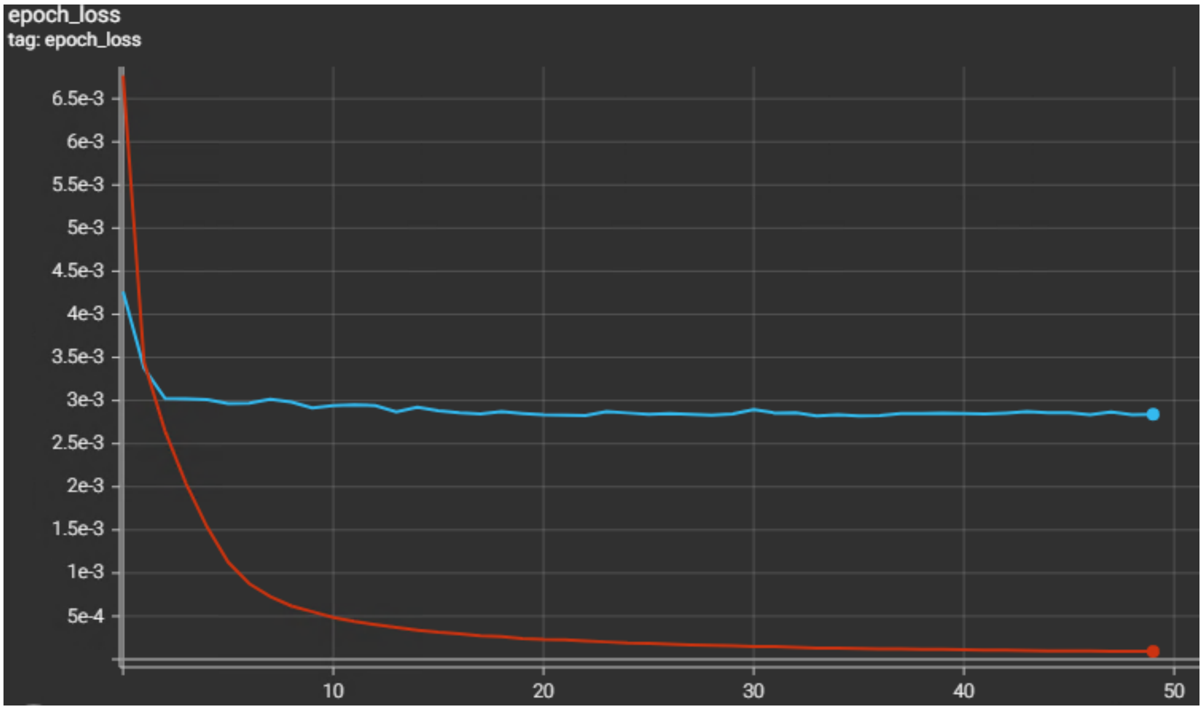


Figure 31: Training of the unoptimized CNN. The MSE per epoch on training data (red) and validation data (blue) is shown.

however was made for D_{EP}. ONN-128 had large deviations from the optimal line. This has completely changed for the unoptimized CNN, which confidently predicts

	MSE per Parameter						
Model	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
bNN-64	0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877
ONN-128	0.00761	0.00539	0.00457	0.00756	0.00477	0.01322	0.00719
UCNN-64	0.00250	0.00186	0.00174	0.00337	0.00101	0.00660	0.00285

Table 10: MSE of different models on MinMax scaled test data for each parameter and overall

D_{EP} with the smallest deviations from the optimal line in comparison with the other parameters. D_{EA} is still the worst parameter, but the predictions of the unoptimized CNN follow the optimal line better in comparison to ONN-128.

Looking at these results we can confidently say that the low performance of J_{out} as well as D_{EP} on the FC-NN models were created by the NN architecture and not the input images. In contrast to the FC-NN models the unoptimized CNN has learned to predict these parameters well. For D_{EA} however the performance is still far behind the other parameters. It is possible that a more optimized CNN, is able to resolve this problem or the input images simply don't allow for a better prediction. For this reason we will try to optimize the CNN approach further in the next subsection.

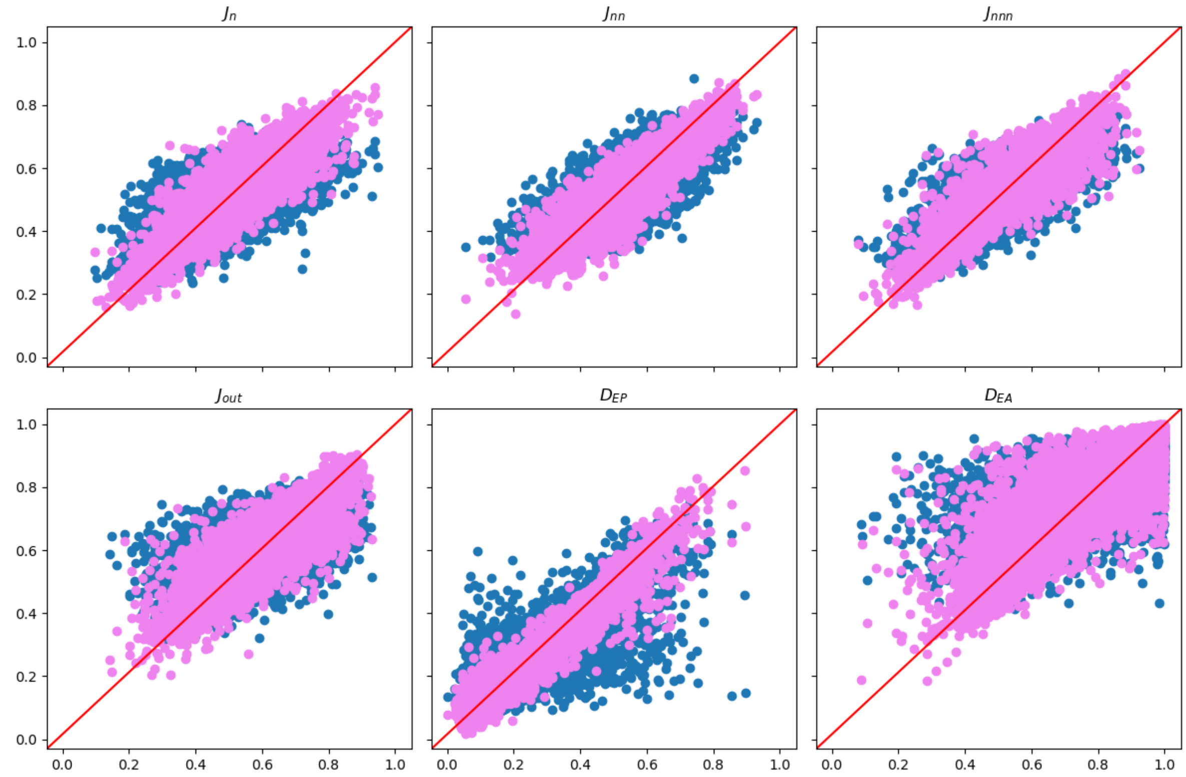


Figure 32: Scatter Plot of ONN-128 (blue) and UCNN-64 (pink) for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

5.7 Optimizing the CNN Approach

Considering that no HPO was performed on the unoptimized CNN the performance increase is already substantial and will likely increase even more when optimizing the CNN approach. For this reason we will perform HPO on the unoptimized CNN in this subsection using KerasTuner. We will search for the best CNN using all tools we have used so far such as BN, Dropout, ...

5.7.1 Training

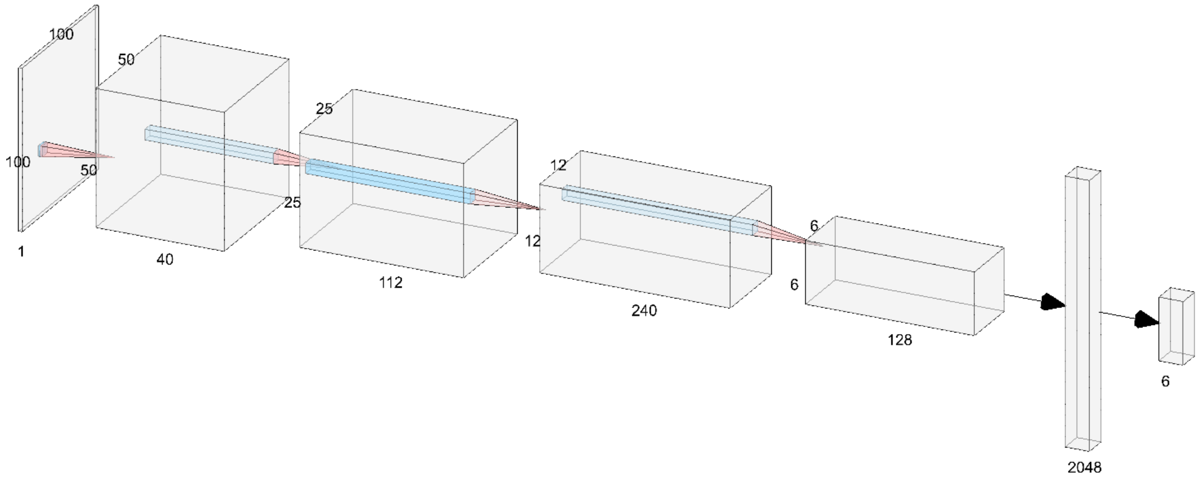


Figure 33: Architecture of the optimized CNN: The hidden layers of the model consists of four Conv-Conv-Max-Pooling Blocks followed by a dense layer with 2048 neurons. All hidden layers use the ReLU activation function. The output layer has six neurons and uses the sigmoid activation function. All convolutional layers have a (3x3) kernel with stride (1,1). All max-pooling layers are (2x2) with stride (1x1). BN is placed before each activation function including the output layer. The number of filters used by the convolutional layers is printed below each block.

The batch size was set to 128. The best model found by KerasTuner is depicted in Figure 33. The Learning Rate for the Adam optimizer is 0.014464. In contrast to the unoptimized CNN, the optimized CNN uses BCE as its loss function as it performed better. BN was placed before each activation function, however no Dropout was used. We will refer to the optimized CNN as OCNN-128.

5.7.2 Evaluation

When looking at Table 11 we can see that the optimized CNN OCNN-128 was able to lower the overall MSE by ~42.8% in comparison to the UCNN-64 model. When looking at the MSE per Parameter all MSE's were reduced considerably while the biggest improvement can be observed for D_{EP} which is ∼70.3% lower than for the UCNN-64 model.

	MSE per Parameter						
Model	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
bNN-64	0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877
UCNN-64	0.00250	0.00186	0.00174	0.00337	0.00101	0.00660	0.00285
OCNN-128	0.00137	0.00097	0.00102	0.00196	0.00030	0.00417	0.00163

Table 11: MSE of different models on MinMax scaled test data for each parameter and overall

When comparing the scatter plots in Figure 34 the biggest improvement can also be seen for D_{EP}, which follows the optimal line really well, with very small deviations.

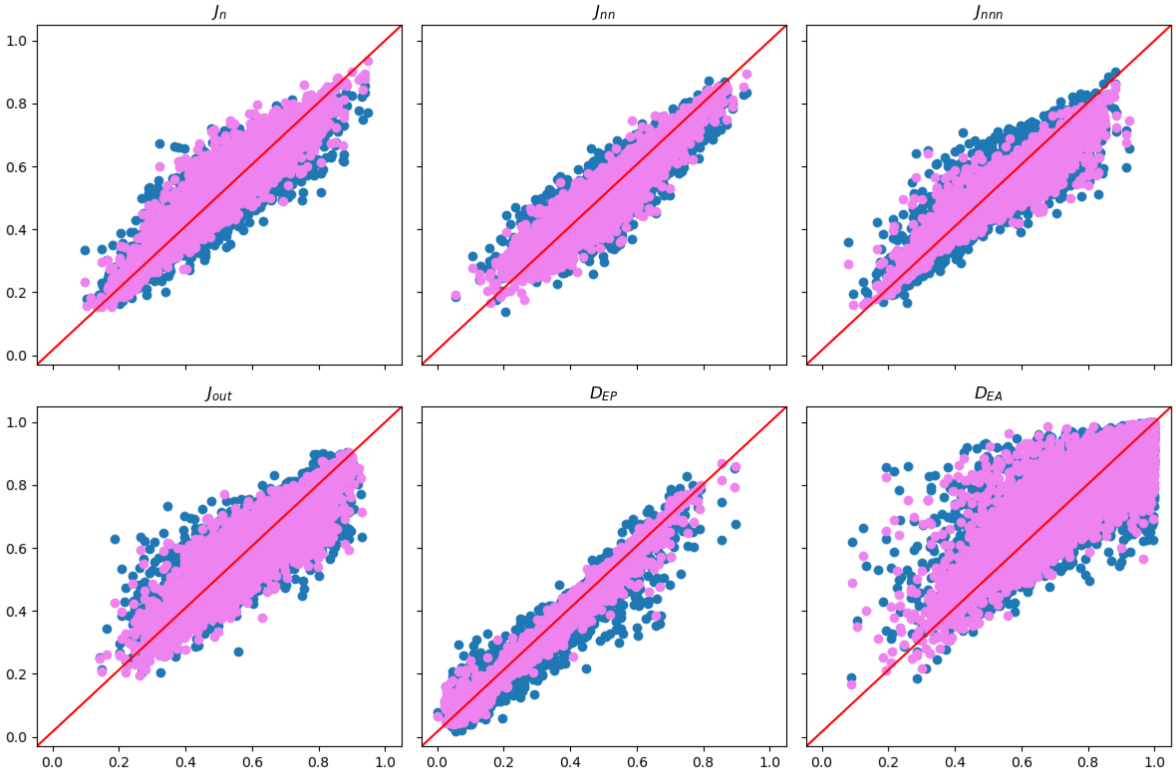


Figure 34: Scatter Plot of UCNN-64 (blue) and OCNN-128 (pink) for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

6 Results

	MSE per Parameter						
Model	J_n	J_{nn}	J_{nnn}	J_{out}	D_{EP}	D_{EA}	Overall
bNN-64	0.00876	0.00634	0.00579	0.01023	0.00689	0.01459	0.00877
bNN+OPP-64	0.00911	0.00653	0.00593	0.00883	0.00652	0.01441	0.00855
BN-64	0.00815	0.00585	0.00469	0.00883	0.00523	0.01454	0.00788
BN-256	0.00834	0.00608	0.00509	0.00867	0.00540	0.01463	0.00803
Dropout-256	0.00865	0.00609	0.00534	0.00843	0.00584	0.01323	0.00793
Dropout+BN-256	0.00810	0.00560	0.00464	0.00820	0.00474	0.01378	0.00750
ONN-128	0.00761	0.00539	0.00457	0.00756	0.00477	0.01322	0.00719
UCNN-64	0.00250	0.00186	0.00174	0.00337	0.00101	0.00660	0.00285
OCNN-128	0.00137	0.00097	0.00102	0.00196	0.00030	0.00417	0.00163

Table 12: MSE of different models on MinMax scaled test data for each parameter and overall

In this section we want to summarize the results of all conducted experiments. In subsection 5.3 we wanted to show how much the Pre-Processing approach alone can improve the performance of the baseline NN. With the optimized Pre-Processing strategy, we were able to reduce the overall MSE by ~2.5%, however the MSE for the parameters J_n, J_{nn}, J_{nnn} actually increased. The reason that the overall MSE is still lower is because the MSE for J_{out} could be reduced by ~13.7% which is a big improvement considering that the NN architecture didn't change.

subsection 5.4 and subsection 5.5 can be grouped together as the goal of these two subsections was to optimize the FC-NN approach that the baseline NN used. By combining the use of BN, Dropout and HPO in the ONN-128 model we were able to reduce the overall MSE by ~18% in comparison to the baseline approach. In contrast to the bNN+OPP-64 model, which increased the MSE for some parameters, we were able to reduce the MSE of every parameter by at least 9% in comparison to the baseline approach. In addition we were able to solve a big problem that the baseline approach had regarding the prediction of J_{out}. When looking at Figure 35 we can see that the ONN-128 model has started to learn the prediction of J_{out}. The baseline approach originally only predicted values around the mean for J_{out}, which has now changed for the ONN-128 model, however the predictions are still far from optimal. The predictions for D_{EP} have also gotten better, however using a FC-NN approach we were not able to solve the bad predictions of D_{EA}.

The following two subsections subsection 5.6 and subsection 5.7 have dealt with the use of CNNs instead of a pure FC-NN. The result of these two subsections was the

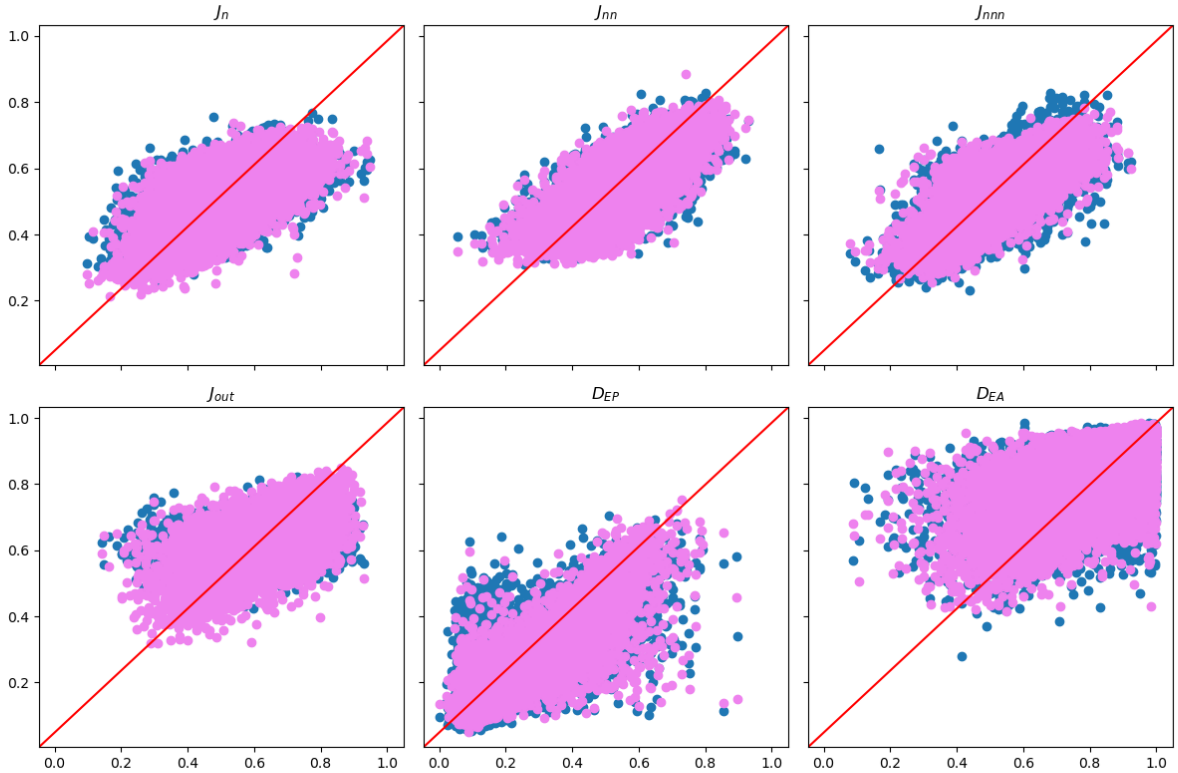


Figure 35: Scatter Plot of bNN-64 (blue) and ONN-128 (pink) for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

OCNN-128 model which achieved the best performance out of all conducted experiments in this bachelor thesis. When comparing the baseline approach to the optimized CNN, we were able to reduce the overall MSE by ∼81.4% which corresponds to a ∼5.4 times better performance. When looking at the MSE per Parameter the smallest improvement was achieved for D_{EA}, which was lowered by ∼71.4%. Although this was the smallest improvement, it still corresponds to a ∼3.5 times better performance. The biggest improvement however was made for D_{EP}. We were able to reduce the MSE by ∼95.6% which corresponds to a ∼23 times better performance. This performance increase is massive and turned D_{EP} into the best predicted parameter by far in comparison to the other parameters. When looking at the scatter plots in Figure 36 we can observe that the OCNN-128 model has learned good predictions for all parameters. The biggest improvements are made for J_{out}, D_{EP} and D_{EA}. The CNN has learned to predict J_{out}, which was a problem for the baseline approach. D_{EP} has made the biggest improvement, as it was one of the worst parameters for the baseline approach when looking at the scatter plot, however this completely changed for the optimized CNN. D_{EP} is now by far the best predicted parameter. The predition of D_{EA} has also made big improvements in comparison to the baseline approach, but it is still the worst performing parameter.

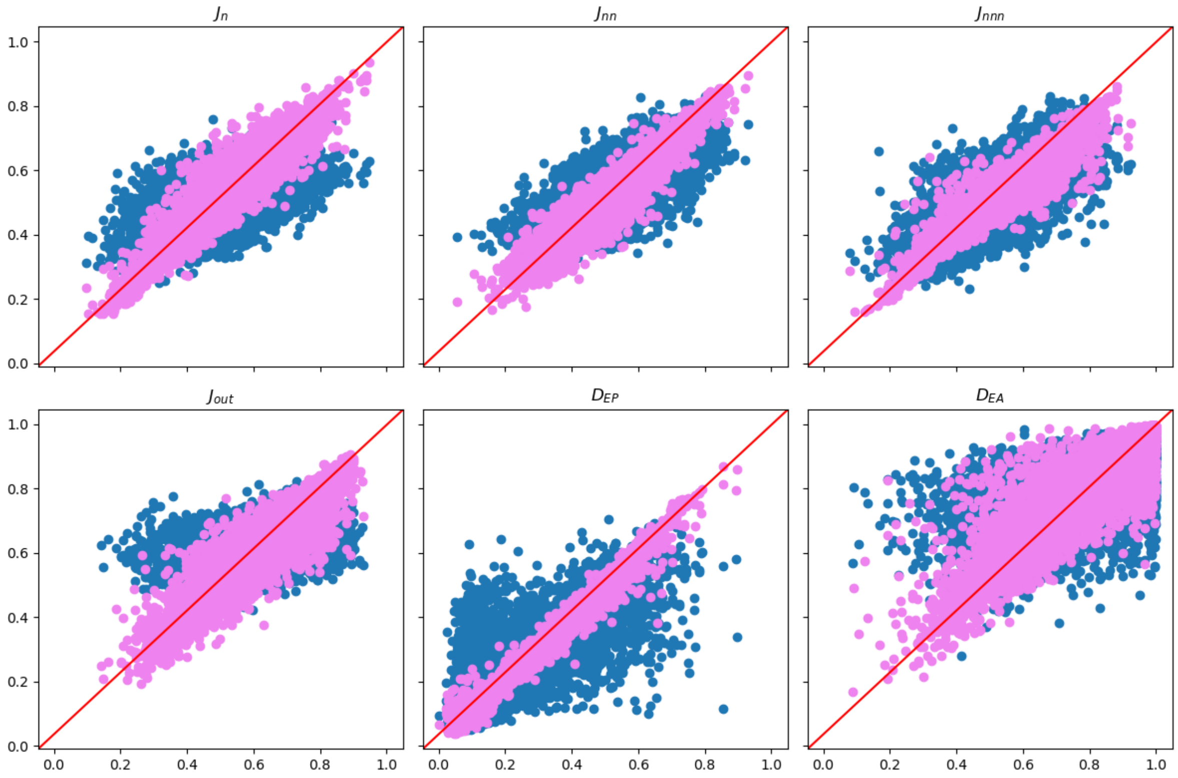


Figure 36: Scatter Plot of bNN-64 (blue) and OCNN-128 (pink) for each parameter. The Red line indicates an optimal fit, X-axis = Label, Y-axis = Prediction

7 Conclusion

In this bachelor thesis we have optimized a Neural Network approach that predicts the parameters of the Hamiltonian from simulated and distorted data of the magnetical compound $BaNi_2V_2O_8$. In order to achieve this goal we have conducted different experiments such as trying different Pre-Processing approaches or the use of Batch Normalization and Dropout. The first part of this bachelor thesis focused on optimizing the FC-NN approach that the baseline Neural Network used. With the use of an optimized Pre-Processing approach, Batch Normalization, Dropout and Hyperparameter Optimization we were able to improve the baseline approach significantly and we were able to resolve problems such as the Neural Network predicting values around the mean for J_{out}. However using a FC-NN approach we were not fully able to improve the unconfident prediction of D_{EP} and especially D_{EA}. This motivated the use of CNNs as they have achieved state-of-the-art performances on image related tasks. Using a simple unoptimized CNN we were able to greatly outperform even our optimized FC-NN and this in turn motivated the further investigation of CNNs. Following the CNN approach we proposed an optimized CNN that has a ∼5.4 times lower MSE on the test data than the baseline approach. The biggest improvements we have seen however are for the parameter D_{EP} which had unconfident predictions for the baseline approach. Using our optimized CNN we were able to achieve a ∼23 times lower MSE on the test data compared to the baseline approach making the prediction of D_{EP} by far the the best out of all parameters for our optimized CNN.

When considering these result we can answer our Research Question and say that Neural Networks are indeed able to perform a reverse transformation of Spin-Wave-Theory using distorted simulated data. We justify this statement with the performance of our optimized CNN which has small MSE values for each parameter. In addition, when looking at the predictions, the optimized CNN has learned to confidently predict each parameter from the input images, although the performance of D_{EA} could be improved even further. We suspect however that the comparatively bad performance of D_{EA} is caused by the input data. It is possible that the simulations simply do not contain enough information regarding D_{EA} and the prediction is made even more difficult by the added artifacts. Further research regarding this problem could lead to a better understanding.

To show the generalization performance of each Neural Network and to compare them, we used a test set which was not used during training or Hyperparameter Optimization. This ensured that the different models are tested on unseen data. It is however possible that the test data is not representative of the true generalization of our models. One potential problem could be that the test set was to small as it only contained about 10% of the whole data set. We wanted to train the models on as much data as possible which was the reason for the comparatively small test set. When conducting further research regarding the problem adressed in this thesis more data could be simulated to obtain a bigger test set.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

[3] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10(978):3, 2018.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[5] RI Bewley, RS Eccleston, KA McEwen, SM Hayden, MT Dove, SM Bennington, JR Treadgold, and RLS Coleman. Merlin, a new high count rate spectrometer at isis. *Physica B: Condensed Matter*, 385:1029–1031, 2006.

[6] François Chollet et al. Keras. `https://keras.io`, 2015.

[7] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.

[8] Antonia Creswell, Kai Arulkumaran, and Anil A Bharath. On denoising autoencoders trained to minimise binary cross-entropy. *arXiv preprint arXiv:1708.08487*.

[9] Ashraf Darwish, Aboul Ella Hassanien, and Swagatam Das. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial intelligence review*, 53:1767–1812, 2020.

[10] Mathieu Doucet, Anjana M Samarakoon, Changwoo Do, William T Heller, Richard Archibald, D Alan Tennant, Thomas Proffen, and Garrett E Granroth. Machine learning for neutron scattering at ornl. *Machine Learning: Science and Technology*, 2(2):023001, 2020.

[11] Cristina Garcia-Cardona, Ramakrishnan Kannan, Travis Johnston, Thomas Proffen, Katharine Page, and Sudip K Seal. Learning to predict material structure from neutron scattering data. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4490–4497. IEEE, 2019.

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

[13] Tony Hey, Keith Butler, Sam Jackson, and Jeyarajan Thiyagalingam. Machine learning and big scientific data. *Philosophical Transactions of the Royal Society A*, 378(2166):20190054, 2020.

[14] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

[15] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[17] TA Kaplan and N Menyuk. Spin ordering in three-dimensional crystals with strong competing exchange interactions. *Philosophical Magazine*, 87(25):3711–3785, 2007.

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

[19] ES Klyushina, B Lake, ATMN Islam, JT Park, A Schneidewind, T Guidi, EA Goremychkin, B Klemke, and Martin Månsson. Investigation of the spin-1 honeycomb antiferromagnet $bani_2v_2o_8$ with easy-plane anisotropy. *Physical Review B*, 96(21):214428, 2017.

[20] Mark Könnecke, Frederick A Akeroyd, Herbert J Bernstein, Aaron S Brewster, Stuart I Campbell, Björn Clausen, Stephen Cottrell, Jens Uwe Hoffmann, Pete R Jemian, David Männicke, et al. The nexus data format. *Journal of applied crystallography*, 48(1):301–305, 2015.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[22] Miroslav Kubat and JA Kubat. *An introduction to machine learning*, volume 2. Springer, 2017.

[23] P Lacorre and J Pannetier. Mcmag: A computer program to simulate magnetic structures. *Journal of magnetism and magnetic materials*, 71(1):63–82, 1987.

[24] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[25] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[26] C-H Liu, Yunzhe Tao, Daniel Hsu, Qiang Du, and Simon JL Billinge. Using a machine learning approach to determine the space group of a structure from the atomic pair distribution function. *Acta Crystallographica Section A: Foundations and Advances*, 75(4):633–643, 2019.

[27] JM Luttinger and L Tisza. Theory of dipole interaction in crystals. *Physical Review*, 70(11-12):954, 1946.

[28] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[29] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[30] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. `https://github.com/keras-team/keras-tuner`, 2019.

[31] Lutz Prechelt. Early stopping—but when? *Neural networks: tricks of the trade: second edition*, pages 53–67, 2012.

[32] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941.*

[33] N Rogado, Q Huang, JW Lynn, AP Ramirez, D Huse, and RJ Cava. $\mathrm{bani}_2\mathrm{v}_2\mathrm{o}_8$: a two-dimensional honeycomb antiferromagnet. *Physical Review B*, 65(14):144443, 2002.

[34] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1328–1347, 2019.

[35] Anjana M Samarakoon, Kipton Barros, Ying Wai Li, Markus Eisenbach, Qiang Zhang, Feng Ye, V Sharma, ZL Dun, Haidong Zhou, Santiago A Grigera, et al. Machine-learning-assisted insight into spin ice $dy_2ti_2o_7$. *Nature communications*, 11(1):1–9, 2020.

[36] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

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

[38] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[41] S Toth and B Lake. Linear spin wave theory for single-q incommensurate magnetic structures. *Journal of Physics: Condensed Matter*, 27(16):166002, 2015.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den February 20, 2023 Schuster