Friedrich-Alexander-Universität
Faculty of Sciences

Bachelor's Thesis

31. März 2026

Marco Zech

Characterization of Quasicrystals with Neural Networks

Inhaltsverzeichnis

1 Quasicrystals: Phasons

According to [SLS86] and [San15], phasonic displacements are obtained by considering a Landau free energy with the ordering parameter being the mass densitiy of the crystal:

$$F[\rho] = \int \sum_{k=0}^{\infty} B_k \rho(x)^k dA, \tag{1}$$

where the integral is taken over the surface the crystal extends over. Here, we expand the mass density as a fourier series

$$\rho(x) = \sum_{G \epsilon L_R} \rho_G \exp(iG \cdot x), \tag{2}$$

where L_R is the reciprocal lattice of the crystal. Since ρ is real valued, we can see from a complex conjugation that for the fourier coefficients the relation $\rho_G = \rho_{-G}$ holds.

Plugging this extension into equation (1) leads us to the following formula:

$$\begin{aligned} F[\rho] &= \int \sum_{k=0}^{\infty} B_k \sum j_1, \ldots, j_k \prod_{l=1}^{k} \exp(iG_{j_l} \cdot x) \\ &= A \sum_{k=0}^{\infty} B_k \sum_{j_1,\ldots,j_k} \delta\left(\sum_{l=1}^{k} G_{j_l}\right) \prod_{l=1}^{k} |\rho_{G_{j_l}}| \exp(i \sum_{l=1}^{k} \phi_{G_{j_l}}), \end{aligned}$$

where $\phi_{G_{j_l}}$ is the phase of the complex number $\rho_{G_{j_l}}$. Since the calculation of the free energy leads to an fourier transform of a constant function, only specific combinations of reciprocal vectors contribute to the result. Note that, in addition, the free energy is a function of only the sum of the phases $\sum_{l=1}^{k} \phi_{G_{j_l}}$.

We inspect the fourier transform of the mass densitiy more closely: According to [SLS86], only a finite subset

$$U = \{G_j; j = 1, \ldots, K\} \subset L_R,$$

that generates the reciprocal lattice/lattice module is required in equation (2). In order to generate a real valued mass densitiy, U has to be symmetric:

$$-U = U.$$

In periodic crystals, we have $K = d$ with d being the spatial dimension of the crystal. In quasicrystals we have $K = N \cdot d$, where n is the number of incommensurate lengths associated with each lattice vector direction. Often, N is called the **rotational symmetry** of the crystal, and the vectors in U are often chosen to span a

regular N-polyhedron around the origin.
Example: For pentagonal crystals, which we will consider in this work, U is chosen to be

$$U = \{(\cos(\frac{2\pi n}{5}, \sin(\frac{2\pi n}{5}); n = 0, \dots, 4\}.$$

It is clear that the free energy depends on the sum of the phases ϕ_{G_n}, therefore the minimization of F fixes also the sum to a constant value:

$$\sum_{G \in U} \phi_G = c = const.$$

For the pentagonal case, we can conclude that four phases can be chosen independently. These degrees of freedom are parametrized by two vectors $u, v \in \mathbb{R}^2$. The formula for the phase ϕ_n, associated to the vector $G_n \in U$ is

$$\phi_n = G_n \cdot u + aG_{3n \bmod 5} \cdot w.$$

If the changes of u and w are chosen as vector fields, they produce special dislocations:

Spatially dependend shifts of u can be associated to the propagation of phonons in the quasicrystal, they are the result the broken euclidian symmetry of the quasicrystal phase. Spatially dependend shifts of w are called **phasonic displacements**. In the superspace description of quasicrystals, they can be interpreted as additional modes that arise since a quasicrystal phase has additional broken symmetries in the internal space. Phasonic modes can lead to rearrangements of atoms in a quasicrystal. See [**?**]janssenaperiodiccrystals), chapter 6 and [Kos01] for further details.

In the following, we will focus on **phasonic dislocations** that can be achieved by defining the vector field w to be

$$w_1(x, y) = \exp(-\frac{(x - \mu_x)^2 + (y - \mu_y)^2)}{2\sigma}) \text{ and } w_y(x, y) = 0. \tag{3}$$

The base for displaying mass densities of quasicrystals and for generating training datasets is the so called **laser potential**, which is a potential created by interfering laser beams. The potential reads[SS12]

$$V(\boldsymbol{r}) = -\frac{V_0}{N^2} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} \cos\left((G_j - G_k) \cdot \boldsymbol{r}\right),$$

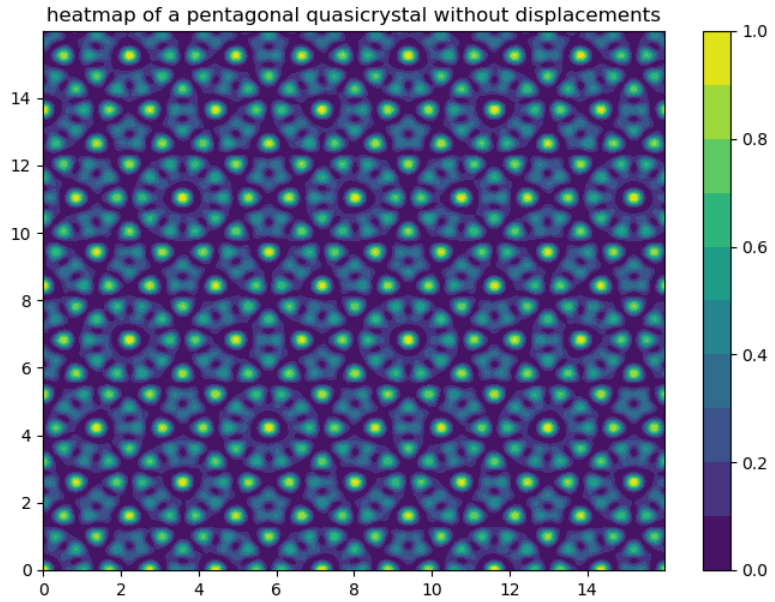
with N being the number of lasers (and the nummeracy of the crystal) and V_0 a proportionality constant, which was chosen as $V_0 = -1$ to convert the potential into a mass density. The vectors

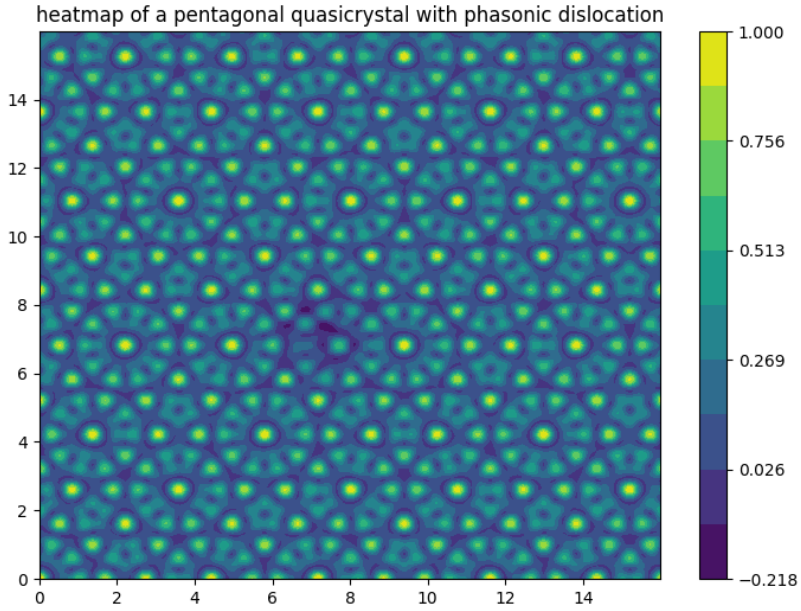
$$G_j = (\cos(\frac{2\pi j}{N}, \sin(\frac{2\pi j}{N})$$

are the generators of the lattice module. If a thin colloid film is iluminated by these laser beams, the dispersed particles will be trapped within the local minima of this potential - therefore this function is a solid theoretical model for the mass density of quasicrystals in two dimensions.

In the following, we will also restrict ourselves to pentagonal quasicrystals with $N = 5$.

The influence of this particular choice for the displacement field w is shown in the following figures:





The second image shows the same surface, but with a phasonic dislocation as in equation (3), which is centered at $\mu_x = 7.25$ and $\mu_y = 7.1$. Here one can observe that the peak at $(6.6, 6.6)$ was shifted to $(7.4, 7, 7)$. In addition, a new symmetric cell emerged arround the peak at $(7.7, 6.7)$.

The following part of this thesis tacles the question wether these dislocations can be detected automatically, based on a graph representation of such quasicrystals, and in particular which geometrical properties of such a dislocation will be relevant.

2 Neural Networks

As [Dö19] pointed out in his preceeding work, the recocgnition of displacements within a quasicrystal is a task that can be hardly detected by using human vision, and this would only be a heuristic approach. In addition, there is not a clear path how a classical algorithm performed by a computer should look like. A promising solution might be the application of neural networks to „interpolate“ a recocgnition algorithm unsing example data.

This section aims to explain some basic deep learning principles. The mathematical part follows [Cal20] while everything related to algorithms comes from [GBC16].

2.1 Definition and Basic Properties

In their most basic form, a singe layer feed forward neural network is a map of the form

$$F : \mathbb{R}^k \to \mathbb{R}^l, \tag{4}$$

$$x \mapsto \sigma^*(Wx + b) \tag{5}$$

with $W \in \mathbb{R}^{l\times k}, b \in \mathbb{R}^l$. The matrix W is called **weight matrix** and the vector b is called **bias**. The function

$$\sigma : \mathbb{R} \to \mathbb{R}$$

is called **activation function** - the superscript * indicates that σ is applied to every component of the input vector:

$$\sigma^*(x_1, \dots, x_l) := (\sigma(x_1), \dots, \sigma(x_l)).$$

In the following, this superscript will be omitted. The weights and the bias are free parameters in a regression model - they are chosen to approximate a given function. This function can be a „mathematical function“, or a function that represents a computer algorithm. The process of determing suitable values for the weight matrix and the bias vector given a set of input data $\{x^j; j \in J\}$ and the corresponding output $\{z^j := F(x^j); j \in J\}$ is called **training**.

If the activation function is linear, a single layer neural net is just a linear regression model and has limited capacities of approximating a complex function (or a complex algorithm, respectivly). For nonlinear activation functions there are different „universal approximation theorems“ ensuring that, in principle, neural networks can be trained to adapt many different types of functions:

Definition: Let μ be a (borel) measure on the unit cube $I_n = [0,1]^n$. A measurable function $\sigma : \mathbb{R} \to \mathbb{R}$ is called **discriminatory** for the measure μ if

$$\int_{I_n} \sigma(w \cdot x + b) d\mu(x) = 0$$

for all $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ implies

$$\mu = 0.$$

One can show that so called **sigmoidal functions** are discriminatory for all borel measures on I_n. A function σ is called **sigmoidal** if

$$\lim_{x \to -\infty} \sigma(x) = 0 \text{ and } \lim_{x \to \infty} \sigma(x) = 1.$$

If σ is chosen to be sigmoidal for the lebesgue measure λ^n, we have two strong approximation results:

Theorem 2.1. *1. Let σ be a discriminatory function. Then single layer neural networks of the form*

$$F(x) = \sigma(w \cdot x + b), \ w \in \mathbb{R}^n, b \in \mathbb{R}$$

are dense in the space of continuos functions on the unit cube, i.e. for any continuos function $f : I_n \to \mathbb{R}$ and a number $\varepsilon > 0$, there exists a number $N \in \mathbb{N}$ and, for each $j \in \{1, \dots, N\}$, paramters $a_j, b_j \in \mathbb{R}$ and $w_j \in \mathbb{R}^n$ such that

$$\max_{x \in I_n} \left| f(x) - \sum_{j=1}^{N} a_j \sigma(w_j \cdot x + b_j) \right| < \varepsilon.$$

2. Let σ be a discriminatory function. Then single layer neural networks as given above are dense in the space of square integrable functions on the unit cube I_n.

Both theorems implicitly imply that the choice of a **loss criterion** is also an important step in designing a deep learning architecture. In addition, the coice of a loss function depends on the task the neural network has to perform:

- If the task is to approximate a continuous or a square integrable function, the universal approximation theorems imply that the loss criterion needs to be a suitable distance on the image set $\mathbb{R}$.
- If the task is to categorize an object based on some of its properties (these properties are the input values), then the loss criterion is usually chosen to

be the so called **cross entropy loss**: The implementation of Pytorch is given as

$$L(x, y) = -\frac{1}{N}\sum_{c=1}^{l} w_c \log\left(\frac{\exp(y_c)}{\sum_{i=1}^{C}\exp(y_i)} z_c\right)$$

with $y = F(x)$ being the output of the neural net and $z = f(x)$ the output of the function/the algorithm one wants to learn.

Using the cross entropy loss is equivalent to a maximum liklihood estimation.

2.2 Training Procedure

As mentioned before, the term **training** refers to the process of finding suitable values for the free parameters of a neural net F such that it is an adequate approximation of a given function/a given algorithm f. The procedure is illustrated with a single layer neural net

$$F(x) = \sigma(Wx + b)$$

with free parameters W and b.

Given a training set

$$\{(x_j, z_j); j \in J\},$$

where each x_j denotes the input, $z_j = f(x_j)$ the corresponding value of the function f at x_j, and $y_j = F(x_j)$ the output of the neural network F, one aims to minimize the cost function

$$C(W, b) := L(z_j, y_j) = L(z_i, \sigma(Wx_j + b)).$$

for some $j \in J$, using an optimazion method.

After the gradients $\frac{\partial C}{\partial W}$ and $\frac{\partial C}{\partial b}$ are calculated, they are updated according to the following formuals:

$$W' = W - \alpha\frac{\partial C}{\partial W},$$

and

$$b' = b - \alpha\frac{\partial C}{\partial b},$$

where α is the **learning rate**.

The basic steps of the updating algorithm are as follows:

1. For each $j \in J$:

- Calculate
$$\frac{\partial C}{\partial W} \text{ and } \frac{\partial C}{\partial b}.$$

2. Update the current parameters W and b to W' and b'.

For the optimization method, there are two standard examples:

1. **Stochastic gradient descent**, which is an exttension of the gradient descent method to find minima of differentiable functions, see e.g. [GBC16], chapter 4.3. While the standard gradient descent calculates directions of descent using the whole input/uses the whole input to calculate updated weights, stochastic gradient uses only parts of the data for the calculation - in the case of deep learning one updates the model parameters after each training input, or after a small subset of training inputs[1]. See [GBC16], chapter 5.9.

2. **Adam** (short for „adaptive moments“) is a relatively new extension of the stochastic gradient descent. See [GBC16], chapter 8.5.3 for detai.s. Since its performance is not too dependent on properties of a model, it is in general the preferable choice.

There are different strategies to updating the parameters: One can update them for each tuple of the training set, or one can group the training set into **batches**.

2.3 Hidden Layers and Backpropagation

Since single layer feed forward neural networks often lack performance in adapting to some tasks, one usually composes multiple single layer functions, one composes multiple functions as in equation (4). This allows to efficiently learn functions „exponentially more efficient“ in the sense that a model with one layer may require $\approx e^n$ parameters to achieve similar results where a model composed from multiple single layer neural networks would only need n parameters. Since the desired output of the layer $F^{(2)}$ of a complex neural network

$$F = F^{(1)} \circ F^{(2)} \circ F^{(3)}$$

is not directly determined by the training dataset, one calls these **hidden layers**.

Updating the weights of a general feed forward neural network effectively follows from the chain rule: Let $i = 1, 2$ and define $F^{(i)}$ to be the single layer function

$$F^{(i)} = \sigma_i(W_i x).$$

[1] Splitting the training set into smaller subsets and updating after the evaluation of such a subset is callem **batching**.

We consider the network

$$F(x) = F^{(1)}(F^{(2)}(x)).$$

For simplicity, our training set is $\{(x, z)\}$, consisting of only one element. If we want to update the parameters W_1 and W_2, we would need to calculate the gradient $\frac{\partial C}{\partial W_i}$. The chain rule then implies

$$\frac{\partial C}{\partial W_i} = \frac{\partial L(z, y)}{\partial y} \cdot \frac{\partial y}{\partial W_i},$$

where $y = F(x)$. So the composition of functions leads to the multiplication of gradients. This is called **Back-Propagation**.

2.4 Graph Neural Networks, Message Passing

The thesis [Dö19] did address the question wether phononic and phasionic impurities of quasicrystals were learnable. In order to determine, how phasonic displacements influence the geometry of a quasicrystal, one can use a similair approach, but compress their mass densities such that simple geometric properties remain. Since physical properties of a (quasi-)crystal are also encoded in the interactons of its atoms, it is natural to take these interactions into consideration for the task of detecting phasonic displacements. The natural framework for this is the framework of **graph-neural-networks**. This section folloows [SLRPW21] and [Vol25].

Graph neural networks are neural networks

$$F : M \to E,$$

that approximate functions/algorithms

$$f : M \to E$$

which are defined on a set of graphs M. The target space can vary for different tasks one wants to perform - these types of tasks are split into different groups:

1. The simpliest kind of tasks are **node-level-tasks**. Here the function f one aims to approximate has the property

   $$f((V, E)) \in \mathbb{R}^{|V|},$$

   i.e. one wants to assign some value to each node of a graph. This is the simpliest form since one has to construct effectively a neural network that accepts for each node some input and performs some computation.

2. **Edge-Level-Tasks** have the property that

$$f((V,E)) \in \mathbb{R}^{|E|},$$

so one wants to assign some value to each edge of a graph.

3. **Graph-Level-Tasks** assign a value to the entire graph:

$$f(G) \in \mathbb{R}.$$

The type of task one wants to perform strongly determines the shape of the neural network F.

In the next chapters we will try to construct a neural network that predicts wether an atom in a quasicrystal is subject to a phasonic dislocation - we therefore want to approximate a function f with

$$f((V,E)) \in \{0,1\}^{|V|}.$$

If we abbreviate numeric properties of the i-th node with $v_i \in V$, then on a node level our function f is

$$v_i \mapsto \begin{cases} 1, & \text{if the atom } i \text{ was subject to a phasonic dislocation,} \\ 0, & \text{otherwise.} \end{cases}$$

Therefore this is a node-level task. We will

To utilize the inherent structure of a graph, **message passing layers** were introduced - These are linear layers that operate as follows: Let $v \in V$ be a node of the graph G[2], and let $N(v)$ be the set of neigbours of v in G. Then we define a **message function** M and a message m as

$$m_v = \sum_{w \in N(v)} M(w)$$

and the output of the message passing layer as

$$y_v = U(v, m_v),$$

where U is the **update function**. Often M is chosen to be the identity and U is the sum over all „messages“ a node recieves from its neigbhbours - then this layer in matrix notation takes the simple form

$$Y = (\mathbb{1} + A)\mathcal{V},$$

[2]As in the paragraph before, we will silently identify v with its features that are relevant for the task.

where $\mathcal{V}$ is the set of node features, arranged as a matrix, A is the adjacency matrix of the input graph and Y is the set of layer outputs of the graph, again arranged as a matrix.

This is the basic idea of **graph convolutional layers**: These combine a simiair message-passing approach with learnable layers, see [KW17]. Advantages of using these layers in graph neural networks are that every node prediction is only influenced by its direct neighbors - if the property of a node influences feature values of other nodes, these then will also be considered.

3 Graph Representations of Quasicrystals

In the following, we aim to create a deep learning model to detect phasonic dislocations within a given quasicrystal structure. A prior thesis did a similair project based on surfaces created from the potential based on tensors representing the mass density of such a crystal.

In that thesis, similair dislocations were injected in a quasicrystal, and a neural net model was trained on tensors

For this thesis, the plan is to compress these tensors to a graph that represents the atoms of a quasicrystal, together with their interactions. These graphs are used to train a graph-neural-network model in order to evaluate, wether a given atom was subject of a phasonic displacement. The attempt of trying this with a graph representing the atoms and their interactions can lead to insights which parameters are actually essential for the determination wether a phasonic displacement has ocurred or wether a given displacement is phasonic.

3.1 Creating Training Data

The starting point is the laser potential discussed in section 1.

The training data for the model consists of graphs that are generated from square surfaces with a side length varying from $(7a_v)^2$to $(17a_v)^2$. The starting point for this procedure was to create a surface/heatmap of the laser potential over these squares S:

$$Z = V(S).$$

Since the atoms are supposed to be at local extremas of the potential V, a function was called to locate the coordinates of these extreme points. Every extreme point with its value below 0.75 was discarded. These pictures are already shown in section 1.

For these points, a Delaunay triangulation was performed to identify interaction partners for each atom: Here, one connects a set of given points in such a way that each point forms a triangle with two of its neighbours. These connections are chosen such that a circle enclosing a triangle does not contain another point.

In this way, each surface is converted into a NetworkX graph object where the nodes represent atoms of a quasicrystal, and edges forces between them. NetworkX was chosen since graph objects have methods which are intuiviely usable.

At first, a graph without dislocations was created. After that, graphs with random dislocations were generated and discarded if the node number did not equal the node number of the graph without dislocations[3].

[3]This occurs if the position of the dislocation leads to effects similair to destructive interference.

Following these discussions, a total amount of 28149 graphs were created. The number of nodes ranges from 28 to 177. Each graph carries a phasonic dislocation as in equation (3), where the value of μ_x and μ_y was chosen ranomly within the initial square S.

Here one can chose which features a neural net model uses to deduct something about them. We will talk about this in detail in the next subsection.

3.2 Extracting Relevant Features

In order to find relevant features that could be influenced by the fact that a given node was displaced, a set of training graphs with and without displacements was created. Since a displacement of an atom should break symmetry patterns, one could expect that these influence the distances between an dislocated atom and its (dislocated as well as non dislocated) neighbours, so these distances might vary more in the dislocated case than in the located case.

Therefore we will work with node features that estimate properties of the distribution of the distance each node i has to its neighbours $N(i)$: these are

- The smallest distance:
$$h_1^{(i)} = \min_{j \in N(i)} d_{ij}.$$
- The mean of all distances to neighbours
$$h_2^{(i)} = \frac{1}{|N(i)|} \sum_{j \in N(i)} d_{ij}.$$
- The standard deviation
$$h_3^{(i)} = \sqrt{\sum_{j \in N(i)} (d_(ij) - h_2^{(i)})^2}.$$

In order to evaluate the meaningfulness of these features, random quasicrystals over quadratic surfaces with the same sizes as the training data were created. These do not contain any phasonic displacements. For each of those graphs $G = (V, E)$, the mean of those features was calculated:

$$h_k := \frac{1}{|E|} \sum_{i \in E} h_k^{(i)},$$

and then averaged for all graphs with the same number of nodes. The same evaluation was performed for the graphs in the training dataset. These are shown in the following figures:

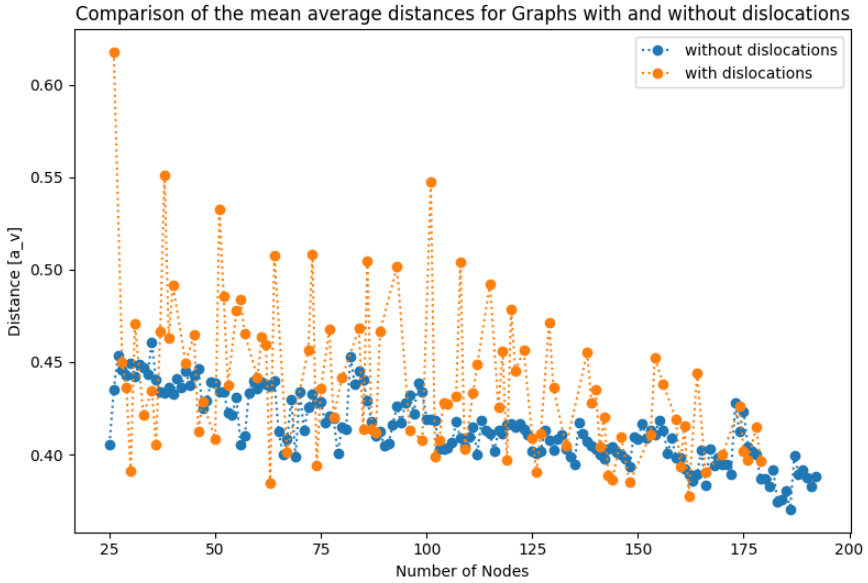


Abbildung 1: Comparison of the mean standard deviation of graphs with and without phasonic displacements, as described in section 3.2. The graphs contiaining dislocations were sourced from the training dataset, the graphs without dislocations were created for this measurement. The latter set contains 8370 graphs.

Here, one can see that these features seem to carry information about the fact wether a node was subject to a phasonic displacement: moving a node leads to a higher variation in the distances to its neighbours as well as to a higher mean of those distances. In addition, the smallest distance to the nearest neighbour seems to be lower. In addition, the standard deviations seem to carry more information than the other features.

We also note that level of those features are indirectly proportional to the number of nodes - this indicates that features of nodes at the boundary of those graphs also have distinct values, and might influence the learning result.

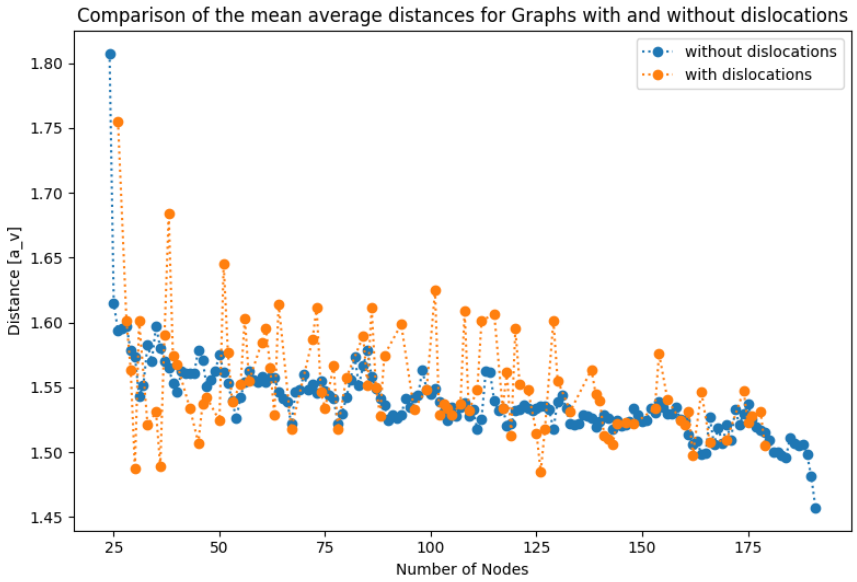


Abbildung 2: Comparison of the mean averaged distance of graphs with and without phasonic displacements, see section 3.2. Here, 8350 graphs were used for this measurement.

4 Deep Learning Models for the Detection of Displacements

4.1 Model Creation and Evaluation

In the following we present some simple GNN-architectures, and how they perform after being trained using the dataset discussed in the previous section. The key building block will be two graph convolutional layers we presented in section to utilize the interactions between a particle and its neighbors.

The first rough step towards an evaluation was the inspection of the training and validation loss curves, as well as the accuracy curves on the training and test datasets.

All models used the Cross-Entropy-Loss shortly discussed in the last section. In the torch implementation, an activation with a softmax function is included. These blocks will be omitted later.

The learning rate for the optimization algorithm Adam was chosen to be $\alpha = 0.0001$.

The first model consisted of two GCNConv layers:

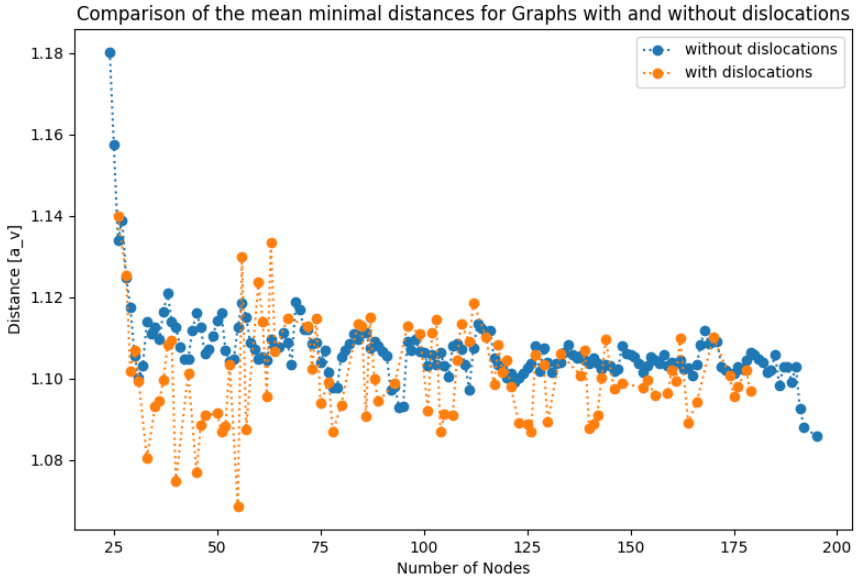


Abbildung 3: Comparison of the mean minimal distance of graphs with and without phasonic displacements, as described in section 3.2. Here, 8442 graphs were used.

Listing 1: Model 1

```
GCNConv(in=3,out=5) -> ReLU -> GCNConv(5,2)
(-> Softmax-> CrossEntropyLoss)
```

In order to proactively prevent overfitting, a dropout layer was included. This layer outputs for every input. The corresopnding loss and accuracy curves are shown in figures 4 and 5.

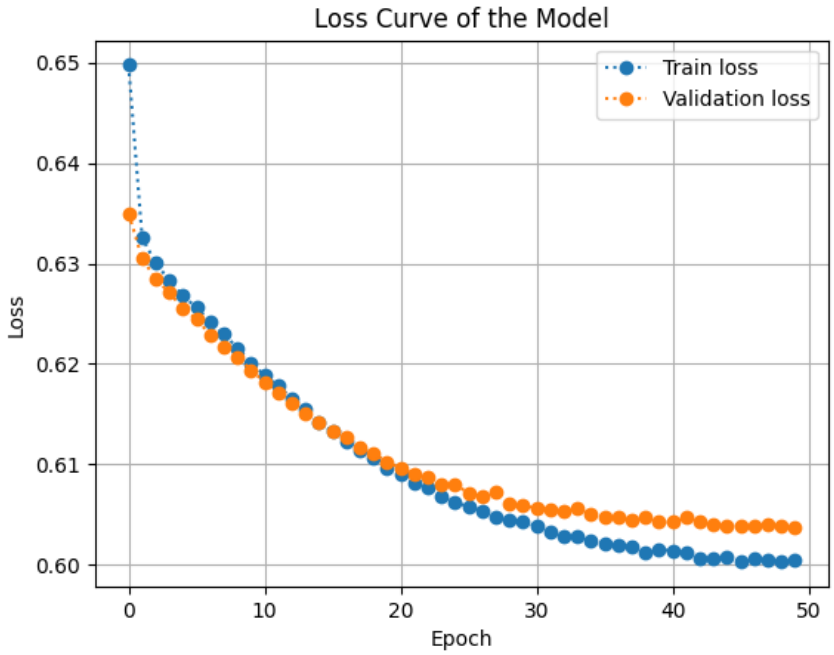


Abbildung 4

The model seems to learn some relations between the node features and their labels, but both loss curves converge to a value that is just slightliy lower that $0.69 \approx \log(2)$. This indicates that the model performs slightly better than random guessing. Since the capacity of this model is relatively low, it is likely that the number of parameters is too low to efficiently generalize the training dataset. Note that the accuracy curves start from a high value. This is not surprising and comes from the fact that the ratio of nodes labeled as being displaced and all nodes is

$$\frac{116362}{2460128} \approx 0.0473,$$

so a model already has a high accuracy of 95.27% if it predicts every node to be not displaced. A desired accuracy curve would achieve a high accuracy later, and not at the start of the training process, so this metric is slightly misleading here.

The next model keeps the structure of the previous one, but increases its capacity:

Listing 2: Model 2

```
GCNConv(3,25) -> ReLU-> GCNConv(25,2)
 (-> Softmax-> CrossEntropyLoss).
```

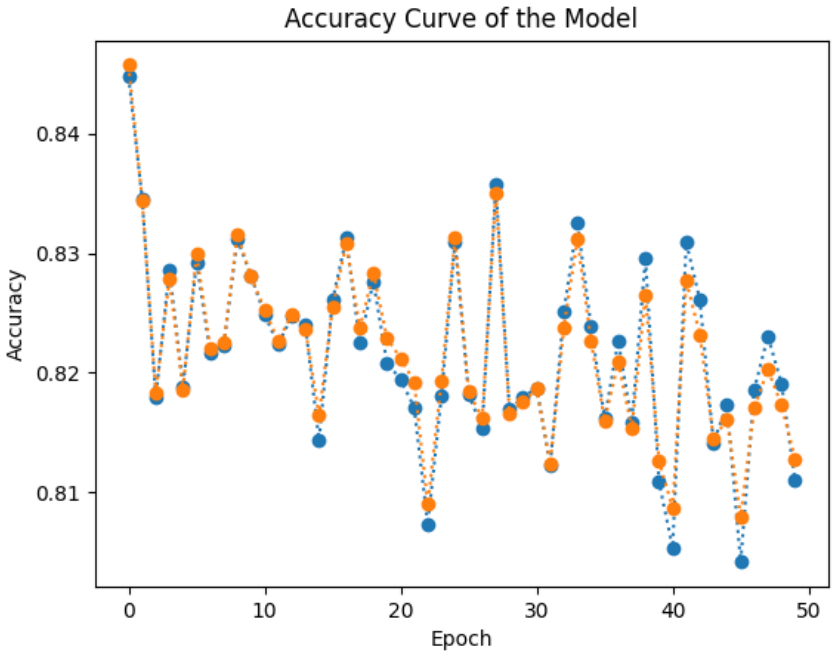


Abbildung 5

Its loss and accuracy are shown below in figure 6 and 7. The increased number of parameters leads to a slightly better performance, but the model still shows signs of underfitting.

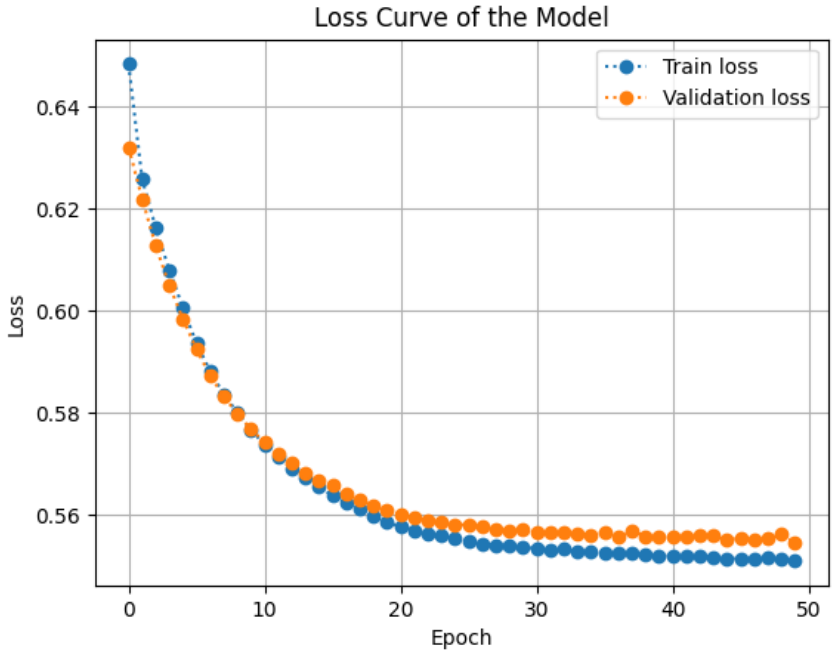

Loss Curve of the Model
Train loss
Validation loss
0.64
0.62
0.60
0.58
0.56
Loss
0
10
20
30
40
50
Epoch

Abbildung 6

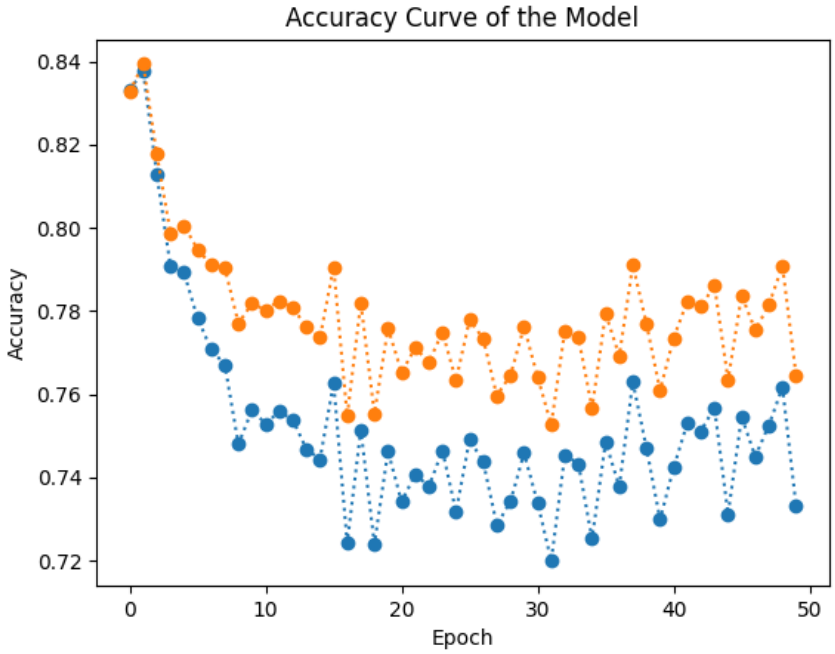

Accuracy Curve of the Model
Accuracy
Epoch
0.84
0.82
0.80
0.78
0.76
0.74
0.72
0
10
20
30
40
50

Abbildung 7

The next step is to add an intermediate linear layer that, for each hidden node, allows their features to be further transformed without the aggregation step. As [UMSM22] states, this can enhance the expressibillity of a model. The structure is

Listing 3: Model 3

```
GCNConv(3,10)->ReLU->Linear(10,10)-> ReLU-> GCNConv(10,2)
```

The loss and accuracy curves are shown in figure 8 and 9.

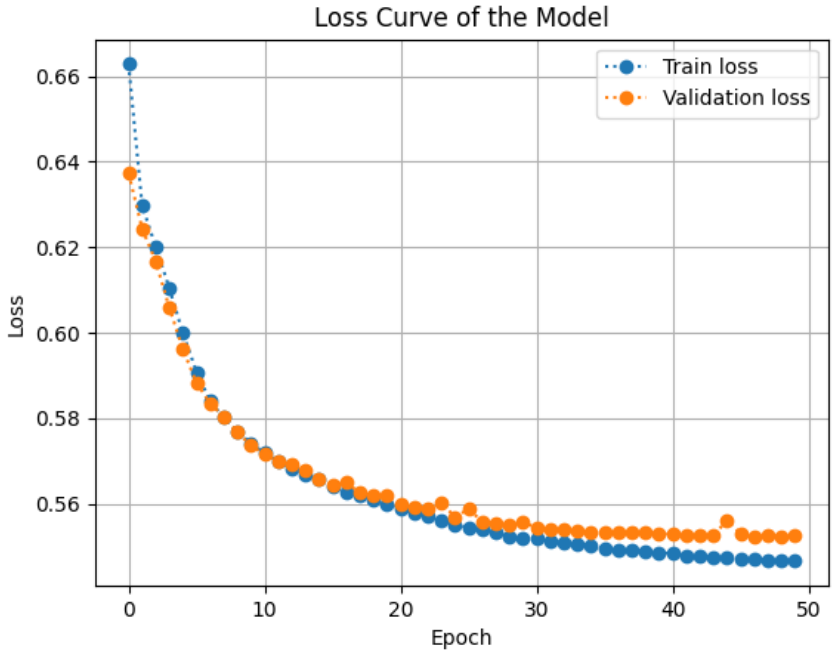


Abbildung 8

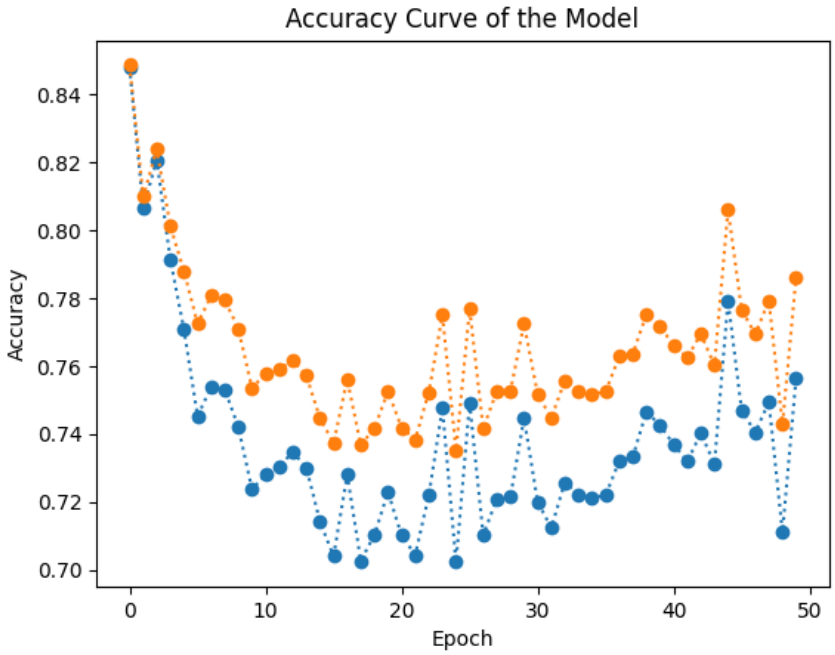


Abbildung 9

Aparreltly, the capacity is still not sufficient, so the number of hidden features will be raised in multiple steps up to 200:

Listing 4: Model 4

```
GCNConv(3,200)->ReLU->Linear(200,200)-> ReLU-> GCNConv
(20,2)
```

These curves are shown in figure 10 and 11. Extending the dimension of the in- and outputs of the hidden layer leads to diminishing loss- and accuracy improvements. This hints that the model's structure is too simple - we therefore add another linear layer, with a reduced number of parameters:

Listing 5: Model 5

```
GCNConv(3,*)->ReLU->Linear(*,*)-> ReLU->Linear(*,5)->ReLU->
GCNConv(5,2),
```

where the omitted hidden dimension * was chosen to be a set of different numbers ranging fom 40 up to 200. For 200, the resulting curves are shown in figure 12 and 13.

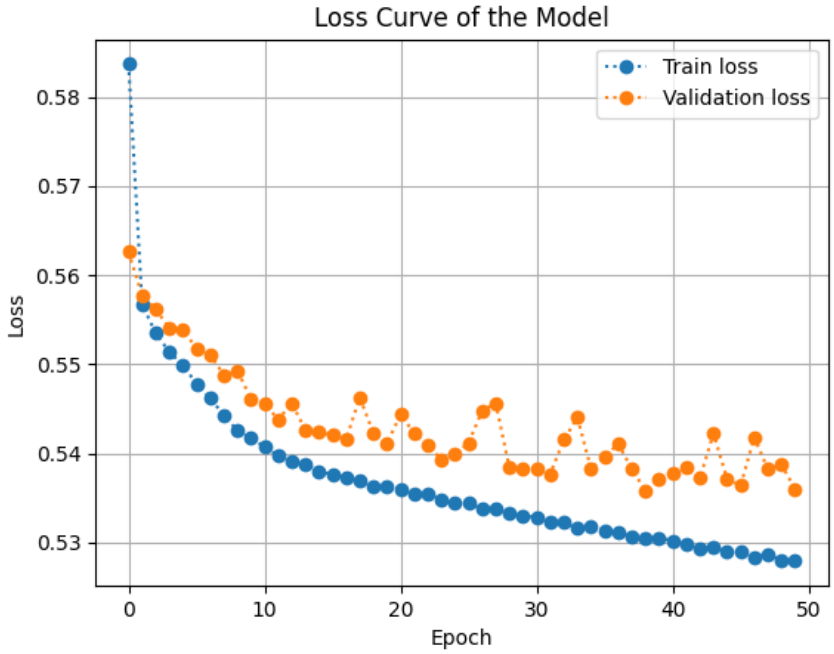


Abbildung 10

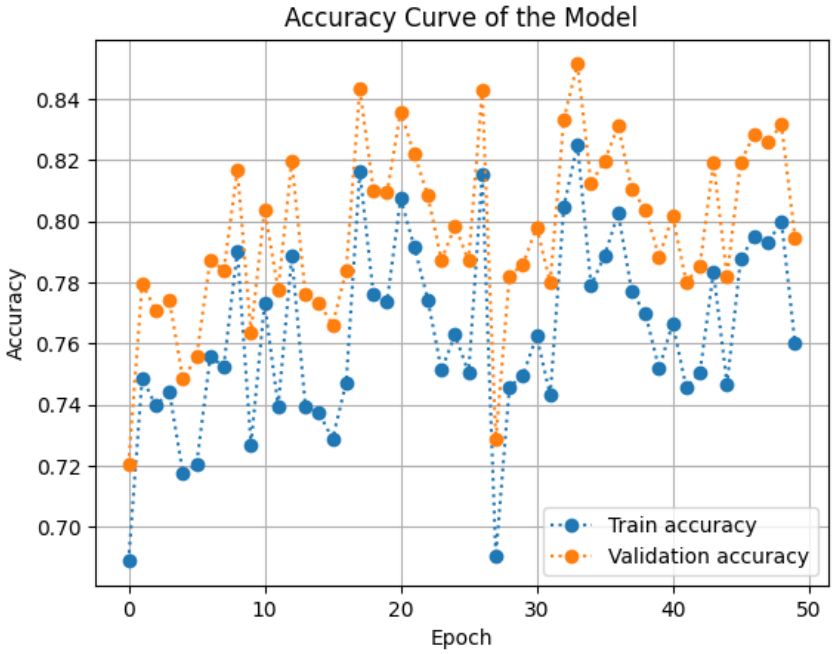


Abbildung 11

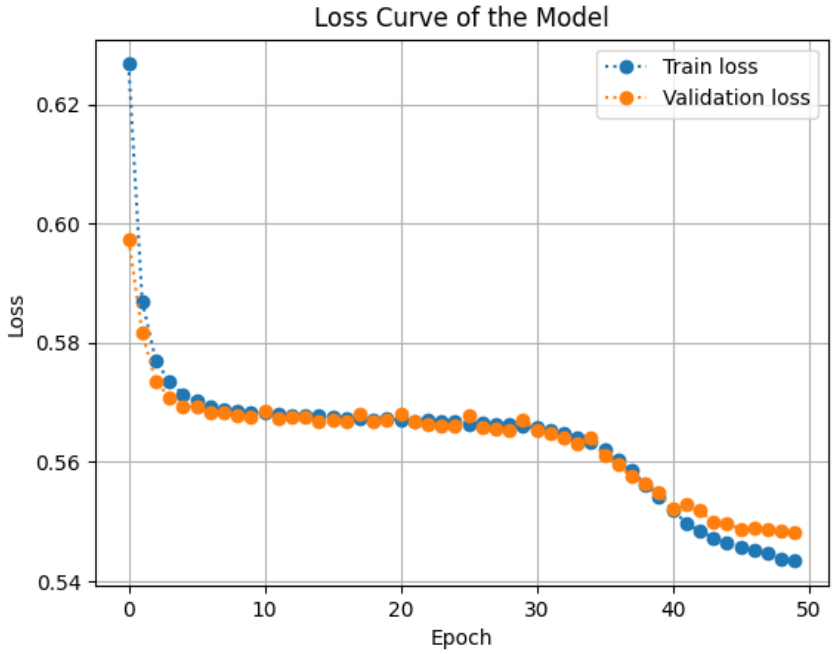


Abbildung 12

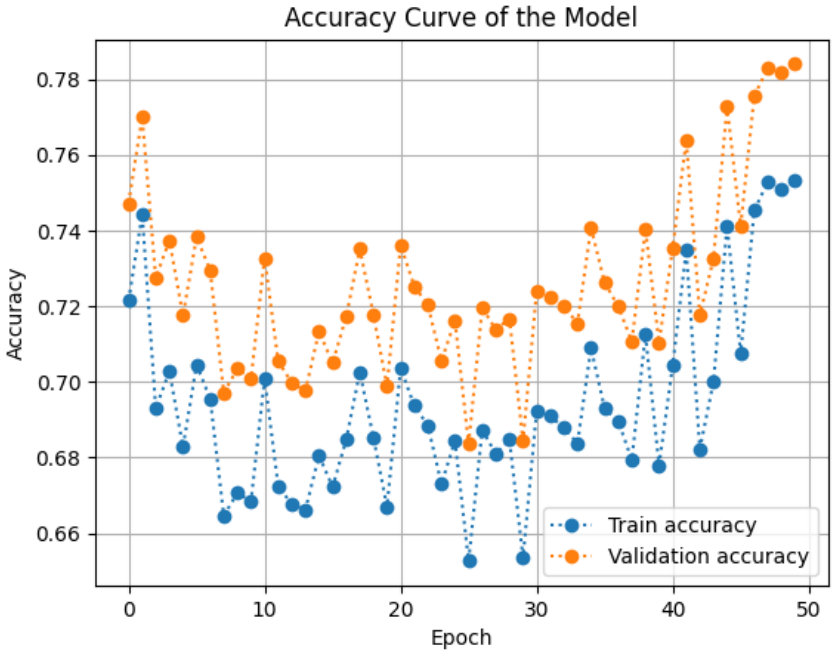


Abbildung 13

After 50 training epochs, each of these models showed a comparable training success with the presented model having the best performance metrics. but even here, the model is not able to improve significanlty over a model/an algorithm, that guesses randomly.

4.2 Analysis of specific Models

As shortly mentioned in the previous section, none of the models reach an accuracy on the training dataset that is remotely close to the accuracy a model outputing $F(v) = 0$ for all nodes would have. This indicates that the training dataset is either not deterministic enough or that some of the not dislocated nodes have features that are similair to the dislocated ones.

In the following, a small test dataset containing 24 graphs was created following the same procedure as the training and validation dataset. The accuracy of identifying a node correctly as displaced or not displaced is 77.9%. In the following, some examples of this test set are presented:

Of those 24 test graphs, at least 22 have the property that the model in 4 wrongly predicts clusters at the boundary of the graoh to be displaced.

Another model that performed decent at training, compared to the other models is that shown in listing 5, with the first hidden dimension equal to 200. Now the accuracy on the test dataset was 77.75%. The same graphs with predictions of that model are shown below:

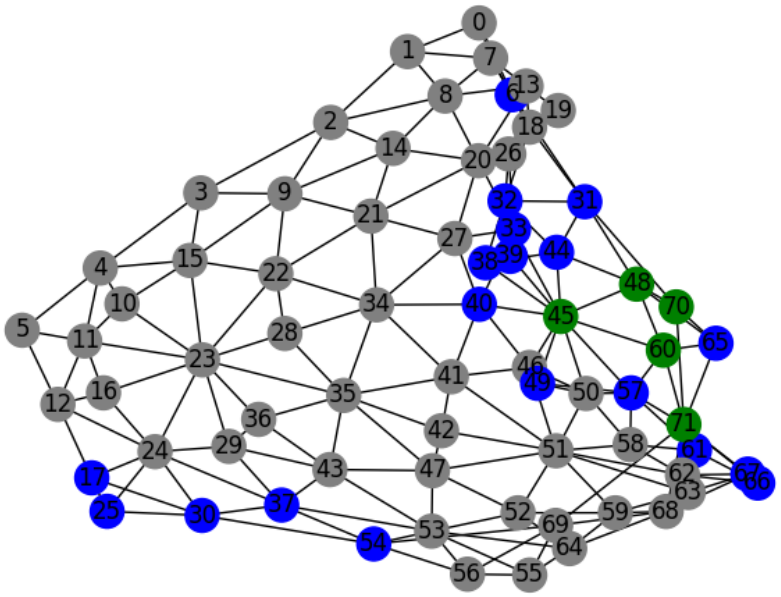


Abbildung 14: A test graph, with node predictions.
Grey: Nodes that were not displaced and correctly recognized as such. Green: Nodes that were displaced and correctly recognized as such. Blue: Nodes that were not displaced but wrongly recognized as such. Yellow: Nodes that were displaced wrongly recognized as not displaced (this did not occur here). The model used to label the nodes is that of 4.

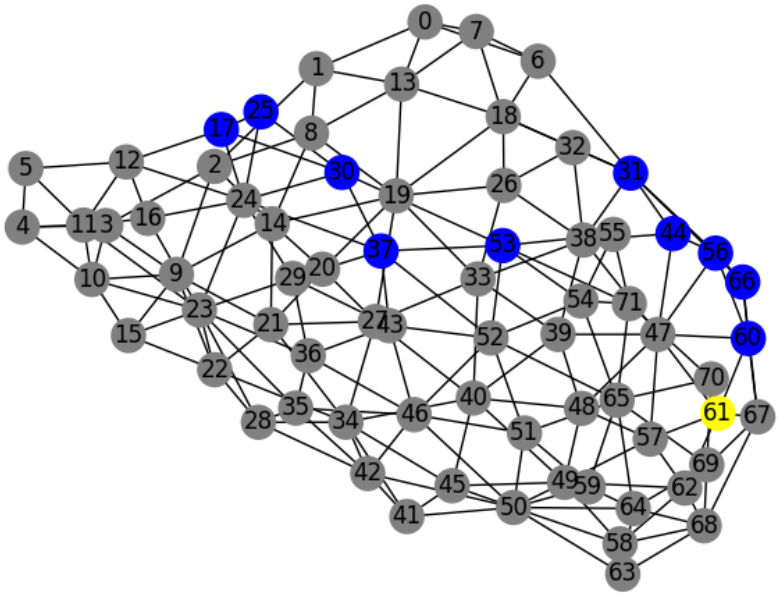


Abbildung 15: A test graph, where the model used to label the nodes is that of 4. For the meaning of the node colors, see figure 14

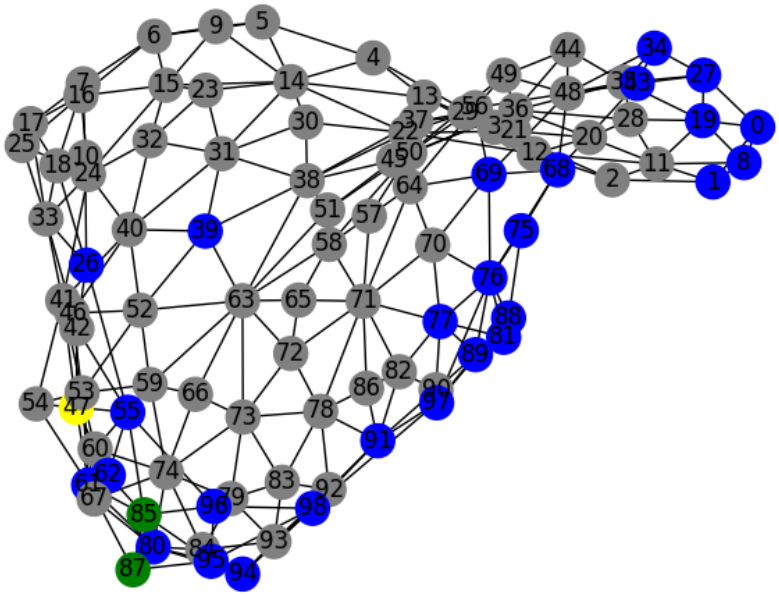


Abbildung 16: A test graph. See 14 for details.

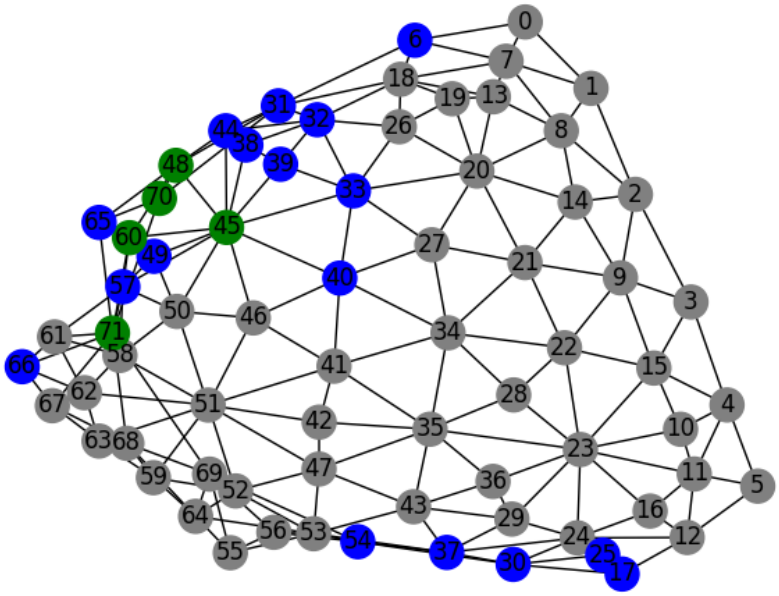


Abbildung 17: The same graph as in figure 14, with the predictions coming from the model 5.

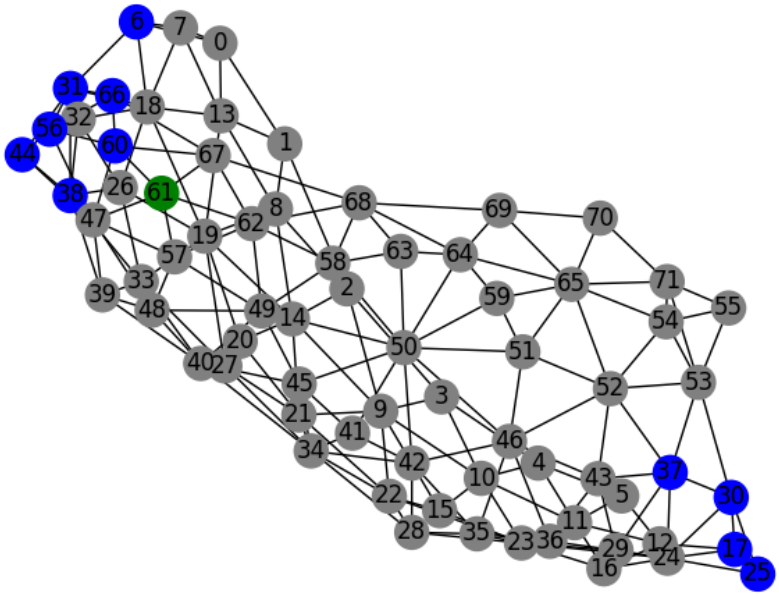


Abbildung 18: The same graph as in figure 15, with the predictions coming from the model 5.

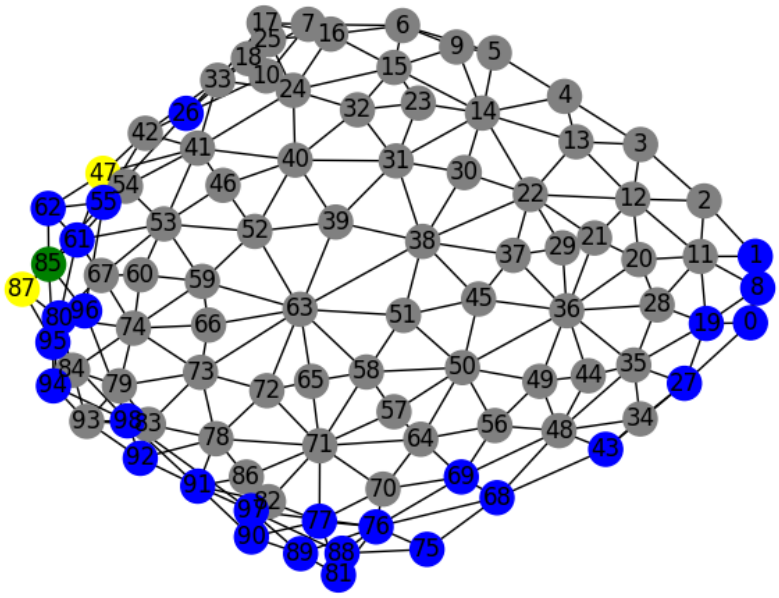


Abbildung 19: The same graph as in figure 16, with the predictions coming from the model 5.

Again, almost all graphs showed clusters of nodes wrongly predicted as displaced at the boundary.

Both example models imply that features of non-displaced nodes at the boundary might be distributed similary, or that node features of displaced nodes do not contain enough relevant information.

We will investigate this claim as follows: For both models and each feature (standard deviation of a node's distance to its neighbours, minimal and mean distance), we will test wether the values of the entire training set, the subset of displaced node feature values and the feature values of the subset of nodes falsly predicted as displaced, are similary distributed.

The test used for this evaluation is the two-sided **Kolmogorov-Smirnov** test[V+26], which evaluates the test statistic

$$D_{n,m} = \sup_{x \in \mathbb{R}} |F_n(x) - G_m(x)|,$$

with F_n and F_m being the empirical distribution functions for the two feature sets. This statistic is a good metric to compare entire distributions since we can estimate the difference of probabillities:

$$\begin{aligned} |P_n((a,b)) - P_m((a,b))| &= |F_n(b) - F_n(a) - F_m(b) + F_m(a)| \\ &\leq |F_n(b) - F_m(b)| + |F_m(a) - |F_n(a)| \leq 2D_{n,m}. \end{aligned}$$

This tells us that probabillities for the feature of the first group being within the interval (a, b) and the feature of the second group being in the same interval will differ by at most two times the statistics value. The null hypothesis is that both feature sets are samples of the same probabillity distribution The p-value states how likely one would observe an outcome for the same experiment that imply stronger differences, assuming the null hypothesis is true.

For the model shown in 4, this is shown below:

	$D_{m,n}$	p
All nodes vs. displaced nodes	0.3223	0.0
All nodes vs. wrongly predicted nodes	0.4944	0.0
Displaced nodes vs. wrongly predicted nodes	0.08582	0.0

Tabelle 1: Comparison of the Kolmogorov-Smirnov statistics a for the standard deviation of a node's distances to its neighbours. The model used for the predictions is that of 4.

	$D_{m,n}$	p
All nodes vs. displaced nodes	0.1110	0.0
All nodes vs. wrongly predicted nodes	0.08014	0.0
Displaced nodes vs. wrongly predicted nodes	0.09588	0.0

Tabelle 2: Comparison of the Kolmogorov-Smirnov statistics a for the minimum of a node's distances to its neighbours. The model used for the predictions is that of 4.

	$D_{m,n}$	p
All nodes vs. displaced nodes	0.2390	0.0
All nodes vs. wrongly predicted nodes	0.4071	0.0
Displaced nodes vs. wrongly predicted nodes	0.08712	0.0

Tabelle 3: Comparison of the Kolmogorov-Smirnov statistics a for the mean distance of a node to its neighbours. The model used for the predictions is that of 4.

Apparently, the p-value was below any numerical accuracy for each feature and dataset. This shows that the distribution of those sets actually differ, and in theory could be distinguished by a neural network. But since the training dataset contains 2460128 nodes, the law of large numbers implies that a statistical test is able to distinguish distributions that show only slight differences. As discussed before, a metric to quantify those differences is the value of the Kolmogorov-Smirnov statistic itself, and from the inspection of the values for $D_{n,m}$, we can actually make some observations that could explain the limited training success:

- For the standard deviation and the mean distance, the comparison between all nodes and displaced nodes lead to a medium value of the Kolmogorov-Smirnov statistic, but the comparison between truly displaced nodes and nodes falsly classified as displaced is rather low. So these falsely classified nodes not entirely differentiable from displaced nodes, based on the used features.

- The minimum does not seem to be a distincting feature for a node to be displaced since its statistics value is rather low.

Here, we assume that a statsitics value below 0.15 is low, and the distributions of the samples used for comparison are similar (since an upper bound for difference of the probabillities for the same outcome is 0.3, but in reality it might be much lower).

The same comparisons for the model listed in 5 are shown below:

	$D_{m,n}$	p
All nodes vs. wrongly predicted nodes	0.4410	0.0
Displaced nodes vs. wrongly predicted nodes	0.06616	0.0

Tabelle 4: Comparison of the Kolmogorov-Smirnov statistics a for the standard deviation of a node's distances to its neighbours. The model used for the predictions is that of 5.

	$D_{m,n}$	p
All nodes vs. wrongly predicted nodes	0.09095	0.0
Displaced nodes vs. wrongly predicted nodes	0.09805	0.0

Tabelle 5: Comparison of the Kolmogorov-Smirnov statistics a for the minimum of a node's distances to its neighbours. The model used for the predictions is that of 5.

	$D_{m,n}$	p
All nodes vs. wrongly predicted nodes	0.3746	0.0
Displaced nodes vs. wrongly predicted nodes	0.08143	0.0

Tabelle 6: Comparison of the Kolmogorov-Smirnov statistics a for the mean distance of a node to its neighbours. The model used for the predictions is that of 5.

Here, the impression from the first model is confirmed: There is a group of non-displaced nodes whose features share strong similarities with those of dispalced nodes. These are possibly nodes that are at the edge of the graphs - here, the number of neighbours is lower than expected, which influences their feature values.

5 Outreach: Possible Improvements

The results of this work suggest that the deep learning models presented in [Dö19] did not only gain their information from simple geometric properties of dislocations in quasicrystals such as the distances to their neighbours. Some information that the ansatz presented here did not include are:

- More geometric information e.g. the angles between two edges.
- Diverse labels: A future work could add additional labels for neighbours and edge nodes, as well as the nodes that are wrongly classified. Since these points often appeared at the edge of a graph, preprocessing the training dataset and removing them after the features were calculated might be helpful as well.
- Inspecting the laser potential and heat maps created with it show patterns of constructive interference beside its local extreme points. These are also influencend by phasonic patterns.

Literatur

[Cal20] Calin, Ovidiu: *Deep Learning Architectures*. 1. Springer Cham, 2020 (Springer Series in the Data Sciences). https://doi.org/10.1007/978-3-030-36721-3. – ISBN 978–3–030–36721–3

[Dö19] Döner, Ali: *Untersuchung von Störstellen in Quasikristallen unter Zuhilfenahme von neuronalen Netzen*, FAU Erlangen, Bachelorarbeit, 2019

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016 http://www.deeplearningbook.org

[JCB07] Janssen, Ted ; Chapuis, Gervais ; Boissieu, Marc de: *International Union Of Crystallography Monographs on Crystallography*. Bd. 20: *Aperiodic Crystals*. 1. Oxford University Press, 2007. – ISBN 978–0–19–856777–6

[Kos01] Koschella, Ulrich: *Zur Phason-Phonon-Kopplung in dekagonalen Quasikristallen*, Universität Stuttgart, Diplomarbeit, 2001. https://elib.uni-stuttgart.de/server/api/core/bitstreams/a63ef4ec-ef0f-4159-8bb9-86c67a8b4b82/content

[KW17] Kipf, Thomas N. ; Welling, Max: *Semi-Supervised Classification with Graph Convolutional Networks*. https://arxiv.org/abs/1609.02907. Version: 2017

[San15] Sandbrink, Matthias: *Tailored Colloidal Quasicrystals*, FAU Erlangen, PhD thesis, 2015. https://www.emergent.physics.nat.fau.de/person/phd-thesis-matthias-sandbrink/

[SLRPW21] Sanchez-Lengeling, Benjamin ; Reif, Emily ; Pearce, Adam ; Wiltschko, Alexander B.: A Gentle Introduction to Graph Neural Networks. In: *Distill* (2021). https://doi.org/10.23915/distill.00033. – https://distill.pub/2021/gnn-intro

[SLS86] Socolar, Joshua E. S. ; Lubensky, T. C. ; Steinhardt, Paul J.: Phonons, phasons, and dislocations in quasicrystals. In: *Phys. Rev. B* 34 (1986), Sep, 3345-3360. https://doi.org/10.1103/PhysRevB.34.3345

[SS12] Schmiedeberg, Michael ; Stark, Holger: Comparing light-induced colloidal quasicrystals with different rotational symmetries. In: *Journal of Physics: Condensed Matter* 24 (2012), jun, Nr. 28, 284101. https://doi.org/10.1088/0953-8984/24/28/284101

[UMSM22] Ullah, Ihsan ; Manzo, Mario ; Shah, Mitul ; Madden, Michael G.: Graph convolutional networks: analysis, improvements and results. In: *Applied Intelligence* 52 (2022), Jun, Nr. 8, 9033-9044. https://doi.org/10.1007/s10489-021-02973-4

[V+26] Virtanen, Pauli u. a.: *Documentation for scipy.stats.ks_2samp*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html. Version: 2026

[Vol25] Volzhin, Nikita: *Graph Convolutional Networks (GCN): All You Need to Know & Code Implementation*. https://medium.com/@volzhinnv/graph-convolutional-networks-gcn-all-you-need-to-know-code-implementati Version: June 2025

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Erlangen, den 31. März 2026 Hierher die Unterschrift