BABEŞ-BOLYAI UNIVERSITY OF CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Dissertation thesis

Rotation, Translation and Scale Invariant Graph Neural Networks

Abstract

Graph neural networks (GNNs) are a generalization of convolutional neural networks meant to work with graph data. A domain of great interest where GNNs can be applied is 3-dimensional object classification. The work done in this thesis focuses on improving these methods' robustness, by making the graph convolution blocks invariant to certain geometric transformations. If a model is invariant to a transformation, it means that applying the transformation to an input will not change the model's output. The main contribution of our work is extending previous models that incorporated translation and rotation invariance to also incorporate scale invariance. The mechanism used to achieve scale invariance is applying a local, graph-neighborhood based normalization to the edge lengths. This approach has the advantage of being outlier resistant in comparison to scaling the inputs as part of a pre-processing step.

Our experimental results show that the three transformations we have considered have a huge performance impact if not properly handled by the model in the ways we have described. The technologies used in this project are python, pytorch and torch geometric.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

JULY 2025 ENG. OLTEANU VLAD

ADVISOR:
PROF. CSATÓ LEHEL, PHD.

BABEŞ-BOLYAI UNIVERSITY OF CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Dissertation thesis

Rotation, Translation and Scale Invariant Graph Neural Networks

SCIENTIFIC SUPERVISOR:
PROF. CSATÓ LEHEL, PHD.

STUDENT:
ENG. OLTEANU VLAD

JULY 2025

UNIVERSITATEA BABEŞ-BOLYAI DIN CLUJ–NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de dizertaţie

Rețele Neuronale pentru Grafuri cu Invarianță la Rotație, Translație și Scalare

CONDUCĂTOR ŞTIINŢIFIC:
PROF. DR. CSATÓ LEHEL

ABSOLVENT:
ING. OLTEANU VLAD

IULIE 2025

Contents

Chapter 1

Introduction

Having its roots in computer science, artificial intelligence evolved from simple graph or heuristic-based methods to increasingly more complex solutions to face the tasks placed upon it. These tasks can range from applying clustering or regression on simple datasets to learning patterns in highly dimensional data using machine learning. The latter of the two has received far greater amount of attention recently, due to its capability to perform tasks that would be intractable using classical algorithms [Russell and Norvig, 2016].

The types of data that can be used to train machine learning algorithms are as varied as the types of architectures that can be used to learn from it. Some data types are easily represented as vectors, while others are not. Among the types of data that are easier to vectorize, we can find tabular data and images. As we try to apply the same methods to more complex things such as video, sound, or natural language, the challenge of effectively encoding the underlying structure of the data comes into play [Goodfellow et al., 2016].

Another type of data which is not trivial to encode as vectors is graph data. A graph has 2 main components: its nodes and their relationships, or edges. Although node attributes are easy to encode, their relationships are not. The core challenge of applying machine learning algorithms on graphs lies in finding a suitable approach to distill the node relationships into vectors [Zhou et al., 2020].

Another challenge is ensuring that the model is invariant with regard to graph isomorphism. The nodes of a graph have no particular order, but they have to be presented to the model in a certain order nevertheless. This has to be also taken into account when building the algorithms, such that the same output should be given for 2 identical graphs whose node order is different [Bronstein et al., 2021].

Graphs are an incredibly versatile data type. Most things in the real world can be described using graphs because the world in its essence is interconnected. Graphs have been used to model things like societies, computer networks, molecules and many more [Brandes et al., 2013; Engel, 2006; Riaz and Ali, 2011]. Something that is of great interest that can be modeled as a graph is physical space and its topology. In 3d graphics, the illusion of a solid object is given by representing its surface using meshes [Shirley et al., 2009]. Physical space can be transformed into a mesh by using Lidar (light detection and ranging) and a face reconstruction algorithms [Wang et al., 2018]. We can then translate the meshes into graphs by turning each triangle or polygon into a number of edges. The ability to represent physical

space in this format gives us a great incentive for developing graph centered machine learning methods.

One problem that generally plagues neural networks is their robustness. Small variations in the input data can drastically decrease a model's accuracy. One example of such a difference is applying a rotation to an image. While a human being would naturally recognize the contents of a picture no matter in what angle it is presented in, machine learning models do not inherently posses this capacity. A technique that tries to counteract this weakness is repeatedly presenting the same input to the model under different transformations [Goodfellow et al., 2016]. Going back to the image example, an approach that is used in some cases is rotating the image by a number of degrees and showing each rotated version to the model individually. This improves the robustness of the model by some amount, but it has the disadvantage of increasing the training time by a factor of n, where n is the number of angles considered during training, This is caused by the need to evaluate the same input n times, once for every transformation applied to it. Rotating the inputs a few times does not however guarantee that the model will learn rotation invariance from the provided examples, it only encourages it to approximate it.

Applying this technique to image data is computationally expensive, but when using the same approach for 3-dimensional meshes, the situation only gets worse. If only considering rotation invariance, the given input would need to be rotated n times along all 3 axes, not just one. This increases the number of training inputs by a factor of n^3, which is intractable. Another approach would be to build this invariance directly into the model. This eliminates the need to repeatedly evaluate the same input under different perspectives, saving a large amount of computation time.

Plenty of approaches implement a combination of translation and rotation invariance or equivariance into the model architecture, but so far no approach attempted to also incorporate scale invariance/equivariance. This observation is the starting point of our research, which has 4 main objectives: The first objective is to ascertain the degree to which existing models are influenced by changes in input rotation, position and scale. This serves to determine the importance of incorporating invariance to these transformations into the models, and the drawbacks we might face if we do not do so.

Another objective is to determine what drawback implementing invariance into the models has on the overall performance. Obtaining invariant features involves hiding information from the input that could be changed by the transformations the model is meant to be invariant to. This information loss is more than certain to result in a lower performance in the best case scenario. The magnitude of this performance drawback needs to be determined and put into balance with the benefits brought by the increased robustness of the model.

The third objective of this research is to devise a method that can be used to create models that are scale invariant/equivariant in addition to the rotation and translation invariance/equivariance proposed by previous work. This objective ties in with the first one, offering a solution to the performance degradation caused by the possible scale differences between the training data and the inputs presented to the model.

The final objective of our work is to build a model that incorporates rotation, translation and scale invariance/equivariance, all at the same time. This will be achived by extending existing work with the approach we will use to obtain scale invariant features. Extending an existing model with another invariance type grounds our approach in previous work and serves as a blueprint for extending other

solutions with the same property.

Finding answers to these research questions has led us to the following contributions:

1. We are the first to propose a model that is invariant/equivariant to rotation, translation and scaling at the same time
2. We solve the scale invariance problem by proposing a local, graph-neighborhood based scale normalization approach that is more robust than global scale normalization.
3. We show experimentally that transforming an object by using translation, rotation or scaling greatly reduces a model's performance, if the model does not have invariance to the transformation built in.
4. We provide formal proofs for the correctness of our approach and of the chosen invariant feature.

The rest of this thesis is structured as follows: In chapter 2 we discuss the principles of geometric deep learning and graph neural networks. In chapter 3 we argue for the importance of scale invariance/equivariance and the drawbacks of global scale normalization. In chapter 4 we propose a model that presents the aforementioned properties and give details about our architectural choices. In chapter 5 we test the performance degradation of our model and several others on inputs transformed using scaling, rotation and translation. In chapter 6 we formally prove that the proposed convolution block is invariant/equivariant to the 3 transformations. Lastly, we give some concluding remarks and proposals for future work in chapter 7.

Chapter 2

Geometric Deep Learning & Graph Neural Networks

The two main topics our work is based on are geometric deep learning and graph neural networks. In what follows we will begin to explore the main ideas of geometric deep learning by referring to [Bronstein et al., 2021].

Geometric deep learning is a branch of deep learning that emphasizes making use of the underlying symmetry groups of a problem when building a model, in order to constrain the possible functions it is able to approximate to ones that are equivariant or invariant to the symmetries of the problem's domain. If a function is equivariant it means that applying a transformation from the symmetry group to its input will yield the same transformation to the output ($Tf(x) = f(Tx)$). When a function is invariant it means that applying a transformation from the symmetry group to the input will yield no effect to the output ($f(x) = f(Tx)$). Some geometric domains that have underlying symmetry groups are sets, graphs, grids, Euclidean spaces, meshes and many more. One example of a symmetry group is translation symmetry for grids. A model that is widely used that takes advantage of this symmetry is the convolutional neural network (CNN). If we interpret an input image as a grid, the CNN's convolutions are translation equivariant, due to the shared parameters of the convolution kernel. In other words, by shifting the input image by a certain amount, the output of the convolution layer is shifted in the same direction by the same amount.

Another example of a symmetry group is permutation symmetry for graph and set data. The elements of a set or the nodes of a graph do not have an inherent order. That is to say that writing the elements as $\{A, B, C\}$ or $\{B, A, C\}$ specifies the same graph or set. Unfortunately, the data has to be processed by the computer in some order or another, so that needs to be taken into account when building permutation invariant/equivariant models.

In the case we are working with spatial data such as positions or directions in Euclidean space, we need to take into account symmetries like rotation and translation. In certain cases, it should not matter if an object is moved or rotated when being considered by the network. In other words, a chair remains a chair even if it is rotated by 90 degrees, and the model's output should not be influenced by the rotation. The same holds true for shifting the coordinates of the object by a certain amount. If the object is not in the center of the scene, it does not mean that it is a different object. Building these symmetries into the model can be helpful in situations when we can expect the object put under scrutiny not to be found in a

canonical position and orientation.

Moving on from geometric deep learning, we will give a brief overview of graph neural networks by referring to [Zhou et al., 2020]. When working with graph type data, one can choose between a number of tasks. These can be node-level, edge-level or graph-level tasks. One example of a node-level task is predicting the attributes of a given node by means of the attributes of the node's neighbors. Other node-level tasks can include classification, clustering and so on. Edge-level tasks can take the form of predicting whether there is some relationship between a node and another based on the nodes' attributes. Graph-level tasks imply combining the information from all the nodes and their relationships into a single embedding, and then using it to perform a task like classification, clustering and more.

Neural networks that work with graph data are called GNNs (graph neural networks). These are generalizations of the well known convolutional networks. They are an extension that is meant to work with graph data instead of image data. The main elements of a GNN are a series of graph convolution layers. The outputs of the graph convolution layers can be processed further using other architectures like feed forward networks, or they can be used as they are. A single graph convolution layer consists of a convolution or recurrent operation that can be optionally paired with a sampling operation for the input, and a skip connection or a pooling operation for the output. The output of the convolution blocks can take the form of node embeddings, edge embeddings or even whole graph embeddings, in the case that all the upstream values are merged into a single embedding using one or more pooling operations.

The approaches used to build graph convolution operators can be split in 2 main categories: spectral approaches and spatial approaches. Spectral approaches first transform the graph into the frequency domain and only then apply the convolution operation. After the convolution is performed, the graph is transformed back using the inverse Fourier transform.

This approach is well studied, but it has the disadvantage that a model built in this manner cannot be adapted from a graph's structure to another's. The other downside is that this method is computationally expensive. The other type of approaches that can be used for graph convolution are spatial methods. In this case, the convolution takes place directly in the spatial domain, avoiding many expensive computations.

There are several types of spatial convolution blocks of varying complexities, ranging from basic spatial approaches to attention and message passing based techniques. Basic spatial approaches perform a simple aggregation, like the sum, or the mean, over the neighbor set, and then combine the result of the aggregation with the state of the current node using different methods.

Looking further into attention based approaches, we will refer to the graph attention networks introduced in [Veličković et al., 2017]. The core idea of the attention based architectures is to employ a learnable, non-linear transformation that takes as input two nodes' features and has as a result an attention score (a scalar). The attention score of node j with respect to node i indicates how relevant node j is to node i when computing node i's representation in the next layer. These scores are used to give different weights to each node in a neighbor set, allowing the network to focus on certain inputs more than on others. The architecture employed by the authors can be described as follows:

$$e_{ij}^l = a^l(W^l h_i^l, W^l h_j^l) \tag{2.1}$$

$$\alpha_{ij}^l = softmax_{N_i}(e_{ij}^l) = \frac{exp(e_{ij}^l)}{\sum_{k \in N_i} exp(e_{ik}^l)} \tag{2.2}$$

$$h_i^{l+1} = \sigma\left(\sum_{j \in N_i} \alpha_{ij}^l W^l h_j^l\right) \tag{2.3}$$

In equation 2.1, e_{ij}^l represents the unnormalized attention score of node j with regard to node i at layer l. h_i^l and h_j^l are the features of node i and node j at layer l, and W^l is a linear transformation that is used to transform the node features into a higher order representation. a^l is a non-linear transformation that takes the transformed nodes features and turns them into a scalar. This computation is applied independently for every node in the neighbor set.

After having computed the unnormalized attention scores, the next step is to normalize them using the softmax function applied over the neighborhood of every node, as seen in equation 2.2. This operation will yield an attention score (α_{ij}^l) for every node j in node i's neighborhood. The scores sum up to 1, giving rise to a convex combination when used in a weighted sum.

The representation of node h at layer $l+1$ is finally computed using equation 2.3. The node features of the neighbors at the current layer h_j^l are first transformed using the same linear transformation used when computing the attention scores (W^l). The transformed node features are then multiplied by their respective attention score (α_{ij}^l) and summed. The convex combination of the transformed node features is passed through a final non-linearity (σ) to obtain the final result. Several attention heads can be used at each layer, and their outputs can be combined by either applying a mean pool or concatenating them.

Moving on to message passing neural networks, we will give a brief description of the general framework used when developing such a network by referring to [Gilmer et al., 2017], where the concept was first introduced. The authors unify previous graph neural network architectures under a common formalism called message passing graph neural networks. This idea was later used to develop many new architectures, being a significant stepping stone in this line of research. The main idea of message passing GNNs is breaking down the network into a series of composable operations: a message function, a vertex update function and a graph readout function.

$$m_{ij}^l = M^l(h_i^l, h_j^l, e_{ij}) \tag{2.4}$$

$$m_i^l = \sum_{j \in N_i} m_{ij} \tag{2.5}$$

$$h_i^{l+1} = U^l(h_i^l, m_i^l) \tag{2.6}$$

$$y = R(h_i^L | i \in G) \tag{2.7}$$

The message function is a learnable, non-linear transformation, that is used to compute the message to be sent from one node to another. In equation 2.4, M^l is the message function used for layer l. This function is applied for every node j in the neighborhood of every node i. The inputs of this function are the features of node i and j (h_i, h_j), and the features of the edge found between node i and node j (e_{ij}). The edge feature is only used in the case that the graph also contains edge information. The result of this operation will be the message passed from node j to node i at layer l (m_{ij}^l).

After the messages have been computed, the next step is the vertex update operation. For every node, the messages sent from its neighbors are aggregated using a permutation invariant function like the sum, as seen in equation 2.5. The aggregated messages (m_i^l) and the features of the node at layer l (h_i^l) are passed through another learnable non-linear function (U^l) to get the node features at the next layer (h_i^{l+1}).

The message passing and vertex update operations are repeated for every subsequent graph convolution layer. After the final convolution is executed, the node features computed at the last layer (h_*^L) can be aggregated using a permutation invariant function (R), to obtain a single graph embedding, as seen in equation 2.7. This is only necessary for graph level tasks, we can use the node embeddings as they are in other cases.

One example of a message passing neural network is the edge convolution architecture introduced in [Wang et al., 2019].

$$h_i^{l+1} = \sum_{j \; in N_i} M^l(h_i^l, h_j^l - h_i^l) \tag{2.8}$$

The update formula for an edge convolution block is described by equation 2.8. The message to a node i from its neighbor j is computed by applying a non-linear learnable function M^l on the features of node i at layer l (h_i^l) and the difference between the features of node j and node i at layer l ($h_j^l - h_i^l$). This convolution block does not have an update function, it constructs the representation of the node at the next layer by using only the message computation's non-linearity instead.

Having discussed the main ideas of geometric deep learning and graph neural networks, we will now look into architectures that incorporate invariance/equivariance to certain 3d transformations. The model introduced in [Satorras et al., 2021] introduces a convolution block that is equivariant to rotation and translation, or the E(3) Euclidean group. In order to achieve these properties, the model computes the messages passed from one node to another by only using the non-spatial attributes of the nodes and the squared distance of their positions. The reason this leads to rotation and translation equivariance is that the distance between 2 points remains the same no matter how they are rotated or translated. After the message passing step, the non-spatial and spatial attributes are updated individually, leading to E(3) equivariance for the operation applied on the spatial information and E(3) invariance for the operation applied to the non-spatial information. We will explore this model in more detail in chapter 4, where we will explain how we extended it to also incorporate scale invariance/equivariance.

When looking for models that incorporate E(3)+scale equivariance/invariance, we were only able to find a model that is rotation invariant/equivariant and that claims it can be modified to also incorporate translation and scale invariance/equivariance [Chen et al., 2022]. This claim only holds true however for the equivariance property and not for the invariance, which we will see in a moment.

$$x_i = \square_{j \in N_i} M^l(s_i^l, h_j^l, s_j^l - s_i^l) \tag{2.9}$$

$$s' = W_v^l s \odot (W_{hv}^l h / ||W_{hv}^l h||) \tag{2.10}$$

$$h' = W_h^l h + W_{vh}^l \Omega(s) \tag{2.11}$$

$$\Omega(s)_c = \left\langle s_c, \frac{\bar{s}}{||\bar{s}||} \right\rangle \tag{2.12}$$

The rotation equivariant/invariant model is described in equations 2.9, 2.10, 2.11, 2.12. Similarly to [Satorras et al., 2021], the model treats the spatial and non-spatial information separately. In equation 2.9, we can see the vertex update function, which is equal to the messages passed to a node from its neighbors aggregated using a permutation invariant function ($\square$). h_i^l represents the non-spatial attributes and s_i^l the spatial attributes of node i at layer l. Equation 2.10 computes the spatial compnents of the message. It uses a number of linear transformations (W_*^l) and a channel-wise multiplication ($\odot$) between the spatial (s) and non-spatial (h) components from the previous layer. In order to update the non-spatial components, the model uses equation 2.11. The computation involves a number of linear transformations and a sum applied on the non-spatial information from the previous layer (h), and the spatial information (s) passed through a transformation the authors call the invariance function (Ω).

The definition of Ω can be seen in equation 2.12. The purpose of this function is to produce an invariant feature from the spatial information. The model is meant to be rotation equivariant, so the output of the function must be rotation invariant. This is achieved by using the dot product between all the vectors passed as spatial information (s_i^l and $s_j^l - s_i^l$ at the input layer), and the normalized mean of these vectors $\left(\frac{\bar{s}}{||\bar{s}||}\right)$. This will reduce every input vector to a scalar that is equal to its norm multiplied by its cosine with the mean of all the spatial input vectors: $\Omega(s)_c = ||s_c|| cos(s_c, \bar{s})$. The cosine does not change if the points are rotated and neither does the norm of the vectors, so the rotation invariance holds. The problems start appearing only when extending the model with translation and scale equivariance.

$$x_i = \square_{j \in N_i} M^l(h_j^l, s_j^l - s_i^l) \tag{2.13}$$

The authors suggest to introduce translation equivariance/invariance by changing equation 2.9 to equation 2.13, by removing the absolute positions from the input of the message computation. This will only leave 1 input vector for the spatial information ($s_j^l - s_i^l$), so the mean of all the input vectors will be equal with the only input vector ($s = \bar{s}$). For the input layer, this will turn omega to $\Omega(s)_c =$

$||s_c||cos(s_c, s_c) = ||s_c||$.

$$h' = W_h^l h + W_{vh}^l \frac{\Omega(s)}{||\Omega(s)||} \tag{2.14}$$

The authors suggest to further extend the model with scale equivariance/invariance by replacing equation 2.11 with equation 2.14. As we have seen, by introducing translation invariance/equivariance $\Omega(s)$ becomes a 1 element vector equal to $||s||$. If we then normalize $\Omega(s)$ by its length, we will get $\frac{\Omega(s)}{||\Omega(s)||} = \frac{[||s||]}{||s||} = [1]$. In essence, the geometric information is entirely discarded from the invariant feature through a series of computations that yield the same result regardless of the input. The input vector can be linearly transformed into several vectors at the latter layers, but a similar situation takes place there aswell. To give a more concise explanation of the problem, if we don't use the absolute positions, the orientations or the lengths of a vector, what else is there to it?

Moving on to other architectures that implement invariance or equivariance, we can see a number of examples that use the attention based approach. In [Le et al., 2022a] the authors implement a transformer based model that uses the distance between 2 points to obtain equivariance to the SO(3) group (rotation equivariance). The authors of [Le et al., 2022b] propose an E(3) equivariant transformer-based GNN. Instead of using the distance between 2 graph nodes as the equivariant feature, they embed the relative positions of 2 nodes into a higher order representation using spherical harmonics. In [Liao and Smidt, 2022], the authors create an SO(3) (rotation group) equivariant transformer-based gnn. The equivariant feature is created by using spherical harmonics, similarly to the approach taken in [Le et al., 2022b]

There are an enormous number of other models that implement either E(3)/SE(3) or O(3)/SO(3) invariance/equivariance, but virtually none that include scale invariance/equivariance alongside one or the other. This observation is the basis of the work done in this thesis.

Chapter 3

Scale Invariance & Equivariance

As we have discussed in the previous chapter, there are plenty of graph neural network models that implement invariance/equivariance to translation, rotation or a combination of the 2. As for now, the literature contains no examples of a model that implements rotation, translation and scale invariance or equivariance at the same time.

Scale invariance/equivariance is useful in situations when we expect the inputs to lie on a wide range of scales or the true scale information about an object cannot be extracted from the dataset. What we want to achieve by incorporating this type of invariance/equivariance into the model is making it only discern the nature of an input by its shape, not by how large or small the shape is.

There would be no reason to also incorporate scale equivariance/invariance into the model, if we could easily normalise the input into a certain range, a common practice in any machine learning project. In this chapter we will discuss the problems that come with using normalization as a preprocessing step in the context of 3-dimensional data, and how our proposed method counteracts these weaknesses.

One of the reasons global normalization is hard to do correctly, is that there is no clear answer to the question "by what amount should the distances in the graph be normalised?". One could choose anything from fitting the entire mesh into a $[0, 1]^3$ cube, to dividing every distance by the maximum distance in the graph. The problem with these approaches is that they are highly sensitive to outliers. If for example, we have a single node that was mistakenly added far away from the correct mesh, both of the aforementioned methods would fail to correctly scale the model. If we tried to fit the model inside a $[0, 1]^3$ cube, the outlier would cause the model to be squished down into a smaller section of space than it would have been if the incorrect point was not present, essentially producing an incorrect result due to a single wrong input node. If we consider dividing the lengths of the edges by the length of the maximum edge, the same problem would arise. If we have a single node that is far away from the main object, the lengths of all the edges of the correct object will be divided by the length of the incorrect edge that connects the noise node to the rest of the mesh. This will result in the edges of the mesh being shorter than they would have been if the incorrect node was not present.

Dividing by the maximum length is in some sense even worse than fitting the entire model in a cube, because there could be a great amount of variation in the lengths of the edges. For example if an object is "rounder", it will most likely consists of shorter and more numerous edges that will try to approximate a

circular shape. If the object has large flat surfaces, they could be defined by just 4 nodes connected by a few long edges. The problem is that the longest edge in the mesh, even if no outliers are present, highly depends on the level of detail the model is trying to encapsulate. If we have have only curved surfaces, the maximum distance will be small, and the curved surfaces would be represented as much larger than they would be, if larger flat surfaces would also be present in the model. One could try to alleviate this by using the median or mean length instead of the maximum, but there's no reason to believe that would yield a better result, because there's no guarantee that the lengths of the edges lie on an unimodal distribution. If for example 70% of the edges belonged to large flat surfaces and 30% to round surfaces, the median would still be an edge belonging to a flat surface, whose length will most likely be a lot larger than its round surface counterparts.

The problems we just discussed arise when we try to scale a single object using these preprocessing mechanism. The situation becomes even more complicated when we introduce multi-object scenes, like rooms or city streets. If we have even just 2 objects in a scene, by trying to fit both of them into a $[0, 1]^3$ cube, the larger object will cause the smaller object to be scaled down much more than it should be. If we try to divide all the edges by the length of the maximum edge, the object with the longest surface would cause the other object to be scaled down more than it would have been on its own. If we try to divide the edges by the median or mean length we run into the same problem as with applying this method for a single object. The lengths of the edges are not guaranteed to lie on an unimodal distribution, if shorter and more numerous edges that approximate rounder surfaces dominate, they would scale up longer edges that belong to flatter surfaces out of proportion. If longer, flat surfaces dominate, they would squish down the finer, shorter edges more than they should be. This problem is only exacerbated by how different the level of detail of the 2 objects can be. The less similar the granularity of the 2 objects' edges is, the more different the final scaling result will be compared to what would have been obtained if they were scaled individually.

We could attempt to apply semantic segmentation to the scene in order to treat each object of the scene individually, but that is usually achieved by using graph neural networks, so the network attempting to segment the scene would suffer from the same scale biases that we are trying to eliminate.

Chapter 4

Scale Invariant Architecture

In order to overcome the shortcomings we enumerated in the previous chapter, we propose a model that uses local, neighborhood-based normalization instead of global normalization.

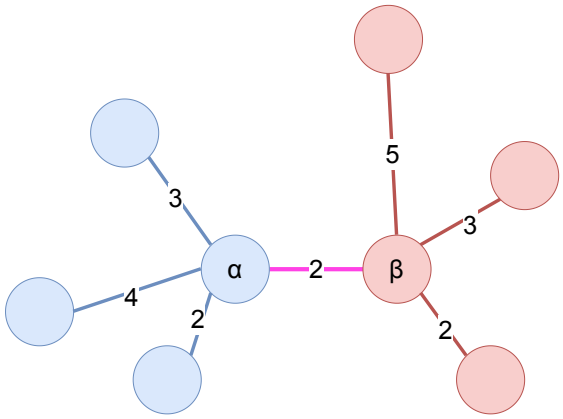


Figure 4.1: Neighborhood of 2 nodes and their distances before local normalization.

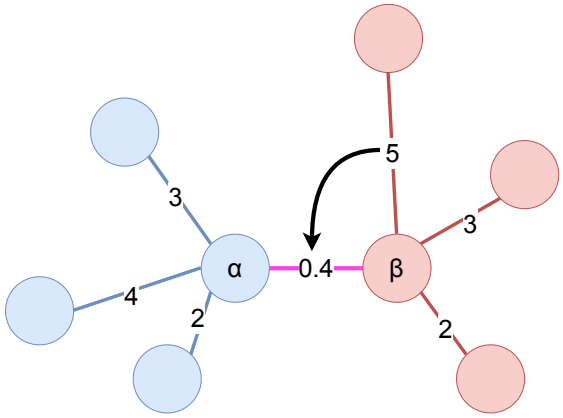


Figure 4.2: Neighborhood of 2 nodes and their distances after local normalization.

Figures 4.1 and 4.2 illustrate the manner in which the neighborhood-based normalization is applied when considering an edge between a node α and a node β. When being applied to an edge $\alpha\beta$, the

normalization works by dividing the length of the edge by the maximum length found within node α's and node β's neighborhoods. In the 2 figures we mentioned previously, the neighborhood of node α is marked with blue, the neighborhood of node β with red and the common edge with purple. The numbers shown on the edges indicate the distance between the 2 nodes that are connected by it. In this example, the edge with the maximum length is the edge with length 5, found within node β's neighborhood. We will take this value and divide the length of the edge $\alpha\beta$ with it.

The only edge that is affected by this normalization is the edge contained between the 2 nodes whose neighborhoods we take into consideration. The neighborhood used to compute this normalization factor will be different for an edge than for another. As a result, we will repeat this neighborhood normalization step for every edge in the graph, by changing α and β to the nodes located at the ends of each edge.

We use this method to obtain scale invariant edge lengths. The reason this property arises is that if we consider a constant c by which the mesh could be scaled, all the lengths of the edges in the mesh will be scaled by that amount. By dividing the query edge by another edge in its neighborhood (in our case the longest one) the constant c gets reduced in the process:

$$\frac{c \cdot \alpha\beta}{c \cdot m} = \frac{\alpha\beta}{m}$$

This approach also solves the problems that arise when using global edge-length normalization or bounding-box normalization. Our approach is noise-resistant, as the only edges that influence the normalization process are the edges adjacent to the ends of the edge taken into consideration. In the case that a noise node or an outlier is present in the input data, the only nodes that will be affected by it will be at most 2-hops away. Using this method is also helpful in dealing with scenes composed of multiple objects. Assuming that 2 distinct objects are only connected at their contact points (locations in which the objects touch), the only nodes that will suffer from this confusion will be at most 2-hops away from the objects' boundary. This assures us that the nodes located more inward of the object's shape will not be affected by other objects that may be present in the scene. Using this method, we can independently handle objects with different levels of detail without the need of actually performing scene segmentation.

Having devised a way to obtain scale invariant edge lengths, we continue by incorporating our method into an existing convolution block that contains rotation and translation invariance/equivariance. We chose to extend the convolution block introduced in [Satorras et al., 2021] because it uses node distances as a rotation/translation invariant feature. This can naturally be extended with our scale invariant, locally normalized edge lengths, to obtain a model that is invariant to rotation, translation and scaling at the same time. The model splits the node information into 2 categories: spatial information (node positions) and non-spatial information (node attributes). The convolution block applies an invariant transformation to the non-spatial information and an equivariant transformation to the spatial information. We extend the invariant transformation to also incorporate scale invariance, by applying the local normalization described previously to the edge lengths. By extending the invariant feature with scale invariance, we implicitly extend the equivariant features with scale equivariance.

The original convolution block introduced in [Satorras et al., 2021] can be described as follows:

$$m_{ij}^l = \phi_e(h_i^l, h_j^l, \|s_i^l - s_j^l\|^2) \tag{4.1}$$

$$s_i^{l+1} = s_i^l + \frac{1}{card(N_i)} \sum_{j \in N_i} (s_i^l - s_j^l)\phi_s(m_{ij}^l) \tag{4.2}$$

$$m_i^l = \sum m_{ij}^l \tag{4.3}$$

$$h_i^{l+1} = \phi_n(h_i^l, m_i^l) \tag{4.4}$$

The convolution block uses the message passing technique to propagate information from one node to another. The message passed from node j to node i at layer l is computed using formula 4.1. The message is composed of the non-spatial information of node i (h_i^l) concatenated with the non-spatial information of node j (h_j^l) and the squared distance between the spatial components of node i (s_i^l) and the spatial components of node j (s_j^l) at layer l. The vector obtained by concatenating these elements is then passed through a learnable, non-linear function approximated by a feedforward neural network. Note that the only spatial information used here is the squared distance between the 2 nodes. The squared distance has the property that rotation and translation transformations applied on 2 points will always yield 2 points that are the same distance apart. This makes the message passed translation and rotation invariant.

After the messages have been computed, the spatial information is updated using equation 4.2. The position of node i at layer $l+1$ (s_i^{l+1}) is obtained by shifting the position of node i at the previous layer (s_i^l) away from each of its neighbors by an amount decided by a learnable function (ϕ_s). The input to the ϕ_s function is the message passed by each neighbor to the node. The amount the node is moved away from each neighbor is divided by the number of neighbors of the node ($card(N_i)$). We discussed earlier that the message m_{ij}^l is obtained by applying an operation that is rotation and translation invariant. The other spatial information used here is the position of the node at the previous layer s_i^l and the direction from each neighbor towards the node ($s_i - s_j$). If we apply a translation or rotation transformation to the inputs, the positions of the nodes and the directions from one node to another will be rotated and/or translated by the same amount. This will result in an output that is translated or rotated by the same amount as the input, making the operation translation and rotation equivariant.

The non-spatial information of node i at layer $l+1$ is computed using equation 4.4. The new non-spatial information is obtained by concatenating the non-spatial information from the previous layer h_i^l with the sum of all the messages passed to node i from its neighbors, and passing the combined information through a learnable function (ϕ_n). We have seen that the messages from one node to another are computed using a translation and rotation invariant operation. There is no other spatial information involved in this transformation, which means that this operation is translation and rotation invariant. We will later use this property to build a model that is invariant to these 2 operations.

We take advantage of this convolution block and extend it to also include scale equivariance for the spatial information and scale invariance for the non-spatial information:

$$M_i^l = max(\{\|s_i^l - s_j^l\|^2, \forall j \in N_i\}) \tag{4.5}$$

$$m_{ij}^l = \phi_e\left(h_i^l, h_j^l, \frac{\|s_i^l - s_j^l\|^2}{\max(M_i^l, M_j^l)}\right) \tag{4.6}$$

$$s_i^{l+1} = s_i^l + \frac{1}{\text{card}(N_i)} \sum_{j \in N_i} (s_i^l - s_j^l)\phi_s(m_{ij}^l) \tag{4.7}$$

$$m_i^l = \sum m_{ij}^l \tag{4.8}$$

$$h_i^{l+1} = \phi_n(h_i^l, m_i^l) \tag{4.9}$$

In order to extend the graph convolution layer with scale invariance/equivariance, we introduce a scale propagation layer that comes before every graph convolution layer. The scale propagation layer computes the maximum squared distance from each node to the nodes in its neighborhood, as seen in equation 4.5.

Our graph convolution block uses a similar message passing approach as [Satorras et al., 2021]. The key difference is the introduction of neighborhood-based normalization in the message passing computation, which can be seen in equation 4.6. The message from node j to node i at layer l is computed by applying a non-linear, learnable function (ϕ_e). The input to this function is the non-spatial information of node i (h_i^l) concatenated with the non-spatial information of node j (h_j^l) and the squared distance between the spatial components of node i and node j normalized by the maximum squared distance within node i and node j's neighborhoods

$$\frac{\|s_i^l - s_j^l\|^2}{max(M_i^l, M_j^l)}.$$

This is the key ingredient in achieving scale invariance. If the graph's spatial components are scaled, the maximum distance will be scaled as well, so the final result is identical. As earlier, if we apply a rotation and/or a translation transformation to the spatial information, the squared distances will not change, thus making the message computation translation, rotation, and scale invariant.

The spatial components at the next layer are computed in the same way as in the original convolution block, as seen in equation 4.7. We have already discussed that this operation is translation and rotation equivariant, but by making the messages passed from one node to another scale invariant, we can see that it also becomes scale equivariant. This follows from the fact that by scaling the node's positions, the position differences and thus the final result are scaled by the same amount. This means that by scaling the inputs, the outputs are scaled in the same way, which makes the operation scale equivariant.

The non-spatial components at the next layer are computed by applying a learnable, non-linear function (ϕ_n) to the non-spatial components at the current layer (h_i^l) and the sum of all the messages from a node's neighbors to itself. We have discussed that the message computation is rotation, translation, and

scale invariant. There is no other spatial information involved in this operation, thus making it rotation, translation, and scale invariant as well. We will use this property when building a model that is invariant to these 3 transformations.

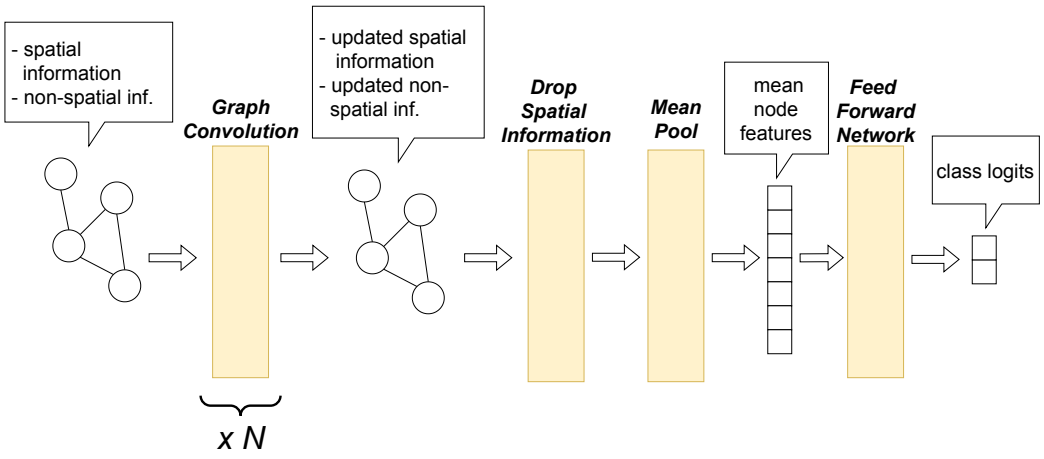


Figure 4.3: Rotation & translation invariant network architecture

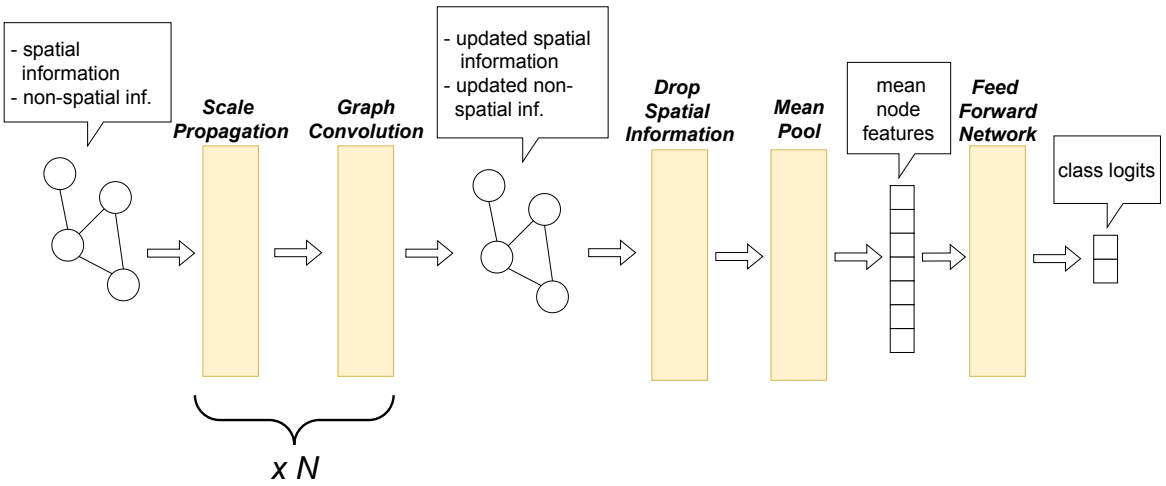


Figure 4.4: Rotation, translation & scale invariant network architecture

In order to build networks that are invariant to translation and rotation, we will use the graph convolution block introduced in [Satorras et al., 2021]. The network architecture can be seen in figure 4.3. We take advantage of the fact that the operations applied to the non-spatial information are translation and rotation invariant, and the operations applied to the spatial information equivariant. We need the convolution blocks to produce an invariant result, so after performing n convolution steps, we will completely drop the spatial information. This leaves us with only the non-spatial information, which is produced using invariant operations. After performing the graph convolutions, we apply a mean pool to aggregate all the nodes' features into a single vector. The mean node features are then passed through a feed forward

network to obtain the output class logits.

Similarly, we use our convolution block to produce networks that are invariant to rotation, translation, and scaling. The network architecture can be seen in figure 4.4. We take advantage of the property that the non-spatial information is computed using operations that are invariant to these 3 transformations, and after performing n graph convolution steps, we discard everything but the non-spatial information. We then apply a mean pool to reduce the node features into a single vector and pass it to a feed forward network. Proofs for the invariance and equivariance of our convolution block can be found in chapter 6.

Chapter 5

Performance Degradation Under Geometric Transformations

The ShapeNet dataset [Chang et al., 2015] is a 3d model dataset aggregated from smaller 3d object repositories. The meshes contained in the dataset are richly annotated, each object being linked to the related WordNet [Miller, 1995] synset and ImageNet [Deng et al., 2009] images. The objects also have other associated metadata, like physical attributes and function. ShapenNetCore is a subset of this dataset that only contains single objects grouped by class, without additional annotations. The dataset is made up of a large amount of examples, containing a grand total of 51300 meshes grouped in 55 categories. We used the ShapenNetCore dataset for conducting our experiments. Out of the 55 classes, we picked the 2 most populated ones, the couch class and the airplane class. Examples of models contained in the 2 classes can be seen in figure 5.1 and figure 5.2. We only picked 2 out of the 55 classes for 2 reasons. On one hand, only processing 2/55 parts of the dataset helped us shorten the training time, which was extremely long even in this scenario due to hardware restrictions. On the other hand, it allowed us not to deal with the discrepancies in the number of examples across the dataset's categories. While some had thousands of examples, others had only hundreds. Our main goal was to demonstrate the effects of scale variations, not to deal with imbalanced classification problems.

We tested the performance of a neural network architecture that used a different type of convolution block in each test scenario. The graph convolution blocks used in the experiments are the following:

- Edge convolution block, introduced in [Wang et al., 2019] (has no invariance/equivariance)
- Graph attention block, introduced in [Veličković et al., 2017] (has no invariance/equivariance)
- E(n) equivariant block, introduced in [Satorras et al., 2021] (has rotation and translation invariance/equivariance)
- Our implementation (has rotation, translation and scale invariance/equivariance)

The experiments involved testing each model variant on inputs transformed using different combinations of rotations and translations, and different combinations of rotations and scaling factors. In the first scenario, the models were tested by randomly rotating and translating the meshes. The rotations

varied from no rotation up to anything between 0 and a complete rotation on each axis. In combination with these rotations, the meshes were also translated in a random direction by an amount between 0 and a maximum that ranged from 0 up to 40 units. As a reference point, the unchanged models are scaled within a $[-0.5, 0.5]^3$ bounding box and centered at 0.

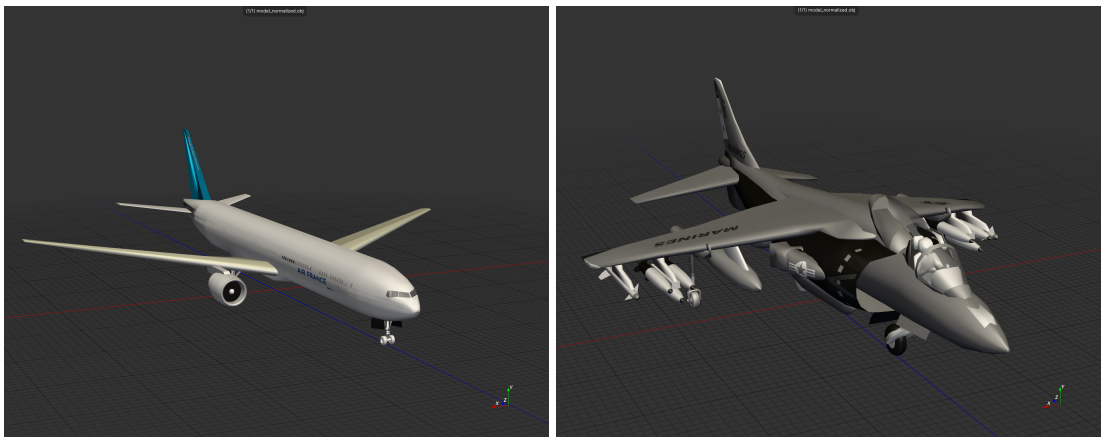


Figure 5.1: Airplane model examples

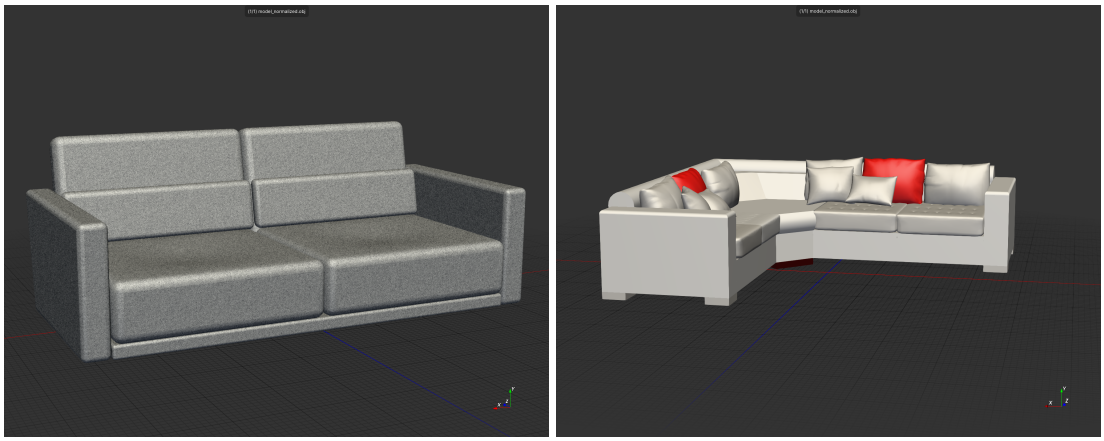


Figure 5.2: Couch model examples

The second set of experiments is similar to the first. We randomly rotated the meshes by an amount that varies from 0 to an upper bound, but instead of translating them we scaled them by a constant. This constant ranges from 1 (the mesh is not scaled) up to 5 (the mesh is 5 times larger).

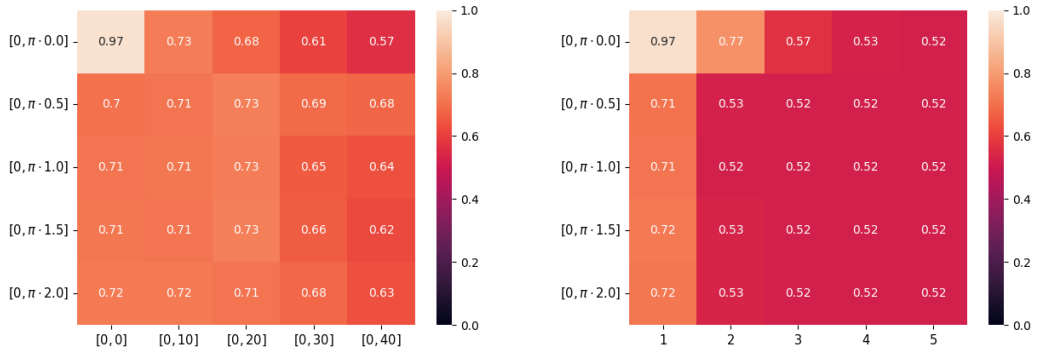
The general network architecture we used in the experiments is described in the previous section. We used $n = 5$ convolution blocks, where the first block outputs node embeddings of size 16 and the next 4 blocks output node embeddings of size 20. After the 5 convolution blocks, we applied a mean pool and passed the aggregated node features to a feedforward neural network. In the case of the 2 convolution blocks that feature invariance, we dropped the spatial information before proceeding with the feedforward network. This step was taken because the spatial information was equivariant, not

invariant, and we wanted to produce an invariant network architecture. The feed-forward network has 3 layers, 2 with output size 256 and one with output size 2 (the number of classes), and uses the ReLU (rectified linear unit) activation function.

Starting off with the baseline models, we will describe the experimental results for the network variant that used the edge convolution block introduced in [Wang et al., 2019]. The first test scenario involves randomly rotating and translating the input mesh in order to observe the performance degradation caused by these transformations. We can see the results of this test in figure 5.3a. The rows represent the range in which the mesh was randomly rotated and the columns the range in which it was randomly translated. When evaluating the examples from the test set without applying any geometric transformations, the network obtained an impressive 97% accuracy. As soon as we started shifting and rotating the inputs however, the performance dropped to 57 to 73%. A small rotation of less than 45 degrees was enough to drop more than 20 percentage points off the network's accuracy. The same effect is observed for just translating the model without applying any rotation. Applying a rotation and a translation at the same time does not seem to make the network perform worse than just rotating or translating the inputs. In the second scenario, the model was tested on inputs that were randomly rotated or scaled by a constant. The results of this test can be seen in figure 5.3b. The rows of the matrix represent the range in which the mesh was randomly rotated and the columns the constant by which they were scaled. Just rotating the inputs yielded a similar degradation as in the previous test case, but scaling the meshes proved to be an even bigger challenge for the model. Scaling the meshes by only 2 times dropped 20% off the model's performance, while scaling them more made the model degenerate to something close to randomness, making it completely useless as a result. Combining the rotations with the scaling only made things worse, as seen on the 2-nd column of the matrix.

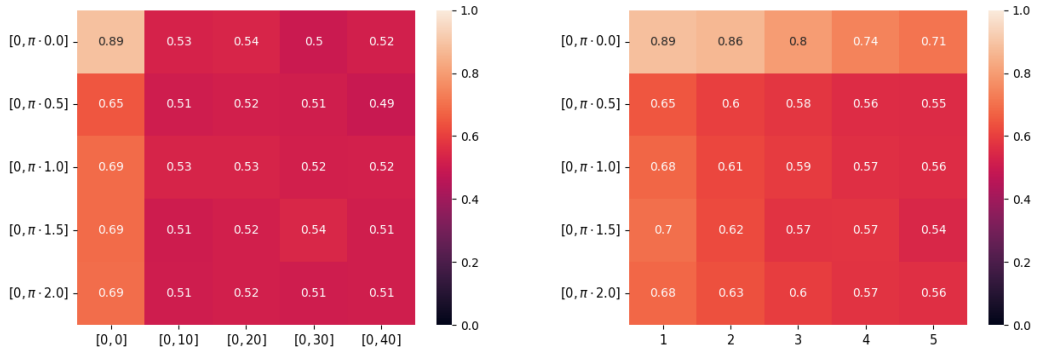
Moving on to the next network variant, we will examine the experimental results for the graph attention network block introduced in [Veličković et al., 2017]. The accuracy for the unchanged inputs in the test set is lower than that of the previous model, with just 89% instead of 97%. The results of the rotation and translation test for this model can be seen in figure 5.4a. Rotating the inputs yielded a lower performance degradation than in the previous case, but it is still a substantial one, ending up at approximately the same performance. Translating the inputs completely disorientated the model, making it degenerate to random choices. The results for the rotation and scaling test can be seen in figure 5.4a. The model performed badly here as well, but not as bad as the edge convolution model. Scaling the input up to 2 times yielded almost no performance degradation, but that didn't hold true for higher scaling constants. When applying both rotations and scaling the model also degenerated to random choice.

Having discussed the baseline models, we will now look at the models that are invariant to 2 or all 3 geometric transformations. We used the E(3) equivariant/invariant convolution block introduced in [Satorras et al., 2021] to build a network that is invariant to rotation and translation. The model's performance for the unchanged inputs was decently high, having an accuracy of 92%. The results for the rotation and translation test can be seen in figure 5.5a. As expected, the model's performance did not degrade when the inputs were transformed using operations from within its symmetry group. The performance stayed constant no matter by what amount we rotated or translated the inputs. Moving



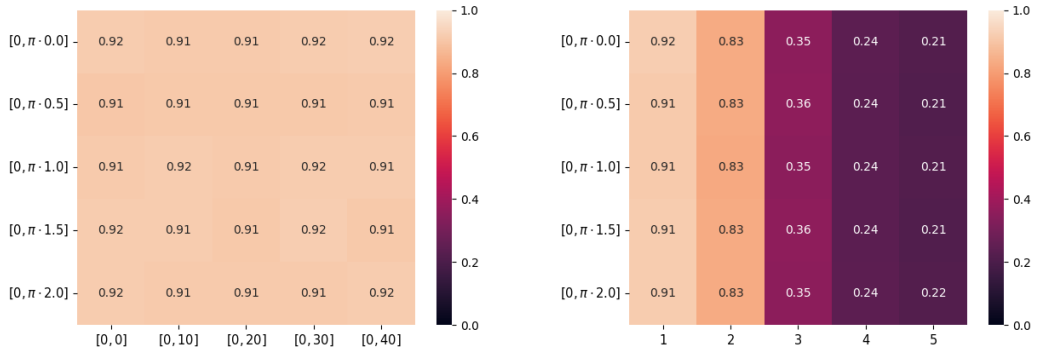
(a) random rotation & translation accuracy (b) random rotation & scaling accuracy

Figure 5.3: Edge convolution block, no invariance [Wang et al., 2019]



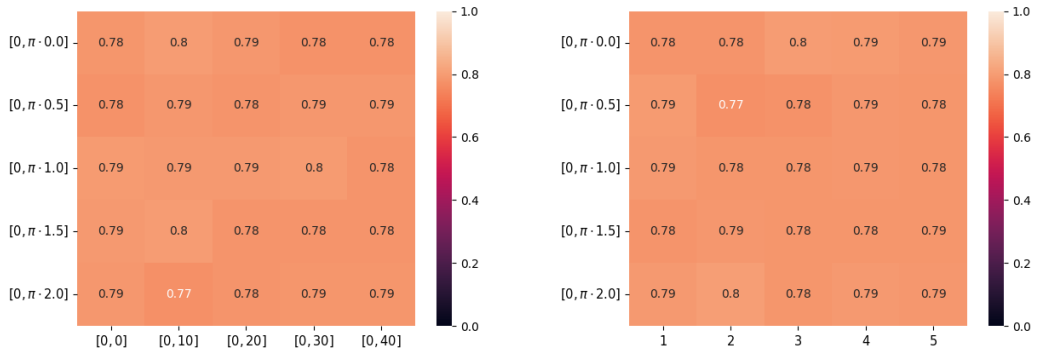
(a) random rotation & translation accuracy (b) random rotation & scaling accuracy

Figure 5.4: Graph attention block, no invariance [Veličković et al., 2017]



(a) random rotation & translation accuracy (b) random rotation & scaling accuracy

Figure 5.5: E(3) block, rotation and translation invariance [Satorras et al., 2021]



(a) random rotation & translation accuracy (b) random rotation & scaling accuracy

Figure 5.6: Our block, rotation, translation and scale invariance

forward to the other test case, the results of the rotation and scaling test can be seen in figure 5.5b. The network was able to cope with a scaling factor of up to 2 times by only incurring a 9% performance degradation. When increasing the scaling factor further however, the model started to predict the opposite class. This can be explained by the fact that the invariant feature used by the network is the length of the edges. By increasing these lengths by more than 3 times, the new domains of the edges might have intersected with the unscaled domains of the opposite class, having as a result repeated incorrect predictions.

Lastly, we will evaluate the model variant that uses the convolution block introduced in our work, a convolution block that is invariant/equivariant to rotation, translation and scaling at the same time. We only used the invariant outputs of the convolution layers to produce an invariant network. The performance obtained by the network on the unchanged inputs is 78%, which is slightly lower than the previous model, but still decently good. The drop of performance can be attributed to the loss of the scale information. By applying the proposed local, neighborhood-based normalization to the edge lengths, we lost sight of the global scale information. In figure 5.6a we can see the results for the rotation and translation test, and in figure 5.6b the results for the rotation and scaling test. The network maintained a constant performance across all input variations, demonstrating the correctness of our approach. The results of the network were the same no matter how the inputs were rotated, scaled or translated.

The experiments were implemented using python, pytorch and torch geometric. The training was conducted using the Adam optimizer with a batch size of 1 and the logit cross entropy loss function. The inputs were randomly split into a 60/20/20 training, cross-validation and test set. In order to run the code for the experiments, the following steps need to be taken:

- Install CUDA 12.1 and Python 3.8.10 or later.
- Download and unarchive the ShapeNetCore dataset, and set the `DATASETS_ROOT` environment variable to the parent directory of the dataset.
- Install the project dependencies:

    * `python3 -m venv venv`
    * `source venv/bin/activate`
    * `pip install -r requirements.txt`

- Train the models: `python3 main.py`
- Run the input transformation tests: `python3 test_random_transformations.py`

The output of the scripts will be stored in a directory called `data` in the project folder. Each model's weights and experimental results will be located in a separate subdirectory. The models are very computationally intensive. The training was conducted on an Nvidia GeForce GTX 1080 with 8 gigabytes of VRAM, so something at least that powerful is required to run the code. Moving on from the experimental results, we will continue by proving the correctness of our method in the next chapter.

Chapter 6

Invariance & Equivariance Proofs

In this chapter we will use the following notation:

Symbol	Meaning
s	spatial components of the convolution block's input
h	non-spatial components of the convolution block's input
f_s	the operation applied to compute the spatial components at the next layer
f_h	the operation applied to compute the non-spatial components at the next layer
R	any rotation matrix
t	any vector of same dimensionality as s
c	any scalar
V	the set of nodes in the graph
m_{ij}	the message passed from node j to node i
M_i	the maximum squared distance from node i to any of its neighbors
N_i	the neighbors of node i
$*^R$	any of the above after the rotation R was applied to the spatial input
$*^t$	any of the above after a translation by t was applied to the spatial input
$*^c$	any of the above after the spatial input was scaled by c
ϕ_*	a learnable non-linear function

6.1 Rotation Invariance & Equivariance

In this section we are going to prove the following:

- The transformations applied by any number of convolution blocks to the non-spatial components of the input nodes are rotation invariant.
- The transformations applied by any number of convolution blocks to the spatial components of the input nodes are rotation equivariant.

We will prove by induction that after any number of stacked convolution blocks, the value of s at the next layer $f_s(s, h)$ is rotation equivariant, and the value of h at the next layer $f_h(s, h)$ is rotation

invariant. We will achieve this by assuming that the input s is rotation equivariant and the input h is rotation invariant. Before proving that s and h preserve these properties at any layer, we first need to prove that this assumption holds for the inputs of the first layer. Because s is the geometric information of the input nodes, applying a rotation to the graph will be equivalent to applying a rotation to s, making s rotation equivariant at the first layer. The rest of the node information h is not changed by the rotation, so the input h is rotation invariant at the first layer. Moving forward we will prove that the spatial output of any layer $f_s(s, h)$ is rotation equivariant, and the non-spatial output of any layer $f_h(s, h)$ is rotation invariant, by using the assumption that the input s is rotation equivariant, and the input h is rotation invariant.

In order to prove that the operations applied to the non-spatial components are rotation invariant we need to show that $f_h(s, h) = f_h(Rs, h)$. Applying the convolution block to all the nodes in the graph is equivalent to applying it to one node at a time and concatenating the results, as seen in equation (6.1). The non-spatial output for a single node is computed using equation (6.2). For any node i, the message m_i is equal to the sum of all the messages passed from a node's neighbors to itself, as seen in equation (6.3). The message from a node j to a node i is computed using formula (6.4). We already assumed that h_i and h_j are rotation invariant, so in order to prove that m_{ij} is rotation invariant we only need to show that $\frac{||s_i - s_j||^2}{max(M_i, M_j)}$ has this property. The maximum squared distance within a node's neighborhood M_i is computed using formula (6.4).

$$f_h(s, h) = [f_h(s_i, h_i) | i \in V] \tag{6.1}$$

$$f_h(s_i, h_i) = \phi_n(h_i, m_i) \tag{6.2}$$

$$m_i = \sum_{j \in N_i} m_{ij} \tag{6.3}$$

$$m_{ij} = \phi_e \left(h_i, h_j, \frac{||s_i - s_j||^2}{max(M_i, M_j)} \right) \tag{6.4}$$

$$M_i = max(\{||s_i - s_j||^2, \forall j \in N_i\}) \tag{6.5}$$

We will begin by showing that M_i^R, the maximum squared distance under a rotation, is equal to the maximum distance in the unaltered input M_i. We obtain M_i^R by substituting s_i with Rs_i and s_j with Rs_j in equation (6.5). We can see that by using the property that the distance between 2 points is not altered by a rotation, $||x - y|| = ||Rx - Ry||$, we can easily deduce that $M_i^R = M_i$, as shown by equation (6.6).

$$M_i^R = max(\{||Rs_i - Rs_j||^2, \forall j \in N_i\}) = \max(\{||s_i - s_j||^2, \forall j \in N_i\}) = M_i \tag{6.6}$$

After proving that M_i is rotation invariant we will move on to show the same for m_{ij}. We obtain m_{ij}^R by replacing s_i and s_j with Rs_i and Rs_j, and by replacing M_i and M_j with M_i^R and M_j^R in equation (6.4). In equation (6.7) we made use of the property that the distance is preserved under rotation ($||x - y|| = ||Rx - Ry||$) and the equivalence of M_i^R to M_i and of M_j^R to M_j, to prove that $m_{ij}^R = m_{ij}$.

$$m_{ij}^R = \phi_e\left(h_i, h_j, \frac{||Rs_i - Rs_j||^2}{\max(M_i^R, M_j^R)}\right) = \phi_e\left(h_i, h_j, \frac{||s_i - s_j||^2}{\max(M_i, M_j)}\right) = m_{ij} \tag{6.7}$$

By using the rotation invariance of m_{ij}, it is easy to show that m_i is also rotation invariant, as seen in equation (6.8). Having proven that m_i is rotation invariant, we use the assumption that h_i is rotation invariant to demonstrate that the operation applied to a single node is rotation invariant as well, as seen in equation (6.9). By combining this result with definition (6.1) we can see that the non-spatial output of the convolution block is rotation invariant, as shown in equation (6.10).

$$m_i^R = \sum_{j \in N_i} m_{ij}^R = \sum_{j \in N_i} m_{ij} = m_i \tag{6.8}$$

$$f_h(Rs_i, h_i) = \phi_n(h_i, m_i^R) = \phi_n(h_i, m_i) = f_h(s_i, h_i) \tag{6.9}$$

$$f_h(Rs, h) = [f_h(Rs_i, h_i)|i \in V] = [f_h(s_i, h_i)|i \in V] = f_h(s, h) \tag{6.10}$$

To prove that the convolution block is rotation equivariant with regard to the spatial information we need to show that $f_s(Rs, h) = Rf_s(s, h)$. We begin by breaking down the entire transformation as a composition of the transformation applied to each input node, as seen in equation (6.11). The operations performed on a single node can be seen in equation (6.12). We have already proven that $m_{ij}^R = m_{ij}$, so by substituting the argument s with its rotated version we can see that applying a rotation to the input yields the same rotation to the output, as seen in equation (6.13).

$$f_s(s, h) = [f_s(s_i, h_i)|i \in V] \tag{6.11}$$

$$f_s(s_i, h_i) = s_i + \frac{1}{\text{card}(N_i)} \sum_{j \in N_i} (s_i - s_j)\phi_s(m_{ij}) \tag{6.12}$$

$$\begin{aligned}
f_s(Rs_i, h_i) &= Rs_i + \frac{1}{\text{card}(N_i)} \sum_{j \in N_i} (Rs_i - Rs_j)\phi_s(m_{ij}^R) \\
&= Rs_i + \frac{1}{\text{card}(N_i)} \sum_{j \in N_i} R(s_i - s_j)\phi_s(m_{ij}) \\
&= Rs_i + R\frac{1}{\text{card}(N_i)} \sum_{j \in N_i} (s_i - s_j)\phi_s(m_{ij}) \\
&= R\left(s_i + \frac{1}{\text{card}(N_i)} \sum_{j \in N_i} (s_i - s_j)\phi_s(m_{ij})\right) \\
&= Rf_s(s_i, h_i)
\end{aligned} \tag{6.13}$$

Taking this result and combining it with definition (6.11), we can see that applying a rotation to the input will give rise to the same rotation in the output, as shown in equation (6.14). This makes the operation applied to obtain the spatial output rotation equivariant.

$$f_s(Rs, h) = [f_s(Rs_i, h_i)|i \in V] = [Rf_s(s_i, h_i)|i \in V] = Rf_s(s, h) \tag{6.14}$$

Relations (6.10) and (6.14) were proven using the assumption that h is rotation invariant and s is rotation equivariant. We have already shown that this property holds for the first layer, and by proving that the same holds for layer $l+1$ by assuming these properties for layer l the induction proof is complete.

6.2 Translation Invariance & Equivariance

In this section we are going to prove the following:

- The transformations applied by any number of convolution blocks to the non-spatial components of the input nodes are translation invariant.
- The transformations applied by any number of convolution blocks to the spatial components of the input nodes are translation equivariant.

We will use a similar assumption as in the previous section in order to prove these properties by induction. We will assume that the input s is translation equivariant and the input h is translation invariant. Using this assumption we will prove that the value of s at the next layer $f_s(s, h)$ is translation equivariant and the value of h at the next layer $f_h(s, h)$ is translation invariant. Before moving on to proving that if these properties hold for layer l, they will also hold for layer $l + 1$, we will begin by proving them for the first layer. For the first layer, the value of s is the geometric information of the graph and the value of h is the non-geometric information. Applying a translation to the graph will cause s to be translated, which makes it translation equivariant. Applying a translation will not change h because it is not part of the geometric properties, which makes h translation invariant.

In order to prove that the convolution block is translation invariant with regard to the non-spatial components we need to show that $f_h(s, h) = f_h(s + t, h)$. In equation (6.15) we define the operation applied on the entire graph as the concatenated results of the same operation applied on each node individually. The result of each node operation is given by equation (6.16). The input h_i was already assumed to be translation invariant, so we only need to show that m_i is translation invariant. The value of m_i is equal to the sum of all the messages from the neighbors of node i to itself, as shown in equation (6.17). Each message is computed using formula (6.18). h_i and h_j have already been assumed to be translation invariant, so we have left to prove that the fraction $\frac{||s_i - s_j||^2}{max(M_i, M_j)}$ is translation invariant in order to prove the same for the entire message. The value of M_i is given by the maximum squared distance within node i's neighborhood, as seen in equation (6.19).

$$f_h(s, h) = [f_h(s_i, h_i)|i \in V] \tag{6.15}$$

$$f_h(s_i, h_i) = \phi_n(h_i, m_i) \tag{6.16}$$

$$m_i = \sum_{j \in N_i} m_{ij} \tag{6.17}$$

$$m_{ij} = \phi_e\left(h_i, h_j, \frac{||s_i - s_j||^2}{max(M_i, M_j)}\right) \tag{6.18}$$

$$M_i = max(\{||s_i - s_j||^2, \forall j \in N_i\}) \tag{6.19}$$

We will begin by showing that M_i is translation invariant. We obtain M_i^t by replacing s_i with $s_i + t$ and s_j with $s_j + t$ in equation (6.19). In equation (6.20) we can see that by applying this substitution the subtraction of s_j from s_i reduces the term t and produces the same result as M_i.

$$M_i^t = max(\{||(s_i + t) - (s_j + t)||^2, \forall j \in N_i\}) = max(\{||s_i - s_j||^2, \forall j \in N_i\}) = M_i \tag{6.20}$$

Having proven that M_i is translation invariant, we will continue to show the same for m_{ij}. We obtain m_{ij}^t by taking equation (6.18) and replacing s_i with $s_i + t$, s_j with $s_j + t$, M_i with M_i^t and M_j with M_j^t. We then use the result that M_i is equal to M_i^t to obtain an equivalence relation between m_{ij}^t and m_{ij}, as shown in equation (6.21). Like in the previous case, subtracting s_j from s_i annulled the term t and yielded the same result.

$$m_{ij}^t = \phi_e\left(h_i, h_j, \frac{||(s_i + t) - (s_j + t)||^2}{max(M_i^t, M_j^t)}\right) = \phi_e\left(h_i, h_j, \frac{||s_i - s_j||^2}{max(M_i, M_j)}\right) = m_{ij} \tag{6.21}$$

$$m_i^t = \sum_{j \in N_i} m_{ij}^t = \sum_{j \in N_i} m_{ij} = m_i \tag{6.22}$$

$$f_h(s_i + t, h_i) = \phi_n(h_i, m_i^t) = \phi_n(h_i, m_i) = f_h(s_i, h_i) \tag{6.23}$$

By proving that $m_{ij} = m_{ij}^t$, it naturally follows that $m_i = m_i^t$, as seen in equation (6.22). Using this result we go on to show that $f_h(s_i + t, h_i) = f_h(s_i, h_i)$ in equation (6.24). Using definition (6.15), we obtain the result that the convolution block is translation invariant with regard to the non-spatial output, with the assumption that s is translation equivariant and h is translation invariant, as seen in equation (6.24).

$$f_h(s + t, h) = [f_h(s_i + t, h_i) | i \in V] = [f_h(s_i, h_i) | i \in V] = f_h(s, h) \tag{6.24}$$

To demonstrate that the convolution block is translation equivariant with regard to the spatial information we need to show that $f_s(s + t, h) = f_s(s, h) + t$. We begin by defining the operation applied on the entire graph as the concatenated results of the operation applied on each node individually, as seen in

equation (6.25). The operation applied on a single node is defined in equation (6.26).

$$f_s(s, h) = [f_s(s_i, h_i)|i \in V] \tag{6.25}$$

$$f_s(s_i, h_i) = s + \frac{1}{card(N_i)} \sum_{j \in N_i} (s_i - s_j)\phi_s(m_{ij}) \tag{6.26}$$

We have already proven that m_{ij} is translation invariant. By substituting s_i with $s_i + t$, we can see in equation (6.27) that applying a translation to the input is equivalent to applying the same translation to the output, which makes the operation translation equivariant.

$$\begin{aligned} f_s(s_i + t, h_i) &= s_i + t + \frac{1}{card(N_i)} \sum_{j \in N_i} ((s_i + t) - (s_j + t))\phi_s(m^t_{ij}) \\ &= s + t + \frac{1}{card(N_i)} \sum_{j \in N_i} (s_i - s_j)\phi_s(m_{ij}) \\ &= t + f_s(s_i, h_i) \end{aligned} \tag{6.27}$$

Combining this result with equation (6.25) we can see that the spatial output of the convolution block is translation equivariant, based on the assumption that s is translation equivariant and h is translation invariant, as shown in equation (6.28).

$$f_s(s + t, h) = [f_s(s_i + t, h_i)|i \in V] = [f_s(s_i, h_i) + t|i \in V] = f_s(s + t, h) + t \tag{6.28}$$

Having already proven that h and s are invariant and respectively equivariant for the first layer, by proving that h is invariant and s is equivariant at layer $l + 1$ if the same is true for layer l, the induction proof is complete.

6.3 Scale Invariance & Equivariance

- The transformations applied by any number of convolution blocks to the non-spatial components of the input nodes are scale invariant.
- The transformations applied by any number of convolution blocks to the spatial components of the input nodes are scale equivariant.

We will use another induction proof to show that any number of convolutions are scale invariant with regard to the non-spatial output, and scale equivariant with regard to the spatial output. We will assume that the input h is scale invariant and the input s is scale equivariant, and we will prove that the values of

h and s at the next layer, given by the functions $f_h(s,h)$ and $f_s(s,h)$, are scale invariant and respectively equivariant.

In order to prove that the convolution is scale invariant with respect to the non-spatial information, we need to show that $f_h(s,h) = f_h(c \cdot s, h)$. We begin by breaking down the operation performed for the entire graph into the concatenated results of the same operation applied on every node of the graph, as seen in equation (6.29). The non-spatial output for a single node is given by relation (6.30). We already assumed that h_i is scale invariant, so we only need to prove the same for m_i. m_i is equal to the sum of all the messages received by node i from its neighbors, as seen in equation (6.31). Equation (6.32) defines the computation of the message from node j to node i. We assumed that h_i and h_j are scale invariant, so we only have to prove that the fraction is as well. The maximum squared distance M_i within node i's neighborhood is defined in equation (6.33).

$$f_h(s,h) = [f_h(s_i,h_i)|i \in V] \tag{6.29}$$

$$f_h(s_i,h_i) = \phi_n(h_i,m_i) \tag{6.30}$$

$$m_i = \sum_{j \in N_i} m_{ij} \tag{6.31}$$

$$m_{ij} = \phi_e\left(h_i, h_j, \frac{||s_i - s_j||^2}{max(M_i,M_j)}\right) \tag{6.32}$$

$$M_i = max(\{||s_i - s_j||^2, \forall j \in N_i\}) \tag{6.33}$$

We obtain the expression for M_i^c by substituting s_i with $c \cdot s_i$ and s_j with $c \cdot s_j$ in equation (6.33). We then proceed to rewrite M_i^c as an expression of M_i and c in equation (6.34).

$$\begin{aligned} M_i^c &= max(\{||c \cdot s_i - c \cdot s_j||^2, \forall j \in N_i\}) \\ &= max(\{c^2||s_i - s_j||^2, \forall j \in N_i\}) \\ &= c^2 max(\{||s_i - s_j||^2, \forall j \in N_i\}) = c^2 M_i \end{aligned} \tag{6.34}$$

$$\begin{aligned} m_{ij}^c &= \phi_e\left(h_i, h_j, \frac{||c \cdot s_i - c \cdot s_j||^2}{max(M_i^c, M_j^c)}\right) \\ &= \phi_e\left(h_i, h_j, \frac{||c \cdot s_i - c \cdot s_j||^2}{max(c^2 M_i, c^2 M_j)}\right) \\ &= \phi_e\left(h_i, h_j, \frac{c^2||s_i - s_j||^2}{c^2 max(M_i, M_j)}\right) \\ &= \phi_e\left(h_i, h_j, \frac{||s_i - s_j||^2}{max(M_i, M_j)}\right) = m_{ij} \end{aligned} \tag{6.35}$$

In equation (6.35), we obtain the message after scaling m_{ij}^c by replacing s_i with $c \cdot s_i$ and s_j with $c \cdot s_j$ in formula (6.32). We then use the previous result to substitute M_i^c with $c^2 M_i$ and M_j^c with $c^2 M_j$. The scaling constant c gets reduced in the fraction, resulting that $m_{ij}^c = m_{ij}$ no matter what scaling factor we choose.

Extending this result to the aggregated message, we can see that $m_i^c = m_i$, as shown in equation (6.36). We then use the invariance of the aggregated message and the invariance assumption of h to demonstrate that the operation applied to a single node is scale invariant, as seen in equation (6.37). We extend this result to prove that the non-spatial output for the entire graph is scale invariant, as shown in equation (6.38).

$$m_i^c = \sum_{j \in N_i} m_{ij}^c = \sum_{j \in N_i} m_{ij} = m_i \tag{6.36}$$

$$f_h(c \cdot s_i, h_i) = \phi_n(h_i, m_i^c) = \phi_n(h_i, m_i) = f_h(s_i, h_i) \tag{6.37}$$

$$f_h(c \cdot s, h) = [f_h(c \cdot s_i, h_i) | i \in V] = [f_h(s_i, h_i) | i \in V] = f_h(s, h) \tag{6.38}$$

To demonstrate that the convolution block is scale equivariant with regard to the spatial information, we need to show that $f_s(c \cdot s, h) = c \cdot f_s(s, h)$. We begin by breaking down the spatial output of the entire graph into the concatenated outputs of each inidivdual node in equation (6.39). The spatial output of a single node is computed using formula (6.40).

$$f_s(s, h) = [f_s(s_i, h_i) | i \in V] \tag{6.39}$$

$$f_s(s_i, h_i) = s_i + \frac{1}{card(N_i)} \sum_{j \in N_i} (s_i - s_j) \phi_s(m_{ij}) \tag{6.40}$$

In equation (6.41) we obtain the scaled spatial output by replacing s_i with $c \cdot s_i$, s_j with $c \cdot s_j$ and m_{ij} with m_{ij}^c in formula (6.40). Using the result that $m_{ij}^c = m_{ij}$, the assumption that h is scale invariant and s scale equivariant, we obtain the result that the spatial output of the convolution block is scale equivariant.

We have proven that the spatial output f_s is equivariant and the non-spatial output f_h invariant based on the assumption that the spatial input s is scale equivariant and the non-spatial input h invariant. In other words, by assuming the equivariance/invariance properties of the input we proved the same properties for the output. Taking into account that at the first layer h is invariant because it is not a geometric property, and s is equivariant because scaling the graph corresponds to scaling s, we have proven by induction that any number of convolution layers will produce an invariant non-spatial output

and an equivariant spatial-output.

$$
\begin{aligned}
f_s(c \cdot s_i, h_i) &= c \cdot s_i + \frac{1}{card(N_i)} \sum_{j \in N_i} (c \cdot s_i - c \cdot s_j) \phi_s(m_{ij}^c) \\
&= c \cdot s_i + \frac{1}{card(N_i)} \sum_{j \in N_i} c \cdot (s_i - s_j) \phi_s(m_{ij}) \\
&= c \cdot s_i + c \cdot \frac{1}{card(N_i)} \sum_{j \in N_i} (s_i - s_j) \phi_s(m_{ij}) \\
&= c \cdot \left(s_i + \frac{1}{card(N_i)} \sum_{j \in N_i} (s_i - s_j) \phi_s(m_{ij}) \right) \\
&= c \cdot f_s(s_i, h_i)
\end{aligned} \tag{6.41}
$$

Chapter 7

Conclusions

This thesis investigated the ways in which we can make use of the symmetry relationships within the context of 3 dimensional data to increase a model's robustness to geometric transformations. The main objectives of our work were the following:

- Find out the performance impact of applying rotations, translations or scaling to a model's input.
- Determine the performance drawback that comes with implementing invariance into a model, and weigh this drawback against the increased robustness of the model.
- Devise a method for obtaining scale invariant features.
- Extend previous work by adding scale invariance/equivariance to a rotation and translation invariant/equivariant model.

We met our first objective by testing the performance degradation of previously proposed models and our own against different combinations of rotations and translations, and rotations and scaling factors. The experimental results have shown that all of these 3 transformations can have a huge performance impact on models that do not implement invariance with regard to them. In some cases, the degradation was so severe that the models degenerated to random choice in predicting the output class. The most serious degradation however, was usually associated with scaling the input, not rotating or translating it. This demonstrated that the models can not generalize shapes across scale differences, adding to our argument for scale invariance. While other architectures failed to deal with the size variations, ours was not affected by the changes thanks to the scale invariance built into the model.

This series of experiments helped us answer our second research question, which posed the problem of weighing the performance cost associated with invariant features against the benefit of added model robustness. While the models that did not incorporate any invariance obtained performance scores of 89% to 97%, the model that incorporated rotation and translation invariance obtained a best-case performance of 92%. Adding scale invariance dropped the best-case performance to 78%. The more invariance types are added, the higher the performance drawback is. This makes sense, because adding invariance to a certain transformation is achieved by hiding any information that could be changed by the transformation from the model. This information loss is the main culprit of the performance drawbacks.

Contrasting this with the performance degradation suffered by the non-invariant models when the inputs were transformed using rotation, translation or scaling, it is obvious that these models can not generalize to these input variations. The performance dropped sharply after applying only a small transformation to the inputs, and degraded to randomness when the magnitude of the transformation was large enough. It is reasonable to conclude that invariance should be used if the inputs are expected to not be found in a canonical position, scale or orientation, and it should not be if we have guarantees about one or more of these factors.

Moving on to our third objective, we introduced a local, graph-neighborhood based normalization technique that served as the key ingredient in achieving scale invariance and equivariance. By only considering the proportion between an edge and the longest edge found 1 step away in the graph, we circumvented using the true sizes of the edges, while still keeping in mind the topology they form in a broader context. In contrast to global scale normalization, our approach is outlier resistant and can be used in contexts where multiple objects are present in a single scene. One shortcoming of our method is the loss of the scale information. While useful in some cases, this reduces the information that can be extracted from the graph, and naturally results in a slightly poorer performance. If the sizes of the objects are known to vary very little or not at all, it is better to not use our method.

Moving forward to our last objective, we incorporate the proposed local normalization into a model from the literature that already incorporates rotation and translation invariance/equivariance. We chose the model proposed in [Satorras et al., 2021] because it uses node distances as a rotation and translation invariant feature. This can naturally be extended with our local scale normalization method to obtain a model that is invariant/equivariant to rotation, translation and scaling at the same time. The extended model performed equally well when presented with unaltered inputs as with inputs that were rotated, scaled, or translated, demonstrating the correctness of our method experimentally. We have not only shown the effectiveness of our method by using experiments, but we have also formally proven the invariance and equivariance properties of the proposed solution.

Achieving these objectives expanded on previous work by both measuring the severity, and providing a solution for the performance degradation brought upon by scale differences between the training and evaluation inputs. Our approach maintains the strengths of previously proposed invariant and equivariant models while also adding the benefit of scale invariance/equivariance. This can be a considerable benefit in situations where the scale of the given inputs can be expected to vary by a large amount, or in situations where the true size of the inputs is not known.

A future improvement could be experimenting with different neighborhood sizes. Currently, when an edge is considered, our method takes into account the maximum length of the edges within a 1-step neighborhood from the end nodes. Using a neighborhood larger than 1 step away could give a better approximation for the relative scale of the edge being considered to the rest of the nearby edges. However, increasing the neighborhood size would also increase the sensitivity of the algorithms to outliers. An outlier edge will affect all edges that are n steps away, and this effect ripples further as the number of steps n increases.

One other direction for future work is implementing the neighborhood-based normalization method

in other models that are only invariant/equivariant to translation and rotation. The only requirement for our method to be applied there as well is that the model uses the distances between 2 nodes as the invariant feature. This distance can be then normalized using our approach just like we did for the model introduced in [Satorras et al., 2021].

Another promising direction is to take these findings from 3-dimensional space back to 2-dimensional images. We could construct graphs from images by representing image patches with graph nodes and connecting adjacent patches with edges in the graph representation. If an image could be embedded in such a graph, we could apply our rotation, translation and scale invariant graph neural network to process it. This would guarantee that rotating, scaling or shifting a given image would not affect the final output of the model.

The work we have done here could be looked at in a broader sense as making computers learn by using reference points. The human mind cannot work with absolute values, whenever we describe something as large or small, slow or fast, ugly or beautiful, we mean that it's one or the other in comparison with something else. No quality can exist in a void, we always need a reference point to compare it to. That reference point is usually implicit, being represented by what we are accustomed to, but it exists nevertheless. For example, a tall person in Romania might be short in Sweden. The person is the same, the only difference is that we use different reference points when assigning that quality. This is the approach we have taken when building our scale invariant feature. We only looked at how long the edge is in comparison to the longest edge nearby, not the actual value of the edge's length. This line of thinking could be used to build models that imitate us in other regards, not just when considering scale differences.

Bibliography

U. Brandes, L. C. Freeman, and D. Wagner. Social networks. 2013.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

Y. Chen, B. Fernando, H. Bilen, M. Nießner, and E. Gavves. 3d equivariant graph implicit functions. In *European Conference on Computer Vision*, pages 485–502. Springer, 2022.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

T. Engel. Basic overview of chemoinformatics. *Journal of chemical information and modeling*, 46(6): 2267–2277, 2006.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

T. Le, F. Noé, and D.-A. Clevert. Equivariant graph attention networks for molecular property prediction. *arXiv preprint arXiv:2202.09891*, 2022a.

T. Le, F. Noe, and D.-A. Clevert. Representation learning on biomolecular structures using equivariant graph attention. In *Learning on Graphs Conference*, pages 30–1. PMLR, 2022b.

Y.-L. Liao and T. Smidt. Equiformer: Equivariant graph attention transformer for 3d atomistic graphs. *arXiv preprint arXiv:2206.11990*, 2022.

G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

F. Riaz and K. M. Ali. Applications of graph theory in computer science. In *2011 third international conference on computational intelligence, communication systems and networks*, pages 142–145. IEEE, 2011.

S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. pearson, 2016.

V. G. Satorras, E. Hoogeboom, and M. Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

P. Shirley, M. Ashikhmin, and S. Marschner. *Fundamentals of computer graphics*. AK Peters/CRC Press, 2009.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

R. Wang, J. Peethambaran, and D. Chen. Lidar point clouds to 3-d urban models : a review. *IEEE Journal*

of Selected Topics in Applied Earth Observations and Remote Sensing, 11(2):606–627, 2018.

Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.