UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022

[Re] G-Mixup: Graph Data Augmentation for Graph Classification

Ermin Omeragić[1] and Vuk Đuranović[1]

[1]University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, 1000 Ljubljana, Slovenia

Edited by
(Editor)

Received
–

Published
–

DOI
–

Reproducibility Summary

Scope of Reproducibility — This paper presents a novel augmentation method used for graph classification tasks: $\mathcal{G}$-Mixup. Our goal is to reproduce eight claims that the authors make in their paper. The first two claims relate to the properties of graphons estimated from graphs, which are the main components of the method. Claims three to eight relate to the superior performance of the method compared to other augmentation strategies.

Methodology — To reproduce the results, we use the open-source implementation of the method provided by the authors, with a few modifications. We write from scratch all the experiments and pipelines needed to defend the claims of the paper. Additionally, we implement three out of four baseline augmentation methods that are compared to the novel method. For one part of the experiments, we use a local computer and run the experiments on a CPU, with a total of 31.7 CPU hours, while for other more demanding experiments, we use a GPU-accelerated machine for a total of 157.3 GPU hours.

Results — Due to many missing implementation details, we were not able to reproduce all of the original results. Some claims can be supported by our results, but most results are very vague. Even though the new method outperforms the baselines in certain scenarios, we find that the superiority of the method is not as strong as presented in the original paper.

What was easy — The novel augmentation method and its theoretical justification are presented intuitively in the paper, and it was easy to grasp the main ideas of the paper.

What was difficult — While the code with the method implementation was given, the reproduction of all the results required much more code and details than what was provided in the paper. This means that we had to make a lot of educated guesses about the experimental settings, choices of hyperparameters, and details about the models used.

Communication with original authors — We contacted the authors on two occasions. We first inquired about the details of the graph estimation methods that they used and which weren't explained in the paper, to which they responded swiftly. On the second occasion, we asked about the experimental settings and hyperparameter details, but we did not receive a reply.

Copyright © 2023 E. Omeragić and V. Đuranović, released under a Creative Commons Attribution 4.0 International license.
Correspondence should be addressed to Ermin Omeragić (eo3031@student.uni-lj.si)
The authors have declared that no competing interests exist.
Code is available at https://github.com/eomeragic1/g-mixup-reproducibility. – SWH swh:1:dir:b3da6c4106727ef884219a1d04275a3306672c34.
Open peer review is available at https://openreview.net/forum?id=XxUlomN-ndH.

1 Introduction

Data augmentation, a process of artificially increasing the amount of data by generating new data points from existing data, has been shown to improve the generalization and robustness of neural networks. It is also present in graph analysis, where synthetic graphs are generated to create more training data for improving the generalization of graph classification models. However, all the prior methods operate on a within-graph level by altering the structure of an individual graph and do not capture between-graph information. In image recognition and natural language processing, Mixup has been shown to improve performance by linearly interpolating continuous values of random samples to generate more synthetic training data. This paper aims to adapt this method to graph data.
A few problems occur when adapting Mixup to graph data: graph data is irregular and not well-aligned, and the graph topology between classes is divergent. The main assumption for $\mathcal{G}$-Mixup is that the graphs of one class are produced under the same generator called *graphon*. A graphon [1] can be regarded as a probability matrix W, where W(i,j) represents the probability of an edge between node i and j. Graphons are regular, well-aligned, and defined in Euclidean space, so they can be easily mixed to generate new synthetic graphs from them. The main idea of $\mathcal{G}$-Mixup is to estimate graphons for each of the classes in the training set, and generate new, unseen examples by mixing the graphons of the random two classes and generating graphs from the mixed graphon.

2 Scope of reproducibility

This report investigates the reproducibility of the original paper by Han et al. [2] and aims to verify its main claims. The main claims, highlighted in the original paper, can be summarized as follows:

- Claims relating to graphons
  - Claim 1: The graphons of a different class of graphs in one dataset are distinctly different
  - Claim 2: The synthetic graphs are indeed the mixture of the original graphs
- Claims relating to $\mathcal{G}$-Mixup
  - Claim 3: $\mathcal{G}$-Mixup can improve the performance of graph neural networks on various datasets
  - Claim 4: $\mathcal{G}$-Mixup can improve the generalization of graph neural networks
  - Claim 5: $\mathcal{G}$-Mixup could stabilize the model training
  - Claim 6: $\mathcal{G}$-Mixup improves the robustness of graph neural networks
  - Claim 7: Using the average node number of all the original graphs is a better choice for hyperparameter K in $\mathcal{G}$-Mixup
  - Claim 8: $\mathcal{G}$-Mixup improves the performance of graph neural networks with varying layers

3 Methodology

3.1 Model descriptions

The core component of $\mathcal{G}$-Mixup is the graph estimation method. The authors reported in the paper that they tried several methods, including Stochastic Block Approximation

(SBA) [1], Largest gap (LG) [3], Smoothing-and-sorting (SAS) [4], Matrix Completion (MC) [5] and Universal Singular Value Thresholding (USVT) [6]. The authors state that they used the LG method in their experiments. However, in their codebase, we only found the implementation of USVT. In our e-mail correspondence, the authors confirmed that they used the LG method and provided us with the repository where we could find the method implementation.
The authors test the method with a few different GNN backbones that can be separated into two categories. The first category consists of Graph Convolutional Network (GCN) [7] and Graph Isomorphism Network (GIN) [8]. Even though the authors use the name GCN throughout the paper, they added a hyperlink to a PyTorch Geometric (PyG) [9] example in which GCNII [10], an extension of the vanilla GCN model, is used. We assumed that the authors copied the model implementation from this hyperlink, so we used the GCNII model. The GCNII model uses 4 GCNII layers with ReLU activations followed by global mean pooling and two linear layers. The GIN model uses 5 GIN convolutional layers with ReLU activations and batch normalization followed by two linear layers. GIN was the only model whose implementation can be found in the authors' GitHub repository. In the paper, they added a hyperlink to another GIN model implementation in PyG. Comparing these two implementations, there is only a slight difference in how batch normalization is used. We decided to use the model implementation that we found in the authors' GitHub repository.
The second category of GNN backbones consists of graph pooling methods. Specifically, the authors use TopK Pooling (TopKPool) [11], Differentiable Pooling (DiffPool) [12] and MinCut Pooling (MinCutPool) [13] backbones. The authors again provided a hyperlink to PyG examples that implement these methods, so we assumed again that they used this code for their implementation. For TopKPool, 3 Graph convolution layers and 3 TopK pooling layers were used, followed by 3 Linear layers. DiffPool and MincutPool are more complex models, and the details about their implementation can be found in our GitHub repository.
The authors also wanted to compare their method to other augmentation methods, so they included the results of several baseline methods: DropNode [14], DropEdge [14], Subgraph [15] and M-Mixup [16]. In each epoch, DropNode and DropEdge pick graphs from the dataset with probability p_{pick} and then augment the graphs by removing nodes or edges from the graph with probability p_{drop}. Subgraph also picks graphs from the dataset in each epoch with probability p_{pick}, and for each graph, it randomly selects a node from the graph and takes an induced subgraph from the neighbourhood of that node. The size of the neighbourhood and the resulting graph is controlled by the parameter p_{drop}, which tells us how many nodes are going to be left out in the end from the induced subgraph. M-Mixup takes two random graphs from the dataset, creates embeddings for each of them, and then interpolates those embeddings in the embedding space. The authors did not provide implementations of these baseline methods, nor did they mention from where they obtained the results. We decided to implement DropNode, DropEdge and Subgraph ourselves, while we weren't able to implement M-Mixup because of the lack of details in the original M-Mixup paper and the deficit of any public repository that implements it.

3.2 Datasets

The authors test their method on several datasets that can be directly imported from the PyG TUDataset [17] repository: IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K, and REDDIT-MULTI-12K. We provide details about the datasets below.

IMDB-BINARY — This is a movie collaboration dataset that consists of 1000 ego networks of different actors/actresses who played roles in movies in IMDB. The nodes in the graph represent actors/actresses, and the link is formed if they performed in the same movie.

The graphs and their labels are derived from Action and Romance genres (binary classification). The average number of nodes in the graphs is 19, and the average number of edges is 93. The nodes and edges in this dataset do not have any features associated with them.

IMDB-MULTI — This dataset is very similar to the IMDB-BINARY dataset. It consists of 1500 graphs, and the labels are derived from 3 genres: Sci-Fi, Romance and Action. The average number of nodes is 13, and the average number of edges is 65. This dataset does not have any node or edge features either.

REDDIT-BINARY — The graphs in this dataset correspond to online discussions on Reddit. Nodes in the graph represent users, and edges are formed between nodes if one of the users responded to another's comment. There are two graph labels. One of them corresponds to a question/answer-based community (r/iAmA and r/AskReddit) and the other to a discussion-based community (r/TrollXChromosomes and r/atheism). There are a total of 2000 graphs in the dataset, with an average of 429 nodes and 497 edges.

REDDIT-MULTI-5K and REDDIT-MULTI-12K — Similar to REDDIT-BINARY, these are balanced datasets from five and eleven different subreddits, respectively. REDDIT-MULTI-5K consists of discussion threads from r/worldnews, r/videos, r/AdviceAnimals, r/aww and r/mildlyinteresting, where each graph in the dataset is labelled with their corresponding subreddit (5-class classification). REDDIT-MULTI-12K consists of 11 different subreddits, namely, r/AskReddit, r/AdviceAnimals, r/atheism, r/aww, r/IAmA, r/mildlyinteresting, r/Showerthoughts, r/videos, r/todayilearned, r/worldnews, r/TrollXChromosomes. The task for both datasets is to predict which subreddit a given discussion graph belongs to. REDDIT-MULTI-5K consists of 5000 graphs with an average node count of 508 and an average edge count of 594, while the REDDIT-MULTI-12K consists of 11929 graphs with an average number of nodes of 391 and an average number of edges of 456.
Since none of the data sets has any node or edge features associated with it, the authors preprocessed every dataset by adding custom node features to each graph. Namely, if the max node degree in the dataset is lower than 2000, the node feature matrix will be one hot encoded degree of each node (the feature vector of one node will be all 0's except in the column that corresponds to that node's degree, where it will be 1). If the max node degree is higher than 2000, each node will only have 1 feature, whose value will be the normalised degree of that node. Normalization is carried out by calculating the mean node degree and its standard deviation across the whole dataset.

3.3 Hyperparameters

The authors report that they've used the same hyperparameters across all experiments to ensure a fair comparison, and we copied their hyperparameter setup. We use the Adam optimizer, with an initial learning rate of 0.01, which is set to decrease by half every 100 epochs. The authors do not mention how many epochs of training they performed, so we chose a number based on the graphs that were in the report that showed 300 epochs. We split the dataset into train/validation/test sets with 7:1:2 proportions. The best test epoch is selected based on the accuracy of the model on the validation set, and accuracy is reported on ten runs. For $\mathcal{G}$-Mixup, we generate 20% more training graphs and use $\lambda \in [0.1, 0.2]$ to mix up the graphons. The authors set the batch size to 128, and we follow that batch size for IMDB datasets, but due to lack of GPU VRAM, we lower it to 32 for REDDIT datasets.
The authors fail to mention what hyperparameters they use for the baseline methods, so we decided to pick them based on our intuition. We choose to augment 20% of the graphs in the dataset in each epoch for DropEdge and DropNode, and 10% for Subgraph.

In each graph that was chosen for augmentation, we eliminate 10% of the edges/nodes for DropEdge/DropNode and 15% of the nodes for Subgraph.
The authors do not report any hyperparameter search being performed in experiments 3 to 6, where $\mathcal{G}$-Mixup was compared to other baseline models. They do, however, test how the method behaves while varying the hyperparameter K (Experiment 7), which controls how many nodes will be generated in the synthetic graphs. In the Appendix, they also test how the performance of $\mathcal{G}$-Mixup changes by varying the mixup parameter λ, but they do not draw any clear conclusions.

3.4 Experimental setup and code

The code with the implementation of the $\mathcal{G}$-Mixup method and graphon estimation is available on the author's GitHub repository[1]. However, in order to test the claims from the paper, we had to write code for each experiment ourselves. Our project is also available as a GitHub repository. Code for each of the experiments is available as a regular python script inside the source folder (example: src/experiment1.py), and as self-explanatory Jupyter notebooks.
For experiments 1 and 2, due to the nature of the claims we do not use any specific measure. We report on the outcome of the experiments and we make some observations about the way experiments were set up in order to make a final statement about their claims. For experiments 3-8 we use classification accuracy to compare results with those from the original paper.

3.5 Computational requirements

We run the experiments on 3 different pieces of hardware. For "lightweight" experiments, we use an Intel i5-10600K CPU. For most of the experiments, we use NVIDIA RTX 2070 SUPER GPU with 6GB of VRAM. For some parts of Experiment 3, this amount of VRAM was not enough, so we used Google Colab Pro+ GPUs and the GPUs on the National Supercomputing Network (both of which have 32GB VRAM). The details about execution times can be viewed in Tables 3 and 4.

4 Results

Our results provide partial support for claims made by authors. While the results we got from experiments gave us a confirmation that **the synthetic graphs are indeed the mixture of the original graphs**, some other claims are highly debatable. For claims relating to the superiority of $\mathcal{G}$-Mixup over other data augmentation methods we were not able to reproduce results from the paper.

4.1 Results reproducing original paper

Result 1: The graphons of a different class of graphs in one dataset are distinctly different — We believe this claim is an overgeneralization. The authors provided a visualization of estimated graphons for datasets used in the paper, which can be seen in Figure 1. While the claim may be true for datasets **IMDB-BINARY** and **REDDIT-BINARY**, the same can not be said for **IMDB-MULTI** dataset. From the plots of estimated graphons, it is very difficult to find differences between classes 1 and 2 for example. The problem with this claim is that we can have a dataset in which classes are not that much different from each other, so the claim doesn't apply.

[1] https://github.com/ahxt/g-mixup

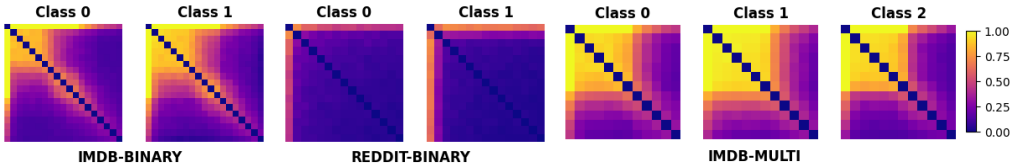

Figure 1. Estimated graphons from experiment 1. Note that we needed to manually update diagonal entries to zeros - something the original authors also did and did not report.

Additionally, the authors wrote that they used the average number of nodes in a dataset for graphon estimation, but for **REDDIT-BINARY** and **IMDB-MULTI** they used different sizes of graphons for visualization. This was done probably to highlight the differences between classes – with a bigger size of the graphon the size of the cell representing each node would get smaller and the plot would be blurry. Nevertheless, this modification is not mentioned by the authors in the paper. To conclude, the experiment is reproducible, but the claim doesn't have strong foundations since graphons are highly dependent on the nature of the dataset.

Result 2: The synthetic graphs are indeed the mixture of the original graphs — This is true. In addition to what the authors did, we confirmed this claim by performing a modified version of the experiment. From **REDDIT-BINARY** dataset we estimated graphons for **Class 0** and **Class 1** respectively. After that, we performed a G-mixup with extracted graphons to get a mixed graphon. Then, we generated 500 synthetic graphs using mixed graphon. Finally, we estimated the graphon from synthetic graphs. From Figure 2 it is easy to observe that the graphon estimated from synthetic graphs is indeed a mixture of two original classes.

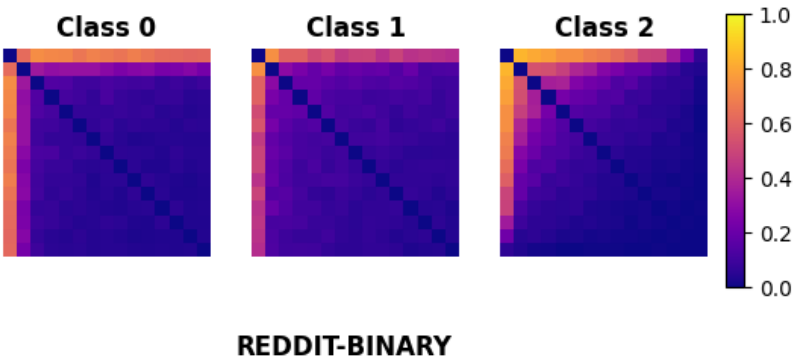

Figure 2. Class 0 and Class 1 are classes from the original dataset. Class 0 has two high-degree nodes, while Class 1 has only one high-degree node. Graphon for Class 2, which was created in a process of $\mathcal{G}$-Mixup of the previous two classes, actually has 1 high-degree node and a dense subgraph.

4.2 Results that differ from the original paper

Result 3: $\mathcal{G}$-Mixup can improve the performance of graph neural networks on various datasets — – We perform the same experiments and create the same table (without one of the baseline augmentation methods) as the authors did in the original paper. However, while authors show that $\mathcal{G}$-Mixup gains 9 best performances among 10 reported accuracies, our experiments show that it performs the best only on 3 occasions out of 10, as shown

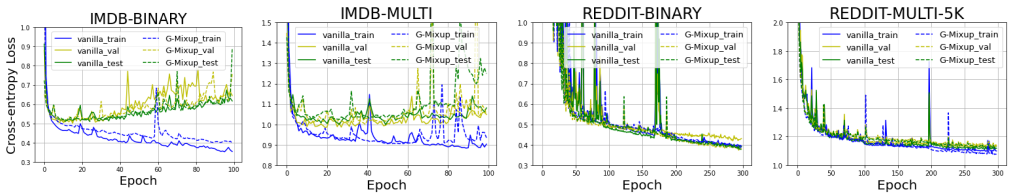

Figure 3. The training/validation/test curves on IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY and REDDIT-MULTI-5K with GCN as the backbone. The curves are depicted on ten runs.

in Table 1. We use the same methodology as the authors, i.e. different augmentation methods adopt the same GNN architecture and the same training hyperparameters.

Model	Methods	IMDB-B	IMDB-M	REDD-B	REDD-M5	REDD-M12
GCN	vanilla	71.40±2.22	**49.86±2.60**	83.75±5.42	**53.15±1.63**	45.90±1.47
	w/ DropEdge	71.70±2.42	48.26±2.90	72.10±2.19	47.15±1.36	36.40±1.57
	w/ DropNode	**72.05±3.36**	49.33±2.27	72.95±2.11	43.51±2.96	31.84±1.39
	w/ Subgraph	72.00±3.95	48.96±2.71	73.61±1.87	48.80±1.22	37.82±0.81
	w/ G-Mixup	71.90±2.92	49.66±2.56	**84.80±4.10**	51.39±2.10	**46.01±1.29**
GIN	vanilla	**72.00±2.98**	47.66±3.07	**91.47±1.70**	**55.81±1.39**	**49.34±0.75**
	w/ DropEdge	71.20±3.16	45.63±2.19	85.37±2.89	49.03±1.63	41.18±1.57
	w/ DropNode	70.30±3.63	46.20±2.44	81.17±2.73	46.85±1.57	38.93±1.14
	w/ Subgraph	69.80±3.32	44.76±2.67	88.80±1.52	49.83±1.86	41.67±1.91
	w/ G-Mixup	70.45±3.94	**49.53±2.59**	90.45±1.54	55.59±1.45	48.81±1.17

Table 1. Performance comparisons of $\mathcal{G}$-Mixup with different GNNs on different datasets. The metric is the classification accuracy along with standard error. Bolded are the augmentations that received the highest mean accuracy on ten runs for that particular GNN architecture, similar to what was done in the original paper.

Model	Methods	IMDB-B	IMDB-M	REDD-B	REDD-M5
TopKPool	vanilla	70.65±3.19	**49.33±2.57**	88.60±2.05	**49.93±1.41**
	w/ DropEdge	68.70±4.17	47.66±2.33	76.55±5.53	43.65±2.97
	w/ DropNode	**71.60±1.76**	46.36±3.15	75.22±4.42	39.05±3.58
	w/ Subgraph	69.40±4.73	45.53±4.96	78.93±4.54	45.24±4.29
	w/ G-Mixup	70.80±2.74	49.00±1.80	**88.82±2.25**	49.86±1.79
DiffPool	vanilla	**72.15±2.04**	**49.10±1.89**	91.90±2.11	**55.71±0.70**
	w/ DropEdge	71.90±2.54	49.06±1.74	89.50±1.62	51.90±3.53
	w/ DropNode	71.05±2.25	47.66±4.42	84.67±3.10	47.25±1.76
	w/ Subgraph	70.15±3.54	47.76±1.30	90.15±1.28	51.11±2.49
	w/ G-Mixup	70.95±3.15	49.03±2.20	**92.22±0.97**	54.24±1.72
MincutPool	vanilla	**71.55±2.67**	48.63±2.58	**84.87±4.02**	48.57±4.23
	w/ DropEdge	71.50±3.69	49.10±2.57	81.06±2.48	44.70±1.35
	w/ DropNode	71.50±2.92	**49.83±3.15**	75.68±1.95	42.49±2.18
	w/ Subgraph	71.05±3.02	48.43±2.82	76.55±3.34	40.33±3.11
	w/ G-Mixup	71.25±2.05	48.93±1.59	84.37±2.65	**48.60±1.88**

Table 2. Performance comparisons of $\mathcal{G}$-Mixup with different Pooling methods. The metric is classification accuracy along with standard error. Bolded are the augmentations that received the highest mean accuracy on ten runs for that particular Pooling method, similar to what was done in the original paper.

Result 4: $\mathcal{G}$-Mixup can improve the generalization of graph neural networks — The authors look at the loss curves of GCN training and state that the test curves with $\mathcal{G}$-Mixup are consistently lower than the vanilla model. The loss curves of our experiments are shown in Figure 3. We do not observe any notable differences between the two models. Furthermore, the test losses of the vanilla model may even be on average lower than the $\mathcal{G}$-Mixup model.

Dataset	Methods	10%	20%	30%	40%	Avg. runtime
IMDB-B	vanilla	**72.05±4.16**	67.65±4.21	66.40±3.60	62.65±5.28	15.44 [s]
	w/ DropEdge	70.00±3.77	68.00±4.52	67.25±4.42	59.75±4.11	36.38 [s]
	w/ G-Mixup	71.05±0.45	**69.50±4.65**	**68.60±4.35**	**65.70±4.28**	17.45 [s]
REDD-B	vanilla	**81.35±5.55**	**77.10±5.31**	74.56±3.86	72.07+-3.94	75.24 [s]
	w/ DropEdge	79.37±6.63	76.02±4.95	**76.97±5.14**	71.67±4.72	118.87 [s]
	w/ G-Mixup	78.65±3.80	76.85±2.94	73.85±3.71	**72.25±2.69**	99.83 [s]

Table 3. Label Corruption Robustness results. We invert the label of a certain percentage of nodes and observe the results. The metric is the classification accuracy along with standard error.

Type	Methods	10%	20%	30%	40%	Avg. runtime
Removing edges	vanilla	82.15±4.50	78.95±4.55	77.50±6.07	75.02±0.35	65.83 [s]
	w/ DropEdge	81.60±5.21	80.40±5.44	77.75±4.68	75.40±3.67	111.60 [s]
	w/ G-Mixup	**82.25±4.55**	**81.35±3.56**	**78.85±0.45**	**76.35±0.50**	88.98 [s]
Adding edges	vanilla	82.37±5.32	82.05±5.09	81.20±6.24	**82.00±4.81**	74.54 [s]
	w/ DropEdge	79.75±6.93	81.02±6.82	80.50±4.93	79.15±3.80	121.82 [s]
	w/ G-Mixup	**85.82±3.64**	**82.25±6.59**	**82.07±4.83**	81.12±4.14	98.48 [s]

Table 4. Topology Corruption Robustness results. We add/remove a certain percentage of edges and observe the results. The metric is the classification accuracy along with standard error.

Result 5 - $\mathcal{G}$-Mixup could stabilize the model training — We do not notice any notable and consistent decrease in the average standard deviation of the classification accuracy in Table 1 between the vanilla and $\mathcal{G}$-Mixup approach. We also do not notice that the loss curves of the $\mathcal{G}$-Mixup approach are smoother than the vanilla method, hence we can not support the authors' claim.

4.3 Results partly supporting the authors' claims

Result 6: $\mathcal{G}$-Mixup improves the robustness of graph neural networks — The authors investigate two kinds of robustness experiments, *Label Corruption Robustness* and *Topology Corruption Robustness*. We perform the same experiments and report the results in Tables 3 and 4. We can see that $\mathcal{G}$-Mixup does well when the topology changes, achieving the highest accuracy in 7 out of 8 runs, which is in line with the authors' results. The results for Label Corruption are not that impressive, as it beats the Vanilla model in only 4 out of 8 runs. For this experiment, we also add the average execution runtime needed to augment the dataset and train it for 300 epochs.

Result 7: Using the average node number of all the original graphs is a better choice for hyperparameter K in $\mathcal{G}$-Mixup — While we get quite similar graphs in our experiments as the authors, we would not say that the graphs support the claim. The graphs are shown in Figure 4. The difference between classification accuracy between the case where the average number of nodes of original graphs is used for hyperparameter K and other cases on the IMDB-BINARY dataset is negligible, while on REDDIT-BINARY it can even be seen that it is one of the worse performing ones.

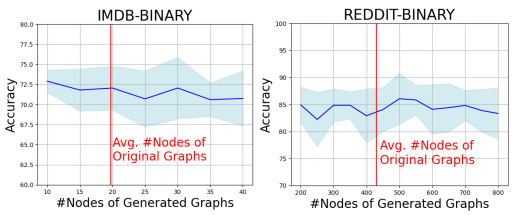

Figure 4. Influence of hyperparameter K to test set classification accuracy

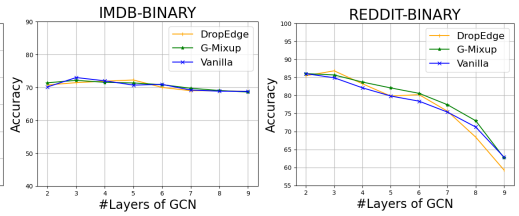

Figure 5. Performance of $\mathcal{G}$-Mixup with GNNs with varying number of layers

Result 8: $\mathcal{G}$-Mixup improves the performance of graph neural networks with varying layers — Our graphs for this claim are shown in figure 5 and are quite different from the ones the authors show in their paper. We do not see any drop in performance when increasing the number of layers on IMDB-BINARY, and all three approaches (Vanilla, DropEdge and $\mathcal{G}$-Mixup) work equally well. On REDDIT-BINARY, we can see that drop in performance, and we can also see that $\mathcal{G}$-Mixup outperforms the other methods by a small margin.

5 Discussion

Our goal was to reproduce the experiments from the paper *$\mathcal{G}$-Mixup: Graph Data Augmentation for Graph Classification* and test the claims that the authors pointed out. While we were able to perform all the experiments from the paper, it did require a lot of additional effort to write the necessary code. We were not able to reproduce some of the results that support the claims from the original paper, as our results in some experiments notably differ from the authors' findings. Regarding the method itself, we were able to reproduce and confirm that synthetic graphs generated with $\mathcal{G}$-Mixup will preserve the key characteristics of both original classes. In contrast to that, we were not able to prove the superiority of this data augmentation method, made in claims 3, 4, and 5. All of the $\mathcal{G}$-Mixup results lie within the standard error of the Vanilla models, which is in our opinion not enough to claim the supremacy of the method. One of the reasons why we didn't get the same results as the authors did may lie in the fact that we didn't have the correct information about the models and hyperparameters used in the experiments, both for $\mathcal{G}$-Mixup and for other augmentation methods. The authors also might have done some data preparation and processing that we were not aware of. We showed, however, that $\mathcal{G}$-Mixup, on average, can improve the robustness of the GNN models, which is useful when the labels or/and the topology of the graphs are noisy.

5.1 What was easy

The paper presented the novel augmentation method precisely, and the authors provided the necessary code for the method implementation. It was also easy to grasp the main ideas of the paper. It was very simple to verify or disprove a certain claim because of the way the experiments were set up.

5.2 What was difficult

The biggest challenge for us was to recreate experiments as they were described in the paper. We needed to write the code for each experiment ourselves, and on top of that, we had to make a lot of educated guesses about the experimental settings and choices of hyperparameters. Additionally, some experiments required vast processing power and a lot of working memory, so we had to perform those experiments on the Google Colab Pro platform and rented virtual machines.

5.3 Communication with original authors

To reiterate, we contacted the authors on two occasions. The first time was very early in the reproduction process when we asked about the details of the graph estimation methods that they used and which weren't explained in the paper, to which they responded swiftly. On the second occasion, we asked about the experimental settings and hyperparameter details, but we did not receive a reply.

References

1. E. M. Airoldi, T. B. Costa, and S. H. Chan. "Stochastic blockmodel approximation of a graphon: Theory and consistent estimation." In: **Advances in Neural Information Processing Systems** 26 (2013).
2. X. Han, Z. Jiang, N. Liu, and X. Hu. "G-Mixup: Graph Data Augmentation for Graph Classification." In: **arXiv preprint arXiv:2202.07179** (2022).
3. A. Channarond, J.-J. Daudin, and S. Robin. "Classification and estimation in the stochastic blockmodel based on the empirical degrees." In: **Electronic Journal of Statistics** 6 (2012), pp. 2574–2601.
4. S. Chan and E. Airoldi. "A consistent histogram estimator for exchangeable graph models." In: **International Conference on Machine Learning**. PMLR. 2014, pp. 208–216.
5. R. H. Keshavan, A. Montanari, and S. Oh. "Matrix completion from a few entries." In: **IEEE transactions on information theory** 56.6 (2010), pp. 2980–2998.
6. S. Chatterjee. "Matrix estimation by universal singular value thresholding." In: **The Annals of Statistics** 43.1 (2015), pp. 177–214.
7. T. N. Kipf and M. Welling. "Semi-supervised classification with graph convolutional networks." In: **arXiv preprint arXiv:1609.02907** (2016).
8. K. Xu, W. Hu, J. Leskovec, and S. Jegelka. "How powerful are graph neural networks?" In: **arXiv preprint arXiv:1810.00826** (2018).
9. M. Fey and J. E. Lenssen. "Fast graph representation learning with PyTorch Geometric." In: **arXiv preprint arXiv:1903.02428** (2019).
10. M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. "Simple and deep graph convolutional networks." In: **International Conference on Machine Learning**. PMLR. 2020, pp. 1725–1735.
11. H. Gao and S. Ji. "Graph u-nets." In: **international conference on machine learning**. PMLR. 2019, pp. 2083–2092.
12. Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. "Hierarchical graph representation learning with differentiable pooling." In: **Advances in neural information processing systems** 31 (2018).
13. F. M. Bianchi, D. Grattarola, and C. Alippi. "Spectral clustering with graph neural networks for graph pooling." In: **International Conference on Machine Learning**. PMLR. 2020, pp. 874–883.
14. Y. Rong, W. Huang, T. Xu, and J. Huang. "Dropedge: Towards deep graph convolutional networks on node classification." In: **arXiv preprint arXiv:1907.10903** (2019).
15. Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. "Graph contrastive learning with augmentations." In: **Advances in Neural Information Processing Systems** 33 (2020), pp. 5812–5823.
16. Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi. "Mixup for node and graph classification." In: **Proceedings of the Web Conference 2021**. 2021, pp. 3663–3674.
17. C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. "Tudataset: A collection of benchmark datasets for learning with graphs." In: **arXiv preprint arXiv:2007.08663** (2020).