Structure recognition with graph neural networks

An intermediate report for the course "Advanced Projects in Computational Physics 2"

From
Stephen Weybrecht
January 25, 2025

Supervisor: Jonas Buba

Abstract

The project described in the following lies at the intersection of solid-state physics and machine learning. On the one hand, there is a physical problem, namely classifying noisy crystal lattices in 2 and 3 dimensions into their corresponding Bravais lattice group. For this graphs are randomly generated in the first step. These graphs then need to be classified efficiently and robustly, regardless of noise and introduced defects. As with other classification tasks, neural networks promise an interesting approach to this goal and will therefore be the second part of this project. For this special networks designed for handling graph-like structures, so-called Graph convolutional networks are employed. These use a concept called message passing to efficiently enable graph-level classification tasks.
By use of these concepts, a test accuracy of over 90% has been achieved for both the 2D and 3D cases. Future goals include testing out other features and network architectures for this classification task, as well as the specialization in defect detection in monoatomic crystal lattices.

needs additional work

Contents

1 Theoretical introduction

1.1 Bravais lattices

In the first part of the project, our task was to deal with crystal lattices in 2 and 3 dimensions. For the following discussion an introduction of the terms "crystal", "basis" and "Bravais lattice" is therefore needed.

Following the discussion of [1] an ideal crystal is a periodic, infinite arrangement of atoms in a solid. These atoms are arranged in blocks, a so-called basis, in a regularly spaced grid, the lattice. In other words, the lattice represents a schema after which individual atoms or groups of atoms (the basis) are arranged to form the crystal. A lattice in d dimensions can be defined by a set of d translation vectors. The superposition of integer multiples of these vectors then makes up the lattice [1]. In principle, the length and direction of these vectors can be arbitrary. In this case, the lattice would generally not map into itself under translations and rotations – it is called oblique. There are however special sets of translation vectors that form lattices of high symmetry. These fundamental lattices are called the Bravais lattices. For $d = 2$ there are 5 (4 special and one oblique) Bravais lattices, while for $d = 3$ there are 14 (13 special cases and one oblique, so-called triclinic lattice). These are depicted in Figure 1 and Figure 2 respectively.

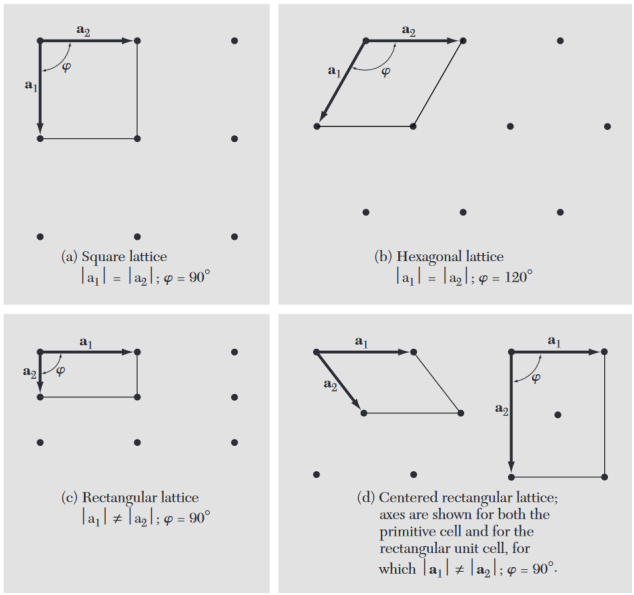

Figure 1: The five Bravais lattices for 2 dimensions. The graphic also illustrates the length of the translation vectors $\boldsymbol{a}_i$ and the angle between them to make up the corresponding lattice. Taken from [1, Fig. 7].

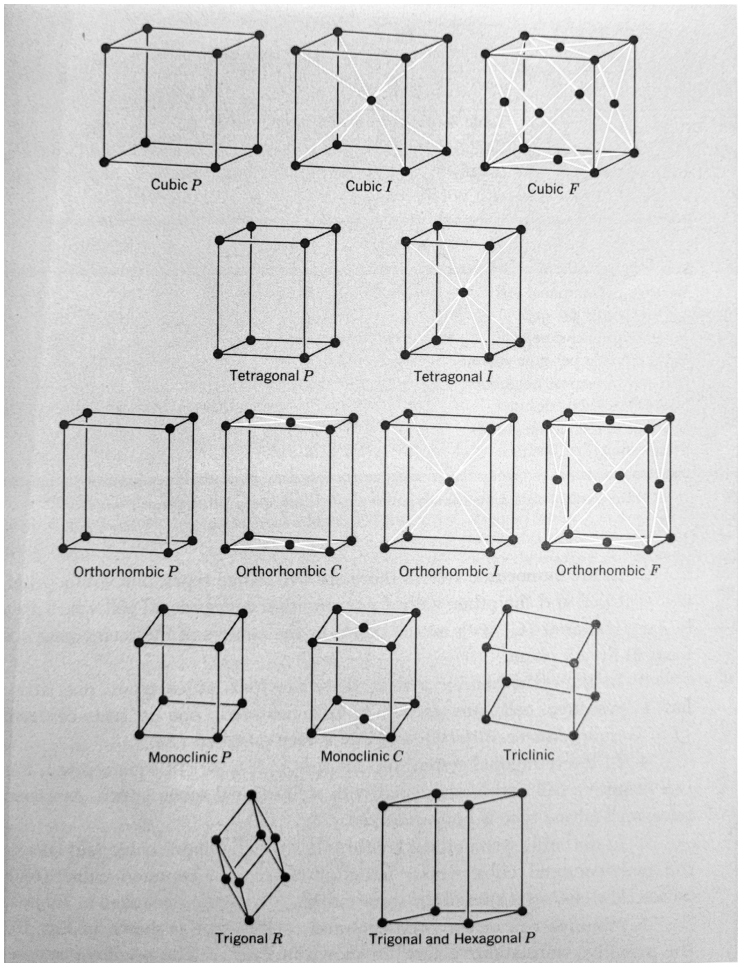

Figure 2: The 14 Bravais lattices for the $d = 3$ case. One further classifies these lattices into seven types of cells: Cubic (a=b=c, α=β=γ=90°), Tetragonal (a=b≠c, α=β=γ=90°), Orthorhombic (α=β=γ=90°), Monoclinic (α=β=90°≠γ), Triclinic, Triagonal (a=b=c, α=β=γ < 120°≠90°), Hexagonal (a=b≠c, α=β=90°, γ=120°). Note that in the prior notation a, b and c denote the lengths of the translation vectors, α, β, γ the angles between them and omitted length or angle relations mean they can be arbitrary. These groups are again subclassified based on their lattice structure into simple "P", body-centered "I", base-centered "C", and face-centered "F". As mentioned, all lattices are special cases of the general, triclinic case. The graphic is taken from [2, Fig. 14].

1.2 Basics of Machine Learning

In the following, the basic principles behind Artificial Neural Networks (NNs) and Machine Learning will be explained. This chapter follows closely the discussion in [3], the upcoming equations and concepts are therefore taken from this source. For this introduction, the simplest model will be used as an example to explain the key concepts, as a generalization to more sophisticated models is straightforward once the basic principles are understood. This simple model is the so-called Perceptron, a network consisting of fully connected units, so-called neurons, arranged in layers. This NN consists at least of an in- and output layer which are usually supplemented by one or more hidden layers in between. A neuron j in layer l has as its attribute a feature vector $x_j^{(l)}$ and each connection is associated by a weight $w_{ij}^{(l-1)}$ where this weight represents the connection between neuron i in layer $l-1$ to neuron j in layer l. The general idea is that each neuron takes as input the "signals" from every neuron it is connected to in the previous layer (ie. in our example of a fully connected network the signals from every neuron in the prior layer) and updates its own value according to the following weighted sum:

$$x_j^{(l)} = f\left(\sum_i w_{ij}^{(l-1)} \cdot x_i^{(l-1)} + b^{(l-1)}\right) \tag{1}$$

Here the weighted sum is further modified by a layer-dependent bias vector $b^{(l-1)}$. Furthermore the aggregated "signal" is usually modified by a non-linear activation function f. Using the above update rule it is now clear, that a forward pass through the model can be accomplished by supplying an input vector $x_i^{(1)}$ and using Equation 1 iteratively to achieve an output at the final layer n, $x_i^{(n)}$.

In this simple supervised training example each input vector $x_i^{(1)}$ is accompanied by a target vector t_i which is the wanted output of the network, given said input vector. For training, one must now specify a so-called loss function, which represents how far the output of the NN deviates from its target (e.g. the mean squared difference between them). The goal of training must be to minimize said loss. This is done via a process called gradient descent where the gradients of the loss function with respect to the model parameters $w_{ij}^{(l-1)}$ and $b^{(l-1)}$ are calculated. In the parameter space of these weights and biasses this gradient points toward regions where the loss changes the most. In the process of backpropagation, the NN parameters are now updated using these gradients. The algorithm goes backward through the network (From layer n to 1) and updates the parameters using calculated gradients such that the loss is minimized. The magnitude of this update is influenced by the learning rate η which is an important hyperparameter that needs to be set for training. How this update works in detail goes beyond the scope of this introduction, further details can for example be found in [3]. Once this training is completed for all input training samples one epoch of training has been completed. Usually, the training of a NN is repeated for many epochs.

Lastly, once the training was deemed sufficient, one used a different, so-called validation or test dataset, which was not used during training to assess the final performance metric of the model.

1.3 Graph neural networks

The lattices as discussed in the prior section are a collection of atoms linked by bonds and can therefore be suitably represented by a graph, consisting of nodes and edges. If we want to apply machine learning to crystal lattices, we therefore need models that are well suited for data organized in a graph-like manner. What follows is a basic introduction to such networks, specifically graph convolutional neural networks (GCNs).

GCNs take inspiration from the already well-established convolutional neural networks in which a typical layer consists of a trainable kernel that can be applied on ordered, grid-like training data of arbitrary size and shape (e.g.. images) [4]. GCNs represent a generalization of this concept onto unordered nodes with a variable number of neighbors. Each GCN-layer uses so-called message passing to update the node state $h_u^{(t)}$ of a time t to the next step $h_u^{(t+1)}$ as shown in Equation 2 [4, eq. 4.1].

$$h_u^{(t+1)} = UPDATE^{(t)}\left(h_u^{(t)}, AGGREGATE^{(t)}\left(\left\{h_v^{(t)}, \forall v \in N(u)\right\}\right)\right) \tag{2}$$

In the above equation $UPDATE^{(t)}$ and $AGGREGATE^{(t)}$ could be any differentiable functions i.e. also neural networks, and $N(u)$ denotes the neighbourhood of u meaning all directly connected nodes in the graph. This equation implies the following update schema that is also depicted in Figure 3:
The starting point is a graph, consisting of nodes with feature vectors and connections, that could also have features. For each node, messages from neighboring nodes are aggregated into a single message by taking a weighted mean of the neighboring nodes' feature vectors. This operation can also be weighted by the use of edge features. This message is then passed to a non-linear update function (e.g. ReLU), that updates the node in question for the next time step. After the update is completed, further operations can be performed depending on the wanted classification scheme. In the following graph classification is used, which means all feature vectors are pooled in a last step, to get a single quantity that is descriptive of the entire graph [4].

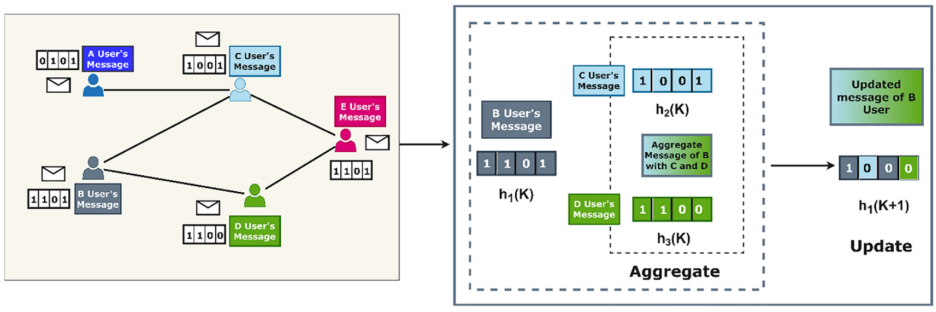

Figure 3: A graphic representation of the update schema for message passing graph neural networks. Taken from [4, Fig. 7].

2 Results

In the following, I will give a chronological overview of the topics I worked on during this project. This started with the generation of training data, namely Bravais lattices in two and three dimensions. After this, I worked on Bravais lattice classification of noisy and defective lattices to gain familiarity with the GNN architecture and a Machine Learning library. Lastly, I worked on defect detection using Graph Autoencoders.

2.1 Generation of Training data

As a first step, Bravais lattices in two dimensions (2D) and three dimensions (3D) needed to be created. For this ideal Bravais lattices with roughly a unit spacing between nodes are created in a first step. Additionally, random Gaussian noise is added. For further variety in training samples, graphs are additionally scaled by random constants to provide non-uniform node spacing within one Bravais lattice group. At this step, different kinds of defects are introduced. These defects either are the removal of randomly many nodes at random positions or the addition of a single, randomly placed node within the lattice. What kind of process has been used will be explained in detail in the following sections, as it is dependent on the task at hand. After the nodes have been placed, noise was added and defects were introduced. The connections between nodes are then determined by searching for the neighbors within a given radius (that also depends on the noise amplitude) and connecting all nodes, that lie within the said radius. By these means the random noise and defects are also included within the structure of the graph and have an influence on message passing.

The features used depend on the task of the GNN – either graph classification into one of the Bravais lattice groups or defect detection. In the following, an in-depth overview of all the different features used is given such that the later sections can focus on explaining the Neural Network.
As edge features a selection of the following was used:

- The 2D or 3D connection vectors between nodes
- Their respective length

The node features tested are the following:

- The amount of nearest neighbors i.e. the count of connected nodes
- The bond orientational order parameter (BOO)
  As the name suggests the BOO quantifies the "order" of the bonds around a node. It can be computed for different orders l and can be used to differentiate between different crystal structures by quantifying their l-fold symmetry, see e.g. [5]. Importantly symmetry is broken near defects, which is why the BOO seems a promising node feature for defect detection. In 2D the BOO of node j and order l is given by

$$\mathrm{BOO}^{2\mathrm{D}}{}_j = \left| \frac{1}{N} \sum_k \exp(il\theta_{jk}) \right|^2 \tag{3}$$

Where we sum over all N neighbors k of node j and θ_{jk} represents the angle of the j-k-connection with respect to some reference direction.
In 3D I used one of the rotational invariants defined as

$$\mathrm{BOO}^{3\mathrm{D}}{}_j = \sqrt{\frac{4\pi}{2l+1}\sum_{m=-l}^{l}\left|\frac{1}{N}\sum_k Y_{lm}\right|^2} \tag{4}$$

Here the sum ranges again over all neighbors k and adds the spherical harmonics Y_{lm} dependent on the given order of the BOO and the angles between nodes j and k and an arbitrary reference direction. As symmetry is also broken at the edges of the generated graphs, for the calculation I take care to apply sufficient padding of extra nodes around the graph (which are later removed) in order to mitigate edge effects.

CITATIONS

2.2 Bravais lattice classification

chapter

2.3 Defect detection

2.3.1 Introduction

The next goal was to build a GNN tasked with defect detection in 2D and 3D lattices utilizing the Dominant network [6]. As this has proven itself to be quite hard, the following chapters will give an overview of the tests I performed and their evaluation rather than straight-forwardly presenting a positive result. For this, the general setup and network architecture is explained in a first step, after which all different tests will be explained. At the end, I will discuss possible reasons for the bad performance of the Dominant network and suggest network architectures that are possibly better suited for this task. It should also be noted that I will focus the following discussion on noise-free 2D graphs, firstly because of time constraints due to the extensive tests I needed to do, and secondly, because it is not helpful or illustrative to investigate noisy or more complex and worse visualizable 3D graphs when even the simplest 2D cases show bad performance.

write this more clearly - ref to conclusion

Data was generated as described in subsection 2.1. As I investigated different feature combinations, the exact features used will be shown for each test specifically. Features were taken out of the feature pool shown in subsection 2.1. Per graph, one defect was added by adding an additional node at a random position. This defect node is different than the others both on a structural level (higher connectivity than the rest of the graph) but also on an attribute level (it breaks symmetry which has an influence on the BOO, additionally, on average, the length of connected edges is smaller and the number of neighbors higher). As mentioned noise was not added for the following tests to make it even easier for the network to detect the outlier node. In Figure 4 a few lattices are shown together with their feature space to illustrate how the features differ between the anomalous node and the more regular ones.

I looked at many different network setups, layers, and features and therefore needed to keep the training time in a reasonable scope which is why I chose to use 250 graphs for each of the 5 Bravais lattice types. This resulted in a total of

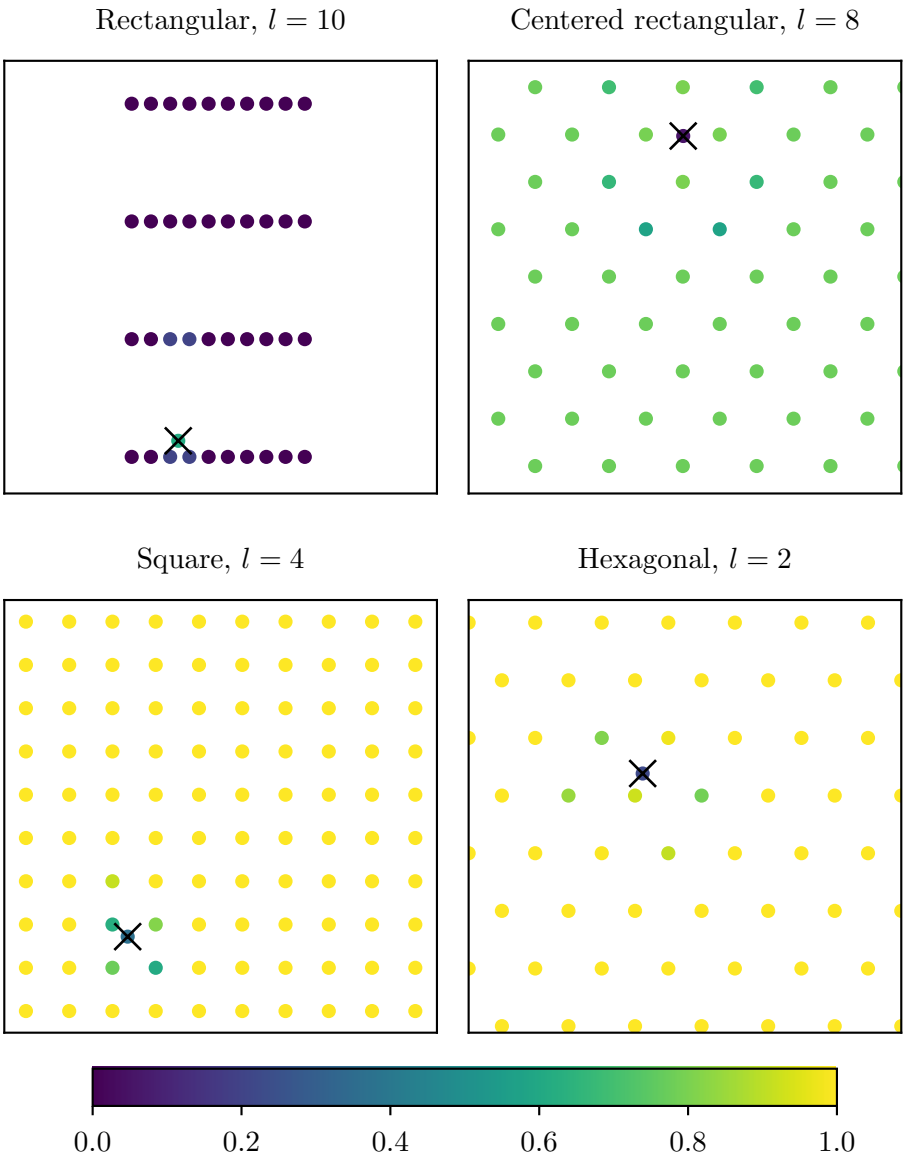

Figure 4: Shown is the feature space of 4 of the 5 Bravais lattices introduced in subsection 1.1. It is visible how the BOO of different orders highlights the additional node in each case. Some lattices were zoomed in for better visibility. The anomalous node is highlighted by a black cross.

1250 graphs, of which 1000 were used for training and 250 for validation. Each network/feature combination test consisted of training the network over 50 epochs and averaging the results over five different random network initializations in order to achieve good convergence of the loss curves and a statistically more meaningful result. For all of the following tests, a batchsize of 16 and the Adam optimizer were used. If the test used edge features GINEConv layers [7] were used for the construction of the encoder and attribute decoder, else GCNConv layers [8] were used. As the learning rate, activation functions, and the number of neurons and layers were varied, they will be stated explicitly when talking about the specific test performed.

2.3.2 The Dominant Architecture

I was tasked to implement the Dominant network as described in [6]. This network uses a graph autoencoder i.e. an autoencoder consisting of message passing layers to handle graphs. Autoencoders are a special kind of neural network that utilizes unsupervised training. The goal of this type of network is to reconstruct the input data as best as possible. This task would be trivial if it were not for an important feature of autoencoders: The hidden layers reduce in dimensionality until a minimum number of trainable parameters is reached at the so-called bottleneck after which the dimensionality again increases until the input dimensionality. It is expected that this compression and de-compression works better for large-scale recurring structures (e.g. a periodic arrangement of nodes) than for outliers (e.g. the defect node) which makes this approach promising for the task at hand [6]. A sketch of the structure of the Dominant network is shown in Figure 5. A training set of graphs

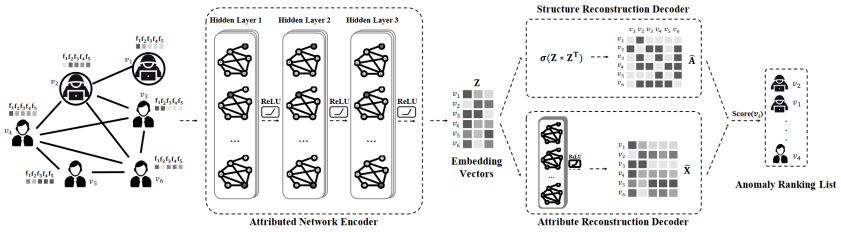

Figure 5: A sketch of the structure of the Dominant network is shown. The shown number of layers and activation functions do not necessarily reflect the ones used in the following, as different tests have been performed. Taken from [6, Fig.1]

with outliers is first input into the encoder. This encoder consists of message-passing layers that reduce in feature dimensionality until the bottleneck is reached. The graph data is maximally compressed into a so-called embedding vector z. This is then decoded in two different decoders: The structure reconstruction decoder aims to reproduce the input adjacency matrix, which is a possible representation of the edges and their features in a graph. This is done by calculating the matrix product of the column vector z with the row vector z^T leading to a matrix of shape $N \times N$ where N is the number of nodes. This can be interpreted as an adjacency matrix. Furthermore, a non-linear activation function (like a sigmoid) can be applied. Additionally, the attribute reconstruction decoder aims to reproduce

the nodal features of the input graph. This is done by the use of an additional GNN which increases in feature dimensionality. In the following, I choose the encoder and decoder network symmetrically. After getting both reconstructions a loss can be calculated (this is not shown in Figure 5) with which the model is trained [6]:

$$\mathcal{L} = (1-\alpha)\left\|A - \hat{A}\right\|_F + \alpha\left\|X - \hat{X}\right\|_F \tag{5}$$

Here α is a fixed hyperparameter weighting the attribute versus the reconstruction loss, A and $\hat{A}$ are the input and reconstructed adjacency matrices, X and $\hat{X}$ the input and reconstructed node feature matrices and the symbol $\|\cdot\|_F$ represents the Frobenius norm. Note that compared to [6] I have omitted squaring the norms as this would have led to extremely large losses which would have needed additional normalization to achieve a good convergence without offering any benefit.

Once training is deemed sufficient, a final anomaly score can be calculated for each node i to evaluate the model [6]:

$$\mathcal{S}_i = (1-\alpha)\left\|a_i - \hat{a}_i\right\|_2 + \alpha\left\|x_i - \hat{x}_i\right\|_2 \tag{6}$$

Here a_i, $\hat{a}_i$, x_i, $\hat{x}_i$ represent the rows of the matrices A, $\hat{A}$, X, $\hat{X}$ resprectively and $\|\cdot\|_2$ represents the usual vector 2-norm. Note that while Equation 5 and Equation 6 look rather similar, they are not quite the same. The loss function computes a single scalar loss for each input graph and is used during training. The anomaly score function computes one scalar anomaly score for each node in the graph and is only used for evaluation. Larger anomaly scores represent a higher probability of the node being an outlier.

When evaluating the model's anomaly score in practice the result is a 1D vector consisting of "unnormalized probabilities" where each entry corresponds to the anomaly score of the node at the given index. In principle, these values could be normalized to a value range of 0-1 such that they could be interpreted as the probability of the given node being an outlier. This is however not done as it is not necessary for the further steps. Instead, all anomaly score vectors of the evaluation dataset (one for each graph) are combined into a single metric (called score in the following) quantifying the performance of the network. This score is given by the average probability of the true anomalous node being in the top five anomaly scores the network predicts. The averaging happens over the whole test dataset at each epoch to consider all Bravais lattice types. Here the top 5 anomaly scores are checked, as all input features are different from the background not only at the additional node but also at the nodes surrounding it (see also Figure 4). Because of this also a guess in the neighborhood of the additional node should be considered correct which is why it also increases the score. Note that because correspondance with the true position of the additional node (i.e. the graph label) is checked in the score evaluation, using the score during training would be a supervised learning approach and defeat the purpose of the autoencoder. Therefore I want to emphasize again, that the training is done in an unsupervised manner (using the loss) and the score only represents an evaluation metric.

2.3.3 Results

I performed multiple tests with differently structured layers, activation functions, learning rates η, loss weights α, dropout, and data parameters. These I will present in the following to show, that no matter the chosen hyperparameters the training and performance of the NN is bad. In the first step η and the use of dropout was varied, after which the best features, network structure, activation functions, and α were explored in subsequent tests. When varying one or two related parameters the others were kept fixed. All hyperparameters for the different tests A-E are tabularised in Table 1 for referencing them in Figure 6 - Figure 10.

As mentioned in the first test the influence of the learning rate η and the use of dropout were examined. To make matters easier a dropout with a fixed probability of 0.5 has been applied between each layer during training. For evaluation, this dropout is automatically switched off. For this first test, it seemed best to supply the network with maximal information, which is why all node features (BOO of order $l = 2, 4, 6, 8, 10$ and the number of neighbors) were used as well as the length of edges as edge features. Because edge features are used, GINEConv layers are employed (They support edge features in contrast to GCNConv layers) in a structure of 6432, meaning the input feature vector dimensionality is reduced from 6 to 4 to 3 to 2 in the encoder and increased again symmetrically in the attribute decoder. As activation a sigmoid was used after each hidden layer for the attribute reconstruction (taking care not to apply a sigmoid to the output) and a ReLu function was used for the structure decoder, because as the adjacency matrix can have values bigger than 1 depending on the edge features. All these choices are also listed in Table 1 and will therefore be discussed less extensively for the following tests.
Now the network was run for larger and smaller η in combination with dropout or without dropout for 5 times 50 epochs. For each training epoch, the network was evaluated by calculating its loss and detection score based on a test sample that was not used during training. These curves are then averaged over the mentioned 5 random network initializations to get a more robust and comparable result, as single runs have proven themselves to be very dependent on initialization. The results are shown in Figure 6, where one can see that the network performs best if dropout and $\eta = 0.001$ are implemented.

However, the central problem that will also govern the tests shown in the following becomes apparent. While the loss is minimized quite well by the network and a stable loss is reached after around 50 epochs indicating the network has trained all it can given the input network and data parameters, the score behaves quite differently. Namely, there are two problems. First of all, it is immediately apparent that the score does not increase steadily with falling loss. Rather it increases and decreases seemingly at random even for the "optimal" parameter choice shown in red. Looking at all curves a slight downward trend is even visible. Secondly one can see the score already starts on quite a high level of roughly 50% correct guesses. As the score is determined by the percentage of test graphs where the true anomalous node is within the top five highest anomaly scores however and there are roughly 100 nodes in each graph the initial score should be much lower if the network would just perform a random guess. This indicates the network in its untrained state is in fact not performing random guesses but rather there is a systematic component "pushing" the network towards the right guess even when all weights are initialized

test	layers	structure	A-act.	S-act.	η	DO	α	NN	BOO
A1	GINE	6432	σ	ReLu	0.001	✗	0.5	✓	✓
A2	-	-	-	-	0.005	-	-	-	-
A3	-	-	-	-	0.01	-	-	-	-
A4	-	-	-	-	0.0005	✓	-	-	-
A5	-	-	-	-	0.001	✓	-	-	-
A6	-	-	-	-	0.003	-	-	-	-
A7	-	-	-	-	0.005	-	-	-	-
A8	-	-	-	-	0.01	-	-	-	-
B1	GINE	6432	σ	ReLu	0.001	✓	0.5	✓	✓
B2	-	-	-	-	-	-	-	✗	✓
B3	-	-	-	-	-	-	-	✗	$l \neq 2$
C1	GCN	6432	ReLu	ReLu	0.001	✓	0.5	✓	✓
C2	-	6543	-	-	-	-	-	-	-
C3	GCN	6531	-	-	-	-	-	-	-
C4	-	65432	-	-	-	-	-	-	-
C5	GINE	6432	σ	ReLu	0.001	✓	0.5	✓	✓
C6	-	6543	-	-	-	-	-	-	-
C7	-	6531	-	-	-	-	-	-	-
C8	-	65432	-	-	-	-	-	-	-
D1	GCN	6432	σ	σ	0.001	✓	0.5	✓	✓
D2	-	-	σ	none	-	-	-	-	-
D3	-	-	ReLu	ReLu	-	-	-	-	-
D4	-	-	ReLu	none	-	-	-	-	-
D5	-	-	ReLu	σ	-	-	-	-	-
E1	GCN	6432	ReLu	ReLu	0.001	✓	0.3	✓	✓
E2	-	-	-	-	-	-	0.4	-	-
E3	-	-	-	-	-	-	0.5	-	-
E4	-	-	-	-	-	-	0.6	-	-
E5	-	-	-	-	-	-	0.7	-	-

Table 1: Listed are the used hyperparameters for each test A-E. Each row corresponds to one network/data combination that was used for training 5 times 50 epochs which were then averaged to produce the curves in Figure 6 - Figure 10. "-" means the parameter is the same as in the row above. Cells with a grey background indicate the best results of each test. The columns represent from left to right: The used layer (either GINEConv or GCNConv), the dimensionality of the encoder layers with mirrored decoder (e.g. 6432 means the input node feature vector of dimension 6 is reduced to a hidden feature vector of dimension 4 in the first, 3 in the second hidden layer and 2 in the bottleneck), the attribute (A-act.) and structure (S-act.) decoder activation functions, the learning rate, whether a dropout with probability 0.5 was used, the loss weighting, whether the number of nearest neighbors (NN) or the bond orientational order parameter (BOO) have been used as features.

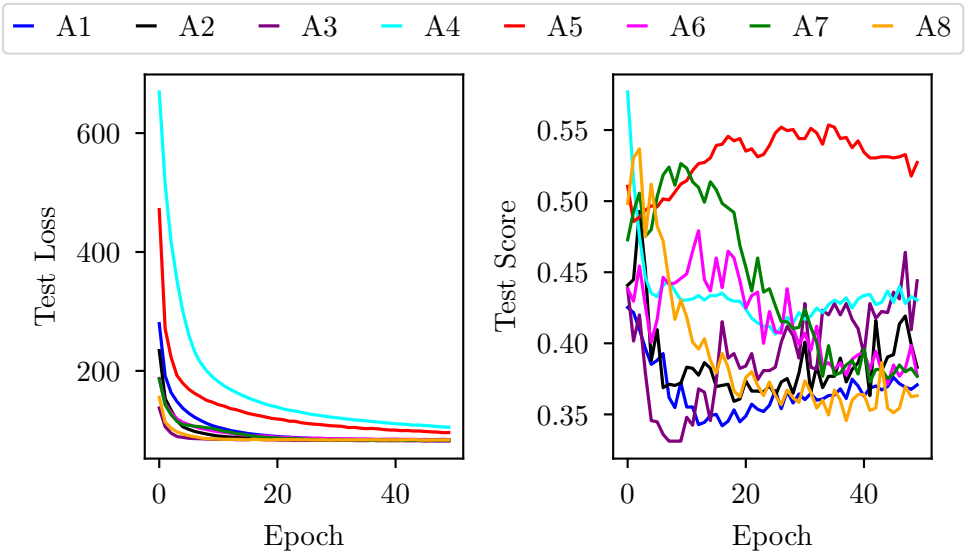

Figure 6: Shown are the test results for Test A. For a listing of hyperparameters see Table 1.

randomly. I reason this follows from the nature of how the anomalous node affects the graph and the network structure. Firstly a spurious node increases connectivity around it as it increases local density and with that the amount of connected nodes as they are determined by connecting all nodes within a said radius. In the message passing layers this highly connected node now receives more messages that are additionally different from those other more regular nodes receive. The number of neighbors for instance is higher on average, while the edge length is lower and the BOO also differs in all orders. I assume that this increases the mean squared reconstruction error of this node on average even for a random initialization and therefore leads to a slightly higher anomaly score in a local neighborhood around the anomalous node. This in turn leads to the fact that the network has quite good performance from the start.
What I want to stress however is that this high score should not be interpreted as a success in achieving the given task. This is because it is not a learned behavior but rather just stems from systematics that indicates that a machine learning approach with this architecture is not well suited for defect detection as learning is the facto not needed. This might further indicate that there could be a simpler algorithm that doesn't use Machine Learning but can extract the anomalous node just from the connectivity and feature space of the graph itself.

This general result is also visible in the next four tests I performed. In the next step, I wanted to establish what kind of node features are useful for the network. The result of this is shown in Figure 7. The motivation behind this test was mainly to see whether the number of neighbors was a good node feature or not and how many degrees of the BOO are useful for the network. I expected the neighbor-information to be already present in the general connectivity of the graph to some extent and therefore wanted to quantify the difference between using this feature and not using

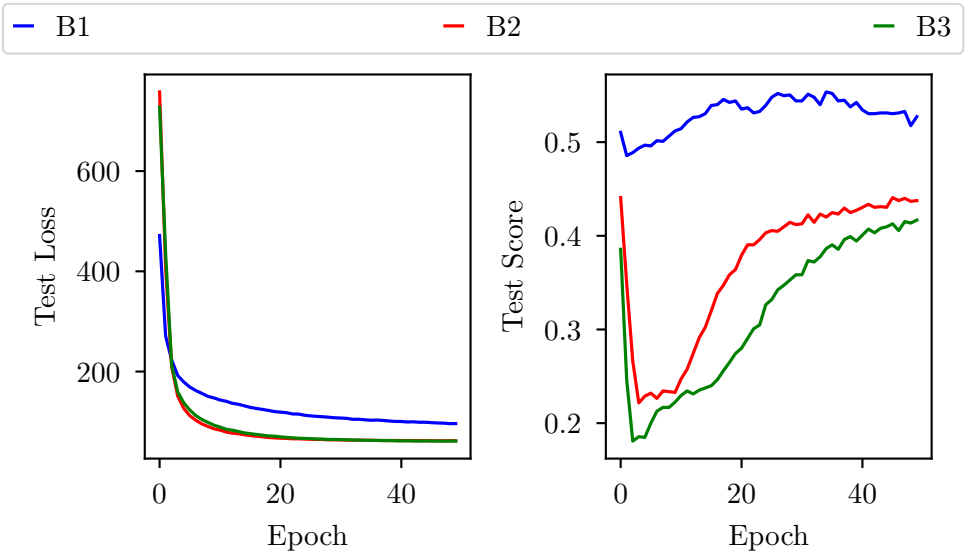

Figure 7: Shown are the test results for Test B. For a listing of hyperparameters see Table 1.

it. The result is that the network performs best when given the maximal amount of information, i.e. the BOO of order $l = 2, 4, 6, 8, 10$ and the number of neighbors. However, it should again be stated that the evaluation metric that interests us in the end, namely the score, has the same behavior as before, leading to the same conclusion that the network is not learning to do the desired task of outlier detection.

As test C I performed a variation of the network layers, the number of layers, and feature dimensionality in them. The switch in layers also meant a switch in features as only the GINEConv layers support edge features, meaning none were used for the GCNConv layers. The result is shown in Figure 8.

As one can see in the graphic the network seemingly works best if no edge features are present and GCNConv layers are used instead. I don't expect the reason for this to be that the network works better with less information however (especially as the edge length usually pinpoints quite well where the additional node is present). Looking at the starting score of the different curves in Figure 8 it becomes apparent that it is already much higher for the GCNConv layers than for the GINEConv layers. During training, both then fluctuate seemingly at random again and partly increase or decrease. The better performance of one layer over the other can therefore be attributed to the same systematic effect that leads to the high score in the beginning for all layer types and is not due to the model being better at classification. Nevertheless, the tests I performed indicated the best performance with GCNConv layers of the structure 6531.

The test shown in Figure 9 now focused on the question which activation function to use. I expected a sigmoid to perform badly as attribute activation as there are nodal attributes that are larger than unity which would be truncated leading to a big reconstruction error. This is indeed what the test shows as well. The best

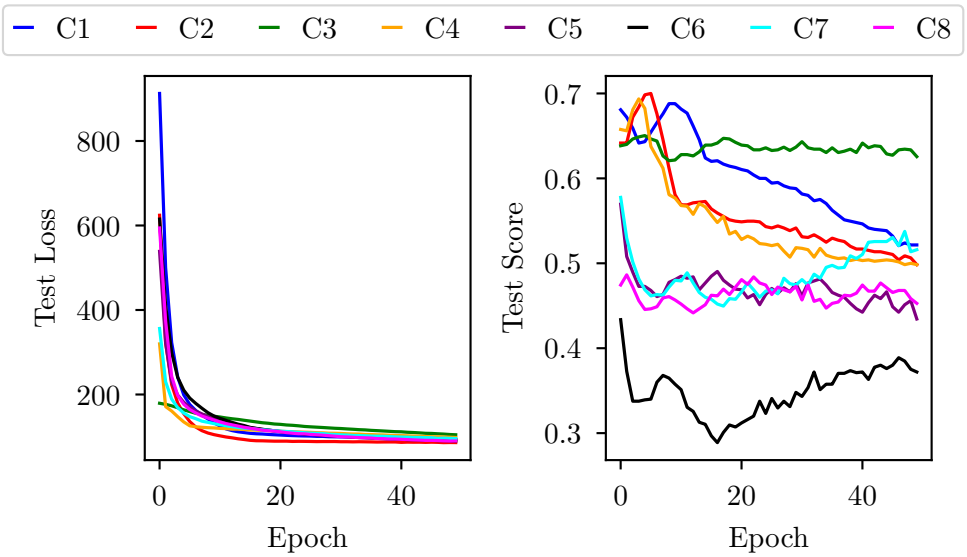

Figure 8: Shown are the test results for Test C. For a listing of hyperparameters see Table 1.

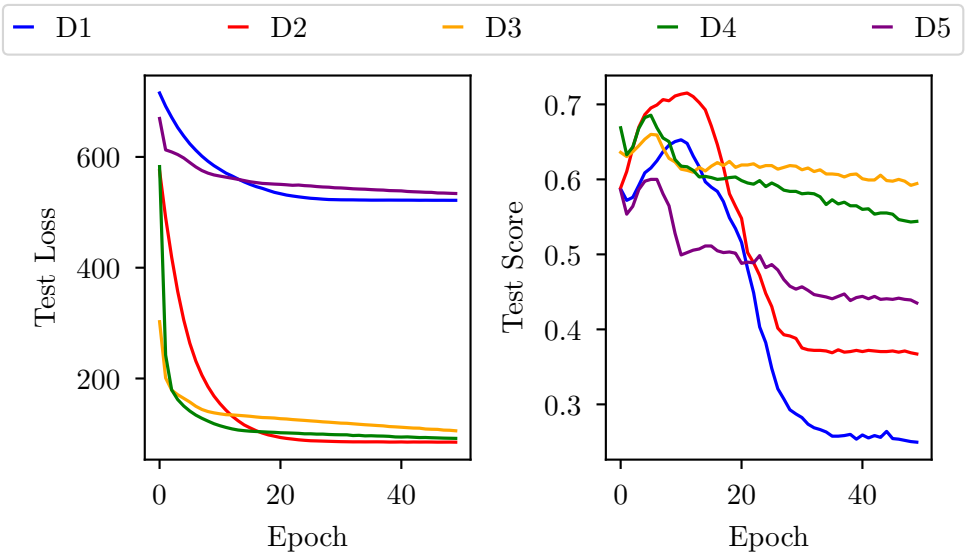

Figure 9: Shown are the test results for Test D. For a listing of hyperparameters see Table 1.

combination has proven itself to be ReLu as both activations. However, again the previously mentioned issues with the score curves are visible.

As a last test, I varied the α parameter. Looking at the calculation of the loss and score function in Equation 5 and Equation 6 it becomes clear that α varies the weight of structure reconstruction loss over the attribute reconstruction loss and score. The test curves shown in Figure 10 indicate the best performance for $\alpha = 0.4$ as it performs only slightly worse than $\alpha = 0.3$ but seems to be more stable.

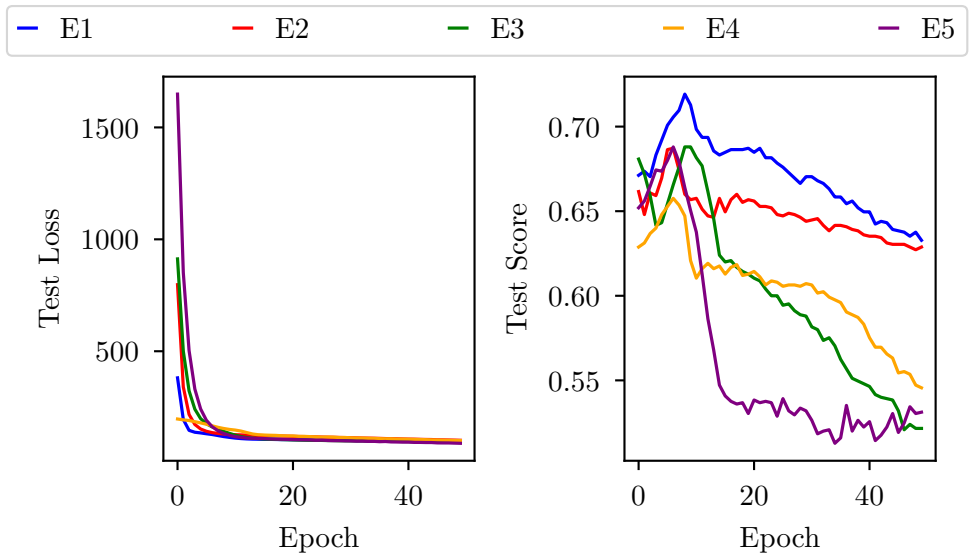

Figure 10: Shown are the test results for Test E. For a listing of hyperparameters see Table 1.

In summary, all tests shown above indicate bad performance of the model in the task of learning to classify defects in 2D lattices. Even though the resulting scores are quite high (at times well above 60%) they are not high because the model has learned to classify lattices through the use of the graph autoencoder and the loss and score functions proposed in [6] and implemented here but rather because there is a systematic component in the structure of the graph data and the GNN itself leading to very high scores from the beginning. These high scores then only fluctuate and mostly even decay while training, all while the training objective function (the loss) gets smaller and smaller. In other words: The model is not bad at the task it trains for. During training the loss function is minimized well by use of gradient descent and backpropagation techniques introduced in subsection 1.2. Therefore it succeeds in reproducing the *features* of nodes and edges well. However, this does not translate to the model getting better at *identifying outliers* which is the task we want it to perform in the end.
With the extensive amount of tests that were performed, I further showed that this is not simply due to a bad choice of hyperparameters or data features but rather a very fundamental problem with the architecture itself. It should further be emphasized again that this result was achieved with the simplest Bravais lattices one can think of. Planar graphs without any noise, boundary effects, or other

special cases like missing nodes have been considered to make learning to identify outliers easy. Additionally, features have been chosen to give a large amount of information, essentially each feature points directly to the spurious node in the graph (see Figure 4). That learning still does not work therefore again indicates that the Dominant-architecture is not well suited for the task at hand.

3 Conclusion

text

Todo:

- Summary and Conclusion of subsection 2.2
- Summary and Conclusion of subsection 2.3
- Reiterate what and why it didnt work
- What could be a better suited architecture for the job? Node classifacation, supervised?
- What did I howerver still achieve -¿ extensive tests, dominant not working, next project differnt approach

4 Results so far

Wie groß war der Datensatz? Wie wurde er aufgeteilt für Training/Validation

Wie lief der Trainingsprozess ab? (Optimizer, batchsize, learning rate etc., gerne loss/accuracy plotten)

* Verwendete Layer und Aufbau des Netzes beschreiben (welche Methode wurde für das Pooling verwendet? wie kommen wir vom Pooling zu unserer Wahrscheinlichkeitsverteilung der Gittertypen?)

The main tasks so far were to gain familiarity with a machine learning Python library by working on graph classification of 2- and 3-dimensional Bravais lattices. For this, I chose PyTorch, as it has a library for working with graph neural networks called PyTorch Geometric (PyG) built on top of it. The first step for both cases was to generate the training data. For this ideal Bravais lattices were created in the first step, to which random (Gaussian) noise and defects (meaning a removal of a random number of nodes at random positions) were added. For further variety in training samples, graphs were additionally scaled by random constants to provide non-uniform node spacing within one Bravais lattice group. After the nodes have been placed and noise was added, the connections between nodes are determined by searching for the neighbors within a given radius (that also depends on the noise amplitude) and connecting all nodes, that lie within said radius. By these means the random noise and defects are also included within the structure of the graph and have an influence in message passing.
The size of the graphs has been chosen as follows. For the plane lattices a size of 10 by 10 nodes has been chosen in order to give the network enough information about each training sample to learn something about the Bravais lattice. For the 3D lattices a size of 10 by 10 by 10 nodes has been tried but deemed unpractical as the time effort for training is quite large. This in turn leads to only being able to use a low number of training samples which leads to low accuracies of around 60%. A better approach proved to be using graphs of size 5 by 5 by 5 nodes which cuts the total number of nodes per graph by a factor of 8 while still giving the network enough information to determine the lattice correctly.
For node features the number of neighboring nodes has been chosen while each edge

has the (two or three-dimensional) connection vector between its corresponding nodes as its feature. This has proven itself to be enough information to classify a test dataset of random graphs with more than 90% accuracy for both cases. For classification, each lattice was labeled by its one-hot encoded type that was then compared by using the cross entropy loss function during training. As connection vectors carry quite a lot of information, for future tests other features like connection lengths or binding angles are planned in order to see, how much information about the lattice is needed to suitably classify it. For these preliminary results a high degree of information in the features and shallow neural network (2D: 2 GINEConv layers [7] with a total of 12 hidden neurons, 3D 3 GINEConv layers with 50 hidden neurons total) was however used as a proof of concept and to see whether graph generation worked.

5 Plans for the future

In the following, the goals for the remaining part of the project will be discussed and a rough time estimate for each step will be given. As graph generation and classification have worked rather well so far, I expect only little changes to be made to the already existing code. Further things to improve or try out with the existing method are varying node and edge features and choosing different layers or a different network structure. By this investigation, one could find out the minimal amount of information the network needs to classify Bravais lattices and the most efficient network architecture. As the tasks discussed so far however only served as a kind of general introduction to the actual topic that is specific to me, extensive experiments with the old goal of lattice classification are not planned. Instead, the focus shifts toward a new goal, namely the detection of defects in mono-atomic crystals. I expect to be able to reuse significant parts of the lattice generation code I have written so far, although significant changes in the network architecture and features will most likely be made. A meeting with Jonas Buba to discuss the specifics of the future project is scheduled for Wednesday, the 11.12. After that, I plan to finish the remaining experiments having to do with the old goal of Bravais lattice classification until the end of December. During the same time, I want to gain familiarity with the theory and suitable approaches for defect detection by reading literature and starting to code. Until the mid of January, I plan to finalize the coding part of the project, to be able to focus on writing the report and preparing the final presentation during the remaining time.

6 Code availability

The code written for this project is made available in the following Git repository: `https://github.com/SteWey0/Computerpraktikum/`

Bibliography

[1] Charles Kittel. “Chapter 1: Crystal Structure”. In: *Introduction to Solid State Physics*. 8. ed. Wiley, 2005. ISBN: 978-0-471-41526-8.

[2] Charles Kittel. “Chapter 1: Crystal Structure”. In: *Introduction to Solid State Physics*. 4. ed. Wiley, 1971. ISBN: 0-471-49021-0.

[3] Miroslav Kubat. “Chapter 5: Artificial Neural Networks”. In: *An Introduction to Machine Learning*. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-63912-3 978-3-319-63913-0. DOI: 10.1007/978-3-319-63913-0. URL: http://link.springer.com/10.1007/978-3-319-63913-0 (visited on 01/17/2025).

[4] Bharti Khemani, Shruti Patil, Ketan Kotecha, and Sudeep Tanwar. “A Review of Graph Neural Networks: Concepts, Architectures, Techniques, Challenges, Datasets, Applications, and Future Directions”. In: *Journal of Big Data* 11.1 (Jan. 16, 2024), p. 18. ISSN: 2196-1115. DOI: 10.1186/s40537-023-00876-4. URL: https://doi.org/10.1186/s40537-023-00876-4 (visited on 12/08/2024).

[5] Paul J. Steinhardt, David R. Nelson, and Marco Ronchetti. “Bond-Orientational Order in Liquids and Glasses”. In: *Physical Review B* 28.2 (July 15, 1983), pp. 784–805. DOI: 10.1103/PhysRevB.28.784. URL: https://link.aps.org/doi/10.1103/PhysRevB.28.784 (visited on 01/10/2025).

[6] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. “Deep Anomaly Detection on Attributed Networks”. In: *Proceedings of the 2019 SIAM International Conference on Data Mining (SDM)*. Proceedings. Society for Industrial and Applied Mathematics, May 6, 2019, pp. 594–602. DOI: 10.1137/1.9781611975673.67. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611975673.67 (visited on 01/17/2025).

[7] PyG Team. *GINEConv*. GINEConv. Dec. 9, 2024. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GINEConv.html#torch_geometric.nn.conv.GINEConv (visited on 12/09/2024).

[8] PyG Team. *GCNConv*. GCNConv. Jan. 20, 2025. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GCNConv.html (visited on 01/20/2025).