The expressive power of pooling in Graph Neural Networks

Filippo Maria Bianchi*
Dept. of Mathematics and Statistics
UiT the Arctic University of Norway
NORCE, Norwegian Research Centre AS
filippo.m.bianchi@uit.no

Veronica Lachi*
Dept. of Information Engineering
and Mathematics
University of Siena
veronica.lachi@student.unisi.it

Github Code Link :- Reproduce-ML-Challenge

Reproduced By :-
Birva Oza (202311050)
Parshwa Dand (202311016)
Harsh Vyas (202311015)
Ayushi Mehta (202311008)

Reproducibility Summary

In our research, we conducted an extensive investigation into the expressive power of Graph Neural Networks (GNNs) by evaluating their ability to preserve graph isomorphism across various pooling operators. Specifically, we explored the impact of pooling operators such as Comp Graclus, Dense Random Pooling, Top-K Pool, ASAPool, MinCutPool, and DiffPool on the preservation of graph isomorphism. Our methodology involved subjecting input graphs to each pooling operator and subsequently testing the resulting graph representations for isomorphism. Graph isomorphism tests were conducted to determine whether the original and pooled graphs retained their structural equivalence, thereby assessing the effectiveness of each pooling operator in capturing and preserving graph topology.

Through meticulous experimentation and analysis, we scrutinized the performance of each pooling operator in maintaining graph isomorphism. By systematically evaluating these operators across a range of datasets and scenarios, we gained valuable insights into their relative strengths and weaknesses in preserving graph structure.

In addition to evaluating various pooling operators' impact on graph isomorphism, we meticulously tuned hyperparameters to optimize the performance of our Graph Neural Networks (GNNs). These hyperparameters, including learning rates, epochs, and batch sizes, were carefully selected to ensure consistency and reproducibility across our experiments. Additionally, we ensured that our experiments were conducted on hardware configurations that met the system requirements necessary for efficient model training and evaluation.

Contribution

Parshwa and Ayushi collaborated closely throughout the project, jointly accomplishing various tasks. They worked together to import the EXPWL1 dataset and conducted preprocessing tasks necessary to make the dataset ready for experimentation. Additionally, they co-led the implementation and optimization of graph isomorphism tests. They explored and implemented several pooling operators, including DiffPool, panpool, graclus, ASAPool, and dense random pooling. They worked collaboratively to fine-tune the parameters of these pooling operators, aiming to optimize their effectiveness in preserving graph isomorphism while ensuring computational efficiency.

Birva and Harsh collectively experimented with MinCutPool, Top-k Pool, SAGPool, EdgePool, and DMON Pooling operators, aiming to provide a comprehensive assessment of different pooling methodologies. They fine-tuning hyperparameters, recognizing the critical role they play in optimizing the performance of Graph Neural Networks (GNNs) with different pooling operators. By meticulously adjusting parameters such as learning rates, epochs, and batch sizes, they sought to achieve superior results while ensuring the reproducibility of experimental findings. They meticulously documented the experimental procedures, code implementations, and parameter configurations.

On challenging tasks like implementing Comp Graclus and k-MIS pooling, as well as incorporating Graph Convolutional Networks (GCNs), the team demonstrated effective teamwork and shared responsibility. Each member contributed to the implementation effort, ensuring accuracy and completeness in documenting the research process. Throughout the implementation process, team members provided support and assistance to one another, fostering a collaborative environment. Notably, each team member possessed a comprehensive understanding of every aspect of the research, including the intricacies of GCN implementation and its integration into the project's framework.

Introduction

In recent years, Graph Neural Networks (GNNs) have garnered significant attention for their ability to analyze and comprehend graph-structured data. A pivotal aspect in advancing GNN research lies in understanding their expressive power, particularly concerning their efficiency in discerning graph isomorphism. This understanding is pivotal for pushing the boundaries of GNN capabilities and ensuring their effectiveness across diverse applications.

This paper centers on the reproducibility of research endeavors aimed at characterizing the expressive power of GNNs. Of particular interest is the exploration of methodologies and techniques employed in evaluating GNN strengths and limitations, with a focus on assessing their capabilities against the Weisfeiler-Lehman (WL) isomorphism test. The introduction of hierarchical pooling operators within GNN architectures has ushered in new challenges and prospects for understanding their expressive powers. However, evaluating the efficacy of these operators remains intricate, often necessitating empirical measures influenced by factors such as network architecture such as activation functions, normalization or dropout layers, and optimization algorithms and dataset peculiarities.

In light of these complexities, this paper undertakes a comprehensive analysis of the reproducibility of research findings concerning GNN expressive power assessment. We propose a principled criterion grounded in theoretical underpinnings and empirical validation for evaluating the effectiveness of graph pooling operators, especially in conjunction with message-passing (MP) layers. Our aim is to facilitate informed decision-making in selecting and designing pooling operators while mitigating criticisms and misconceptions surrounding graph pooling methodologies.

Scope Of Reproducibility

- **Evaluation Methodologies:** Detail the evaluation methodologies used in existing research for assessing the expressive power of Graph Neural Networks (GNNs), including the Weisfeiler-Lehman (WL) isomorphism test and message-passing (MP) layers.

- **Hierarchical Pooling Operators:** Investigate the reproducibility of findings related to hierarchical pooling operators introduced in GNN architectures and their impact on the expressive power of GNNs.
- **Theoretical Analysis:** Conduct a theoretical analysis to elucidate the conditions under which graph pooling operators preserve the expressive power of GNNs, drawing insights from previous research and proposing new criteria for evaluation.
- **Comparison with Existing Criteria:** Compare and contrast the proposed criteria for evaluating graph pooling operators with existing methodologies, highlighting their strengths, limitations, and implications for reproducibility.
- **Practical Applications:** Explore the practical implications of reproducible research findings in guiding the selection and design of pooling operators for real-world applications of GNNs in graph analysis tasks.

Methodology

The core objective of evaluating the expressive power of GNNs, which lies in their ability to discriminate between non-isomorphic graphs. We delve into the Weisfeiler-Lehman (WL) isomorphism test, a computationally efficient algorithm used as a benchmark for graph isomorphism detection. Through iterative color assignments based on the multiset of labels of neighboring vertices, the WL test achieves graph differentiation. We draw parallels between the aggregation scheme of message-passing (MP) layers in GNNs and the iterative process of the WL test. Notably, prior research has established that GNNs utilizing MP operations are at most as powerful as the WL test in discerning distinct graph features. Moreover, the injective nature of MP operations ensures that GNNs achieve parity with the WL test in discriminatory capabilities. Our focus predominantly centers on the Graph Isomorphism Network (GIN), which embodies such an injective multiset function, equating its expressive power with that of the WL test under specific conditions. Here, x is node features.

$$\mathbf{x}_v^l = \mathtt{MLP}^{(l)}\left((1+\epsilon^l)\mathbf{x}_v^{l-1} + \sum_{u\in\mathcal{N}[v]} \mathbf{x}_u^{l-1}\right)$$

Transitioning to the realm of graph pooling operators, we adopt the Select-Reduce-Connect (SRC) framework to articulate their fundamental functions. The selection function (SEL) serves to cluster nodes into supernodes, governed by membership scores that dictate the assignment of nodes to distinct supernodes. The reduction function (RED) consolidates the features of nodes within each supernode, thereby generating pooled vertex features. Finally, the connect function (CON) orchestrates the establishment of connections between supernodes. A critical component of our methodology involves devising a formal criterion to ascertain the expressive power of graph pooling operators. Leveraging theoretical underpinnings, we delineate three sufficient conditions that delineate when a pooling operator preserves the expressive power of preceding MP layers.

1. $\sum_i^N \mathbf{x}_i^L \neq \sum_i^M \mathbf{y}_i^L$;
2. *The memberships generated by* SEL *satisfy* $\sum_{j=1}^K s_{ij} = \lambda$, *with* $\lambda > 0$ *for each node* i, *i.e., the cluster assignment matrix* $\mathbf{S}$ *is a right stochastic matrix up to the global constant* λ;
3. *The function* RED *is of type* RED : $(\mathbf{X}^L, \mathbf{S}) \mapsto \mathbf{X}_P = \mathbf{S}^T \mathbf{X}^L$;

then $\mathcal{G}_{1_P}$ *and* $\mathcal{G}_{2_P}$ *will have different nodes features, i.e., for all rows' indices permutations* $\pi : \{1, \ldots K\} \rightarrow \{1, \ldots K\}$, $\mathbf{X}_P \neq \Pi(\mathbf{Y}_P)$, *where* $[\Pi(\mathbf{Y}_P)]_{ij} = \mathbf{y}_{P_{\pi(i)j}}$.

Here, x and y are node features.

X and Y contain all nodes features.

S is a membership matrix of nodes in supernodes

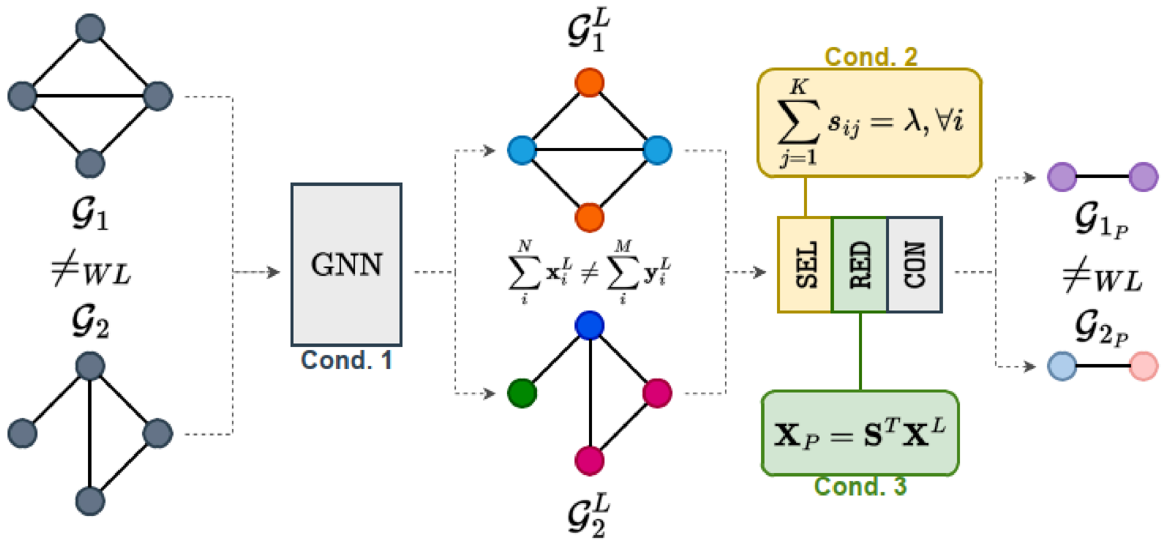

Fig 1. A GNN with expressive MP layers

EXPWL1 dataset loads preprocessed data and slices from a saved file during initialization. The class provides properties for specifying the names of raw and processed data files. The process method reads raw data from a pickle file, applies optional pre-filtering and pre-transformation functions, and collates the data into a format compatible with PyTorch. The processed data and slices are then saved to a specified path.

Initialization: The GIN_Pool_Net class initializes the GNN with various parameters, including the size of node features, the number of output classes, the number of GIN layers before and after pooling, the dimensionality of node embeddings, the activation function, and others.

Pre-Pooling Block: The GNN comprises a pre-pooling block with multiple layers of GINConv or PANConv operations. These layers process the node features before pooling.

Pooling Block: Depending on the specified pooling method, different pooling operations are applied to the node embeddings. The supported pooling methods include DiffPool, MinCutPool, DMoNPooling, Dense-Random Pooling, TopKPooling, SAGPooling, ASAPooling, EdgePooling, KMISPooling, Graclus, and Comp-Graclus. The pooling operation reduces the size of the graph by aggregating nodes.

Post-Pooling Block: After pooling, the GNN applies additional GINConv or DenseGINConv operations to the pooled embeddings.

Readout: Finally, the pooled node embeddings are aggregated to obtain a graph-level representation, followed by a multi-layer perceptron (MLP) to generate logits for classification.

Reset Parameters: The method reset_parameters initializes the parameters of the network.

Forward Pass: The forward method executes the forward pass of the network, applying the pre-pooling block, pooling operation, post-pooling block, and readout. Optionally, auxiliary loss terms are computed during pooling operations such as DiffPool, MinCutPool, and DMoNPooling.
Then we train our GNN model by using different pooling operators to check their effectiveness. The `train` function conducts training epochs, iterating over batches of data to compute gradients and update model parameters using the specified optimizer. It also incorporates an optional auxiliary loss for certain pooling methods. Meanwhile, the `test` function assesses the trained model's performance on validation and test datasets by calculating accuracy and loss metrics. The `log` function facilitates organized logging of key training and testing metrics. The training loop executes multiple runs,

where the dataset is shuffled and split into training, validation, and test sets for each run. Within each run, the GNN model is instantiated with specific configurations, trained, and evaluated. Notably, the best test accuracy achieved during each run is recorded for comparison. After all runs are completed, the mean and standard deviation of test accuracies are computed, providing insight into the model's overall performance across different pooling methods. This systematic approach enables comprehensive evaluation and comparison of GNN models with varying pooling strategies, contributing to a deeper understanding of their effectiveness in graph representation learning and classification tasks.

Pooling Operators

- ### Dense Pooling Operators

  The density of a pooling operator is defined as the expected value of the ratio between the cardinality of a supernode and the number of nodes in the graph. A method is referred to as dense if the supernodes have cardinality O(N). Besides being dense, all these operators are also trainable and fixed. All dense methods preserve the expressive power of the preceding MP layers. The types of dense pooling operators are:-

  DiffPool :- The 'diffpool' pooling method is integrated into the graph neural network architecture through a combination of a linear layer for pooling and a dense graph convolution operation. In the initialization stage, a linear layer is created with specific input and output sizes to facilitate the pooling process. During the subsequent pooling block, if 'diffpool' is chosen as the pooling method, a dense graph convolution operation is conducted using a predefined MLP architecture. In the forward pass phase, if 'diffpool' is specified, the input graph data undergoes conversion to a dense batch representation alongside a mask, while the adjacency matrix is transformed into a dense adjacency matrix. Following this, the pooled representations, updated adjacency matrix, and associated loss components are computed utilizing a dedicated pooling function. Finally, an auxiliary loss is derived based on the calculated losses. Upon completion of the pooling operation.

  MinCutPool :- The implementation of MinCut Pool is the same as DiffPool, like taking a linear layer for pooling and a dense graph convolution operation. The difference is just in calculating auxiliary loss.

DMoN :- Initially, when the pooling method is 'dmon', an instance of the 'DMoNPooling' class is instantiated with specific parameters. This pooling mechanism is designed to perform DMoN (Dense Matrix Operation Network) pooling. Moving on to the post-pooling block, an architecture for a Multilayer Perceptron (MLP) is defined, comprising three hidden layers, all with the same size. If the specified pooling method belongs to a predefined set ['diffpool', 'mincut', 'dmon', 'dense-random'], a dense graph convolution operation is incorporated into the list of convolutional layers using the provided MLP architecture. During the forward pass phase, if the chosen pooling method is 'dmon', the input graph data undergoes a transformation into a dense batch representation, accompanied by a mask to identify active nodes. Additionally, the adjacency matrix is converted to a dense format to facilitate computations. Following the pooling operation, an auxiliary loss is computed based on the generated loss components. These losses are weighted and combined to form the auxiliary loss, which contributes to the overall training objective.

- ## Sparse Pooling Operators

A pooling operator is considered sparse if the supernodes generated have constant cardinality O(1). Some sparse pooling operations retain the expressiveness and some cannot. These methods are trainable and adaptive.

Non-expressive pooling operators :-

These operators produce a pooled graph that is a subgraph of the original graph and discard the content of the remaining parts. This hinders the ability to retain all the necessary information for preserving the expressiveness of the preceding MP layers. Some Operators of this kind are:-

Top-k Pooling :- During the forward pass phase, if the chosen pooling method is 'top-k', the input graph data is processed through the top-k pooling mechanism. This operation selects the top-k nodes in the graph based on a specific criterion, such as node features or edge attributes. The resulting pooled representations, along with the updated adjacency matrix and other relevant parameters, are obtained from the 'TopKPooling' function. Specifically, the Top-k method ranks nodes based
on a score obtained by multiplying the node features with a trainable projection vector.

ASAPool :- If the designated pooling method is 'ASAPool', the input graph data undergoes processing through the 'ASAPooling' mechanism. This operation dynamically adapts the graph structure by considering the features of the nodes. It intelligently selects nodes and edges to construct a refined graph representation that captures essential information for downstream tasks. ASAPool, instead, examines all potential local clusters in the input graph given a fixed receptive field and it employs an attention mechanism to compute the cluster membership of the nodes. The clusters are subsequently scored using a particular MP operator.

SAGPool :- If the designated pooling method is 'SAGPool', the input graph data undergoes processing through the 'SAGPooling' mechanism. This operation leverages self-attention mechanisms to assign importance scores to individual nodes based on their features and relationships within the graph. Subsequently, nodes with higher importance scores are selected for retention, while less significant nodes are pruned, effectively reducing the size of the graph representation. SAGPool simply replaces the projection vector with an MP layer to account for the graph's structure when scoring the nodes.

PanPooling :- If the designated pooling method is 'PANPool', the network's pre-pooling layers are constructed using 'PANConv' modules. These modules enable the aggregation of information across neighborhoods by convolving node features with a specified filter size, enhancing the network's ability to capture local and global graph structures effectively. This operation involves aggregating information across different neighborhoods in the graph, facilitated by the 'PANPooling' module. The module dynamically adjusts the pooling ratio to optimize information exchange and aggregation, ensuring comprehensive coverage of the graph's structure and features.

Expressive Pooling Operators :-

Graclus :- The network applies the Graclus algorithm to partition the graph into clusters. This partitioning process involves identifying cohesive groups of nodes within the graph, ensuring that nodes within the same cluster are densely connected while nodes across different clusters have sparse connections. Once the graph is partitioned into clusters, the network aggregates information within each cluster to produce cluster-level representations. This aggregation is performed by summarizing the node features within each cluster, capturing the essential characteristics of the cluster's subgraph. A greedy bottom-up spectral clustering technique, Graclus is non-trainable and matches each vertex with the neighbor that is nearest based on graph connection. When performing graph pooling with Graclus, the RED function is typically implemented as a max_pool operation among the points designated for the same cluster.

k-MISPool :- In K-MIS Pooling, each node in the input graph is assigned a score, indicating its importance or relevance. This scoring process can be based on various criteria, such as node features, connectivity patterns, or learned parameters. Common scoring functions include linear transformations followed by sigmoid activations, random scores, constant scores, or custom functions tailored to the problem context. Optionally, a heuristic can be applied to refine the node scores. One common heuristic is the "greedy" approach, where nodes are iteratively selected based on their scores while considering their neighbors.

After scoring and applying, the top-K nodes with the highest scores are selected to form the Maximal Independent Set (MIS). The parameter K determines the size of the MIS and influences the level of graph coarsening. The selected nodes serve as the basis for restructuring the original graph, resulting in a smaller, more compact representation. This restructuring process involves updating the graph connectivity information to reflect the selected nodes and their relationships. Information from the selected nodes is aggregated to create a condensed representation of the graph. This aggregation can involve various operations, such as summing node features. It aggregates the features of the vertex assigned to the same centroid with a sum_pool operation to create the features of the supernodes.

Edge-Pool :- In EdgePooling, the focus is on selecting a subset of edges from the original graph based on their importance or significance. This selection process typically involves scoring each edge according to certain criteria, such as edge weights, edge features, or their importance to the overall graph structure. Once the edges are scored, a subset of edges with the highest scores is chosen to form the pooled graph. This selection is essential for retaining the critical connectivity patterns and preserving the graph's structural integrity. After selecting the edges, the graph is restructured based on the chosen edges.

- ## Other Pooling Operators :-

Comp Graclus :- It is a graph pooling technique that reduces the size of the input graph while preserving its structure. It uses the Graclus clustering algorithm to form initial clusters, then identifies complementary clusters to capture remaining connectivity patterns. The combined clusters form the pooled graph representation, maintaining critical structural properties.

Dense Random Pooling :- It downsizes graphs by randomly selecting a subset of nodes. This method ensures diversity in node selection without considering specific connectivity patterns. The selected nodes form the basis of the pooled graph, potentially resulting in dense connectivity. This approach effectively reduces the graph's size while preserving its key features.

Dataset

This research focuses on assessing how combining MP layers with pooling layers affects expressive power. However, current benchmark datasets, both real-world and synthetic, aren't suitable for this comparison because they weren't designed to directly measure GNNs' power against the WL-test. A recent dataset called EXP was created for this purpose, but it's beyond the scope of our study. Hence, we've developed a modified version named EXPWL1. It includes graphs representing propositional formulas, each consisting of pairs (Gi, Hi) that are distinguishable by the WL test, encoding formulas with opposite SAT outcomes. Any GNN with the same expressive power as the WL test should be able to distinguish them with close to 100% accuracy. They've expanded the dataset to 3000 graphs, with each graph now averaging 55 to 76 nodes, to allow for aggressive pooling without simplifying the graph structure excessively. The EXPWL1 dataset and the code to reproduce our experiments are publicly accessible.

Hyperparameters

In our experimentation, we employed various hyperparameters to train our models effectively. We utilized the in_channels parameter to specify the size of node features in our input data. Similarly, out_channels was determined based on the number of classes in our dataset. For the architecture of our models, we configured the number of GIN (Graph Isomorphism Network) layers before pooling with num_layers_pre, and the number of GIN layers after pooling with num_layers_post. These parameters determined the depth of our model's architecture. To control the dimensionality of node embeddings, we set hidden_channels to a value of 64. This parameter influenced the complexity of our model's internal representations. We chose to normalize the layers in the GIN MLP using the norm parameter, which helped stabilize the training process. The choice of activation function for the MLP in the GIN model was specified with the activation parameter. We opted for the ELU activation function. For dense pooling methods, such as Comp Graclus, we used the average_nodes parameter, which was necessary for calculating the average number of nodes per graph. For the random pool method, we specified the maximum number of nodes to select with the max_nodes parameter.

Additionally, we specified the type of pooling method to use with the pooling parameter, allowing us to choose between different pooling strategies such as Comp Graclus or Dense Random Pooling. Loss functions are of two types, 1) Auxiliary Loss :- Some Pooling Operators return this loss and some do not have this loss and 2) Primary Loss :- it is just the negative log likelihood between original graph and output graph.

Finally, we set the pooling ratio with the pool_ratio parameter, which determines the fraction of nodes to retain after pooling and by default its value is 0.1. During training, we maintained consistency across all experiments by using a fixed learning rate of 1e-4, running each experiment for 500 epochs with a batch size of 32. Additionally, we conducted a single run for each experiment to ensure reproducibility of results.

Computational requirements

Table 1 presents the overall duration of each experiment. It's evident that incorporating the differentiable Pooling Techniques into the training process notably affects the total training time due to the Different time required for Different pooling operations. Our experiments were conducted on a machine with the specifications outlined in Table 2.

Pooling	**s/epoch(In paper)**	**s/epoch(Our observation)**
No-pool	0.33s	1.98s
DiffPool	0.69s	2.00s
DMoN	0.75s	2.28s
MinCut	0.72s	1.90s
ECPool	20.71s	14.78s
Graclus	1.00s	1.92s

Pooling	**s/epoch(In paper)**	**s/epoch(Our observation)**
k-MIS	1.17s	2.4s
Top-k	0.47s	1.55s
PanPool	3.82s	5.03s
ASAPool	1.11s	3.37s
SAGPool	0.59s	1.55s
Rand-dense	0.41s	1.67s
Cmp-Graclus	8.08s	10.00s

Table 1. Time statistics for each experiment.

OS	Windows 11
Storage	1TB SSD
CPU	AMD Ryzen-7840HS (3.80 GHz)
GPU	GeForce RTX 4050
RAM	16 GB Ram (5600 MHz)

Table 2. Hardware Components

Results

Our findings corroborate the assertions made by the authors. The model trained with Dense pooling operators and Expressive sparse pooling operators, satisfies the expressiveness conditions while Non-expressive sparse pooling operators doesn’t. If accuracy is close to 100% then we can say that the pooling operator is expressive. The results that are provided in this paper and the results that we are getting are very close enough.

Results reproducing original papers

Pooling	Test Acc(In paper)	Test Acc(Our observation)	Expressive
No-pool	99.3	99.7	✓
DiffPool	97.0	98.3	✓
DMoN	99.0	98.6	✓
MinCut	98.8	99.6	✓
ECPool	100.0	100.0	✓
Graclus	99.9	99.7	✓
k-MIS	99.9	100.0	✓
Top-k	67.9	49.6	X
PanPool	63.2	57.6	X
ASAPool	83.5	84.3	X
SAGPool	79.5	65.3	X
Rand-dense	91.7	93.0	✓
Cmp-Graclus	91.9	89.6	✓

Table 3. Results reproducing

Work Beyond Paper

Utilization of Graph Convolutional Networks (GCNs)

In addition to reproducing the research outlined in the original paper, this project incorporated the utilization of Graph Convolutional Networks (GCNs) as an extension to the existing methodology. GCNs have gained significant attention in recent years due to their effectiveness in modeling graph-structured data.

Background and Rationale:

Graph Convolutional Networks (GCNs) are a class of neural network architectures designed to operate directly on graph-structured data. Unlike traditional convolutional networks, which operate on regular grid-like data such as images, GCNs can effectively capture the relational information inherent in graph structures.

The decision to incorporate GCNs into the project stems from their ability to handle graph-structured data and their potential to enhance the performance of the model in tasks related to graph analysis and representation learning.

Results and Analysis:

Pooling	**Test Acc**
No-pool	52.33
DiffPool	53.67
DMoN	47.33
MinCut	53.67
ECPool	51.33
Graclus	51.00
k-MIS	46.00
Top-k	48.67
PanPool	55.67
ASAPool	51.33
SAGPool	50.00
Rand-dense	46.00
Cmp-Graclus	48.33

Table 4. Results of GCN

Using Graph Convolutional Networks (GCNs) for isomorphic testing could be an interesting approach, but it may face some challenges due to the nature of isomorphism and the capabilities of GCNs

GCNs are powerful tools for learning representations of graph-structured data and performing tasks such as node classification, link prediction, and graph classification. However, directly applying GCNs to isomorphism testing might not be straightforward due to the following reasons:

- **Permutation Invariance:** Isomorphism testing requires that the algorithm is invariant to the permutation of node labels. GCNs typically rely on fixed node orders or node features, which may not be invariant to permutations. Therefore, directly applying GCNs to isomorphism testing without considering this invariance property may lead to incorrect results.

- **Complexity:** Isomorphism testing is computationally expensive, especially for large graphs. GCNs may struggle with capturing the subtle structural similarities between graphs required for accurate isomorphism testing, especially in the presence of noise or sparse data.

- **Limited Expressiveness:** While GCNs are effective for learning local and global graph features, they may not capture the fine-grained structural details necessary for isomorphism testing. Isomorphism testing often requires comparing the entire graph structure, which may be challenging for GCNs to accomplish.

Conclusion

Our methodology involved a systematic evaluation of various pooling operators within the context of Graph Neural Networks (GNNs) to ascertain their impact on graph isomorphism preservation. We carefully curated a selection of pooling operators, including PanPool, k-MISPool, Top-K Pool, DMoNPool, ASAPool, SAGPool, MinCutPool, DiffPool, Graclus, Comp-Graclus and Dense Random Pooling. Each operator offers unique mechanisms for aggregating and condensing graph information. We established a consistent experimental setup across all pooling operators to ensure fair comparison.

This included fixed hyperparameters such as learning rates, epochs, batch sizes, and a single run for each experiment to maintain result consistency.

We implemented each pooling operator within our GNN framework and evaluated their performance in preserving graph isomorphism. This involved training the GNN models on graph datasets and assessing the similarity between original and pooled graphs using graph isomorphism tests. We meticulously analyzed the results obtained from each pooling operator, examining their effectiveness in preserving graph structures. This analysis was informed by both quantitative measures, such as accuracy and efficiency metrics, and qualitative assessments of graph representations.

Finally, we compared the results across different pooling operators and discussed their implications. By identifying trends and patterns in the performance of each technique, we gained insights into the relative strengths and weaknesses of various pooling strategies. Our methodology facilitated a comprehensive exploration of pooling operators in GNNs, shedding light on their role in graph representation learning and their impact on graph isomorphism preservation.

What was Easy

It's notable that the dataset used for this task is publicly available (EXPWL-1), facilitating its processing for experimentation. Moreover, many of the pooling layers utilized in this study are directly available in torch.geometric, simplifying the implementation process for evaluating different pooling methods' expressiveness. Additionally, the paper presents explanations in a concise and straightforward manner, making it easier for readers to understand each aspect of the research methodology and findings.

What was Hard

It's pertinent to highlight that Graph Neural Networks (GNNs) represent a relatively recent and rapidly evolving field of study. As such, comprehending the intricate terminology and concepts associated with GNNs can be time-consuming and challenging due to the wealth of ongoing research in this area. Furthermore, compared to more established neural network architectures, resources and libraries specific to GNNs may be less readily available, adding complexity to the implementation process.

Moreover, some pooling methods explored in the study may not be readily available in existing libraries, necessitating their implementation from scratch. This task can be daunting, as it requires a deep understanding of various concepts and algorithms associated with graph representation learning. Additionally, while the dataset used in the paper was well-processed and readily accessible, sourcing similar high-quality data for future research endeavors can pose a significant challenge.

Furthermore, creating one's dataset tailored to specific research requirements can be arduous and resource-intensive. It involves meticulous data collection, preprocessing, and annotation processes, which may require domain expertise and substantial time investment. Training GNN is computationally expensive and time consuming.

References

[1] A. Menzli, “Graph Neural network and some of GNN applications: everything you need to know,” *neptune.ai*, Sep. 11, 2023. Available: https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications

[2] J. Lee, I. Lee, and J. Kang. Self-attention graph pooling. In the *International conference on machine learning*, pages 3734–3743. PMLR, 2019.

[3] Antonio Longa, “Pytorch Geometric tutorial: Graph pooling DIFFPOOL,” *YouTube*. Jun. 24, 2021. Available: https://www.youtube.com/watch?v=Uqc3O3-oXxM

[4] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, “Understanding pooling in graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 2, pp. 2708–2718, Feb. 2024, doi: 10.1109/tnnls.2022.3190922. Available: https://arxiv.org/abs/2110.05292

[5] “torch_geometric.nn — pytorch_geometric documentation.” Available: https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#pooling-layers

[6] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller. Graph clustering with graph neural networks. *J. Mach. Learn. Res.*, 24:127:1–127:21, 2023.