Boosting GNN's Generalization via Graph Data Augmentation Group 10 - CS249 Project Report (Spring 2022)

Siddhant Patil
siddhantpatil@g.ucla.edu
University of California, Los Angeles (UCLA)

Dipti Ranjan Sahu
diptisahu11@g.ucla.edu
University of California, Los Angeles (UCLA)

Nischal Reddy Chandra
nrchandra@g.ucla.edu
University of California, Los Angeles (UCLA)

Shivam Patel
shivambpatel@g.ucla.edu
University of California, Los Angeles (UCLA)

ABSTRACT

Data augmentation, creating new plausible variations from the original training data, has been proved effective in improving the generalization of machine learning models on images and natural language. However, comparatively little work studies data augmentation for graphs [16] and [5]. Augmentation operations commonly used in vision and language cannot be naturally applied to graphs primarily because of two reasons. Firstly, graph is a more complicated data abstract, i.e. the graph structure is not euclidean while an image is. Secondly, the nodes in graphs are described by some (high-dimensional) characteristics while an image pixel is just scalars. This project aims to explore how to generate reasonable data on graphs that can boost the generalization of graph neural networks. Link to codebase: Github Repo

KEYWORDS

Graph Neural Networks, Data Augmentation

ACM Reference Format:
Siddhant Patil, Dipti Ranjan Sahu, Nischal Reddy Chandra, and Shivam Patel. 2022. Boosting GNN's Generalization via Graph Data Augmentation Group 10 - CS249 Project Report (Spring 2022). In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data is the primary interest when it comes to Deep Learning. Despite having an architecture capable of learning hidden latent representation, without diverse data, the model fails typically either overfitting or underfitting. This can prove fatal especially in domains such as self-driving and healthcare where MRI scans can be misclassified can ML models. The scarcity of data in several sectors motivated researchers to develop efficient methods to generate what is called **synthetic data** based on existing data. These family of techniques have proven to be instrumental especially in the domains of Computer Vision, Natural Language Processing to ensure more training. Similarly data augmentation has also been show to reduce over-fitting especially in Graph Neural Networks.

Graph Neural Networks, being another type of neural network targeted towards understanding and inferring from graph structures. Their ability to learn better representations give them the edge to handle several prediction tasks. Some of the tasks tackled by GNN's include node classification, edge classification and graph classification. However there ability to learn better representations sometime make the model overfit thus resulting in poorer results. Similar to images and text, data augmentation can also be use to reduce overfitting. However the graph structure is not euclidean unlike an image. Additionally the nodes in graphs are described by some (high-dimensional) characteristics while an image pixel is just scalars. As a result, similar techniques cannot be applied to graphs. Motivated by these challenges we explore different approaches to develop better augmentation techniques for graphs. Some of the approaches we explore includes Sampling, Mixup and Edge Manipulation.

2 PROBLEM DEFINITION AND FORMALIZATION

The performance of most ML models, and deep learning models in particular, depends on the quality, quantity and relevance of training data. However, insufficient data is one of the most common challenges in implementing machine learning. Data Augmentation is a technique that can be used to artificially expand the size of a training set by creating modified data from the existing one. It acts as a regularizer and helps reduce overfitting when training a machine learning model. It is closely related to oversampling in data analysis.

Data Augmentation have seen widespread adoption in fields such as computer vision [14] and natural language processing [6]. At the same time, graph neural networks (GNNs) have emerged as a rising approach for datadriven inference on graphs, achieving promising results on tasks such as node classification, link prediction and graph representation learning.

Despite the complementary nature of GNNs and data augmentation, few works present strategies for combining the two. One major obstacle is that, in contrast to other data, where structure is encoded by position, the structure of graphs is encoded by node connectivity, which is irregular. The hand-crafted, structured, data augmentation operations used frequently in CV and NLP therefore

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

cannot be applied. Furthermore, this irregularity does not lend itself to easily defining new augmentation strategies.

Based on the components of interest, the graph augmentation can be divided into four categories.

- Node augmentation: adding new nodes or removing existing nodes
- Edge augmentation: adding new links or removing existing links
- Graph augmentation: adding new graphs or removing existing graphs (only for graph-level tasks such as graph classification)
- Feature augmentation: modifying the node features

Another problem that needs to be addressed is the class imbalance problem. Graph Convolution Networks (GCNs) have shown the state of the art performance in node classification tasks. These networks work on data that have balanced classes. There are many real-world scenarios where the classes are imbalanced. For example, the Twitter - fake account detection data set, wherein the fake accounts are very less compared to genuine accounts. Also, data related to blogs is highly imbalanced wherein there can be more articles on trending topics. We address this class imbalance problem in the project.

3 RELATED WORK

The class imbalance problem has been prevalent in the Machine Learning domain for a long time. Many real-world problems like fine-grained image classification [13] and medical diagnosis [1] suffer from this issue. The classes are generally divided into categories, namely, the majority classes and the minority classes. Classes that have a large number of samples fall in the majority classes and the ones with a fewer number of samples fall in the minority classes. In the literature, there are three types of ways that can be used to handle such a kind of problem. The three types of solutions are algorithm level, data level and hybrid. In the algorithm level approach, the general idea is to penalize the wrong classification of minority classes more as compared to that of majority classes. A cost-sensitive matrix [12, 17] is used to assign different misclassification penalties to different classes. This approach increases the effect of minority classes in training without actually modifying the data.

In the data-level approach, we update the sample size of classes by over-sampling or under-sampling the data samples. A vanilla method of over-sampling is replicating the nodes of minority classes. Though this method sounds like it solves the problem, it does not. As we are not adding new information to the existing data, the model tends to overfit. This problem has been addressed in the SMOTE paper [3]. The problem is addressed by generating new samples by performing interpolation between samples of minority classes. Multiple articles [2, 8] have been published that extend the work of SMOTE. One more approach [10] called the Cluster-based Over-sampling method has been proposed in the literature that clusters samples into different groups and then over-samples different groups separately. Under-sampling nodes of majority classes is not a good method of generating class balance as information is lost during the process.

The combination of the above two approaches is called the hybrid approach. The authors of [9] combine over-sampling with cost-sensitive learning to address the class imbalance. The method proposed in [4] combines boosting and SMOTE.

The major drawback of all the methods described above is that they assume that the samples are independent identically distributed (i.i.d). The methods discussed above cannot be directly applied to graph-structured data. The node features do not take into account the graph structure of the data. Performing interpolation on them to generate synthetic nodes wouldn't be logical. Also, the nodes that are generated synthetically are not connected with the graph. They cannot be used for training on graph-based classifiers like the Graph Neural Network (GNN). To overcome these drawbacks, the methodology of the project focuses on the classification tasks for the graph-structured data.

4 DATASETS

We use the following datasets.

- Citation graphs:
  - Cora: CORA (standing for Coriolis Ocean database ReAnalysis) is a global oceanographic temperature and salinity dataset produced and maintained by the French institute IFREMER. Most of those data are real-time data coming from different types of platforms such as research vessels, profilers, underwater gliders, drifting buoys, moored buoys, sea mammals and ships of opportunity.
  - CiteSeer: CiteSeer is a public search engine and digital library for scientific and academic papers, primarily in the fields of computer and information science.
  - PubMed: PubMed is a free search engine accessing primarily the MEDLINE database of references and abstracts on life sciences and biomedical topics.
- Social networks:
  - BlogCatalog: BlogCatalog is a graph and network repository containing hundreds of real-world networks and benchmark datasets.

The statistics and experimental setup for the four evaluation datasets is given in table 1.

5 METHODS DESCRIPTION

5.1 Class Imbalance:

The synthetic minority nodes are generated using the proposed framework shown in Figure 1. It consists of 4 parts: the feature extractor, synthetic node generator, edge generator and graph neural network (GNN) classifier. Each part of the pipeline is explained below.

- **Feature Extractor:** As explained in the literature review section, the raw node feature space cannot be directly used to generate the synthetic nodes. The raw feature space is very sparse and finding neighbor nodes similar to the actual node using Euclidean distance may give inappropriate results. Thus, a Graph Neural Network is used to build a feature extractor using the node features and edges. The model representation is shown in the leftmost block of Figure 1. h_v

	Cora	CiteSeer	PubMed	BlogCatalog
# Nodes	2,708	3,327	19,717	10,312
# Edges	5,278	4,552	44,338	333,983
# Features	1,433	3,703	500	64
# Classes	7	6	3	38
# Training Nodes	140	120	60	2578
# Validation Nodes	500	500	500	2578
# Test Nodes	1,000	1,000	1,000	5156

Table 1: Statistics and experimental setup for the four evaluation datasets

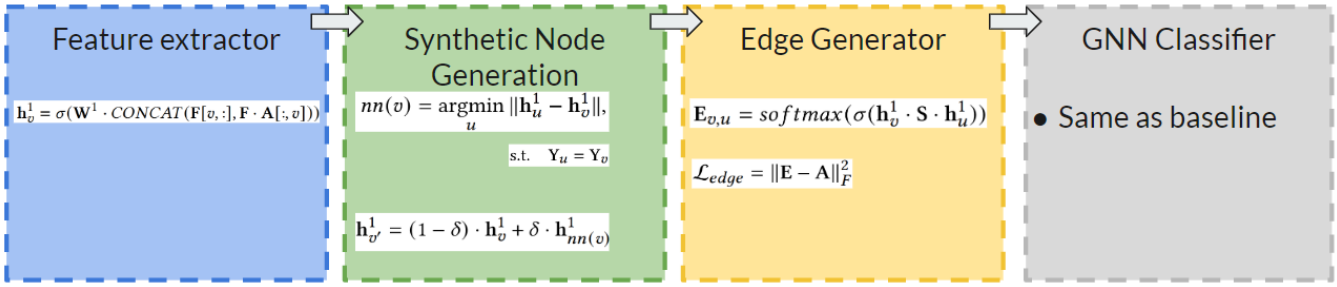

Figure 1: Architecture of solution for class imbalance problem

is the new feature embedding for the node v. F and A represent the input node attribute matrix and adjacency matrix respectively. W is the weight matrix that is trainable and σ is any activation function. The activation function used for the project is ReLU.

- **Synthetic Node Generator:** The feature space generated by the feature extractor is passed on to the synthetic node generator for up-sampling the minority classes. The base framework used for up-sampling is referred from SMOTE [3]. It replaces vanilla up-sampling with interpolation. Given a node of a minority class, a neighboring sample i.e. whose features are similar to the node is found. The distance between nodes is calculated using Euclidean distance on the node features generated by the feature extractor. Using the base node and the neighboring node, a new synthetic node is generated by interpolation.

$$h_{v'}^1 = (1-\delta) \cdot h_v^1 + \delta \cdot h_{nn(v)}^1 \tag{1}$$

h_v^1, $h_{nn(v)}^1$ and $h_{v'}^1$ are the features of the base node, the nearest neighboring node of h_v^1 and new node respectively. δ is the hyperparameter that can be tuned to get better performance.

- **Edge Generator:** The nodes that are generated using the synthetic node generator are isolated. They are not connected with the graph. The edge generator performs the task of connecting the synthetically generated nodes with the existing graph. A vanilla edge generator is trained as described in the third block of Figure 1. The original graph is fed in as input to the model and a matrix S is trained to capture the interaction between nodes. $E_{v,u}$ refers to the predicted relation information between the nodes v and u. The new nodes and edges are added to the initial adjacency matrix A and passed on to the Classifier for training.
- **Graph Neural Network (GNN):** The classifier is the same as the baseline classifier described above. This classifier classifies the nodes based on the updated graph obtained from the edge generator.

5.2 Mix-Up:

Mixup technique focuses on interpolating input sample features as well as their labels to produce synthetic data. This approach helps the model generalize better and learn from rare samples which are often unavailable in the original data. Additionally, incorporating Mixup has proven to be effective in terms of regularization especially for images. It helps smoothen decision boundaries and thus improve the hidden representations arrangement. The domain of Computer Vision and NLP have relied on the utilizing mixup technique several tasks such as classification. Mixup focuses on randomly sampling 2 instance from the same mini-batch and performing an interpolation

$$\tilde{x} = \lambda x_i + (1-\lambda) x_j \tag{2}$$
$$\tilde{y} = \lambda y_i + (1-\lambda) y_j \tag{3}$$

Mixup extends the training distribution by incorporating the prior knowledge that interpolations of features should lead to interpolations of the associated labels [15]. However the traditional Mixup approach assumes that the features dealt with comprise of plain vector format. Since graph data isn't fixed, this methodology cannot be directly applied to graphs and proves to be more challenging

- **Node Classification:** We formulate our task focuses on node multi-class classification problem. GNN layer goes through a *message passing* mechanism wherein the GNN

layer updates a specific node representations based on its neighbors. Using this GNN goes makes a nodes prediction on the last layer called the *receptive field*. To elaborate receptive fields of the two sub-graphs should be mixed for interpolating two nodes. This is done via two steps, mixing interpolating the nodes before input layer (assuming layer l = 0)

$$\tilde{x_{ij}} = \lambda x_i + (1 - \lambda) x_j \tag{4}$$

The next step involves conducting the graph convolutions based on nodes i and j's topologies separately at each layer followed by mixing the aggregated features from the two topologies together before the next layer l+1

$$\tilde{h}^l_{ij} = \lambda \tilde{h}^l_{ij,i} + (1 - \lambda) \tilde{h}^l_{ij,j'} \tag{5}$$

However the nodes are susceptible to interference wherein during a Mixup of node i and j, i can mix with a node k, which is outside the receptive field of both i and j. This adds unwanted noise perturbation leading to poor results.

To allievate this, a two stage-approach is utilized wherein feed-forward to GNN is constructed for each mini-batch graph to obtain latent representations of the nodes without Mixup. Secondly, two nodes are randomly samples from the batch for Mixup of node attributes. Finally, a two-branch Mixup graph convolution is performed for both the nodes.

5.3 Edge Modification

In this section, we discuss another graph augmentation technique by manipulating graph via adding and removing edges over the fixed node set. We tried to mimic the work done in Data Augmentation for Graph Neural Networks [16]. We use an edge predictor function to obtain edge probabilities for all possible and existing edges in $\mathcal{G}$. Using the predicted edge probabilities, we add(remove) new(existing) edges to create a modified graph $\mathcal{G}$, which is used as input for node classification. These edge manipulation technique can encode class-homophily to promote intra-class edges and inter-class edges.

We use Graph Auto-Encoder(GAE) from [11] as the edge predictor module due to its simple architecture and competitive performance. GAE consists of a two layer GCN encoder and an inner-product decoder.

$$M = \sigma(ZZ^T), where\ Z = f(A, X) \tag{6}$$

Z denotes the hidden embeddings learned by the encoder, M is the predicted (symmetric) edge probability matrix produced by the inner-product decoder, and $\sigma(\cdot)$ is an elementwise sigmoid function.

Figure 2 shows the architecture of the GAUG framework that we follow for the experiment. We use a hyper-parameter α to interpolate and sample from the modified adjacency and the original adjacency matrices. The obtained interpolated matrix is normalised and thus used for our downstream application i.e. node classification.

We evaluate the performance of this method on two settings:

- **GAUG-M:** The modified graphs generated using variational GAE are used at inference time to calculate the test accuracy and F1 score.
- **GAUG-O:** GAUG-O uses the original-graph setting, where we cannot benefit from graph manipulation at inference time.

5.4 Super-Nodes

In this section, we will be discussing a novel node addition technique called super-node addition. A "super-node" is defined as a node which is connected to every input node in the graph. [7] incorporate a similar node addition technique, however their latent "master" node has several differences from ours:

(1) It is connected to every node with a special edge type
(2) It has a separate feature dimension
(3) It has separate learnable weights for the internal update function

Our approach is rather simple as in we add a single node and connect it to all the existing nodes using the same type of edge. It is shown in the middle image of Figure 3. This type of augmentation is desirable since it is plug-and-play, and thus requires no changes to the learning algorithm. Adding a super-node allows information to travel long distances during the propagation phase. This is because all inputs nodes become 2-hop neighbours of each other and thus can facilitate better feature learning. This ease of information propagation instantly yields better results.

However, this approach has a significant drawback since it contradicts with the very idea of the graph. The edges lose their significance as every node can communicate with every other node, not just its neighbours. One way to overcome this drawback is to connect only those nodes which need to communicate with each other, but are not directly connected. This leads us to the next subsection in which we explore clustering as a way to decide which nodes need connecting.

5.5 Clustering

First we will discuss some clustering algorithms. Our idea is to use clustering to define similarity between nodes and then insert a single super-node for these similar nodes. This will ensure that every node does not become 2-hop neighbour of every other node, rather only the nodes that are similar to it. Using different types of clustering algorithms will give us different types of similarity between nodes. In particular, we experiment with two types of clustering:

(1) Structure-based Clustering: Graph structure is used to assign nodes to a cluster, and thus nodes in a cluster have similar structure to each other. We use a serial graph-partitioning algorithm called METIS to implement structure-based clustering.
(2) Feature-based Clustering: Closeness between node features will be used for clustering and thus nodes in a cluster will have similar features. In particular, we use the standard clustering technique of K-Means to implement feature-based clustering.

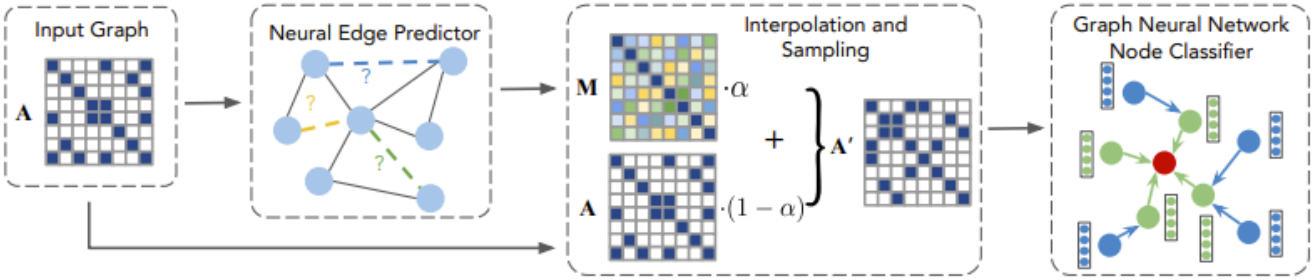

Figure 2: GAUG Framework

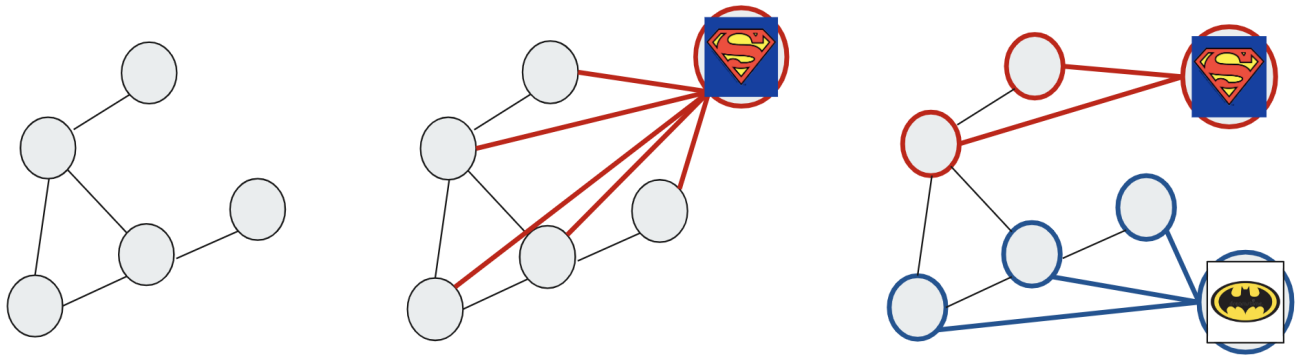

Figure 3: Leftmost image shows the input graph. Middle image shows super-node addition. Rightmost image shows clustering and then super-node addition.

Figure 3's right image shows the data augmentation technique using two clusters. The number of clusters is a hyper-parameter which needs fine-tuning. We use grid-search and find the optimal number of clusters from a search space of k={2,3,5,10,20,30,50,100,200,500}.

6 EVALUATION

Based on the given tasks at hand, it is evident that our primary downstream task is node classification. Hence, we used the following evaluation metrics pertaining to classification primarily.

- Accuracy: As it is a classification task, we have accuracy as our prominent evaluation metric as it is the measure of number of nodes that are predicted correctly
- F1-Score: Since we plan on evaluating the augmentation technique on task like node classification we would compute the accuracy of the model via augmentation followed by computing an F1 score.
- AUC: Area under ROC curve is a widely used metric in evaluating classification problems. It is primarily used in case of imbalanced data.

Loss Function: BCE Loss- We plan on using the Binary Cross Entropy loss function which is the measure of the difference between two probability distributions for a given random variable

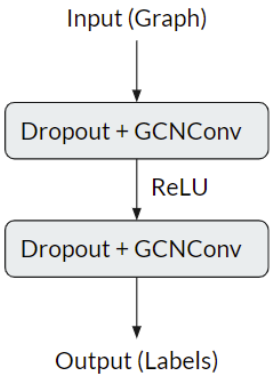

Figure 4: Baseline Architecture

7 EXPERIMENTS AND RESULTS

Basline: We used a traditional GCN architecture for our downstream task which involves two GCNConv layer. This architecture is kept constant among all the methods to check the performance with varying strategies. The architecture of the baseline model is shown in Figure 4.

Class Imbalance: Experiments using the described architecture for handling class imbalance problem are done using the Cora

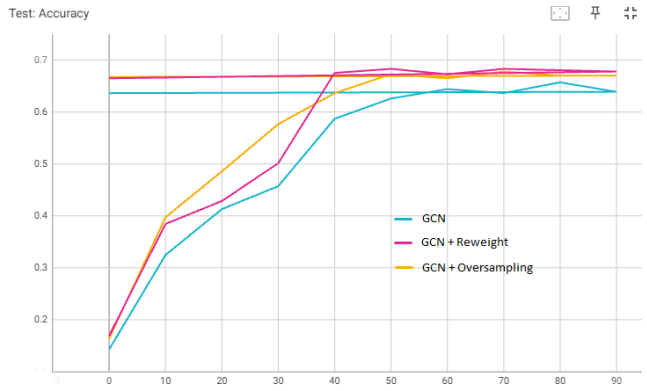

Figure 5: Graph of Test Accuracy vs Iterations for different methods - Cora Dataset

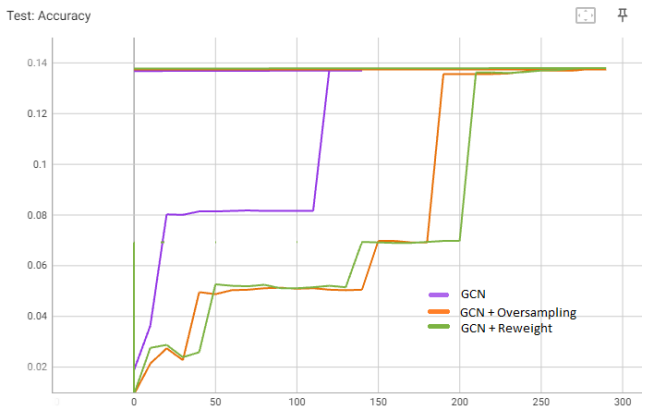

Figure 6: Graph of Test Accuracy vs Iterations for different methods - BlogCatalog Dataset

dataset and BlogCatalog dataset. Cora dataset is a class-balanced dataset. We synthetically imbalance the dataset and use it for analysis. We use 20 samples per class for training. We randomly pick 3 classes and downsample them by a factor of λ. The BlogCatalog dataset is imbalanced and we use it as-is for training. We train 4 different models on each dataset. The results of experimentation are summarized in table **??**. The graph of test accuracy vs iterations for different methods is shown in figure 5 for Cora dataset and 6 for BlogCatalog dataset. GCN is the baseline model. GCN + Oversampling is the model in which we perform vanilla oversampling on the graph data and pass it through the GCN baseline model for classification. GCN + Reweight is the model where we penalize the incorrect classification of minority classes more as compared to majority classes. We do not over-sample or down-sample the data in this model. GCN + GraphSMOTE is the model proposed in Figure 1.

The different hyperparameters used while training have the following values. The learning rate is set to 0.01, weight decay is 5e-4, the number of hidden dimensions in the graph convolution network is 16 and the dropout factor is 0.5. The detailed instructions on running the code can be referred to from the README.md file present in the code repository.

Mixup: We consider Cora, Citeseer and PubMed as a part of our experiment for Mixup. Additionally we ensure that the train-val-test split ratio is in accordance with the traditional experiment-ratio

Models	Cora	CiteSeer
Baseline	0.8170	0.7108
GAUG-M	**0.8343**	**0.7228**
GAUG-O	0.8318	0.7223

Table 2: Edge Modification Experimental Results

Edge Modification: We ran the experiments on Cora and CiteSeer. After all the hyper-parameter tuning, we got an improvement of around 2% in case of Cora and 1% in case of CiteSeer for both GAUG-M and GAUG-O frameworks. The data distribution in this case is same as the traditional format that we have been following. Table 2 summarize the results for edge manipulation experiments

Models/Dataset	Cora	CiteSeer	PubMed
Baseline	0.8170	0.7108	0.7916
Super-Node	0.8264	0.7154	0.7894
K-Means and Super-Node	0.819 (5)	**0.7234** (100)	**0.8016** (3)
METIS and Super-Node	**0.8282** (20)	0.7196 (20)	0.801 (5)

Table 3: Super-Nodes and Clustering Experimental Results. Numbers in the bracket are number of clusters

Super-Nodes and Clustering: Cora, CiteSeer, and PubMed are the three benchmark datasets used to test the effectiveness of our novel technique. To ensure that the newly added nodes are used just to improve message propagation, we exclude those nodes from being part of either train, val or test split. Table 3 summarizes the results for super-nodes and clustering experiments. As we can see, we get roughly 1% increase in accuracy using our novel technique. It's important to note that these values are averaged over 5 random seeds and thus it shows that the jump in accuracy is not because of some random seed, rather because of the augmentation. A few observations include:

- Simply super-node addition increases test accuracy by 0.5% for Cora and CiteSeer datasets.
- K-Means outperforms METIS for CiteSeer and PubMed, whereas METIS is better for Cora dataset.

Mixup and SuperNode/Clustering: We utilize the same 3 datasets used in the Mixup and Clustering based approaches. However, we perform an ensemble of Mixup and cluster based approaches to further understand how well GNN can generalize during the node classification task. 5 shows our experiments based on our ensemble technique. Using Metis and Mixup provides an increase in about 3% accuracy for cora and 1% accuracy for PubMed. Upon using Kmeans clustering algorithm we notice a 5% increase compared to the baseline results.

Model \Data	Cora*			BlogCatalog		
	Accuracy	AUC-ROC	F1 Score	Accuracy	AUC-ROC	F1 Score
GCN	63.64%	0.8858	0.6434	13.68%	0.4973	0.0065
GCN + Oversampling	66.75%	0.9096	0.6774	13.74%	0.5019	0.0077
GCN + Reweight	66.49%	0.9122	0.6731	13.78%	0.5015	0.0076
GCN + GraphSMOTE	**67.68%**	**0.9172**	**0.6804**	**14.48%**	0.5013	**0.0081**

Table 4: Class Imbalance: Experimentation Results

Ensemble Test Accuracy			
Model	Cora	Citeseer	PubMed
Baseline	0.78	0.65	0.76
Mixup	0.799	0.69	0.757
Mixup + Single	0.811	0.661	0.768
Mixup + KMeans	0.808	**0.704**	0.764
Mixup + Metis	**0.818**	0.507	**0.775**

Table 5: Mixup + Clustering: Experimentation Results

8 CONCLUSION

We propose show multiple data augmentation methods for better generalization of graphs. We also show how well our approaches perform when compared to the baseline models. We also propose ensemble methods which successfully show 3%- 5% increase in the test accuracy. Additionally we also observe that GCN+GraphSMOTE works well on imbalanced data while edge manipulation and clustering / mix-up work well for the other datasets.

9 FUTURE WORK

An important future direction for us is to jointly utilize the the various augmentation techniques we explored in this project. Combining the edge manipulation technique with clustering and supernode addition has promise. Mix-up can also incorporated to make it a three-stage augmentation.

In terms of individual techniques, utilizing new clustering techniques (different from the ones explored in the project) for supernodes and clustering is an interesting direction. How these techniques affect different model architectures (e.g. GAT, SplineCNN) is a fascinating question to answer. For the class imbbalance problem, one future work is to implement it for new datasets such as Twitter - fake account detection dataset.

Another interesting direction is using Reinforcement Learning to rewire nodes which are 2-hop neighbors based on the RL update rules. Rewiring nodes which are 2-hop neighbors can preserve the graph properties as the Laplacian matrix won't change much. The paper **??** uses this technique to generate adversarial examples, but we think it can be applied for obtaining more synthetic data as well. As there is lot of similarity between adjacent nodes, we think it will be a good exercise to check it for data augmentation.

10 TASK DISTRIBUTION

All the team members have contributed equally for the project work. Everyone contributed towards literature review, experimentation and comparison of results. The specific distribution of tasks can be referred from table 10.

ACKNOWLEDGMENTS

We thank Song Jiang and Prof. Yizhou Sun for guiding us through the project work.

REFERENCES

[1] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. 2018. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks* 106 (oct 2018), 249–259. https://doi.org/10.1016/j.neunet.2018.07.011

[2] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. 2009. Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for Handling the Class Imbalanced Problem. In *Advances in Knowledge Discovery and Data Mining*, Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 475–482.

[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority over-Sampling Technique. *J. Artif. Int. Res.* 16, 1 (jun 2002), 321–357.

[4] Nitesh V. Chawla, Aleksandar Lazarevic, Lawrence O. Hall, and Kevin W. Bowyer. 2003. SMOTEBoost: Improving Prediction of the Minority Class in Boosting. In *Knowledge Discovery in Databases: PKDD 2003*, Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 107–119.

[5] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. 2022. Data Augmentation for Deep Graph Learning: A Survey. *arXiv preprint arXiv:2202.08235* (2022).

[6] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075* (2021).

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 1263–1272.

[8] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. 2005. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In *Advances in Intelligent Computing*, De-Shuang Huang, Xiao-Ping Zhang, and Guang-Bin Huang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 878–887.

[9] Haibo He and Edwardo A. Garcia. 2009. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering* 21, 9 (2009), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

[10] Taeho Jo and Nathalie Japkowicz. 2004. Class Imbalances versus Small Disjuncts. *SIGKDD Explor. Newsl.* 6, 1 (jun 2004), 40–49. https://doi.org/10.1145/1007730.1007737

[11] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[12] Charles X. Ling and Victor S. Sheng. 2008. Cost-Sensitive Learning and the Class Imbalance Problem.

[13] Yuxin Peng, Xiangteng He, and Junjie Zhao. 2018. Object-Part Attention Model for Fine-Grained Image Classification. *IEEE Transactions on Image Processing* 27,

Sr. No.	Task	People
1	Literature Review	Siddhant, Shivam, Dipti, Nischal
2	Class Imbalance	Siddhant
3	Node mix-up	Nischal
4	Node-wise data augmentation	Shivam
5	Edge-wise data augmentation	Dipti
6	Data collection, evaluation and comparing algorithms	Nischal, Dipti, Shivam, Siddhant
7	Writing Report	Shivam, Siddhant, Nischal, Dipti

Table 6: Task Distribution

3 (2018), 1487–1500. https://doi.org/10.1109/TIP.2017.2774041

[14] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.

[15] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. 2018. mixup: Beyond Empirical Risk Minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=r1Ddp1-Rb

[16] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2020. Data augmentation for graph neural networks. *arXiv preprint arXiv:2006.06830* (2020).

[17] Zhi-Hua Zhou and Xu-Ying Liu. 2006. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge and Data Engineering* 18, 1 (2006), 63–77. https://doi.org/10.1109/TKDE.2006.17