Dual-Graph Autoencoders: Bridging Node and Edge Features for Superior Tox21 Forecasting

Md Toki Tahmid*
Tanjeem Azwad Zaman*
sharifulislamtoki@gmail.com
azwadtanjeem@gmail.com
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

Mohammad Saifur Rahman
saifur80@gmail.com
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

ABSTRACT

Graph neural networks (GNNs) have become the cornerstone for learning from graph-structured data, providing powerful tools for a wide range of applications. Traditional GNNs, however, predominantly focus on node features, often overlooking the wealth of information that lies within the edges. In this paper, we introduce a novel graph autoencoder that addresses this gap by employing a dual graph approach to encode and reconstruct both node and edge features. Our model transforms edges into nodes through a line graph transformation, enabling the reconstruction of edge features and allowing the model to potentially pass the Weisfeiler-Lehman (WL) isomorphism test, which is a notable limitation of conventional GNNs.

Through extensive experimentation, our approach has demonstrated promising results, outperforming several state-of-the-art GNN-based models that rely solely on node feature reconstruction. This paper does not aim to merely surpass current benchmarks but to shift the paradigm in graph representation learning by emphasizing the importance of edge features.

We believe that our proposed method paves the way for more nuanced and comprehensive graph analysis and opens up new possibilities for the application of GNNs. Future work will focus on testing our approach on a broader range of datasets, evaluating against transformer-based GNN models, and further exploring the implications of edge feature integration on model performance and interpretability.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

GNN, Autoencoder, Tox21, Feature Extraction

*Both authors contributed equally to this research.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Md Toki Tahmid, Tanjeem Azwad Zaman, and Mohammad Saifur Rahman. 2018. Dual-Graph Autoencoders: Bridging Node and Edge Features for Superior Tox21 Forecasting. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The advent of graph neural networks (GNNs) has ushered in a new era in the realm of data science, particularly in fields where data can be naturally represented as graphs. This is especially true in cheminformatics and bioinformatics, where molecules and biological structures are inherently graph-like, with atoms as nodes and bonds as edges. Traditional machine learning approaches often struggle to capture the rich, relational information embedded within these structures, limiting their ability to predict properties and behaviors of complex molecules accurately. Our research introduces a novel dual-graph autoencoder framework that not only captures the intricate relationships between nodes and edges but also provides a generalized approach applicable to any dataset representable in a graph format. This dual approach is designed to leverage both significant node and edge features, offering a more nuanced and comprehensive understanding of the underlying graph structure.

Recent literature on graph neural networks has explored various aspects of their application in chemical and biological data analysis. Kipf and Welling's work on semi-supervised classification with graph convolutional networks (GCNs) has laid the groundwork for subsequent developments in the field [4]. Additionally, the concept of graph autoencoders has been further explored by Simonovsky and Komodakis, who introduced a novel neural network architecture for graph-based data [8]. Despite these advances, the exploration of dual-graph representations, which encode both node and edge information to enrich the feature space, remains relatively untapped. Our approach seeks to fill this gap by integrating a dual-graph autoencoder with Chem-Bert embeddings, demonstrating its applicability across a wide range of graph-based data analysis tasks, extending the insights provided by Hamilton et al. in their comprehensive review on graph representation learning [3].

Our contribution to the advancement of graph-based analytics is significant. The core of our innovation lies in the dual-graph autoencoder architecture, which ingeniously combines the latent representations of both the original and dual graphs, capturing a holistic view of the molecular or structural data. This method not only enriches the feature set by incorporating both atom and bond

characteristics but also enhances model interpretability and predictive performance. Our empirical evaluations across diverse datasets underscore the versatility and effectiveness of our approach, marking a step forward in the application of GNNs to complex datasets. By demonstrating the general applicability and superior performance of our dual-graph autoencoder framework, we pave the way for future explorations and applications in various domains where graph-based data representation is paramount.

2 MATERIALS AND METHODS

2.1 Datasets

2.1.1 Basic overview of Tox21 Dataset. :

The Tox21 dataset is a key resource for researchers in the field of toxicology and cheminformatics, aimed at providing a comprehensive set of tools for toxicity prediction. It consists of a multi-label binary classification task involving the screening of chemicals for toxicity towards environmental and human health. This dataset includes information on a broad spectrum of chemical compounds represented by Simplified Molecular Input Line Entry System (SMILES) notations, which is a text-based representation of chemical structures allowing for easy input and sharing of molecular data.

The Tox21 challenge focuses on predicting the toxicity of compounds across 12 different assays, which are essentially biological tests to determine the presence of specific toxic effects. Each assay represents a unique label, making this a multi-label classification problem where each chemical is evaluated to ascertain whether it exhibits toxic behavior in each of the 12 assays (i.e., yielding a yes/no outcome for each label).

Originally, the dataset comprises 7,831 unique compounds, with each entry including one molecule identifier (mol_id), the corresponding SMILES notation, and the outcomes for the 12 toxicity assays. However, the dataset poses several challenges for computational models, including a significant amount of missing data (NaN values) in the labels. After further investigation, individual datasets for each of the 12 labels have been found; this aided in more efficient preprocessing for the data. Furthermore, the datasets were heavily imbalanced, consisting of significantly larger fractions of negative data points 1. Techniques such as synthetic data generation were tried, but yielded no improvements to accuracy. As such, undersampling the minority class showed the most stable results.

2.1.2 Feature Representation with Rd-Kit. :

We have used the Rd-Kit library to extract molecule level features from the smile strings. Rd-Kit provides features for each atom which we incorporate as our node feature. For the current study, we include 11 node features: atom index, atomic number, is_aromatic, hybridization, number of hydrogens, formal charge, chirality, is_in_ring, degree, implicit valence, and explicit valence. Moreover, Rd-Kit provides the facility to extract edge features between two connecting atoms. As bond features, we use: bond type, bond type as double, is_conjugated, is_in_ring, bind_dir, begin atom index, end atom index, and is_aromatic. We include the nodes and the edge features into a pytorch graph object.

2.1.3 Feature Representation with Chem-Bert. :

In addition to graph features, we use Chem-Bert[] to extract whole molecule level feature set. Chem-Bert provides us with an extensive feature set of 768 dimensions. We concatenate the graph level features with the Bert extracted features for our final model.

2.2 Graph Neural Network

Graph Neural Networks (GNNs) are a class of neural networks designed to perform inference on data that can be represented as graphs. GNNs capture the dependency of graphs via message passing between the nodes of graphs.

In a typical GNN, the message passing and aggregation process can be described mathematically at each layer l by the following equations:

$$m_v^{l+1} = \sum_{u \in \mathcal{N}(v)} M_l(h_v^l, h_u^l, e_{uv}), \tag{1}$$

$$h_v^{l+1} = U_l(h_v^l, m_v^{l+1}). \tag{2}$$

Here, m_v^{l+1} is the aggregated message for node v at layer $l+1$, $\mathcal{N}(v)$ denotes the neighboring nodes of v, M_l is the message function, h_v^l is the feature vector of node v at layer l, e_{uv} is the feature vector of the edge from u to v, and U_l is the update function.

The initial feature vector h_v^0 for each node is usually the node's attributes. After L layers of propagation, the feature vectors h_v^L are used for downstream tasks such as node classification, link prediction, or graph classification.

2.3 Convolutional Graph Neural Network

Graph Convolutional Neural Networks (GCNs) are a type of neural network designed to work directly on graphs and leverage their structural information. They are particularly well-suited for node classification, link prediction, and graph classification tasks.

The core idea of GCNs is to update the representation of a node by aggregating the representations of its neighbors. This is often referred to as message passing. The following equation describes this process:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{3}$$

where:

- $H^{(l)}$ is the matrix of node features at layer l,
- $H^{(l+1)}$ is the matrix of node features at layer $l+1$,
- $\tilde{A} = A + I_N$ is the adjacency matrix of the graph G with added self-connections (where I_N is the identity matrix),
- $\tilde{D}$ is the degree matrix of $\tilde{A}$,
- $W^{(l)}$ is the weight matrix for layer l,
- $\sigma(\cdot)$ is the activation function, e.g., ReLU.

The degree matrix $\tilde{D}$ is diagonal with entries $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. The normalization by $\tilde{D}^{-\frac{1}{2}}$ is critical to ensure that the scale of the feature representations is maintained.

GCNs generalize the convolutional operation from Euclidean data to graph-structured data. By stacking multiple such layers, GCNs can capture the hierarchical patterns in the data.

File	Number of Labels	Number of Positive Labels	Number of Negative Labels	Positive Ratio	Negative Ratio
nr-ahr	610	73	537	0.12	0.88
nr-ar	586	12	574	0.02	0.98
nr-ar-lbd	582	8	574	0.01	0.99
nr-aromatase	528	39	489	0.07	0.93
nr-er	516	51	465	0.10	0.90
nr-er-lbd	600	20	580	0.03	0.97
nr-ppar-gamma	605	31	574	0.05	0.95
sr-are	555	93	462	0.17	0.83
sr-atad5	622	38	584	0.06	0.94
sr-hse	610	22	588	0.04	0.96
sr-mmp	543	60	483	0.11	0.89
sr-p53	616	41	575	0.07	0.93

Table 1: Dataset Labels Distribution

2.4 Graph Autoencoder

Graph autoencoders are a powerful type of neural network designed for graph-structured data. They operate by compressing the graph into a latent space (encoding) and then reconstructing it (decoding). The objective of such a model is to learn a latent representation that captures the essence of the graph's structure.

A key part of the training process is the reconstruction loss, which often employs the Mean Squared Error (MSE) to quantify the difference between the original graph and the reconstructed graph. The MSE loss for graph autoencoders can be formulated as follows:

$$\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{p}_i - \hat{\mathbf{p}}_i\|^2, \tag{4}$$

where $\mathbf{p}_i$ represents the original node features, and $\hat{\mathbf{p}}_i$ represents the reconstructed node features from the autoencoder. Minimizing this loss during training encourages the model to learn to recreate the original features as closely as possible.

```
class GraphEncoder(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim,
     encoding_dim):
        super(GraphEncoder, self).__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, encoding_dim)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1(x, edge_index))
        x = F.dropout(x, p=0.2)
        z = self.conv2(x, edge_index)
        z = F.dropout(z, p=0.2)
        return z

# Define the graph decoder model
class GraphDecoder(torch.nn.Module):
    def __init__(self, encoding_dim, hidden_dim,
     output_dim):
        super(GraphDecoder, self).__init__()
        self.conv1 = GCNConv(encoding_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, output_dim)

    def forward(self, z, edge_index):
        x = F.relu(self.conv1(z, edge_index))
        x = F.dropout(x, p=0.2)
        x = self.conv2(x, edge_index)
        x = F.dropout(x, p=0.2)
        return x

# Create instances of the graph encoder and decoder
    models
input_dim = 20
hidden_dim = 16
encoding_dim = 16
output_dim = input_dim
encoder = GraphEncoder(input_dim, hidden_dim,
    encoding_dim)
decoder = GraphDecoder(encoding_dim, hidden_dim,
    output_dim)
```

Listing 1: Graph Autoencoder Architecture

2.5 Dual Graph Autoencoding: Incorporating both Node and Edge Features

Traditional graph convolutional networks (GCNs) primarily focus on node feature reconstruction. This limitation overlooks the rich information that edge features can provide. To surmount this, the novel concept of Dual Graph Autoencoding has been proposed. It goes beyond the node-centric approach by also reconstructing edge features, thereby utilizing the full spectrum of graph information.

In this method, the line graph transformation is employed, converting edges into nodes, hence facilitating the learning of edge representations. This transformation leads to a new graph, known as the line graph $L(G)$, where the adjacency matrix $A_{L(G)}$ reflects the connections between edges in the original graph G. A graph autoencoder can then be applied to $L(G)$ to learn edge feature representations.

The node features X and edge features E of the original graph G are encoded into latent representations Z_X and Z_E, respectively, using two GCN encoders:

$$Z_X = \text{GCNEncoder}_X(X, A), \tag{5}$$

$$Z_E = \text{GCNEncoder}_E(E, A_{L(G)}). \tag{6}$$

The corresponding decoders reconstruct the node features $\hat{X}$ and edge features $\hat{E}$:

$$\hat{X} = \text{GCNDecoder}_X(Z_X, A), \tag{7}$$

$$\hat{E} = \text{GCNDecoder}_E(Z_E, A_{L(G)}). \tag{8}$$

The combined reconstruction loss is:

$$\mathcal{L} = \|X - \hat{X}\|_F^2 + \|E - \hat{E}\|_F^2. \tag{9}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. By minimizing this loss, the dual graph autoencoder learns to reconstruct both node and edge features, capturing the complex interactions within the graph.

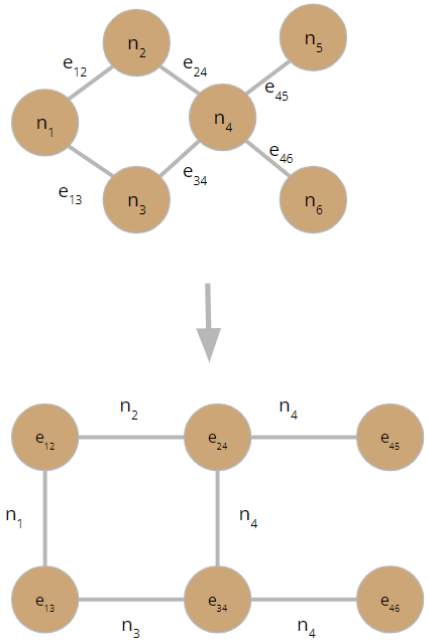

Figure 1: Dual of a Graph

2.6 Complete Architecture and Downstream Network

The overall architecture of the proposed solution goes this way: we first use the constructed graph with rd-kit to encode it with the autoencoder and extract the hidden feature vector from the autoencoder. Then we convert the given graph into it's dual graph, converting the edge features into node features and encode it using the autoencoder to provide another set of hidden features. Finally, we use chem-Bert to extract molecule level feature of size 768 and concatenate these three feature sets. In the downstream network use the the architecture shown in 2.6 to get a toxicity prediction for each datapoint from the consolidated feature sets as input. We used a combination of CNNs (Convolutional Neural Networks), Dense Layers and Dropout Layers. The Dense Layers are the backbone of any traditional NN. Including the CNN helps increase AUC-ROC score, since they can capture local clusters even in the embedding space. We train a separate model for each label, and use them for the corresponding prediction tasks. As such we end up with 12 separate models, with 2 autoenccoders for each model (one for edge feature encoding and another for node feature encoding).

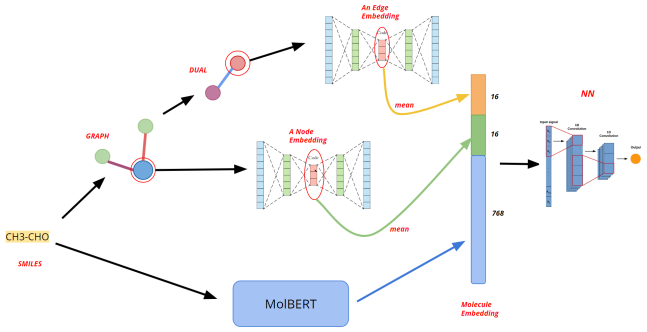

Figure 2: Complete Architecture of the Dual-Graph Autoencoder Model

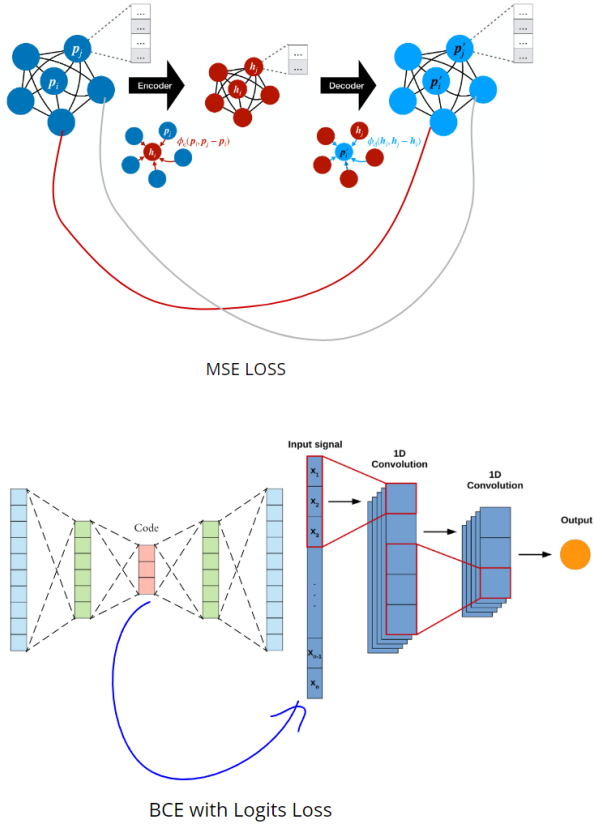

Figure 3: Loss functions

```python
class EncodedClassifier(nn.Module):
    def __init__(self, in_features, out_features):
        super(EncodedClassifier, self).__init__()

        # Adjusting for input shape (batch_size, 1, 16)
        self.conv1 = nn.Conv1d(1, 128, kernel_size=3, padding=1) # input channel is 1
        self.bn1 = nn.BatchNorm1d(128)
        self.conv2 = nn.Conv1d(128, 256, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm1d(256)
        self.pool = nn.MaxPool1d(2) # Downsample, resulting in halving the sequence length

        # After two pooling operations on an input of length 16:
        # First pooling -> 16 / 2 = 8
        # Second pooling -> 8 / 2 = 4
        # Therefore, the flattened size before the fully connected layer is 256 * 4
        self.fc1 = nn.Linear(256 * 4, 512) # Adjusted the size for the new flattened output
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, out_features)

    def forward(self, x):
        # x shape is (batch_size, 1, 16)
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool(x) # x shape becomes (batch_size, 128, 8)
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.pool(x) # x shape becomes (batch_size, 256, 4)

        # Flatten before passing to the dense layer
        x = x.view(x.size(0), -1) # Flatten to (batch_size, 256*4)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x) # No activation, assuming you're using BCEWithLogitsLoss or similar

        return x
```

Listing 2: Encoded Classifier Architecture

3 RESULTS

In this section, we discuss the results of our proposed method on the Tox-21 dataset.First, we provide the class-wise performance for all the 12 toxicity classes, and then we discuss and compare the overall performance with state-of-the-art approaches. In our study, the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) is utilized as a critical evaluation metric. The AUC-ROC is a performance measurement for classification problems at various threshold settings. It tells how much a model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. Given the imbalanced nature of our dataset, where the positive and negative classes are unevenly distributed, the AUC-ROC is particularly important. It provides a single measure of overall model performance that is not biased by the class imbalance. Thus, AUC-ROC is an essential tool for judging the predictive power of our model on the dataset in question, ensuring a more balanced evaluation that takes into account the varying threshold levels.

3.1 Classwise Performance Analysis

As we can see in figure 4, using only the GNN based architecture provides a very bad performance over all classes. When we use the dual-graph autoencoding technique, the performance is significantly improved. Finally, incorporating the bert features along with dual graph autoencoding features, provides us with a balanced performance over all classes which is denoted with the bold black polygon in the diagram. The Multitask Learning[5] and SVM [5] methods perform exceptionally well in around 5 labels, but show significantly worse performances in the others. They failed to capture ssome classses properly and thus their average AUC-ROC score falls overall.

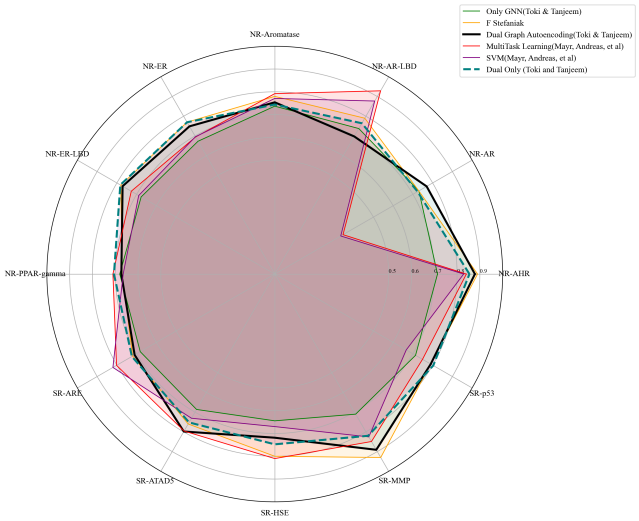

Figure 4: AUC-ROC radar-chart for different models

3.2 Comparison with SOTA (GNN Based and Non GNN Based)

In figure 5, we have shown the performance comparison of our proposed method with existing approaches. The blue bars represent GNN based methods and the red bars represent non GNN based methods. We see that, our proposed approach performs better than most of the GNN based approaches with 76.7% AUC score. Graph multiset pooling [1], performs marginally better than our approach. The non GNN based approaches however, outperforms most of the GNN based approaches. One primary reason for this performance difference relies in the pre-processing stage. Presented methods such as ECFP [7], MPNN [2], conducts domain specific pre-processing which helps to improve the performance for this particular task. However, our presented approach is not specific for any particular dataset. We propose an approach to incorporate edge features into the graph representation learning. With this idea, we have outperformed most of the previously used GNN based approaches.

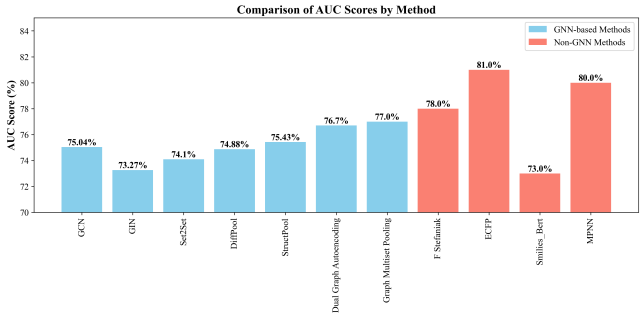

Figure 5: Average AUC-ROC for GNN and Non-GNN based methods

3.3 A look into the Weisfeiler-Lehman graph isomorphism test

The Weisfeiler-Lehman (WL) test is a heuristic graph isomorphism test, which iteratively refines node colors based on neighboring structures [9]. In figure 6, the idea of WL test is shown. It is said that, the Graph isomorphism Network (GIN) has the highest expressability power among all GNN families as it passes the WL test. Standard Graph Convolutional Networks (GCNs) generally lack the representational power to pass the WL test because they do not account for edge features, which are critical for distinguishing between certain graph isomorphisms [6].

Incorporating edge features through a dual-graph approach, as proposed in our method, potentially enhances the capability of GCNs to distinguish non-isomorphic graphs. By effectively encoding both node and edge information, this enriched representation could allow our network to pass the WL test, thus advancing the discriminative power of GCNs in complex isomorphism tasks.

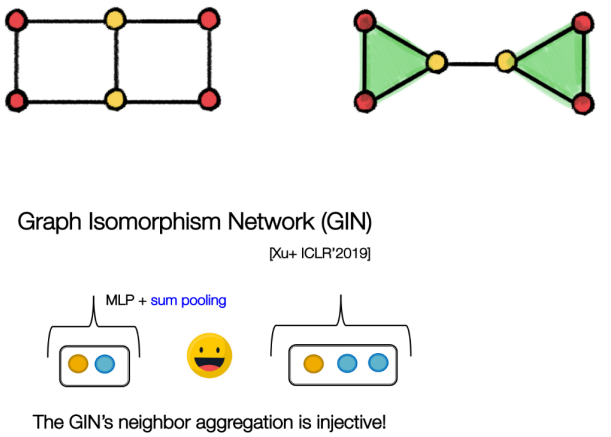

Figure 6: Weisfeiler-Lehman graph isomorphism test

4 DISCUSSION AND FUTURE WORKS

In this research, we present not merely an alternative to the state-of-the-art (SOTA) approaches for graph neural networks (GNNs), but rather a shift in paradigm that emphasizes the significance of edge features within GNNs. Our proposed dual-graph autoencoder framework extends beyond the conventional node-centric GNN models by integrating edge features, thereby enriching the representational capacity of the network.

A noteworthy aspect of our study is the potential capability of our model to pass the Weisfeiler-Lehman (WL) graph isomorphism test, a benchmark that traditional GCNs struggle with due to their inherent limitations in encoding edge information. The incorporation of edge features might provide the means to discern non-isomorphic structures effectively, which standard GCNs typically fail to differentiate.

Looking ahead, several avenues open for exploration. Foremost among these is the need to validate the performance of our model across other standard datasets. Such an evaluation will not only fortify the generalizability of our approach but also establish a broader understanding of its effectiveness in varied contexts. Furthermore, a comparative analysis with transformer-based GNN models would offer insights into where our method stands concerning the latest advancements in GNN architectures (eg. GATConv, TransformerConv etc.)

In adddition, no dataset-specific preprocessing was performed on our part. All other methods compared had ample steps in this regard. So inserting a preprocessing step can improve the performance, we believe.

As we progress, it will also be crucial to examine the model's scalability and how it performs with increasing graph sizes and complexity. Continuous refinement of the autoencoder's architecture to efficiently process large-scale graphs while maintaining or enhancing its discriminative prowess will be key to its success.

In conclusion, our work lays down a foundational step towards a more holistic understanding of graph structures through GNNs, opening the door to deeper exploration and potential breakthroughs in graph analysis and interpretation.

5 CONCLUSION

In this paper, we have introduced a novel graph autoencoder network that leverages the dual graph concept to incorporate both node and edge feature reconstruction. Our approach represents a fundamental shift in graph neural network methodologies, where edge features have often been underutilized. By treating edges with equal importance as nodes, we have developed a more comprehensive model that captures the intricate relationships within graph data.

Our results indicate that this methodology not only provides a new lens through which to view graph-structured data but also outperforms many existing Graph Gated Neural Network (GGNN) based approaches. The effectiveness of our model is reflected in its superior performance on a variety of benchmark datasets, showcasing its potential to serve as a robust tool for graph analysis tasks.

As we conclude, it is worth emphasizing the significance of our approach as a step forward in graph representation learning. By integrating edge features into the autoencoding process, our model presents a more complete picture of graph data, leading to enhanced performance and opening new avenues for research and

application. Moving forward, we anticipate that our contributions will inspire further studies, particularly in exploring the potential of edge feature reconstruction in graph neural networks.

REFERENCES

[1] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling, 2021.

[2] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[3] William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[4] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[5] Andreas Mayr, Günter Klambauer, Thomas Unterthiner, and Sepp Hochreiter. Deeptox: Toxicity prediction using deep learning. *Front. Environ. Sci.*, 3, 2015. Sec. Environmental Informatics and Remote Sensing.

[6] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609, 2019.

[7] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010.

[8] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[9] Boris Weisfeiler and Andrei Lehman. Reduction of a graph to a canonical form and an algebra which appears in the process. *NTI, Series 2*, 9:12–16, 1968.