university of groningen / faculty of science and engineering

UNIVERSITY OF GRONINGEN,
THE NETHERLANDS

MASTER'S THESIS
ARTIFICIAL INTELLIGENCE

Bandgap prediction for molecular crystals using Geometric Deep learning

Avinash Pathapati
S3754715

August 27, 2020

supervised by:
Primary Supervisor: **prof. Dr. L.R.B. (Lambert) Schomaker** (Scientific Director, Artificial Intelligence and Cognitive Engineering (ALICE), University of Groningen)
Secondary Supervisor: **prof. Dr. Alexander Balatsky** (Professor, NORDITA)

Abstract

Deep Learning (DL) has achieved the state of the art results on many machine learning tasks starting from image classification, video processing to natural language understanding and speech recognition. The input data to the model in all of these tasks lie in the Euclidean space. However, there are many applications where the input data is not present in Euclidean space and are represented using graphs. For example, in chemistry, molecules are represented as graphs. A citation network can also be modelled as a graph where the nodes represent papers and an edge between the nodes indicate citation relationship. Graph Neural Networks (GNNs), i.e., Geometric Deep Learning concerns generalized convolution methods that can work on a non-linear structure like graphs. GNNs can also be applied on images where the pixels (nodes) are connected by neighbouring pixels.

GNNs are classified into spectral and spatial approaches depending on how the convolutions are defined. In this thesis, we have examined the performance of GNNs and SchNet (a continuous filter convolution) model on OMDB dataset. The dataset contains the band gap values of 12,500 3D organic molecular crystals calculated using Density Functional Theory (DFT) and the task is to predict the bandgap value given the structure of the molecule. Among all of the models tested, we found that our slightly modified version of SchNet achieved the Mean Absolute Error (MAE) of **0.28eV** which is better than the state of the art. In this SchNet model, we also identified the atoms that are significant in contributing to the bandgap value of the molecule. Finally, we built an ensemble of SchNet models which attained a slightly lower MAE of **0.268eV**. Although the GNN approaches that were tried did not improve the estimation accuracy, they still hold a promise in terms of improved explainability of results due to the graph-based nature of molecules.

Acknowledgements

I would like to express my sincere appreciation to my internal supervisor, Dr Lambert Schomaker for his guidance and support throughout the project. The feedback and ideas provided by Dr.Lambert during our meetings were truly insightful. I also would like to pay my special regards to Dr.Alexander Balatsky for devoting his valuable time for this project and also for providing feedback.
Finally, I would like to thank the Center for Information Technology of the University of Groningen for providing access to the Peregrine high performance computing cluster and also for their quick responses for any queries raised.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

In recent years, we observed that the application of Deep Learning (DL) has expanded to many fields. The reason is that DL models can learn complex relationships between a large number of attributes in successfully classifying data. Chemistry is one of the new fields where DL is being used for many sub-applications like structure prediction, material design, quantum chemistry and property prediction as explained in [7]. The specific problem which this thesis will be focused on is predicting the property of a crystalline material. Many of the properties of a crystalline material can be derived from its electronic structure. Learning this relationship is very important as it will enable us to combine the materials appropriately to obtain the desired output material. The amount of data available with respect to these materials increased significantly in recent years. So DL models can be quite promising in achieving better results in this specific task. [8] is one of the earlier machine learning methods which was quite successful in this task. However, this model computes representations of atoms which are rotation, translation and permutation invariant (which are necessary for successful identification) manually. The approach in [9] learns the representations automatically from the input data. Specifically, this paper employed "Graph Neural Networks" (GNNs) approach which can extract features for an atom based on its neighbourhood. But, all of these methods used the QM9 dataset which has molecules containing a small number of atoms. The Organic Materials Database (OMDB) is a complex dataset than QM9 containing a large number of atoms. [6] used an ensemble of 'SchNet' from [10] and "Kernel Ridge Regression with the Smooth Overlap of Atomic Positions (SOAP) kernel" methods to achieve the state of the art results on this dataset. In this thesis, different possible input representations and architectures of multiple GNNs currently available will be explored on OMDB dataset. The final goal is to find the model which produce the best results and also has the lowest computational complexity at the same time.

Recently, Convolution Neural Networks (CNN), a class of deep learning models have achieved the state of the art results in object classification tasks. They achieve this by extracting higher-level features automatically from the pixels that are necessary to identify the required objects in images. Deeper CNN architectures [11] and [7] were able to achieve state of the art results on the difficult ImageNet classification benchmark. However, CNNs can be applied directly on images defined in Euclidean space but generalize poorly on non-linear data structure like graphs. Graph Neural Networks, i.e., Geometric Deep learning (coined by Michael Bronstein in [12]) concerns techniques designed to generalize the convolutions to work on non-Euclidean domains. These techniques can be classified into Spectral and Spatial approaches which mainly differ in the way how convolutions are defined. We used the PyTorch Geometric Library [13] to implement GNNs in this thesis.

1.1.1 Spectral Graph Approaches

In Spectral approaches [14], convolutions are defined in the frequency domain as a filter on the spectral components of the graph laplacian. The first Graph CNN based on this theory is presented in [14]. This model generalized poorly and over-fitted the input graph structure seen during training. The reason is that the filters learnt in the convolution layer are dependent on the eigenvalues of the graph. Better convolution kernels were later introduced in ([15] and [16]) to alleviate this problem. However, these filters were still domain dependent. The current state of the art method in this approach is GraphConvNets (GCN by [17]) which is explained in detail in 3.2.2. This method simplified the Chebyshev polynomial (used as a kernel) to order 2 since it minimized the over-fitting issue.

1.1.2 Spatial Graph Approaches

Spatial graph neural networks define convolutions directly in the Euclidean domain. These models can capture the local position relations between the nodes and generalize better to graphs of different shapes. So these models can be quite effective in predicting the property of crystalline material based on the underlying structure of it. GeodesicCNN [18] is one of the initial approaches which finds local polar coordinates around a point and Gaussian kernels (with learnable weights) is computed with respect to each of the bins for these local coordinates. [19] proposed using anisotropic heat kernels which produced better results than the existing spatial approaches on benchmark datasets. Monet [20] also used the Gaussian kernels but with learnable mean vector and covariance matrix. SplineCNN [21], the current state of the art approach in many applications uses B-spline convolution kernel. A more recent approach Graph Attention Networks in [22] added a self-attention mechanism to the existing convolution kernels and this approach matched the state-of-the-art performance while being computationally efficient. However, this approach is for node-classification and has to be extended to graph classification.

1.1.3 Graph Neural Networks in Chemistry

In Chemistry, [23] is one of the earliest papers which proposed CNN directly on graph structures (molecules) for learning molecular fingerprints. Approaches before this used hash function to compute circular fingerprint values of each atom whereas the approach in [23] estimates the hash function using a single layer neural network. [24] defined graph convolutions directly on molecule features such as atoms, bond order and didn't use any molecular fingerprints. However, this approach was tested only on small size molecule dataset. [22] defined Message Passing Framework (MPNN) for graphs which can describe many of the popular GNNs and the small variations made to this proposed framework demonstrated very good performance on the QM9 dataset. [22] also mentioned that these methods have to be tested on molecular datasets that are much more complex than QM9. [10] proposed continuous filter convolutional layers in SchNet neural network which can work on the non-grid structure and achieved state of the art results on 'QM9' dataset. [6] trained this SchNet model on OMDB dataset and tested on "Crystallography Open Database (COD)" dataset. This model has achieved the best results so far on this dataset.

1.1.4 Pooling methods

Designing a pooling strategy is also an active area of research in GNNs. An effective pooling strategy outputs a representation that has low computational complexity, permutation invariant while retaining the useful information. The primitive and effective way of pooling is the mean/max/sum pooling of the graph. The disadvantage of this simple technique is that the output representation has a fixed size irrespective of the size of the graph. Ideally, the pooled output should scale with the size of the input representation. Set2Set pooling [25] generates pooled output that increases with the input size. A detailed description of this method is presented in Section 3.2.4. [16] proposed a different approach where the graphs are coarsened into multiple levels using the Graclus algorithm ([26]). Then a binary search tree is constructed where the ordering of the nodes at lower levels is done based on the arbitrary order of nodes chosen at the coarsest level. This preserves the regular ordering among the nodes at the initial level. Then max-pooling is performed on this reordered graph which is more efficient than pooling the original graph. All of these approaches discussed so far perform pooling based on node features but does not take the graph structure into consideration. The SAG pooling method considers both node features and topological information of the graph and performs pooling using the self-attention method. More details on this method can be found in Section 3.2.5.

1.2 Research Question

The main aim of this study is to examine the performance of the state of the art GNN approaches on complex molecular datasets. Particularly, we will investigate the performance of different Graph Neural Network models on OMDB dataset. We also attempt to modify the SchNet architecture in the process of improving the performance while making sure that the computational complexity is not increased. The specific questions that will be answered are as follows.

- Experimenting with different architectures, parameter values of the SchNet model (which is the baseline method) and finding the model with the lowest MAE.
- Experimenting with the state of the art methods in GNNs (both spectral and spatial domains) and finding the model with the lowest MAE.
- Experimenting with different pooling methods that are currently available for GNNS and finding the pooling method that gives the best results.

- Finding the model or an ensemble of models with the best performance while being computationally optimal.
- Interpreting the results of the best model and identifying the atoms that are significant in contributing to the bandgap value of the molecule.

1.3 Outline

The rest of the thesis is structured as follows.

- **Theoretical background**: This section provides a detailed description of the Spectral Graph theory necessary to understand the Spectral Graph Neural Networks (which is a relatively new field). Also, a brief introduction to the Convolution Neural Networks (CNNs) is provided.
- **Methodology**: This section describes the GNN methods used in this thesis and also the OMDB dataset.
- **Experiments and Results**: In this section, the experimental setup as well as the results of the experiments conducted will be described in detail.
- **Discussion**: In this final section, an attempt is made to answer the research questions posed earlier and also an interpretation of the results is performed along with a discussion of possible directions for future work.

2 Theoretical Background

In this chapter, we provide a brief background on CNNs that achieved the state of the art results on image classification tasks. CNNs are introduced by LeCun et al.[27] that operate on inputs in the Euclidean domain. The Neocognitron architecture proposed by Fukushima et al. [28] has a similar architecture of CNN but it is trained using a layer-wise unsupervised clustering algorithm. Understanding the workings of CNN will provide a basic foundation for comprehending the working of GNNs on non-Euclidean domains. Also, we provide brief information on spectral graph theory which is necessary to understand the working principles of spectral graph neural networks.

2.1 Convolution Neural Networks (CNNs)

A CNN consists of a stack of layers that takes the input signal or image (2-dimensional grid as input) and produces an output (probability score for each class label) through a sequence of steps. Figure 1 shows an example of a CNN model that is used for recognizing the digits (0-9) when an image containing a digit is presented as input. A typical CNN contains mainly three components which are described in detail below. The mathematical expressions used in this subsection are based on [4].

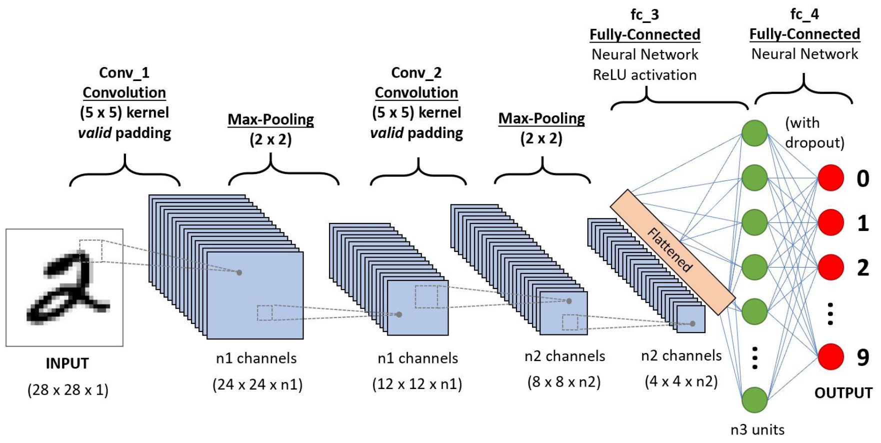


Figure 1: An example of a CNN model from [1]

2.1.1 Convolution

This is the most important block in a CNN where the feature extraction from input happens through convolution operation. The parameters of each convolutional layer are a set of learnable filters that operate on a small region (also known as receptive field). Each 3-D filter will have chosen height and width (generally less than that of image height and width) but same depth as that of the image. During the forward propagation,

the filter is slid across the height and width of the image and a dot product is computed between the values of the filter and the pixels present in the window of the image. The output of the filter is a 2-dimensional grid known as the activation map. In this way, the network learns filters that activate when they identify a particular feature at some arbitrary position in the input. There will be a stack of these filters placed along the depth dimension in each layer and the activation maps produced constitute the output volume.
Let the convolution layer be denoted by g = $\mathcal{C}_\Gamma$(f) operating on n-dimensional input $\mathbf{f}(x) = (f_1(x),\ldots,f_n(x))$ using a set of filters $\Gamma = \left(\gamma_{l,l'}\right), l = 1,\ldots,n$ denotes n-dimensional input, $l' = 1,\ldots$ denotes the q-dimensional output and ξ denotes the non-linear activation function, the output of the convolution layer is computed as shown below.

$$g_{l'}(x) = \xi\left(\sum_{l=1}^{n}\left(f_l * \gamma_{l,l'}\right)(x)\right)$$

Some of the key features of CNNs are "Sparse Connectivity" and "Parameter sharing".

Local (Sparse) Connectivity
Multi-Layer Perceptron models (MLP) use matrix multiplication in each layer where the interaction between each input and output has a separate parameter. However, CNNs typically have sparse interactions. This is achieved by having kernel/filter size less than the input size. For example, when the input is an image, it might have a large number of pixels, but the key features such as edges might be present in a relatively small number of pixels. Hence a small-sized kernel is enough to recognize the features in the images. This also greatly reduces the number of parameters to store in the memory and improves the statistical efficiency of the model as well.

Weight/Parameter sharing
If the parameters are used more than once in the model, it is called weight sharing. Parameter sharing reduces the total number of parameters in the CNN architecture. Intuitively, in a CNN, it also makes sense to share the filter (parameters) across the image as the key features like edges can be present at any region in the image and the same kernel can be used to identify such locations. Hence parameter sharing causes the layers to have translation invariance property. An example of the convolution with a 2-D image is shown in Figure 2.

2.1.2 Pooling

Generally, a pooling layer will be present in between successive convolution layers. The purpose of pooling layers is to reduce the spatial size of the representation, reduce the number of parameters and also minimize over-fitting. Pooling operation also provides translation invariance to the CNN. There are many non-linear functions that can be used for pooling (like max, average). Among them, the most commonly used is Max pooling.

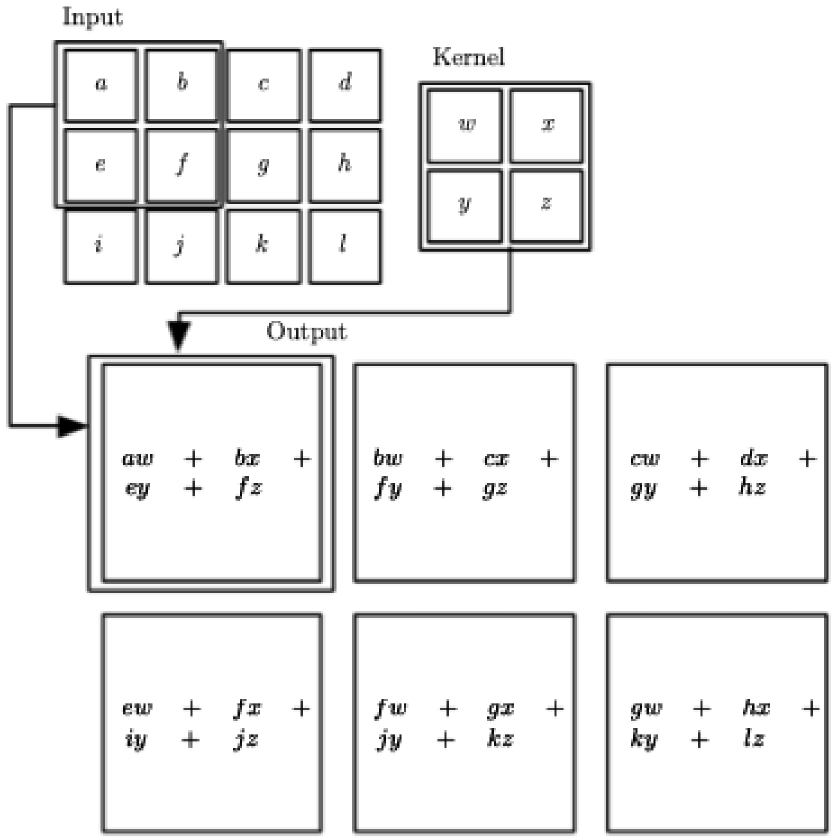


Figure 2: An example of convolution in CNN taken from [2]

The pooling operation is applied independently along each depth of the input. An example of a pooling layer shown in Figure 3 where a filter of size 2×2 with a stride of 2 reduces each depth by half, ignoring 75% of the activations.

General pooling layer
A generic pooling layer can be defined as follows

$$g_{l'}(x) = P\left(\left\{f_l\left(x'\right) : x' \in \mathcal{N}(x)\right\}\right), \quad l = 1, \ldots, n$$

where $\mathcal{N}(x)$ is neighbourhood around x and P is any permutation-invariant function (average or max pooling).

2.1.3 Activation layer

The output of a convolution layer is sent to activation layer before pooling is applied. In this section, we will describe some of the most commonly used activation functions.

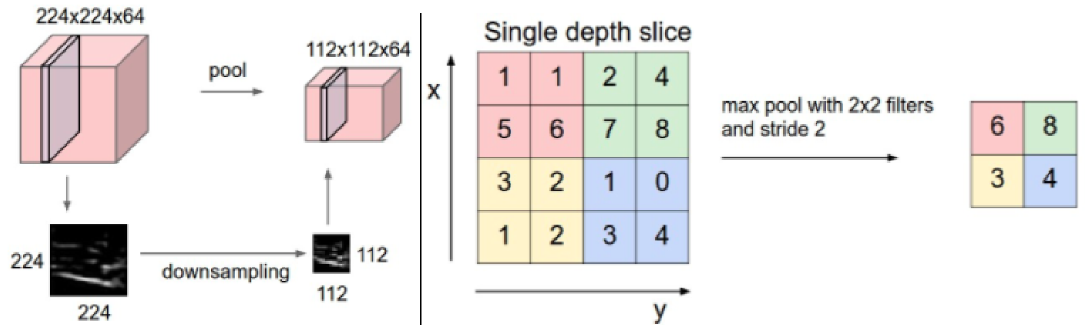


Figure 3: The image is taken from [3]. In the left pane, the input volume of size $224 \times 224 \times 224$ is downsampled (pooled) into output volume $112 \times 112 \times 64$ using a filter of size 2×2 with stride 2. We can see that the depth of the image remains the same in both input and output. In the right pane, Max pooling on a single depth of an image is shown. Each output value is the maximum taken over 2×2 square

Rectified Linear Unit (ReLU)
The ReLU function on an input x is defined as shown below

$$\mathrm{ReLU}(x) = \max(0, x)$$

LeakyReLU is slightly different from ReLU which has a small, positive gradient when the unit is inactive.

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

Sigmoid function

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

2.1.4 Fully Connected layer (MLP)

The final layers in a CNN are usually fully connected. These layers together perform the complex reasoning (based on the features extracted from the convolution layers) in successfully predicting the correct output value. The neurons in these layers are connected to all of the activations in the previous layer. Hence the activation of neurons in this layer is computed with a matrix multiplication before adding a bias value.

2.2 Spectral Graph Theory

In this section, we will describe the important concepts in Fourier analysis and how these ideas can be utilized to build graph convolution models in the spectral domain.

Also, the relation between the filtering in the spectral domain and the vertex domain (spatial domain) is derived. [4] provided a very nice introduction on the spectral graph theory and this section is a revised version of the same.

2.2.1 Graph representation and Basic notations

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ denote an undirected connected graph with a set (denoted by $\mathcal{V}$) of n vertices, set of edges $\mathcal{E}$ and a weighted adjacency matrix $\mathbf{W}$. Each entry in the weight matrix w_{ij} denotes the weight of the edge between the nodes i and j. If there is no edge between the nodes, the corresponding entry w_{ij} is 0. The weight matrix is called Adjacency matrix (denoted by $\mathcal{A}$) if the entry w_{ij} = 1 when there is an edge present between the nodes i ,j and 0 otherwise.
In the scenarios where the weight matrix is not provided upfront, one possible approach is to define the edge weight between any two vertices i and j using a Gaussian weighting function as shown below

$$W_{ij} = \begin{cases} \exp\left(-\frac{[dist(i,j)]^2}{2\theta^2}\right) & \text{if dist}(i,j) \leq \mathcal{K} \\ 0 & \text{otherwise} \end{cases}$$

Here θ and $\mathcal{K}$ are chosen parameters. $dist(i, j)$ represent the distance between vertices i and j. The distance value can be either Euclidean distance between the feature vectors of i and j or the physical distance between i and j. There are various ways in which the graphs (weighted adjacency matrix) can be constructed and details about these methods can be found in [29].
Let $f : \mathcal{V} \to \mathbb{R}$ be a vertex function defined on the vertices of the graph and is represented by a vector $\mathbf{f} \in \mathbb{R}^N$ where the i^{th} component of the vector represents the value of f at i^{th} vertex in $\mathcal{V}$. An example of a signal (vertex function) defined on a graph is shown in Figure 4.

2.2.2 Graph Laplacian

The non-normalized graph laplacian is defined as $\Delta = \mathbf{D} - \mathbf{W}$. Here $\mathbf{D} = \text{diag}\left(\sum_j W_{ij}\right)$ is the degree matrix and $\mathbf{W}$ is the weight matrix. For any arbitrary signal $\mathbf{f} \in \mathbb{R}^N$, the graph laplacian can be seen as a difference operator as shown below

$$(\Delta f)(i) = \sum_{j \in \mathcal{N}_i} W_{ij}[f(i) - f(j)]$$

This equation can be derived easily from the definition of Δ and f. Here $\mathcal{N}_i$ is the immediate neighbours (adjacent vertices) of vertetx i.
Since the graph laplacian is a real symmetric matrix (un-directed graph), there exists a set of orthogonal eigenvectors denoted by $\Phi = (\phi_1, \ldots, \phi_n)^1$. These eigenvectors have corresponding non-negative, real eigenvalues $\lambda_1, \ldots, \lambda_n$ which are the solutions of the equation $\Delta\phi_i = \lambda_i\phi_i$ for $i = 1, 2, \ldots, N$. Zero is the minimum eigenvalue with frequency

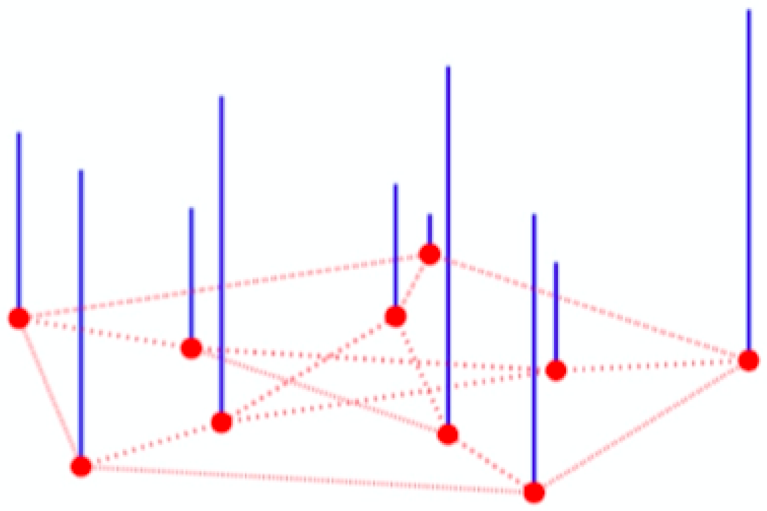


Figure 4: An example of a signal defined on vertices of a graph from [4]

equal to the number of connected components of the graph [30]. Hence the range of the eigenvalues will be $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N := \lambda_{\max}$.

Normalized graph laplacian In normalized graph laplacian, each weight w_{ij} is normalized by value of $\frac{1}{\sqrt{d_i d_j}}$ where d_i, d_j represents the degree of vertex i and j respectively. The normalized graph laplacian denoted by $\tilde{\Delta}$ is defined as follows

$$\tilde{\Delta} = \mathbf{D}^{-1/2} \Delta D^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

The eigenvalues of the normalized graph laplacian will be in the range $0 \leq \tilde{\lambda}_1 \leq \tilde{\lambda}_2 \leq \ldots \leq \tilde{\lambda}_{\max} \leq 2$. $\tilde{\lambda} = 2$ when $\mathcal{G}$ is bipartite; in a bipartite graph, the set of vertices in $\mathcal{V}$ can be divided into two disjoint subsets such that every edge $e \in \mathcal{E}$ connects one vertex from set 1 to the vertex from set 2. In general, normalized graph laplacian eigenvectors are used as filtering basis when defining graph convolutions as their eigenvalues are constrained to lie in the interval [0,2].

Eigendecomposition of Δ or $\tilde{\Delta}$: As both Δ or $\tilde{\Delta}$ are symmetric and positive semi-definite matrices, they have an eigendecomposition $\Delta = \Phi \bigwedge \Phi^T$ where diagonal matrix $\bigwedge = diag(\lambda_1 \ldots \lambda_N)$

2.2.3 Fourier Transform

The Fourier transform that decomposes a signal into the frequency components is defined as follows

$$\hat{f}(\xi) := < f, e^{2\pi i \xi t} > = \int_{\mathbb{R}} f(t) e^{-2\pi i \xi t} dt$$

From the equation, we can see that Fourier transform is the expansion of signal (in time domain) in terms of complex exponential values.

Graph Fourier Transform : Similarly, the graph Fourier transform $\hat{f}$ of function $\mathbf{f} \in \mathbb{R}^N$ on the vertices of $\mathcal{G}$ can be defined as the expansion of **f** in terms of the eigenvectors of the graph Laplacian as shown below:

$$\hat{f}(\lambda_l) :=< \mathbf{f}, \phi_l >= \sum_{i=1}^{N} f(i)\phi_l^*(i)$$

Rewriting this in terms of matrix representation

$$\hat{\mathbf{f}} = \Phi^T \mathbf{f}$$

Inverse Graph Fourier Transform : The inverse graph Fourier is given by the following equation

$$f(i) = \sum_{i=1}^{N} \hat{f}(\lambda_i)\, \phi_l(i)$$

and the equivalent matrix vector notation is

$$\mathbf{f} = \Phi \hat{\mathbf{f}}$$

Properties of the graph with respect to the eigenvalues : The eigenvalues can be interpreted as frequencies in classical Fourier analysis. For smaller frequency values, the corresponding eigenfunctions are smooth, slowly oscillating functions. In case of high frequency values, the respective eigenfunctions change more swiftly across the graph. Similarly, in case of graphs, the graph laplacian eigenvalues are equivalent to frequency of a signal. The eigenvectors of the graph laplacian associated with small eigenvalues (frequencies) vary slowly across the graph; if a pair of vertices are connected by edge with large weight, the eigenvalues at these vertices will be almost similar. For example, the laplacian eigenvector (Φ_1) corresponding to the 0 eigenvalue has constant value (equal to $1/\sqrt{N}$) for the entire graph. The eigenvectors corresponding to larger eigenvalues oscillate more swiftly and are likely to have different values at vertices connected by an edge having large weight.

2.2.4 Signal smoothness

In this section, we describe the smoothness property of signal with respect to the intrinsic structural information (W) of the graph.

Edge derivative: The edge derivative of signal f with respect to edge e = (i,j) is

$$\left.\frac{\partial \mathrm{f}}{\partial e}\right|_i := \sqrt{W_{ij}}[f(j) - f(i)]$$

Graph gradient: The graph gradient of f with respect to vertex i is the vector

$$\nabla_i \mathrm{f} := \left[\left\{ \frac{\partial \mathrm{f}}{\partial e}\bigg|_i \right\}_{e \in \mathcal{E} \text{ s.t } e=(i,j) \text{ for some } j \in \mathcal{V}} \right]$$

Using the above definitions, we can now define the **local smoothness** of f at vertex i as follows

$$\|\nabla_i \mathbf{f}\|_2 := \left[\sum_{e \in \mathcal{E} \text{ s.t } e=(i,j) \text{ for some } j \in \mathcal{V}} \left(\frac{\partial \mathbf{f}}{\partial e} | \right)^2 \right]^{\frac{1}{2}}$$

$$= \left[\sum_{j \in \mathcal{N}_i} W_{ij} [f(j) - f(i)]^2 \right]^{\frac{1}{2}}$$

Defining **global smoothness** of f (known as discrete p-Dirichlet form) using the definition of local smoothness (defined around a vertex) as follows

$$S_p(\mathbf{f}) := \frac{1}{p} \sum_{i \in \mathcal{V}} \|\nabla_i \mathbf{f}\|_2^p = \frac{1}{p} \sum_{i \in \mathcal{V}} \left[\sum_{j \in \mathcal{N}_i} W_{ij} [f(j) - f(i)]^2 \right]^{\frac{p}{2}}$$

When p=1,$S_1(\mathbf{f})$ is the total variation of the signal with respect to the graph. For p=2, the equation becomes

$$S_2(\mathbf{f}) = \frac{1}{2} \sum_{i \in \mathcal{V}} \| \nabla_i \mathbf{f}\|_2^p = \frac{1}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} W_{ij} [f(j) - f(i)]^2 = \sum_{i,j \in \mathcal{E}} W_{ij} [f(j) - f(i)]^2 = \mathbf{f}^\top \Delta \mathbf{f} \tag{1}$$

$S_2(\mathbf{f})$ is also called graph Laplacian quadratic form [31].
The Courant-Fischer theorem [32] states that the eigenvalues and eigenvectors can be found iteratively using the Rayleigh quotient as follows

$$\lambda_1 = \min_{f \in \mathbb{R}^N} \left\{ \mathbf{f}^\top \Delta \mathbf{f} \right\}$$
$$\|\mathbf{f}\|_{2=1}$$

$$\lambda_l = \min_{f \in \mathbb{R}^N} \left\{ \mathbf{f}^\top \Delta \mathbf{f} \right\}, l = 2, 3, \ldots, N$$
$$\|\mathbf{f}\|_{2=1} \tag{2}$$
$$\mathbf{f} \perp \operatorname{Span}(\phi_1, \ldots, \phi_N)$$

Here the eigenvector ϕ_l is the solution to the l^{th} minimization problem. From equation 1 and 2, it is clear that the eigenvectors of smaller eigenvalues vary smoothly across the graph. Hence the global smoothness value of the graph is kept as minimal as possible for smaller eigenvalues.

2.2.5 Filtering

In this section, we will describe the frequency filtering on signals in the frequency domain (converted using Fourier transform) and later extend this concept to graphs. Also, the localized filtering in the spatial domain is discussed.

Frequency filtering for signals
In classical signal processing, frequency filtering is performed by converting the signal into the frequency domain using Fourier transform, then amplifying or attenuating certain frequencies and finally converting the signal back to the time domain using inverse Fourier transform. The filtering of a signal in the frequency domain is shown below.

$$\hat{f}_{\text{out}}(\xi) = \hat{f}_{\text{in}}(\xi)\hat{h}(\xi)$$

where $\hat{h}(\xi)$ is the filter transfer function. The inverse Fourier transform of $\hat{f}_{\text{out}}(\xi)$ is equivalent to convolution in time domain as shown below.

$$\begin{aligned} f_{\text{out}}(t) &= \int_{\mathbb{R}} \hat{f}_{in}(\xi)\hat{h}(\xi)e^{2\pi i\xi t}d\xi \\ &= \int_{\mathbb{R}} f_{in}(\tau)h(t-\tau)d\tau := (f_{\text{in}} * h)(t) \end{aligned}$$

Spectral filtering of graphs We can extend the signal filtering definition to filtering in graphs as follows

$$\hat{f}_{out}(\lambda_l) = \hat{f}_{in}(\lambda_l)\hat{h}_{in}(\lambda_l) \tag{3}$$

where λ_l corresponds to the eigenvalue of the graph laplacian Δ. Taking an inverse Fourier transform of this equation will result in $f_{out}(i)$ in spatial domain as shown

$$f_{out}(i) = \sum_{l=1}^{N} \hat{f}_{in}(\lambda_l)\,\hat{h}(\lambda_l)\,\phi_l(i) \tag{4}$$

In matrix form, the equations 3 and 4 can be written as $f_{out} = \hat{h}(\Delta)f_{in}$ where

$$\hat{h}(\Delta) := \Phi \begin{bmatrix} \hat{h}(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{h}(\lambda_N) \end{bmatrix} \Phi^{\top}$$

Filtering in the Spatial Domain
The final output $f_{out}(i)$ at vertex i is linear combination of the input signal values at vertices within K-hop local neighbourhood of i.

$$f_{out}(i) = b_{i,i}f_{in}(i) + \sum_{j \in \mathcal{N}(i,K)} b_{i,j}f_{in}(j) \text{ for some constants } \{b_{i,j}\}_{i,j\in\mathcal{V}}$$

Relation between filtering in graph spectral domain and vertex domain
When the frequency filter is a K order polynomial $\hat{h}(\lambda_l) =$

$\sum_{k=0}^{K} a_k \lambda_l^k$ for some constants $\{a_k\}_{k=0,1,\ldots,K}$, the equation 4 becomes

$$\begin{aligned}
f_{\text{out}}(i) &= \sum_{l=1}^{N} \hat{f}_{\text{in}}(\lambda_l)\,\hat{h}(\lambda_l)\,\phi_l(i) \\
&= \sum_{j=1}^{N} f_{\text{in}}(j) \sum_{k=0}^{K} a_k \sum_{l=1}^{N} \lambda_l^k \phi_l^*(j)\phi_l(i) \\
&= \sum_{j=1}^{N} f_{\text{in}}(j) \sum_{k=0}^{K} a_k \left(\Delta^{\mathrm{k}}\right)_{i,j}
\end{aligned}$$

Since $\left(\Delta^{\mathrm{k}}\right)_{i,j} = 0$ when the shortest path between the vertices i and j is greater than k, the above equations becomes

$$f_{out}(i) = b_{i,i} f_{in}(i) + \sum_{j \in \mathcal{N}(i,K)} b_{i,j} f_{in}(j) \text{ for some constants } \left\{b_{i,j}\right\}_{i,j\in\mathcal{V}} \tag{5}$$

We can clearly see that the filtered signal at vertex i ($f_{out}(i)$) is a linear combination of the signal values defined at vertices within a K-hop local neighborhood of vertex i.

2.2.6 Graph Convolution

The graph convolution in terms of graph laplacian eigenvectors is defined as

$$(f * h)(i) := \sum_{l=1}^{N} \hat{f}(\lambda_l)\,\hat{h}(\lambda_l)\,\phi_l(i)$$

Rewriting the same in matrix notation

$$\mathbf{f} * \mathbf{h} = \Phi\, diag\left(\hat{h}(\lambda_1), \ldots, \hat{h}(\lambda_N)\right) \Phi^\top \mathbf{f} = \hat{h}(\Delta)\mathbf{f}$$

3 Methodology

3.1 OMDB-GAP1 dataset

Bandgap refers to the energy difference between the top of the valence bond and the bottom of the conduction band in a material. It is a characteristic of the material and combining the design of a material with the optimal bandgap value can lead us to finding new electronic devices. The dataset [5] contains the band gap values of 12500 3D organic molecular crystals. Among all of the materials in OMDB, only the materials with a calculated magnetic moment of less than 10^{-4} micros were considered in this dataset. More information on this dataset can be found in [33]. The structural information of the molecules is taken from the Crystallographic Open Database (COD) and the bandgap values were calculated in the DFT framework by applying the Vienna Ab initio Simulation Package (VASP). The dataset can be downloaded from "https://omdb.mathub. io/dataset".
The significance of OMDB-GAP1 dataset is that it has more complex molecules than the existing popular molecular datasets. Table 1 contains the statistics of the commonly used molecular datasets. $\bar{N}$ refers to the average number of atoms in a molecule. Consistency means whether all molecules come from identical computational setup ('Y' indicates yes and 'N' indicates No). Size indicates the size of the dataset.

Table 1: Statistics of common molecular datasets

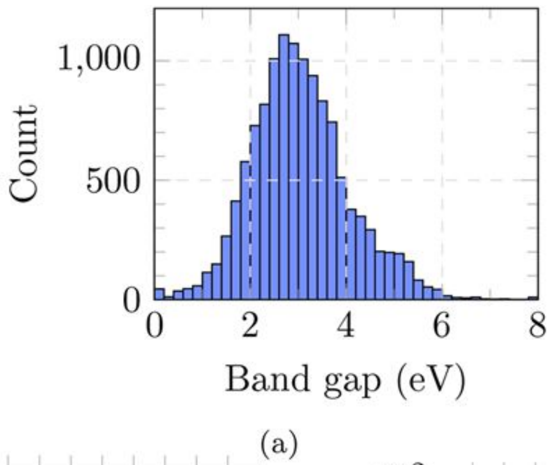
Name	Size	Type	$\bar{N}$	Consistent
QM9	133885	Organic molecules	18	*Y*
Materials Project	53340	Crystals	27	*N*
OMDB-GAP1	12500	Organic crystals	82	*Y*

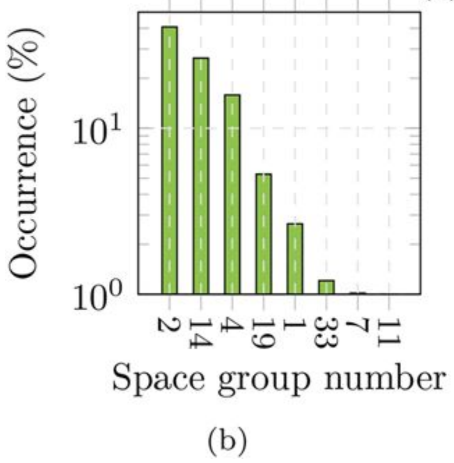
The dataset comprises 65 atomic elements with Uranium being the heaviest and spans 69 space groups. Figure 5 (a) shows the distribution of the dataset. It follows Wigner-Dyson distribution: $x^{4.61}e^{-0.28x^2}$. The mean, median and standard deviation of the dataset are 3.05, 2.96 and 1.03 respectively. Figure 5 (b) shows the most common space groups and Figure 5 (c) shows the most common atomic elements excluding Carbon (C) and Hydrogen (H) in the dataset.

3.2 Methods

3.2.1 SchNet

Schütt et al. [20] initially proposed DTNN which used many-body Hamiltonian method to model the interaction between atoms. They showed that combining this with deep





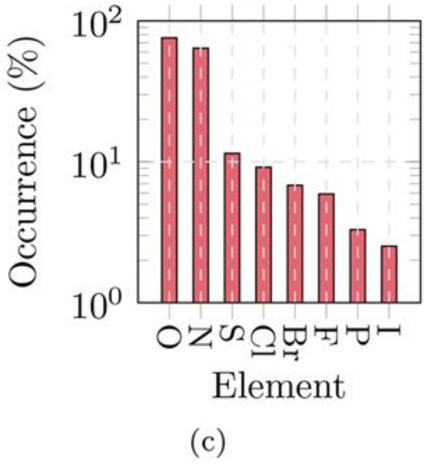


Figure 5: Statistics of OMDB-GAP1 dataset from [5]

tensor networks can achieve chemical accuracy on common datasets such as QM9. However, this method does not employ the continuous filter convolution to model the interaction between the atoms. In order to predict molecular properties such as potential energy or forces, minor changes in atomic positions are to be captured which is possible only through continuous convolution kernel. Figure 6 shows the drawback of the discrete filter which can be handled using continuous filter convolutions. In the left

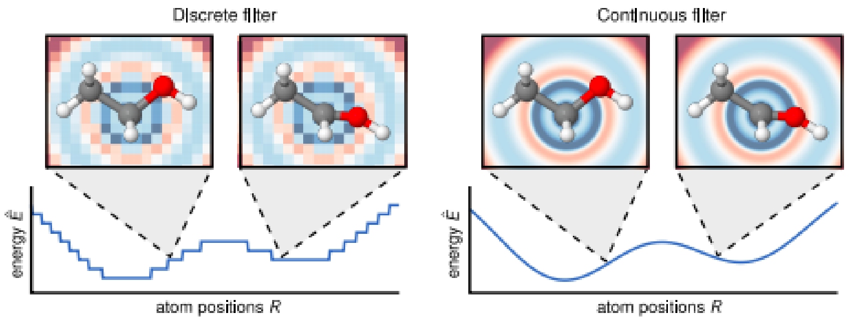


Figure 6: Discrete vs Continuous filter

sub-figure, we can see that the discrete filter is not able to identify the slight variations in the atomic positions and predicts similar output for all of these minor variations. Hence this resulted in discontinuities in the predicted energy values (can be seen in the bottom left). The continuous filter can capture these variations (top right) and this resulted in smoother energy predictions (bottom right).

Architecture

SchNet is mainly designed to model the molecular energies and forces of molecules. However, it can also be used for predicting other properties of molecules (bandgap in our case). SchNet follows the fundamental laws of physics by learning representations that are rotational, translational invariant in addition to being invariant with respect to the indexing of the atoms. The overall architecture of the SchNet is shown in Figure 7 and a detailed description of the individual components is provided in the Section 3.2.1.

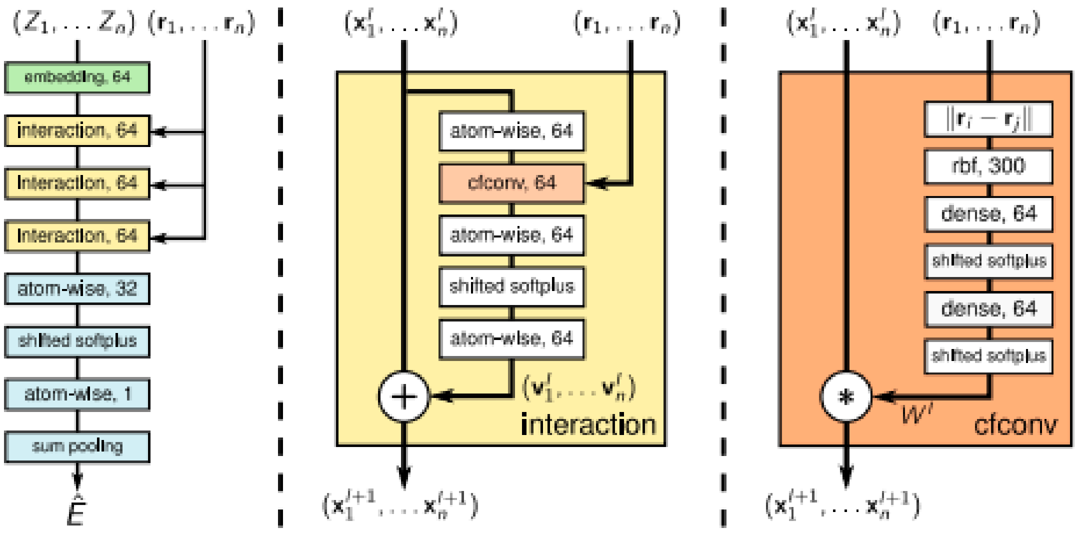

$(Z_1,\dots Z_n)$
$(\mathbf{r}_1,\dots \mathbf{r}_n)$
embedding, 64
interaction, 64
interaction, 64
interaction, 64
atom-wise, 32
shifted softplus
atom-wise, 1
sum pooling
$\hat{E}$
$(\mathbf{x}_1^l,\dots \mathbf{x}_n^l)$
$(\mathbf{r}_1,\dots \mathbf{r}_n)$
atom-wise, 64
cfconv, 64
atom-wise, 64
shifted softplus
atom-wise, 64
$(\mathbf{v}_1^l,\dots \mathbf{v}_n^l)$
interaction
$(\mathbf{x}_1^{l+1},\dots \mathbf{x}_n^{l+1})$
$(\mathbf{x}_1^l,\dots \mathbf{x}_n^l)$
$(\mathbf{r}_1,\dots \mathbf{r}_n)$
$\|\mathbf{r}_i - \mathbf{r}_j\|$
rbf, 300
dense, 64
shifted softplus
dense, 64
shifted softplus
W^l
cfconv
$(\mathbf{x}_1^{l+1},\dots \mathbf{x}_n^{l+1})$

Figure 7: SchNet architecture

Components

1. Atomic embeddings

A set of n atoms that constitute the molecule along with their corresponding nuclear charges ($Z = (Z_1,\dots,Z_n)$) and positions ($R = (\mathbf{r}_1,\dots \mathbf{r}_n)$) is enough to describe a molecule. The atoms in the SchNet model is represented by a vector of features $X^l = \left(\mathbf{x}_1^l,\dots \mathbf{x}_n^l\right)$ where $\mathbf{x}_i^l \in \mathbb{R}^F$. Here F, n, l represents the number of feature maps, number of atoms and the current layer in the model respectively. Atoms represented using an embedding dependent on it's type Z_i is fed as input to the model.

$$\mathbf{x}_i^0 = \mathbf{a}_{Z_i}$$

These embeddings are initialized randomly and updated during the training process.

2. Atom-wise layers
From Figure 7, we can see that the atom-wise layers occur at multiple places in the network. These layers are applied to each of the atomic representations ($\mathbf{x}_i^l$) separately as follows:

$$\mathbf{x}_i^{l+1} = W^l \mathbf{x}_i^l + \mathbf{b}^l$$

Here W^l is the weight matrix which is common to all of the atoms in layer l. Since the weights are shared among all of the atoms, it is clear that the architecture is independent of the number of atoms present in the molecule.

3. Interaction blocks
An interaction block is the place where the convolutions are defined on the atoms. In convolution neural networks (CNN) [27], the convolutions are defined on pixels in a 3-dimensional grid. However, this is not possible in case of molecules as the atoms are located at arbitrary positions in space. Hence a new type of convolution called continuous filter convolution (cfconv) is defined as follows.

$$\mathbf{x}_i^{l+1} = \left(X^l * W^l\right)_i = \sum_{j=0}^{n_{\text{atoms}}} \mathbf{x}_j^l \circ W^l \left(\mathbf{r}_j - \mathbf{r}_i\right)$$

Here 'i' is the atom for which convolution is defined, X^l is the atomic representation, j corresponds to the neighbouring atoms and W^l is the filter generating network described in the subsequent section. From the interaction block (middle in Figure 7), we can see that the atomic representation obtained from the atom-wise layer is sent through the cfconv layer and then a couple of atom-wise layers with a shifted softplus ($\ln\left(0.5e^x + 0.5\right)$) non-linear layer in between them. The use of shifted softplus layers ensures the energy predictions of the model to be smooth. Finally, the initial atomic representation (input to interaction block) is added to the output of the interaction block which is inspired by ResNet [34] as shown below.

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{v}_i^l$$

Varying Interaction block architecture

ResNet is a CNN architecture which achieved the state of the art performance on image classification tasks. The trait that separates ResNet from other popular CNN architectures is "skip connections". Skip connections address the problem of vanishing gradient in training deep neural networks. As the gradient is back-propagated in a neural network while training, repeated multiplication makes the gradient infinitely small. So the weights in the initial layers do not get updated. ResNet deals with this problem by having identity connections that skip one or more layers as shown in Figure 8.

We can see from Figure 8 that ReLU activation is applied to the immediate input ($F(x)$) before adding to x but not in the case of SchNet (ReLU is not applied to v_i).

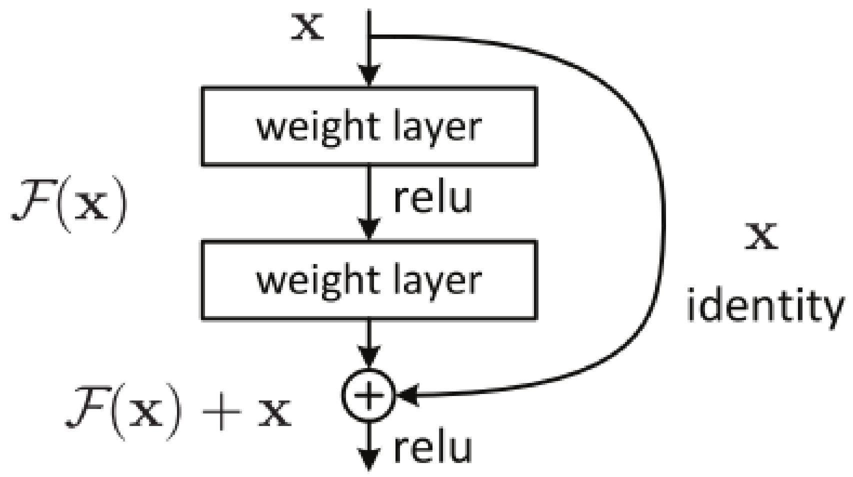

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 8: ResNet skip connections

Hence we modified the equation in SchNet accordingly to add ReLU activation. This produced better results in our case.

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + ReLU(\mathbf{v}_i^l)$$

4. Filter Generating network

Filter Generating network defines the interaction between the atoms in the interaction module i.e W^l in the equation. A fully connected neural network is used as a filter generating network which takes the distance between the atoms (obtained by computing the absolute difference between their positions) as input.

4.1 Ensuring Rotational variance

As mentioned before, the model should learn representations that are rotational and translational invariant. This is achieved by using inter-atomic pairwise distance function shown below.

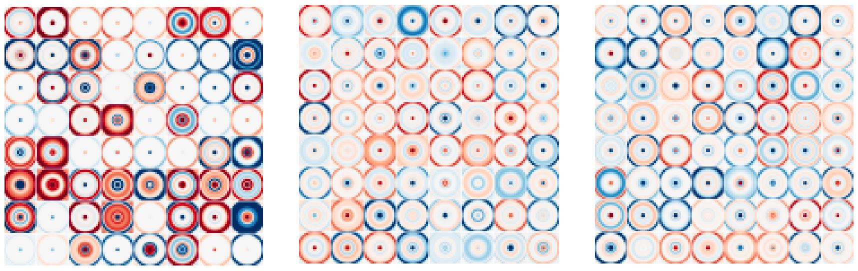
$$d_{ij} = \left\|\mathbf{r}_i - \mathbf{r}_j\right\|$$

One drawback of this type of filters is that they become highly correlated and may learn almost the same features during training. To resolve this, the distances are expanded using Gaussian basis functions as shown below.

$$e_k\left(\mathbf{r}_j - \mathbf{r}_i\right) = \exp\left(-\gamma\left(\left\|\mathbf{r}_j - \mathbf{r}_i\right\| - \mu_k\right)^2\right)$$

Here μ_k represents the mean of the Gaussian k. γ and the number of Gaussians (configurable parameters) define the resolution of the filter. Higher values indicate larger resolution of the filter. These expanded distances are then passed to the dense layers with a shifted softplus layer in between to obtain the value $W(r_i - r_j)$ (as shown in Figure 7 right).

Figure 9 shows the 2-dimensional cuts of the SchNet learned filters for all the 3 interactions blocks during training on ethanol molecular dynamics trajectory. The color ’blue’ indicates negative and ’red’ indicates positive values respectively. We can see that each filter focused on learning particular range of inter-atomic distances. These interaction blocks in sequence (through filters) learn complex representations that will assist the model in making better predictions.



(a) 1st interaction block (b) 2nd interaction block (c) 3rd interaction block

Figure 9: Learned kernel values in interaction blocks

3.2.2 GraphConvNets (GCN)

As mentioned before, the equation in 5 relates the filtering mechanism in the spectral and the spatial domain. When the filter is of K-order polynomial $\hat{h}(\lambda_l) = \sum_{k=0}^{K} a_k \lambda_l^k$ for some constants $\{a_k\}_{k=0,1,\ldots,K}$, the resulting filtered output for a node i in the spatial domain is given by

$$f_{out}(i) = \sum_{j=1}^{N} f_{in}(j) \sum_{k=0}^{K} a_k \left(\Delta^k\right)_{i,j}$$

If we further constrain that $\left(\Delta^k\right)_{i,j} = 0$ when the shortest path between the vertices i and j is greater than k and also assume that the spectral filter is of K-order polynomial, we can observe from the above equation that it is K-hop localized in the spatial domain. The following derivation of the GCN convolution expression is a revised version of the derivation provided in [4]

Defferrard et al.[16] used this property and created a convolution kernel of polynomial parametrization. Each element of the filter matrix is of the form as shown below.

$$\hat{h}(\lambda_i) = \sum_{j=0}^{r-1} \alpha_j \lambda_i^j, i = 1, \ldots, n$$

The matrix form of the filter is given by

$$\hat{h}(\Lambda) = \begin{bmatrix} \sum_{j=0}^{r-1} \alpha_j \lambda_1^j & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sum_{j=0}^{r-1} \alpha_j \lambda_n^j \end{bmatrix}$$

Convolution Layer The convolution of the filter with f_l is given by

$$\xi\left(\sum_{l}^{p}\phi\hat{h}(\Lambda)\phi^{\top}\mathbf{f}_l\right)$$

where $\{\mathbf{f}_l : \mathbf{f}_l \in \mathbb{R}^n\}_{l=1,\dots,p}$ represents the p-dimensional input signal (equivalent to 3-dimensional image input for CNN) and ξ is a non-linear function We can see that the computational complexity of this filter is quite high because of the multiplication with the forward and inverse Fourier transformation matrices ϕ and $\phi^{\top}$. One possible solution is to write $\hat{h}(\Lambda)$ as a polynomial function that can be recursively computed from the previous values. Chebyshev polynomial [35] can be used as the higher-order polynomials as they are computed iteratively from the lower order ones.

Chebyshev polynomial The Chebyshev polynomial is defined according to the below equation

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

Defferrard et al.[16] used the Chebyshev polynomials to define a convolution filter as shown below.

$$\hat{h}(\Lambda) = \sum_{j=0}^{r-1}\alpha_j T_j(\hat{\Lambda})$$

Here $\{\alpha_j\}_{j=0,\dots,r-1}$ are the coefficients of the polynomial, $\hat{\Lambda} = 2\Lambda/\lambda_{\max} - I$. We know from 2.2.2 that eigenvalues of the normalized graph laplacian lie in $[0,2]$. Hence it is clear that the eigenvalues of the $\hat{\Lambda} = 2\Lambda/\lambda_{\max} - I$ lie in $[-1,1]$.
The convolution operation is then defined as below.

$$\xi\left(\sum_{l}^{p}\Phi\sum_{j=0}^{r-1}\alpha_j T_j(\hat{\Lambda})\Phi^{\top}\mathbf{f}_l\right)$$

This can be rewritten as

$$= \xi\left(\sum_{l}^{p}\sum_{j=0}^{r-1}\alpha_j T_j(\hat{\Delta})\mathbf{f}_l\right)$$

$$= \xi\left(\sum_{l}^{p}\sum_{j=0}^{r-1}\alpha_j \mathbf{F}_{j,l}\right)$$

where $\mathbf{F}_{j,l} = T_j(\hat{\Delta})\mathbf{f}_l \in \mathbb{R}^n$. Now we can see how to utilize the recurrence relation to rewrite the filter. If $\mathbf{F}_{0,l} = \mathbf{f}_l$ and $\mathbf{F}_{1,l} = \hat{\Delta}\mathbf{f}_l$, then for any j,l the Chebyshev recurrence relation can be used to compute the value $\mathbf{F}_{j,l}$ as shown below.

$$\mathbf{F}_{j,l} = 2\hat{\Delta}\mathbf{F}_{j-1,l} - \mathbf{F}_{j-2,l}$$

In **GraphConvNets**, the filter is further simplified by considering Chebyshev polynomials until order 2. Also, we know that $\lambda_{\max} = 2$ since normalized graph Laplacian is used. Using these new constraints, the convolution equation can be simplified as shown below.

$$\begin{aligned}
&\xi\left(\sum_{l=1}^{p}\sum_{j=0}^{r-1}\alpha_j T_j(\hat{\Delta}) f_l\right)\\
&\xi\left(\sum_{l=1}^{p}\sum_{j=0}^{r-1}\alpha_j T_j\left(2\Delta/\lambda_{\max} - I\right) f_l\right)\\
&\xi\left(\sum_{l=1}^{p}\sum_{j=0}^{2-1}\alpha_j T_j(2\Delta/2 - I) f_l\right)\\
&\xi\left(\sum_{l=1}^{p}\alpha_0 T_0(\Delta - I) f_l + \alpha_1 T_1(\Delta - I) f_l\right)\\
&\xi\left(\sum_{l=1}^{p}\alpha_0 I f_l + \alpha_1(\Delta - I) f_l\right)\\
&\xi\left(\sum_{l=1}^{p}\alpha_0 I f_l + \alpha_1\left(\left(I - D^{-1/2} W D^{-1/2}\right) - I\right) f_l\right)\\
&\xi\left(\sum_{l=1}^{p}\alpha_0 I f_l - \alpha_1\left(D^{-1/2} W D^{-1/2}\right) f_l\right)
\end{aligned}$$

The reason for considering the order of the Chebyshev polynomial as 2 is to reduce over-fitting in the convolution operation. This can happen for graphs with large node degree. Furthermore, Kipf and Welling [17] added an extra constraint $\alpha = \alpha_0 = -\alpha_1$ to reduce over-fitting and also to reduce the number of computations per layer. Now the equation can be simplified as follows

$$\begin{aligned}
&= \xi\left(\sum_{l=1}^{p}\alpha_0 I f_l - \alpha_1\left(D^{-1/2} W D^{-1/2}\right) f_l\right)\\
&= \xi\left(\sum_{l=1}^{p}\alpha I f_l + \alpha\left(D^{-1/2} W D^{-1/2}\right) f_l\right)\\
&= \xi\left(\sum_{l=1}^{p}\alpha\left(I + D^{-1/2} W D^{-1/2}\right) f_l\right)
\end{aligned}$$

Since the eigenvalues of the normalized graph laplacian lie in the range $[0,2]$, repeated application of this operator (when the networks go deep) can result in numerical instability. So Kipf and Welling [17] further normalized the above equation to

$$\xi\left(\sum_{l}^{p}\alpha\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{W}}\tilde{\mathbf{D}}^{-1/2}\mathbf{f}_l\right) \qquad l = 1,\ldots,p; l' = 1,\ldots,q$$

where

$$\tilde{\mathbf{W}} = \mathbf{W} + I \text{ and } \tilde{\mathbf{D}} = \sum_j \tilde{W}_{ij}$$

3.2.3 Message Passing Neural Networks (MPNN)

All the existing popular graph neural networks like GCN, DTNN (Deep Tensor Neural Networks) can be explained using a framework called Message Passing Neural Networks (MPNNs) according to [9]. As there are a large number of GNN algorithms, explaining them under this common framework will help us to gain a deeper understanding of these models and possibly modify them to achieve better results. Let G be a directed graph with node features x_v and edge features e_{vw} between the nodes v and w. The forward pass in this framework has two stages namely "message passing stage" and a "readout stage". In "message passing stage" which runs for T time steps, the message m_v^{t+1} at time step 't+1' for a node 'v' is computed based on it's hidden representation h_v^t and also the hidden representation of it's neighbours h_w^t as shown below.

$$m_v^{t+1} = \sum_{w \in N(v)} M_t\left(h_v^t, h_w^t, e_{vw}\right)$$

Then the node representation h_v^{t+1} for node v at time step $t+1$ is updated using it's representation h_v^t at time t and message m_v^{t+1} as shown below.

$$h_v^{t+1} = U_t\left(h_v^t, m_v^{t+1}\right)$$

In the readout stage, the feature vector for the whole graph is computed using some readout function R as shown below

$$\hat{y} = R\left(\left\{h_v^T \mid v \in G\right\}\right)$$

The readout R, vertex update U_t and message functions M_t are all differentiable functions updated during the training process. The constraint on the readout function is that it must be invariant to the permutation of the node states. This is necessary to make the MPNN model invariant to graph isomorphism. Now we will describe some of the common graph neural networks using MPNN framework.

Convolutional Networks for Learning Molecular Fingerprints, [23]

The message function used in this model is $M\left(h_v, h_w, e_{vw}\right) = \left(h_w, e_{vw}\right)$ where (.,.) indicates concatenation. The vertex update function is $U_t\left(h_v^t, m_v^{t+1}\right) = \sigma\left(H_t^{\deg(v)} m_v^{t+1}\right)$. Here σ is the sigmoid function, $deg(v)$ is the degree of the node v, H_t^N is a weight matrix for each time step t and N indicates degree of the vertex. The readout function R is equal to $f\left(\sum_{v,t} \operatorname{softmax}\left(W_t h_v^t\right)\right)$ where f is a MLP and W_t is learned read out matrix at time 't'. We can also notice that the readout function has skip connections to all the previous hidden states h_v^t. However, one disadvantage of this approach is that the resulting message vector ($m_v^{t+1} = \left(\sum h_w^t, \sum e_{vw}\right)$) has summations separately over nodes and edges. It does not take the correlation between the nodes and edges into consideration.

Molecular Graph Convolutions, [24]

In this model, the edge representations e_{vw}^t are updated (instead of the node representations) in the message phase i.e. $M\left(h_v^t, h_w^t, e_{vw}^t\right) = e_{vw}^t$. The vertex update function is $U_t\left(h_v^t, m_v^{t+1}\right) = ReLU\left(W_1\left(\alpha\left(W_0 h_v^t\right), m_v^{t+1}\right)\right)$ where (.,.) denotes concatenation, W_1 and W_0 are weight matrices. The edge state update is performed by $e_{vw}^{t+1} = U_t'\left(e_{vw}^t, h_v^t, h_w^t\right) = \alpha\left(W_4\left(\alpha\left(W_2, e_{vw}^t\right), \alpha\left(W_3\left(h_v^t, h_w^{t+1}\right)\right)\right)\right)$ where W_i are learned matrices.

Deep Tensor Neural Networks, [36]

The message function is given by $M_t = \tanh\left(W^{fc}\left(\left(W^{cf}h_w^t + b_1\right) \odot \left(W^{df}e_{vw} + b_2\right)\right)\right)$ where W^{cf}, W^{fc}, W^{df} are weight matrices and b_1, b_2 are bias vectors. The update function is $U_t\left(h_v^t, m_v^{t+1}\right) = h_v^t + m_v^{t+1}$. The readout function sends the updated node representation (h_v^T) through the Neural Network (NN) separately and sums the resulting output over all the nodes as shown below.

$$R = \sum_v \mathrm{NN}\left(h_v^T\right)$$

GraphConvNets (GCN)

[17] proposed the below propagation rule for each layer

$$H^{l+1} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^l W^l\right)$$

Here $\tilde{A} = A + I_N$ where A is the adjacency matrix for an undirected graph G. To add self-loops to the graph, Identity matrix I_N is added to A. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the degree matrix for the graph with self loops,W^l is a learned weight matrix in layer 'l', H^l is $\mathbb{R}^{N\times D}$ matrix where each of the N nodes have D-dimensional representation. If we assume $L = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, then the above equation can be re-written as

$$H_{(v)}^{l+1} = \sigma\left(L_{(v)}H^l W^l\right) = \sigma\left(\sum_w L_{vw}H_{(w)}^l W^l\right)$$

Converting the above row vector (H_v^{l+1}) into a column vector (obtained by transposing the above equation) and relabelling it in terms of 't' (H_v^{t+1}), then the above equation is equivalent to

$$h_v^{t+1} = \sigma\left(\left(W^l\right)^T \textstyle\sum_w L_{vw}h_w^t\right)$$

Now this equation can be explained using the MPNN framework where the message function is $M_t\left(h_v^t, h_w^t\right) = L_{vw}h_w^t = \tilde{A}_{vw}(\deg(v)\deg(w))^{-1/2}h_w^t$ and update function is $U_t\left(h_v^t, m_v^{t+1}\right) = \sigma\left(\left(W^t\right)^T m_v^{t+1}\right)$

SchNet

Section 3.2.1 provides a detailed explanation of the SchNet. In this section, we will try to explain the SchNet model using the MPNN framework.
The message function in SchNet is $M_t\left(h_v^t, h_w^t\right) = m_w^t = W^t\left(\mathbf{r}_w - \mathbf{r}_v\right)$ and the vertex update function is $U_t\left(h_v^t, m_v^{t+1}\right) = \sum_{j=0}^{n_{\text{atoms}}} \mathbf{x}_j^t \circ m_j^{t+1}$. Finally, the readout function takes the average over all the atoms $R = \frac{1}{n_{\text{atoms}}}\sum_v\left(h_v^t\right)$

Best model among the MPNN variants

[9] took GGNN (Gated Graph Neural Network) [37] as the baseline and experimented with different variations of it on molecular datasets like QM9. The best MPNN variant found has the below message and readout functions.
Let d denote the dimension of each node's representation and n represent the number of nodes in the graph. [9] implemented MPNNs on directed graphs with a separate message channel for incoming and outgoing edges m_v^{in} and m_v^{out} respectively. To apply this method on an undirected graph, each edge is considered as both incoming and outgoing edge with the same label. So the size of the message channel will be double (2d) in case of an undirected graph.
The input to the MPNN model is a set of node feature vectors x_v (information about different bonds in the molecule) and an adjacency matrix A containing the pair-wise distance between the atoms. The initial hidden state h_v^0 is the atomic feature vector x_v which is padded to some large dimension d and GRU unit with weight tying (same weight matrix across multiple time steps) is used at each time step t.

Message function
The message function from node w to node v along edge e is $M(h_v, h_w, e_{vw}) = m_{wv} = A(e_{vw})h_w$ where $A(e_{vw})$ is a neural network that converts the edge vector e_{vw} to a $d \times d$ matrix.

Readout function
The readout function used is Set2Set model from [25] as it showed better results than other functions on QM9 dataset. Set2Set model is designed to work on sets (where order does not matter) and is a better pooling technique than average/sum pooling. Detailed explanation of Set2Set is presented in 3.2.4.

3.2.4 Set2Set Pooling

Set2Set pooling is based on content-based attention. This mechanism ensures that the final representation (vector retrieved from memory) does not change much when the input data is shuffled. This is necessary for Set inputs where the order of input does not matter. Particularly, for molecules, the pooling operation should be invariant to the order of the atoms in that molecule. The architecture of the pooling method is shown in Figure 3.2.4. We can see that there are three components namely "Read", "Process" and "Write" blocks which are explained in detail below.
Read block: This block creates an embedding for each element x_i using a neural network onto a memory vector m_i

Process block: The steps in this block is shown below

$$
\begin{aligned}
q_t &= \mathrm{LSTM}\left(q_{t-1}^*\right) \\
e_{i,t} &= f\left(m_i, q_t\right) \\
a_{i,t} &= \frac{\exp\left(e_{i,t}\right)}{\sum_j \exp\left(e_{j,t}\right)} \\
r_t &= \sum_i a_{i,t} m_i \\
q_t^* &= [q_t r_t]
\end{aligned}
$$

Here m_i is the memory vector (the number of m_i is equal to the size of the input X). r_t is read from memory and stored in q_t. f returns a scalar value computed between m_i and q_t. $a_{i,t}$ is a softmax value computed for $e_{i,t}$ and the LSTM is a Long Short Term Memory cell ([38]) that takes only the previous state as input q_{t-1}^* (which is a concatenation of q_t and r_t). t indicates the number of processing steps that are run before sending it to the decoder for output. One can clearly see from the equation of r_t that it is invariant to the order in which the memory values (m_i equivalent to x_i) is given as input.
Write block: This block contains LSTM pointer network which takes q_T^* as input and the output is a pointer to the elements in memory (m_i) one at a time over t time steps.

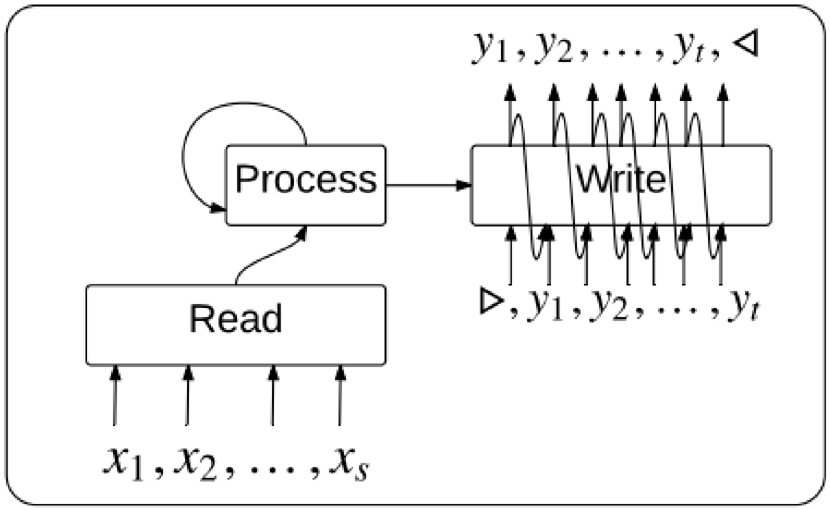

$y_1, y_2, \ldots, y_t, \triangleleft$
Process
Write
Read
$\triangleright, y_1, y_2, \ldots, y_t$
$x_1, x_2, \ldots, x_s$

Figure 10: Set2Set model

3.2.5 SAG pooling

SAG pooling ([39]) uses self-attention method for pooling in GNNs. Basically, the self-attention mechanism is used to identify the nodes that should remain after pooling. GNNs are used in the self-attention to compute the attention scores for each node.

The reason for using GNN is that both the node features as well as the topology of the graph information can be used for pooling. This is not the case in earlier pooling methods where either node features or topology of the graph is considered for pooling. The GNN used for pooling in this thesis is GNN from [17] where the self-attention score ($Z \in \mathbb{R}^{N\times 1}$) is computed as shown below.

$$Z = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta_{att}\right)$$

Here σ is the activation function, $\tilde{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix with self-loops, $\tilde{D} \in \mathbb{R}^{N\times N}$ is the degree matrix, $X \in \mathbb{R}^{N\times F}$ is the feature matrix and $\Theta_{att} \in \mathbb{R}^{F\times 1}$ is the learnable parameter. We can see from the equation above that the SAG pooling is using both node features (feature matrix) and the topology (adjacency matrix) of the graph. The pooling ratio is a hyper-parameter that determines the number of nodes to keep after the pooling method.

$$\text{idx} = \text{top}-\text{rank}(Z, \lceil kN \rceil), \quad Z_{\text{mas}k} = Z_{\text{idx}}$$

where the top rank returns the indices of the top $\lceil kN \rceil$ values and z_{mask} is the feature attention mask. The input graph is filtered using the mask operation shown in Figure

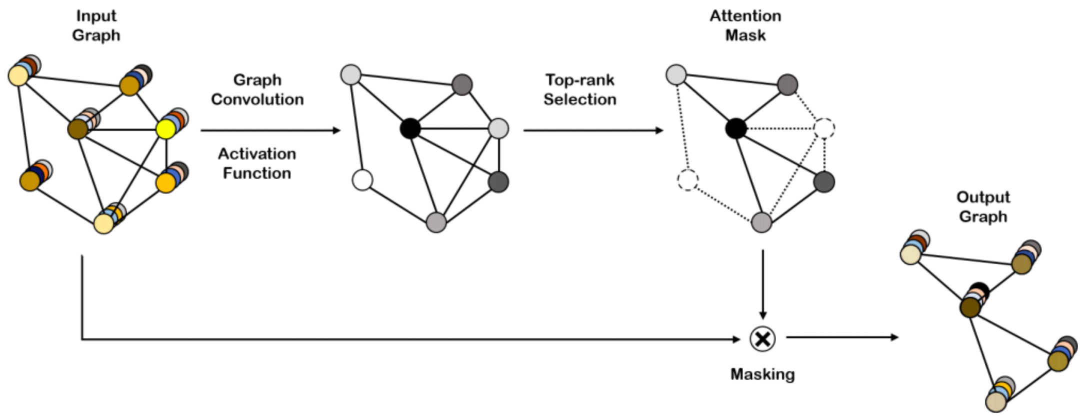


Figure 11: SAG pooling

11.

$$X' = X_{\text{idx},:,}, \quad X_{out} = X' \odot Z_{mask}, A_{out} = A_{\text{idx},\text{idx}}$$

Here $X_{idx,:}$ is the row-wise indexed matrix, $\odot$ is the elementwise product, $A_{idx,idx}$ is the row-wise, column-wise indexed adjacency matrix. X_{out} and A_{out} are the resulting feature matrix and adjacency matrix respectively.

3.2.6 GlobalMax Pooling

This is the basic pooling method where the pooled output is obtained by taking the maximum value among all nodes for each dimension in a graph. For a single graph $\mathcal{G}_i$, the output is computed by

$$\mathbf{r}_i = max_{n=1}^{N_i} \mathbf{x}_n$$

where node feature matrix $\mathbf{X} \in \mathbb{R}^{(N_1+...+N_B)\times F}$ and N_i denotes the number of nodes in graph $\mathcal{G}_i$. The final output for a given batch of graphs is obtained by merging the individual graph outputs.

4 Experiments and Results

In this chapter, the experimental settings as well as the results of the experiments are described. Detailed information on the fine-tuning of the hyper-parameters and the reason behind choosing those values are also provided. First, the performance of the SchNet with different possible configurations are discussed and the best SchNet model (taken as baseline) obtained after fine-tuning the parameters is presented. Then the performance of different GNNs models obtained after fine-tuning the parameters is compared with the baseline model. Similar to [5], 9000 molecules are used for training, 1000 for validation and the remaining 2500 for testing in all of the experiments. The Mean Squared Error (MSE) is used for training and the Mean Absolute Error (MAE) is used for reporting the loss on both validation and test sets in all of the models. Also, during training, the epoch having the least validation loss is recorded and the corresponding loss on the test set is reported.

4.1 SchNet

We started with an “SchNet” architecture similar to the model (from [5]) that achieved the state of the art results on OMDB dataset (Section 3.1). The parameters of the model are shown in Table 2.

Table 2: SchNet configuration from [5]

Parameter	Value
Interaction blocks	3
Atomic embedding size	64
Cutoff radius	5A
learning rate	0.001
decay factor	0.6
optimizer	Adam
batch size	32

The mean squared error (MSE shown as loss[eV] on Y-axis) measured with respect to time for both training and validation sets is shown in left pane of Figure 12. The mean absolute error (MAE) with respect to time for validation set is shown in right pane of the Figure 12. We can see that the validation curve is not smooth and has lot of spikes. The model that achieved the lowest validation error (during training)is chosen as the final model and the error on the test set is reported using this model. The MAE of the model on the test set is 0.51eV. This is worse than the model with similar configuration from [5] which achieved a much lower MAE of 0.415eV.

We also have gone through the preprint version ([6]) of the paper ([5]) submitted in 2018. In this preprint version, a slightly different architecture of SchNet was reported to have achieved an MAE of 0.378eV on the test set. The architecture of the model

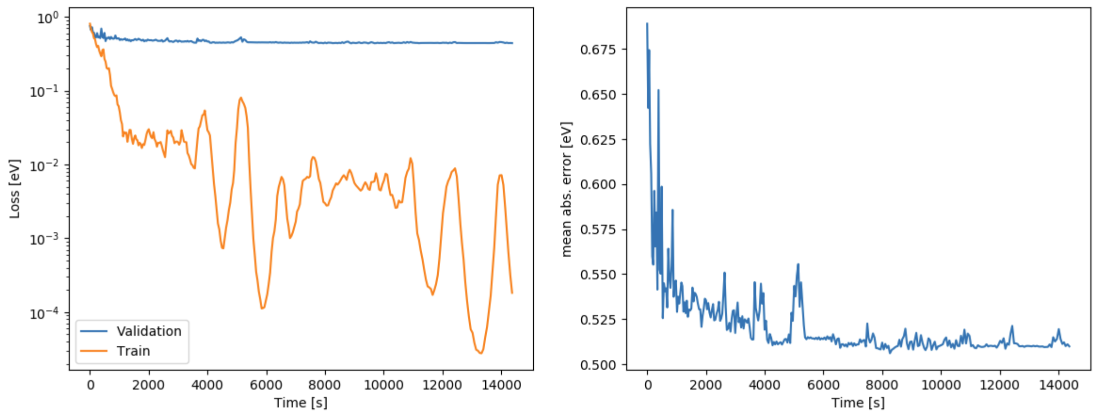


Figure 12: Training (MSE) and Validation (MSE (left) and MAE (right) Loss of model from [5]

is shown in Table 3. We trained a model with the similar configuration but there was no improvement in the accuracy on the test set. We then chose to increase the number

Table 3: SchNet configuration from [6]

Parameter	Value
Interaction blocks	3
Atomic embedding size	64
Cutoff radius	5A
learning rate(η)	0.001
decay factor	η* 0.96 i/10000(i denotes training step)
optimizer	Adam
batch size	32

of interaction blocks to 6, atomic embedding size to 128. As the molecules are quite complex, increasing these values might capture better representations of atoms (with respect to it's neighbours) and this can result in lower MAE on the validation set. In this new setting, the training starts with a learning rate of 0.001 which is reduced by a factor of 0.8 (until a value of $1e-6$) after 25 epochs if there is no improvement in the validation loss. The other parameters are shown in Table 4. We also slightly changed the interaction module in SchNet as mentioned in section 3.2.1. The MAE of this model on both the training and validation sets are shown in Figure 13. We can clearly see that the validation curve is quite smooth now and the MAE is much lower than the previous models. This is the best SchNet configuration we found that achieved an MAE of **0.28eV** on the test set.

Table 4: Best performing SchNet model

Parameter	Value
Interaction blocks	6
Atomic embedding size	128
starting learning rate	0.001
decay factor	0.8
Cutoff radius	5A
optimizer	Adam
batch size	16

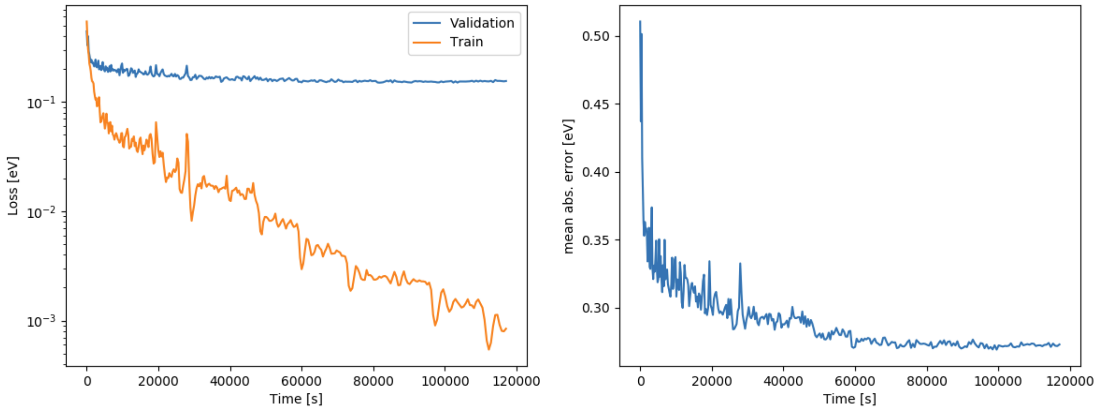


Figure 13: Training (MSE) and Validation (MSE (left) and MAE (Right)) Loss:Best model

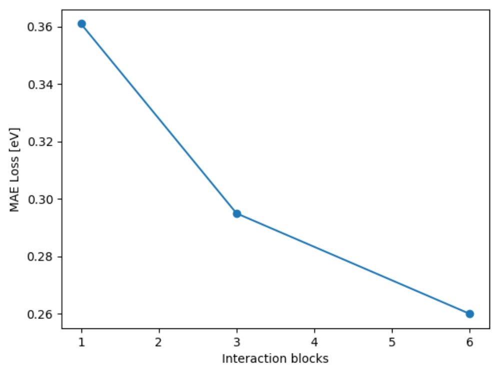
4.1.1 Effect of different parameters

We will vary the number of interaction blocks and the size of atomic embeddings separately and study their impact on the performance of the model. The MAE loss on the test set for different number of interaction blocks and sizes of atomic embeddings are shown in Figures 14a and 14b respectively.

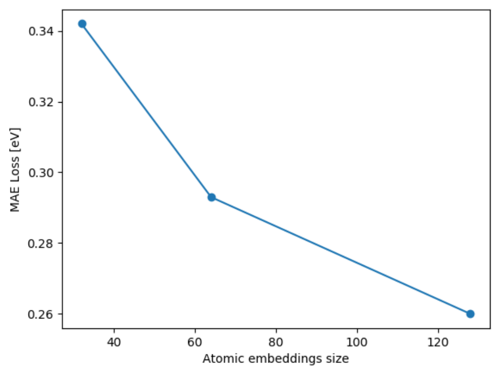
For both the interaction blocks and atomic embedding parameters, we can see that the loss on the test set reduced with increase in the parameter value.

4.2 NNCONV with Set2Set pooling

In this section, the results of the MPNN (NNCONV) with set2set pooling are discussed. The architecture of the model with the best performance on the validation set is shown in Figure 15. The input graph is initially passed through a linear layer of size 32 followed by ReLU activation layer. Then there is a set (NNCONV layer with 64 filters, ReLU and Gated Recurrent Unit (GRU)) of modules which computes the message function (mentioned in Section 3.2.3). There are 8 such modules (equivalent to 8 time steps) in sequence followed by set2set pooling (equivalent to readout function as



(a) Interaction Blocks vs test MAE Loss (eV)



(b) Atomic embeddings vs test MAE Loss (eV)

Figure 14: Effect of interaction blocks and atomic embedding parameters

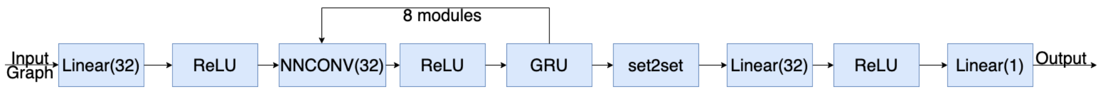


Figure 15: NNCONV model with Set2Set pooling architecture

in Section 3.2.3) with sequence of steps equal to 1. Finally, there are a couple of linear layers (64 unit and 1 unit respectively) with ReLU activation in between. The linear unit of size 1 outputs the required bandgap value.
The parameters of the model are presented in Table 5. The learning rate is decayed by a factor of 0.8 (before it reaches a minimum value of $1e-6$) after 25 epochs if there is no improvement on the validation loss. The loss plot of the model on the training (MSE), validation (MAE) and test sets (MAE) are shown in Figure 16.

Table 5: NNCONV model with Set2Set pooling parameters

Parameter	Value
starting learning rate	0.001
decay factor	0.8
optimizer	Adam
batch size	32

The model is trained for 500 epochs and we can see from Figure 16 that all the 3 loss curves are quite smooth throughout the training process. The loss on the training set is reducing gradually as expected and became almost constant at the end. However, the loss value on the validation and test sets reduced gradually before 200 epochs and started to increase slightly later. The reason could be attributed to over-fitting on the training set. Also, the loss value on the validation and test sets are almost the same throughout the training process. This indicates that the model has generalized well because the loss on unseen data (test set) is similar to the loss on the validation set.

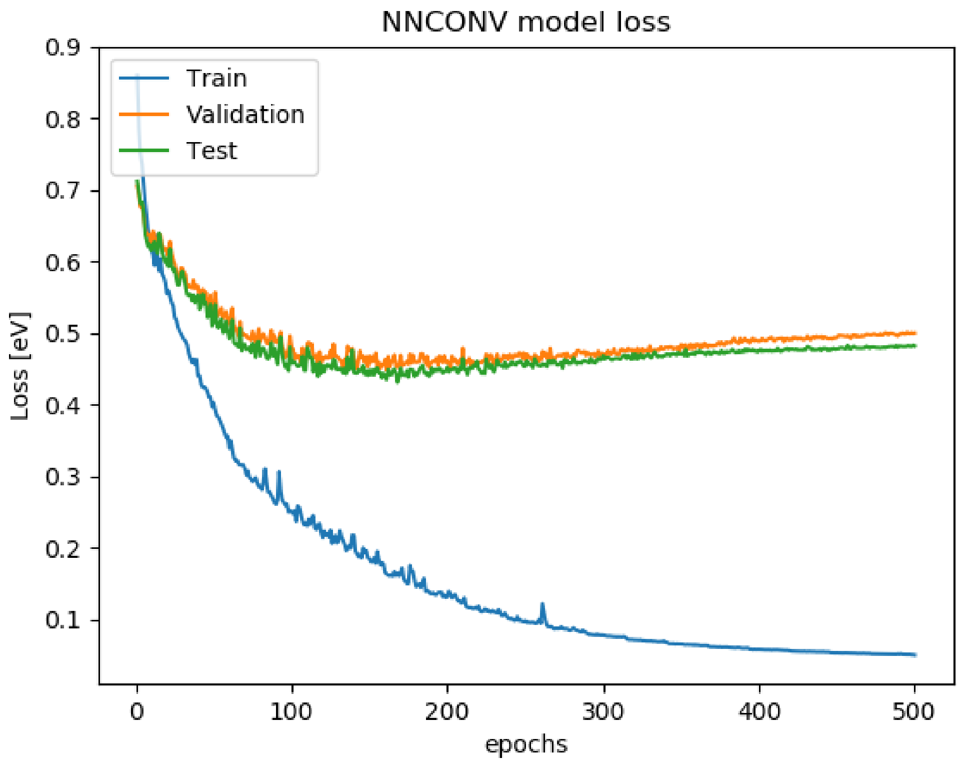


Figure 16: NNCONV model with Set2Set pooling results

The model reached a minimum loss value of 0.447eV on the validation set and the corresponding test loss is **0.443eV**.

4.2.1 Effect of message propagation time step t

Since NNCONV with set2set pooling achieved the best performance among all GNNs in this thesis, we will explore the impact of time step t on this model. As mentioned before, there are 8 sets of modules (NNCONV, ReLU, GRU) in sequence for computing the message propagation step. The size of this module set is equivalent to time step t and the performance of the model on the test set for different possible values of t is shown in Table 6. The MAE loss on the test set decreased with increase in the time

Table 6: MAE loss on the test set for different values of t

t	MAE loss
3	0.49
5	0.47
8	0.447

step t. As the molecules are quite complex, it seems that the message propagation step has to go deep (until 8 in this case). Because of this deeper message propagation, the representation learnt for each node enabled the model to perform better on unseen data (test set).

4.3 NNCONV with SAG pooling

In this section, we will discuss the performance of NNCONV model with SAG pooling. We can see from the results of NNCONV (with Set2Set pooling 4.2) and GCN (4.4) models that NNCONV achieved the best performance. This NNCONV model used the Set2Set pooling method. Set2Set method uses only the node features for pooling but recent approaches such as SAG also considers the structural information of the graph which is important. More information on this can be found in Section 1.1.4. So we decided to test the performance of this pooling method. The new model has similar configuration as that of model in section 4.2 except the new pooling method (SAG) and it's architecture is shown in Figure 17. We can see from Figure 17 that the only differ-

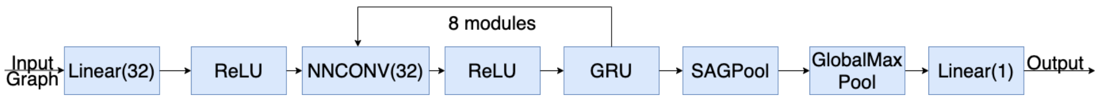


Figure 17: NNCONV with SAG pooling architecture

ence from the architecture of NNCONV model with Set2Set pooling (15) are SAGPool (3.2.5) and GlobalMaxPool (3.2.6) modules. The reason for using GlobalMaxPool is that SAGPool outputs graphs of different sizes for a given batch of graphs. To make all of the graphs have a constant size output, GlobalMax pooling is applied. Finally, the linear layer of size 1 outputs the bandgap value of the molecule.
The model is trained for 500 epochs and the loss on the training, validation and test sets are shown in Figure 18. We can see from the Figure 18 that the training and test loss curves have more spikes than the NNCONV model with Set2SetPooling (see Figure 16). However, the validation and test loss curves became flat after 300 epochs. This wasn't the case when Set2Set pooling was employed (Section 4.2) where the validation and test loss values started to increase again after 200 epochs. The minimum loss of the model on the validation set during the training process is 0.471eV and the corresponding test loss is **0.485eV**. Therefore, we can conclude that the performance of NNCONV model with Set2Set pooling is slightly better than SAG pooling as the loss on the test set is comparatively lower in the former case.

4.4 GCN

We experimented with different architectures and parameters of the GCN model. In this section, we will present the architecture as well as the results of the GCN model that achieved the best performance on the OMDB dataset. The architecture of the GCN model is shown in Figure 19. The model has 3 GCN convolution layers (having 256,

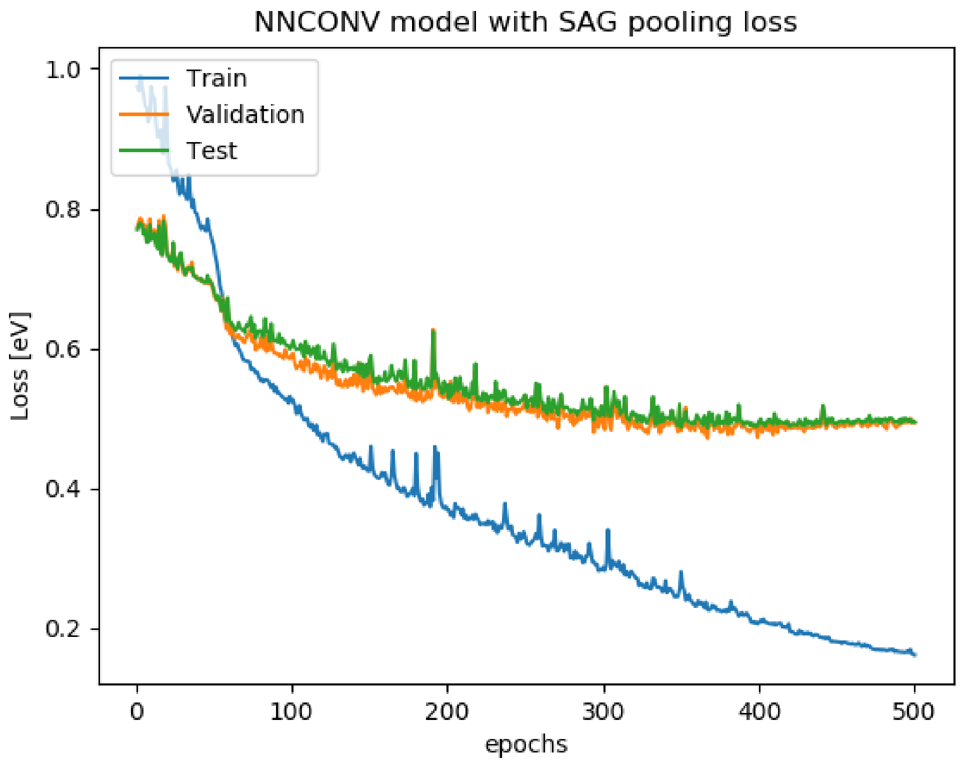


Figure 18: NNCONV with SAG pooling loss

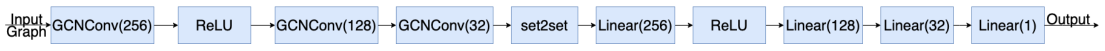


Figure 19: GCN model architecture

128 and 32 filters respectively) at the start with a ReLU activation after the first layer. Then a set2set pooling (with the sequence of steps=1) is applied to the resulting output. This is the only place in the model where the pooling is applied. Then there is a set of 4 linear layers with 256, 128 and 32 neurons respectively. A ReLU activation is applied to the output after the first linear layer. The training starts with a learning rate of 0.001 which is reduced by a factor of 0.8 (until a value of $1e-6$) after 25 epochs without an improvement in the validation loss. The rest of the hyper-parameters of the model are shown in Table 7.

The model is trained for 500 epochs and plot of the MSE loss on training, MAE loss on the validation and test sets are shown in Figure 20. We can see that the loss on the test dataset has more oscillations than the rest until 250 epochs and then became quite smooth. The validation and test loss curves are quite close to each other throughout the training process. The loss on the training set is steadily decreasing up to 400 epochs and remains almost unchanged later. However, it can be seen that the validation and test loss curves became almost flat earlier than the training set (from 300 epochs). The model achieved a least validation loss of 0.566eV and the corresponding test loss value

Table 7: GCN model parameters

Parameter	Value
starting learning rate	0.001
decay factor	0.8
optimizer	Adam
batch size	32

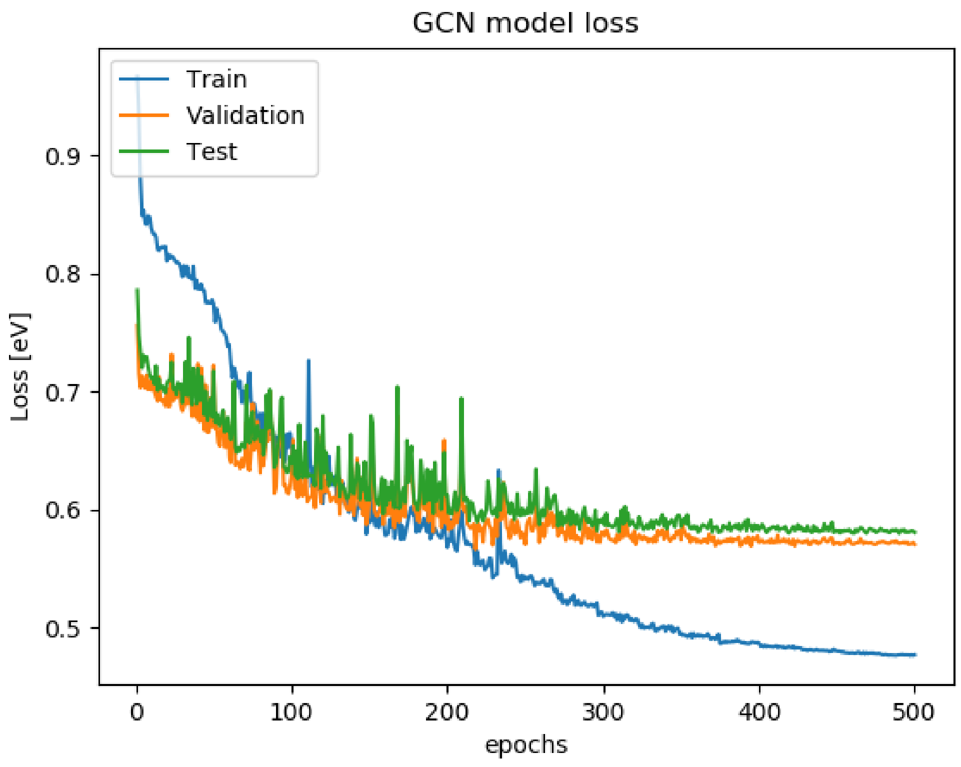


Figure 20: GCN model loss (MSE)

is **0.594eV**. Hence this model has the worst performance among all the models we tested.

4.5 An ensemble of SchNet models

In general, Neural Networks have high variance and may produce different results each time they are trained. To reduce this variance, one possible solution is to combine the predictions from different models. Combining the predictions from multiple models increase the overall bias which in turn reduce the variance. This is called "Ensemble Learning". We built an ensemble of 3 SchNet models and examined the resulting model's performance on the test set. The final output/prediction of the combined model is the average of the individual model predictions. The MAE loss of the ensemble model on the test set is **0.268eV**.

5 Discussion

In this chapter, we will answer the research questions posed earlier based on the knowledge gained from the results of the experiments. Also, at the very end, we will provide possible directions for future work to achieve better results.

5.1 Answers to research questions

5.1.1 Experimenting with different architectures, parameter values of the SchNet model (which is the baseline method) and finding the model with the lowest MAE

The results of SchNet with different architectures and parameter values are presented in Section 4.1. The reason for choosing SchNet is that it was one of the methods that achieved the state of the art results on molecule datasets. Also, this was the only DNN model tested on OMDB dataset. As mentioned in Section 4.1, we started with the SchNet setting reported in [5] that produced the state of the art results. However, we achieved better than the state of the art results using a different configuration and slightly modified architecture of SchNet. We also observed that the SchNet model converged to the lowest MAE in the least number of epochs (less than 100) than other GNN models tested in our thesis. Hence, the training of SchNet is much faster compared to other GNN models.

5.1.2 Experimenting with the state of the art methods in GNNs (both spectral and spatial domains) and finding the model with the lowest MAE

As stated in the Introduction section 1.1, the GNNs can be classified into spectral and spatial approaches. In this thesis, we chose the state of the art methods in each of these approaches namely GCN, NNCONV and analyzed their performance on OMDB dataset. The results of these methods are presented in Sections 4.4 and 4.2 respectively. we observed that both of the models are learning from the training data. As a consequence, the loss on the test set (generalization loss) was gradually decreasing along with the validation loss before reaching a certain value. Among these two methods, NNCONV had a much better performance (MAE 0.44eV) on the test set than GCN (MAE 0.59eV). Hence we can conclude that the spatial approach performed much better than the spectral approach on the OMDB dataset.

5.1.3 Experimenting with different pooling methods that are currently available for GNNs and finding the pooling method that gives the best results

In this thesis, we experimented with primitive pooling methods (mean/max) and also advanced pooling methods that are designed to work on graphs. As mentioned in Section 1.1.4, the pooling methods designed for graphs can be classified into "methods that work based only on node features" (set2set pooling) and "methods that consider structural information of the graph along with node features" (SAG pooling). The results of NNCONV with the same configuration but with different pooling methods Set2Set pooling and SAG pooling are reported in Sections 4.2 and 4.3 respectively. Ideally, the model with SAG pooling should perform better than the model with Set2Set pooling. The reason is that SAG pooling includes necessary information such as the topology of the graph for pooling. Contrary to our expectations, the model with set2set pooling demonstrated better performance.

5.1.4 Finding the model or an ensemble of models with the best performance while being computationally optimal

We can clearly see that SchNet performed significantly better than all of the other models on OMDB dataset. Our configuration of SchNet surpassed the state of the art results (reported in [5]) on OMDB dataset. However, it is known that DNN models have high invariance. This means that the same model may not produce similar results when trained multiple times as they are sensitive to the specifics of the training process (for instance, random weight initialization). Hence we trained multiple SchNet models with the same configuration and made an ensemble of these models. The final output of this ensemble model is the average of predictions made by each of these individual models. The final MAE loss of the ensemble model on the test set is 0.268eV which is less than the MAE loss of the individual models.

5.1.5 Interpreting the results of the best model and identifying the atoms that are significant in contributing to the bandgap value of the molecule

Deep learning (DL) models are considered as "black box" models as we cannot explain why the DL model arrived at a particular solution. "Explainable AI" is an unsolved problem and is currently an active area of research. Multiple methods have been developed in trying to explain the predictions of the DL models but there is no one superior method that works in all of the cases. More details on the current explainable AI techniques can be found in [40]. As GNNs are advanced DL models and a new research field, there are a few explainable AI techniques that work on these GNN models. [41] is one of the papers that discusses the extension of DL explainability methods to GNNs. In this thesis, we also attempted to explain the results produced by SchNet (as this is the best performing model). For a given molecule, SchNet predicts bandgap value for each of the atoms. The final output of the molecule is the average over the predictions of the individual atoms. More information on SchNet can be found in Section 3.2.1. In our explainability method, the atoms for which the bandgap values lie within one standard deviation of the molecule bandgap value are considered significant. We then used the GraphViz [42] software to plot the molecule as a graph where atoms are described using nodes and an edge indicates a bond between the atoms. Atoms (nodes) that are found to be significant will have orange as background colour and white otherwise. A part of a molecule **C9H14N2O2** (showing the entire molecule is unreadable as it is quite complex) using our explainability method is shown in Figure 21. The atoms shown in this Figure are C (carbon) and N (Nitrogen) and the only atom that is found to be significant in this region is the carbon (C) atom present at the bottom-left. More information (SMILES and other properties) on this molecule can be found in "http://www.crystallography.net/cod/4505309.html".

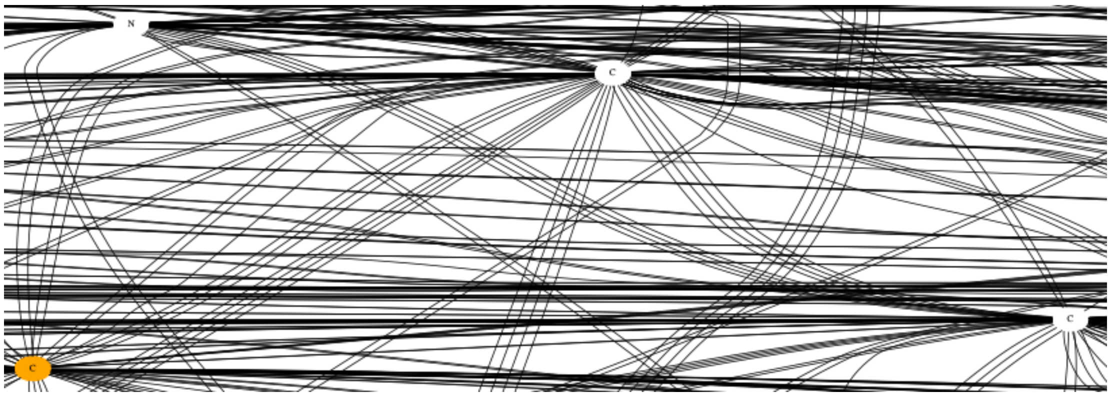


Figure 21: Part of Molecule (C9H14N2O2) with significant atoms (orange coloured nodes)

5.2 Future work

From the results (Section 4), we can infer that GNNs (both spectral and spatial approaches) can be applied to molecular datasets. Even though the performance of GNNs is poor than SchNet in case of OMDB dataset, we cannot conclude that SchNet is always superior. It would be interesting to see the performance of these methods on different molecule datasets. Also, the size of the OMDB dataset does not appear to be large enough to arrive at any conclusion. CNNs (like ResNet that achieved the state of the art results on image classification tasks) were trained on ImageNet dataset which had around 14 million images. In our case, the OMDB dataset contains only 12500 graphs (molecules). We can increase the size of the dataset by adding new molecules either manually or through data augmentation (using Generative Adversarial Networks (GANs) [43]). However, generating molecules (graphs) through GANs is still in the early stages of research. Also, we were not able to extend the number of interaction modules beyond 6 when testing SchNet due to the GPU limitations. In the future, we would like to investigate the accuracy of SchNet model for larger interaction modules and representation embeddings. One possible disadvantage of SchNet is that it is less flexible compared to GNNs. Of course, there are certain parameters which we can configure but it is clear from the SchNet architecture (shown in Figure 7) that it has a certain defined structure. Finally, we would like to mention that GNNs is a fast-growing research field. A new GNN model called DimeNet [44] that demonstrated the state of the art results on molecule datasets was published recently. It would be interesting to see the performance of this model on the OMDB dataset. Although the GNN approaches that were tried did not improve the estimation accuracy, they still hold a promise in terms of improved explainability of results due to the graph-based nature of molecules.

References

[1] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`, 2018.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[3] Stanford. Cs231n convolutional neural networks for visual recognition. `https://cs231n.github.io/convolutional-networks`, 2014.

[4] Shunwang Gong. *Geometric Deep Learning*. PhD thesis, Imperial College London, 2018.

[5] Bart Olsthoorn, R Matthias Geilhufe, Stanislav S Borysov, and Alexander V Balatsky. Band gap prediction for large organic crystal structures with machine learning. *Advanced Quantum Technologies*, 2(7-8):1900023, 2019.

[6] Bart Olsthoorn, R Matthias Geilhufe, Stanislav S Borysov, and Alexander V Balatsky. Band gap prediction for large organic crystal structures with machine learning. *arXiv preprint arXiv:1810.12814*, 2018.

[7] Garrett B Goh, Nathan O Hodas, and Abhinav Vishnu. Deep learning for computational chemistry. *Journal of computational chemistry*, 38(16):1291–1307, 2017.

[8] Sandip De, Albert P Bartók, Gábor Csányi, and Michele Ceriotti. Comparing molecules and solids across structural and alchemical space. *Physical Chemistry Chemical Physics*, 18(20):13754–13769, 2016.

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[10] Kristof T Schütt, Huziel E Sauceda, P-J Kindermans, Alexandre Tkatchenko, and K-R Müller. Schnet–a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24):241722, 2018.

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

[12] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[13] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[15] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[18] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[19] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197, 2016.

[20] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[21] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[22] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[23] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[24] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[25] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[26] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834, 1983.

[29] Leo J Grady and Jonathan R Polimeni. *Discrete calculus: Applied analysis on graphs for computational science*. Springer Science & Business Media, 2010.

[30] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[31] Daniel A Spielman. Spectral graph theory and its applications. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 29–38. IEEE, 2007.

[32] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[33] Stanislav S Borysov, R Matthias Geilhufe, and Alexander V Balatsky. Organic materials database: An open-access online database for data mining. *PloS one*, 12(2), 2017.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[35] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[36] Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8(1):1–8, 2017.

[37] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[39] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

[40] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (xai): towards medical xai. *arXiv preprint arXiv:1907.07374*, 2019.

[41] Federico Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv: 1905.13686*, 2019.

[42] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001.

[43] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[44] Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*, 2020.