COMP8221 ASSIGNMENT 2

Real-world applications of GNNs

Submitted by:
Rohan Junaid Khan
SID: 47843276

Topic: Semi-supervised Node Classification using Graph Convolutional Network on CORA dataset.

Section 1: Motivation and Explanation of dataset and task.

Graph Neural Networks are useful and powerful tools for solving machine learning problems on graph-based datasets. We are familiar with convolutional neural networks that work well on grid-based datasets. However, unlike grid-based datasets, graphs have different dimensions and properties that must be addressed while working on machine learning problems. Therefore, there are multiple breakthroughs made to find a solution to use the neural network method on graphs. Our task is to classify at the node level for a simple undirected graph using a convolutional network. Research has been done on Graph Convolutional Networks where they motivated their convolutional architecture selection with a localized first-order approximation of spectral graph convolutions. The model grows linearly with the number of network edges and learns hidden layer representations that encode both local graph structure and node attributes (Kipf & Welling, 2017). It is a simple and straightforward design to start with and gather intermediate knowledge on node classification problems using GNN.

For this task, we have chosen the CORA dataset from the torch dataset which is a popular choice for node classification tasks using Graph Convolutional Networks (GCNs). For those new to graph neural networks, Cora provides a simple approach to examine and investigate the influence of graph topology on learning processes without the need for sophisticated feature preprocessing or graph construction from scratch. It is a dataset with different publications with its word vectors as properties and citation information. Cora provides a single graph with a well-understood baseline for evaluating the performance of graph neural networks. Let's observe the different aspects of the graph dataset below.

```
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
Number of graphs: 1
Number of features/node: 1433
Number of classes: 7
Number of nodes: 2708
Number of edges: 10556
Number of training nodes: 140
Number of test nodes: 1000
Number of validation nodes: 500
Training node label rate: 0.05
test node label rate: 0.37
Validation node label rate: 0.18
Is undirected: True
```

Figure 1: CORA graph features

Observing Figure 1, we can say CORA is a single undirected graph having 2708 nodes where each node has 1433 node features. There are 10556 edges connecting those nodes in an undirected way. Each node is classified between 7 classes. The dataset is well-processed with split masks attached for training, testing, and validation. Also, we tuned up the training, validation, and test mask in the code to use the full dataset.

Graph dataset structure:

- **Node:** Each node is a publication of a topic relating to AI, Data Science, and many others closely related to these topics. This means that the graph consists of 2708 publications.
- **Edges:** Edges are citation connections, which means one text cites another. This structure naturally produces a graph, making it ideal for GCNs, which excel at using link/edge information. This graph is undirected and has around 5278 citation links.
  In Figure 2, we can see the output tensor vector of the node connection explaining the

```
tensor([[ 633,    0],
        [1862,    0],
        [2582,    0],
        ...,
        [ 598, 2707],
        [1473, 2707],
        [2706, 2707]])
```

Figure 2: Edge representation

edges. Notably, there can be multiple edges connected with a single node. These are the representations of one single publication cited throughout different other publications: maybe of different labels. In Figure 3, most of the nodes have degrees of 2 to 7 which indicates less complexity of the graph.

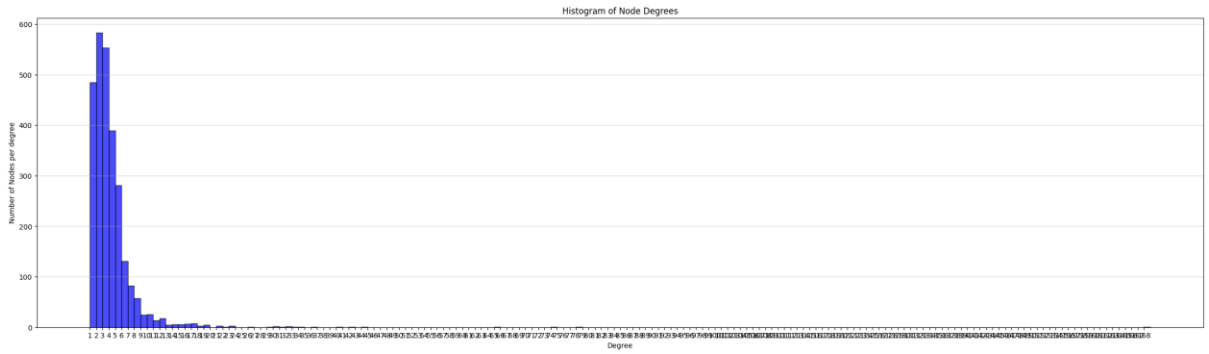

Figure 3: Count of nodes on different levels of degree

- **Labels:** Nodes are divided into seven classes based on their field of publication (e.g., Machine Learning, Genetic Algorithms, etc.). This enables a multiclass classification task. The classes are numerically encoded in the dataset from 0 to 6. It is shown from the code output (Figure 4) below:

```
All the labels in each node: tensor([3, 4, 4,  ..., 3, 3, 3])
Total labels exist: tensor([0, 1, 2, 3, 4, 5, 6])
```

Figure 4: All node classes and unique labels of classes

- **Node Features:** Each node in the Cora dataset represents a scientific article, and the node features are obtained from binary word vectors that indicate the presence or absence of certain terms in the text. These extensive feature sets are appropriate for evaluating GCNs' feature aggregation capabilities. We normalized the node features using the built-in torch function. We can see that in Figure 5 where a portion of transformed node features are illustrated.

```
torch.Size([2708, 1433])
tensor([0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0500, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0500, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000])
```

Figure 5: Normalized node features

The goal of normalizing features is to make the values of node characteristics more consistent across the dataset. **NormalizeFeatures()** changes the feature vectors such that each feature (for each node) has a total sum of one (or extremely close to it, depending on floating-point accuracy). This is commonly accomplished by dividing each feature value by the total of all feature values for the node.

This form of normalization is especially important in graph neural networks since it helps to stabilize the learning process. Scaling the features guarantees that no single

feature with particularly high values dominates the learning process, which can be critical for models like Graph Convolutional Networks (GCNs) where feature aggregation from neighbors is a crucial part.

Cora has around 2,708 nodes and 5,278 linkages, making it large enough to deliver relevant insights and training challenges without requiring expensive processing resources in normal research contexts. This balance makes it accessible to researchers who have low resources. The graph is sparse, which means that each node has a manageable number of connections than the total number of nodes. This is a prevalent occurrence in real-world graphs, posing a genuine challenge to graph neural networks. Using Cora demonstrates how node characteristics (word vectors) and graph structure (citation) may be coupled to increase prediction accuracy, highlighting GCNs' main benefit. Because the dataset is stable in terms of features, labels, and splits, studies on the Cora dataset are repeatable, allowing other researchers to validate and build on previous work.

Graph visualization:

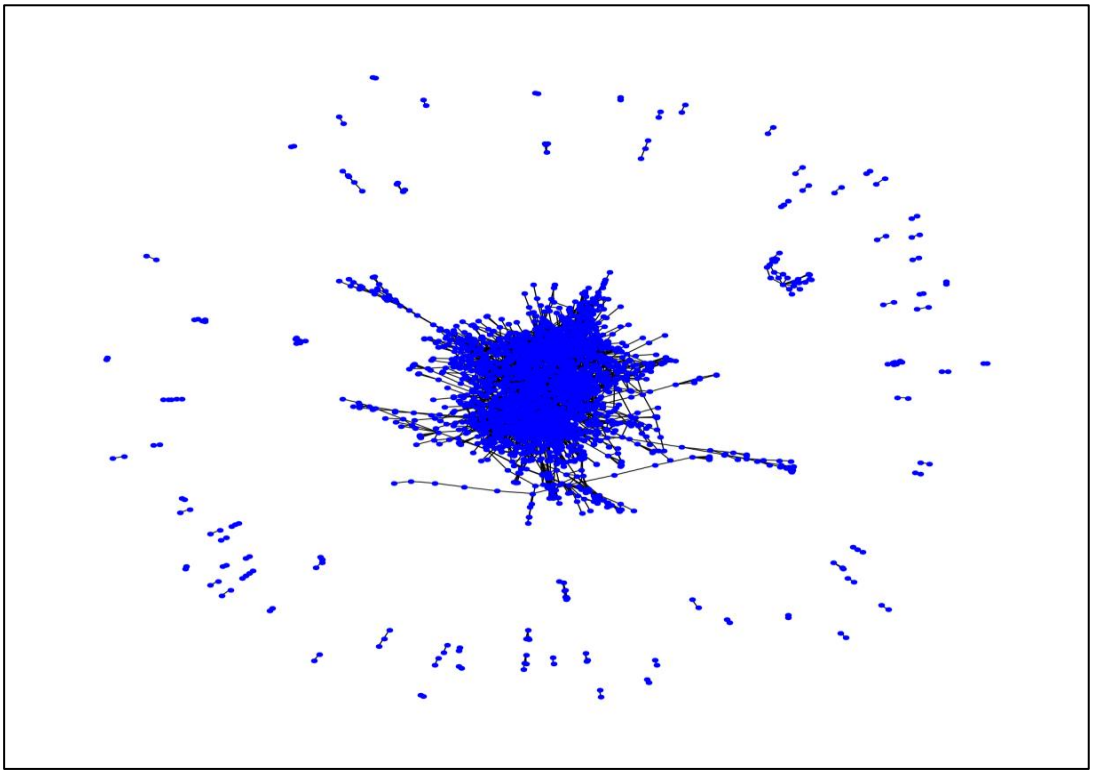

Figure 6: CORA graph visualization

Figure 6 depicts a network constructed from the CORA dataset with a highly linked center. This cluster of strongly coupled nodes most likely reflects a separate community or group within the dataset, having comparable traits or features. The visualization reveals diverse community structures within the network, with the dense core area possibly playing an important role due to its influence or centrality. Furthermore, many nodes are located on the perimeter, with fewer connections, which might imply entities with less direct interconnections in the network. Isolated nodes or tiny clusters separated from the major mass may indicate outliers or smaller sub-communities.

Section 2: Appropriateness and Explanation of the Model

Graph Convolutional Network:

The core mathematical principle underlying Graph Convolutional Networks (GCNs) is to update a node's representation based on its attributes as well as the features of its neighboring nodes into a low-dimensional space in which comparable or closely related nodes are clustered together. This is done in a method that extends the concept of convolution from Euclidean spaces (such as pictures) to graphs (Kipf & Welling, 2017). This is accomplished by a mechanism called message forwarding, or neighborhood aggregation.

- Aggregation: Each node collects information (features) from its neighbors.
- Transformation: The collected information is then transformed using a neural network, often followed by a non-linear activation function.
- Update: Each node's features are updated based on its features and the aggregated information.

Mathematical components of GCN:

- **Adjacency matrix,** A_{ij} **:** This is a square matrix representing the graph. It simply shows if there is any connection between node ***i*** and node ***j***.
- **Degree matrix,** D_i **:** It is a diagonal matrix that shows how many edges are connected to node ***i***.
- **Self-loop representation:** It allows a specific node to consider its features with its neighbor's features. This is done by adding the identity matrix (***I***) with the adjacency matrix and generating a new adjacency matrix and updated degree matrix.
    - **New adjacency matrix,** $\tilde{A} = A + I$, where I is the identity matrix.
    - **Updated Degree matrix,** $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, Here it sums over all nodes ***j***, counting the number of edges between node ***i*** and each other node ***j*** (including itself due to the self-loop).
- **Normalized Graph Laplacian:** This normalization aids in smoothing the features throughout the graph, preventing the features from becoming too big. For the GCN model, it is given by,

    $$L_{norm} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$$

- Thus, the propagation rule for a GCN layer can be described as,

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

    Where *l* is the recent layer and,
    - H^l is the matrix representation of the node features.
    - W^l is the weight matrix.
    - σ is the non-linear activation function (in this case, ReLu)

Operation of GCN:

From the propagation rule of GCN, the operation can be broken down into three sections,

1. **Feature Aggregation:** $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{l}$, Here, each node combines characteristics from itself and its neighbors, weighted by their degree. This step guarantees that the feature aggregation is normalized and does not disproportionately benefit nodes with high degrees.
2. **Feature Transformation:** $H^{l}W^{l}$, Then the aggregated features are then modified using a weight matrix, like the filters found in classic CNNs.
3. **Non-linear Activation:** $\sigma(\square)$, Lastly, a non-linear activation function is used to add nonlinearities to the model, allowing it to learn more complicated patterns.

Proposed GCN Architecture:

We have chosen GCN Architecture of four layers. The first layer size is 64, the second layer size is 32, the third layer size is 16, and the last layer is just one with softmax activation as it is a classification problem, and we need to generate probabilities of different labels respective to the nodes. Figure 7 illustrates the architecture of our GCN model,

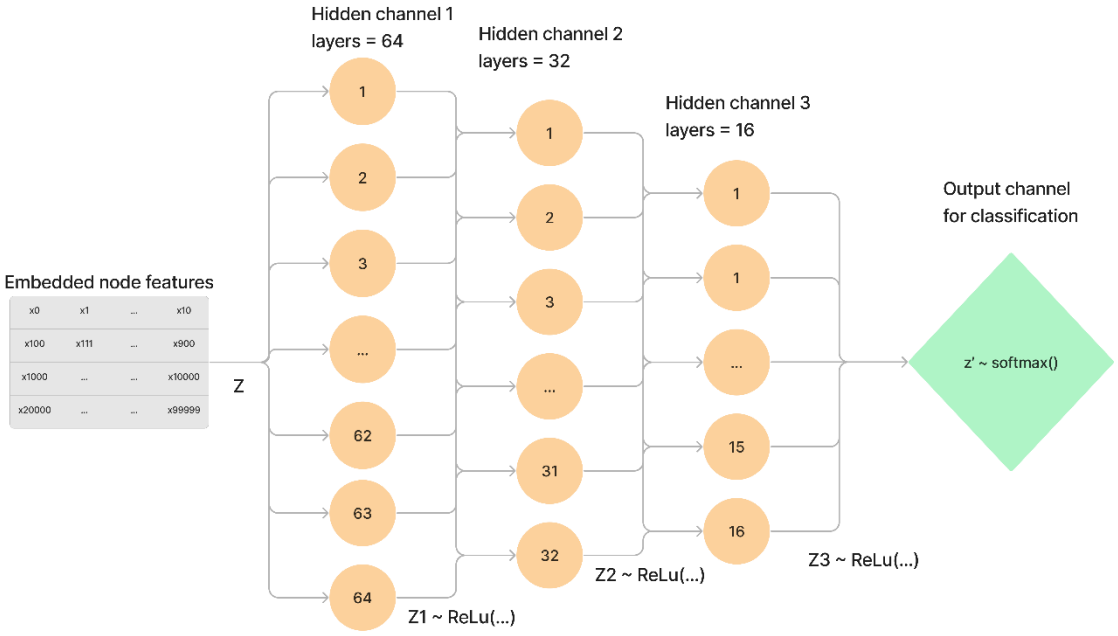

Figure 7: Proposed GCN architecture with 4 layers.

Choosing 3 hidden layers for the task and dataset at hand might be appropriate. As node features propagate through these hidden layers with different dimensions, it is crucial to have the right number of layers set for an efficient and accurate result. By selecting the appropriate number of layers, the danger of nodes losing their traits due to over-smoothing is reduced. This careful balancing guarantees that nodes retain enough uniqueness for activities that need fine-grained differences between node attributes while minimizing performance damage. This also ensures

that each node's characteristics are impacted by a sufficiently extensive neighborhood, allowing the capture of broad structural and feature-based contexts without going too far. The chosen number of layers adds parameters to the model at a reasonable level, improving its capabilities without dramatically increasing training time or the danger of overfitting. This enables solid performance without extensive regularisation or a large volume of data.

We will observe an ablation case on this network, where we reduce the layers and its size and evaluate the results in Section 4.

Rationale for Selecting GCN Model:

While there are better models like GAT, SSP, SplineCNN, etc. for node classification tasks, GCN is chosen for its simplicity and scalability. Additionally, for beginners, it is a great model to get introduced to GNNs with ease of understanding the aspects of the model clearly. It employs a simple approach for feature aggregation—taking the normalized sum of the neighbors' features—which reduces computing complexity. This simplicity is especially useful when computing resources are limited or the model needs to be deployed in contexts where computational efficiency is critical. The averaging techniques utilized in GCNs can properly represent nodes' preference to link to other nodes in the network that have comparable characteristics. This makes GCN an excellent match for datasets with edges implying similarity or direct linkages that do not fluctuate much in significance. Historically, GCNs have performed well on the CORA dataset for node classification tasks. The reason for this efficacy is that GCN's layer-wise propagation algorithm makes use of the graph structure and node attributes, which are well-suited to the characteristics of conventional citation networks such as CORA. Also, Other model architectures are complex with additional parameters that can potentially overfit more easily than GCN, especially on smaller datasets or where the number of node features is limited. Since CORA has a fixed and relatively small size, the simpler model (GCN) might generalize better.

Section 3: Results and Insights

In this section, we present the results of our model aimed at classifying the node of the CORA graph dataset into seven class labels. We evaluate the model's performance using loss, accuracy, precision, recall, F1-score, and AUC-ROC, which provide insights into important factors like the overall correctness, the balance between precision and recall, and the model's ability to distinguish between classes. First, we will evaluate the loss and accuracy during the training and validation phase, which is visualized in Figure 8 below,

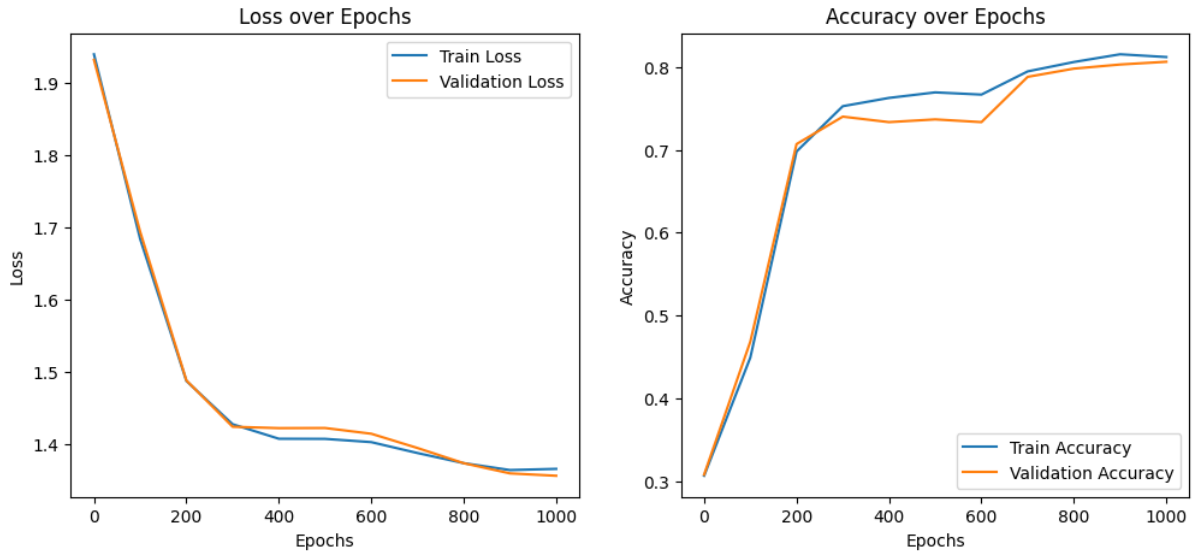

Figure 8: Loss and Accuracy chart on training and validation over a thousand epochs.

Loss Analysis:

- **Training Loss:** The training loss decreased rapidly in the first epochs, demonstrating that the model was fast learning from the training data. As the epochs advanced, the pace of loss reduction decreased, and the loss curve began to plateau, indicating that the model was reaching its optimal state.

- **Validation Loss:** The validation loss closely followed the training loss trajectory, indicating effective model generalization. The comparable drop and eventual plateau in both training and validation loss indicate that the model did not overfit the training data.

Accuracy Analysis:

- **Training Accuracy:** The accuracy of the training dataset improved dramatically in the early phases of training, eventually stabilizing after around 400 epochs. This stabilization indicates that the model has effectively learned the patterns in the training data.

- **Validation Accuracy:** Interestingly, the validation accuracy occasionally outperformed the training accuracy, which might indicate improved generalization or anomalies in the validation dataset that made it simpler for the model to predict. After about 600 epochs, the validation accuracy remained slightly higher than the training accuracy, confirming the concept of successful model generalization.

The tight alignment of training and validation loss, together with comparably high accuracy metrics, implies successful model training without significant overfitting, as the model efficiently learned from the training data and generalized well to new data. The rare greater validation accuracy compared to training accuracy, albeit exceptional, might be attributed to factors such as the degree of representativeness or relative simplicity of the validation set compared to the training set, indicating the need for more exploration into dataset features.

Class Accuracy Evaluation:

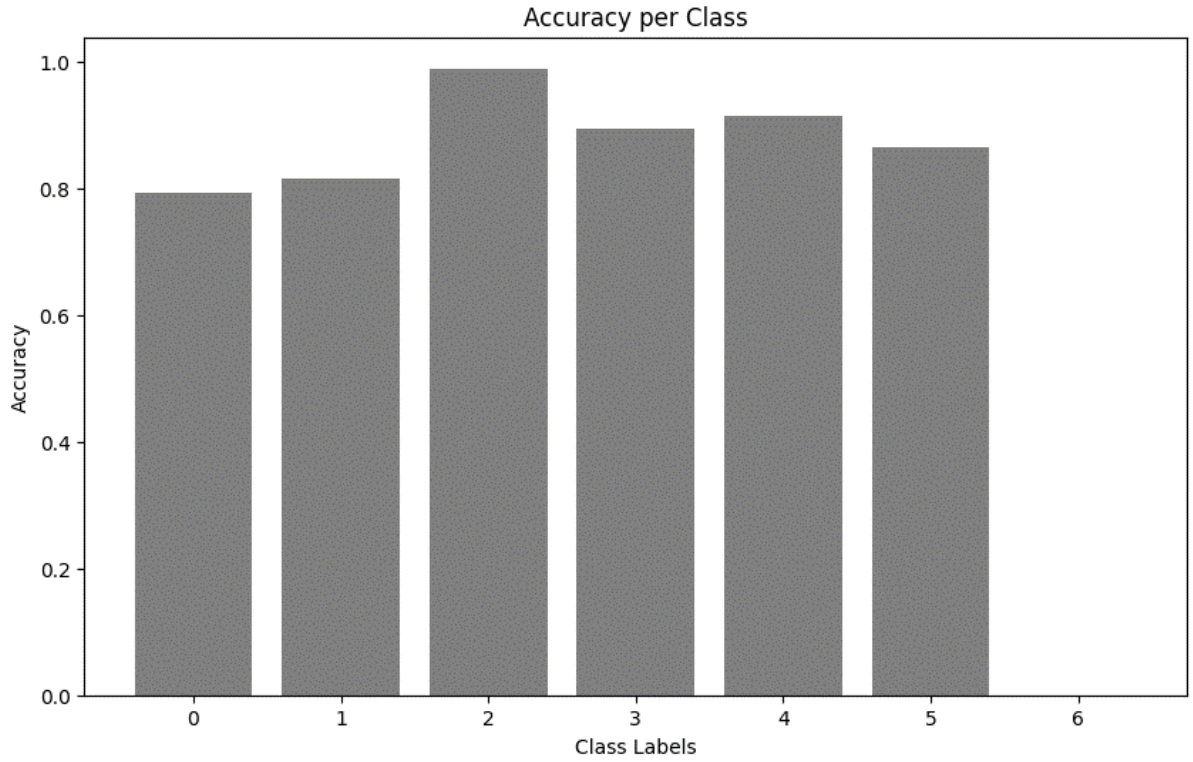

Figure 9: Overall classification accuracy on different labels of the graph nodes.

Figure 9 illustrates the accuracy metrics for a classification model across seven distinct classes, labeled from 0 to 6. The model is highly accurate in classes 0 through 5, with accuracies of around 0.82 and 0.84, respectively, with class 2 reaching the maximum accuracy of around 0.88. However, there is one major exception: class 6, where the accuracy is 0, suggesting that the model does not properly predict any instances of this class. This mismatch shows that class 6 may have difficulties, such as inadequate or unrepresentative training data, or features that the model is missing. Improving the model's performance for class 6 may entail reevaluating the training procedure, feature engineering, and maybe boosting the variety and number of training data for this class to obtain more consistent accuracy across all classes.

Testing phase:

Our model performed well during the training and validation phase and the prediction accuracy of most of the classes is high. We will now evaluate the result (Figure 10) of the model on the testing set.

```
Test Set Evaluation Results:
+-----------+--------+
|  Metric   | Value  |
+-----------+--------+
|   Loss    | 1.3486 |
| Accuracy  | 0.8162 |
| Precision | 0.7090 |
|  Recall   | 0.7530 |
| F1 Score  | 0.7296 |
|  AUC-ROC  | 0.9489 |
+-----------+--------+
```

Figure 10: Different metrics to evaluate the model on test set.

The model's test set assessment results provide a variety of performance measures, offering a thorough picture of its efficacy and dependability. The model has a modest loss of 1.3486 and an acceptable accuracy of 0.8162, demonstrating a high ability to accurately identify in most cases. Precision is recorded as 0.7090, indicating that a large fraction of affirmative identifications was right, but there is still space for improvement. The recall of 0.7530 indicates that the model is generally successful at recognizing all relevant cases but may overlook some positive cases. The F1 Score of 0.7296 indicates a fair trade-off between precision and recall. Notably, the AUC-ROC score is 0.9489, indicating great discrimination across classes, particularly, efficiently discriminating between positive and negative classes, highlighting the model's strong predictive capability. This extensive study indicates that, while the model performs admirably overall, particularly in differentiating class features, there is a need for future adjustments to improve precision and recall.

Visualization of graph embeddings:

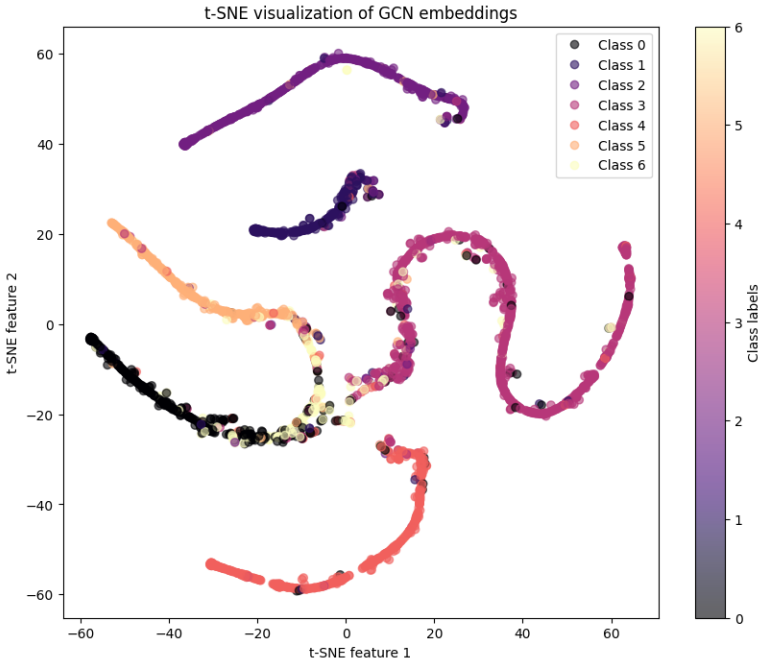

Figure 11: Graph embeddings in 2D space.

The t-SNE visualization of GCN embeddings in 2D space (Figure 11) shows separate clusters corresponding to class labels ranging from 0 to 6, demonstrating the model's ability to distinguish between most classes based on their learned embeddings. Notably, classes 0–5 show rather obvious division, with each forming coherent clusters over the two-dimensional feature space. However, Class 6 has a significant overlap with other classes, especially where it overlaps with the routes of multiple other class clusters. This overlap is a likely reason for the poorer accuracy seen for Class 6, as previously noted. The mixing of Class 6 with others indicates that the model struggles to distinguish its characteristics, resulting in misclassifications. The overlapping distribution indicates that the model does not sufficiently differentiate the embeddings for Class 6, which could be due to underrepresentation in the training data, insufficient model complexity to capture its characteristics, or inherent similarities with other classes that confuse the model. This visualization emphasizes the need for model refining or data resampling to improve the uniqueness and accuracy of Class 6 forecasts.

Observation of Class Prediction:

	Node ID	Predicted Label	True Label
0	694	5	5
1	475	0	0
2	1138	4	4
3	1283	1	1
4	1983	4	0
5	1292	2	2
6	2588	3	3
7	1068	4	4
8	1892	0	0
9	754	2	2
10	2649	5	6
11	1475	0	0
12	79	0	0
13	1759	2	2
14	1740	2	2
15	1191	3	3
16	379	5	5
17	1975	0	6
18	1119	3	3
19	1174	0	0

Figure 12: Predicted vs. True labels of random 20 nodes.

Notably, the model predicted precisely as per our evaluation above. Only 3 out of 20 predictions were mismatched which aligns with our accuracy rate. The model predicted poorly in terms of “class 6” as we discussed above its poor accuracy in terms of this label.

Section 4: Comprehensive analysis

Ablation Studies:

Evaluation of ablation studies on the same model has been done on a different file named “Ablation_studies-GCN.ipynb”. For this matter, we reduced the hidden layers to 2. We also reduced the specific layer size with the first layer having a size of 32, and the second one’s

size is 16. The output layer remains the same. Figure 13 below illustrates the ablation GCN architecture,

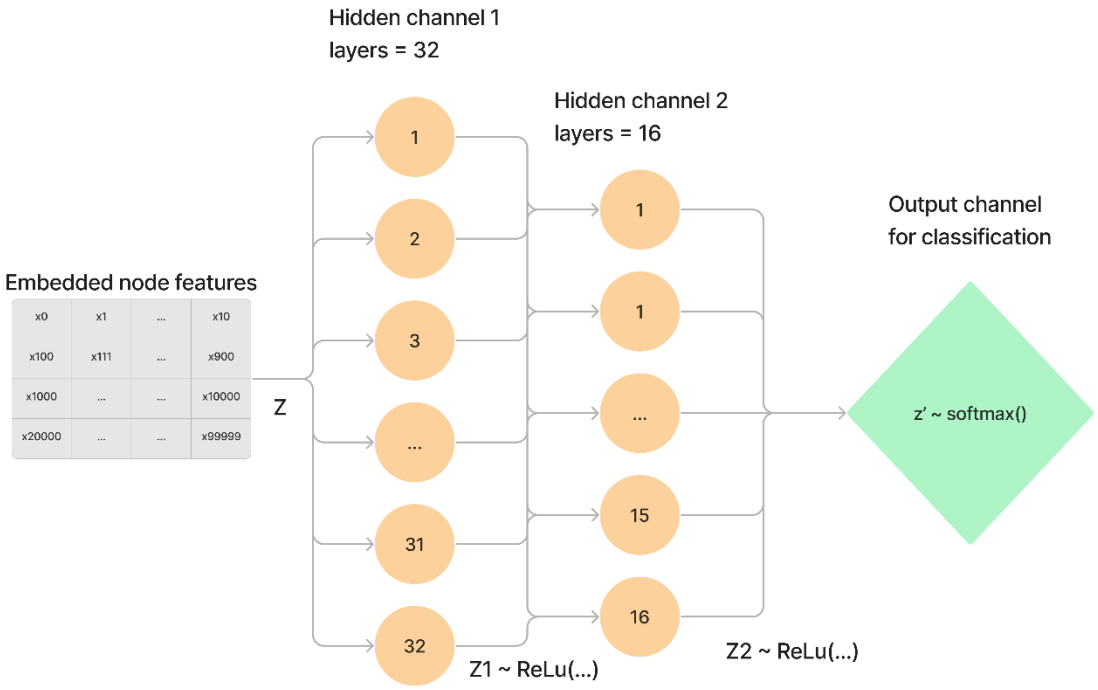

Figure 13: GCN architecture of Ablation study part

This will significantly change the performance of the model. We will evaluate this model with some illustrations below,

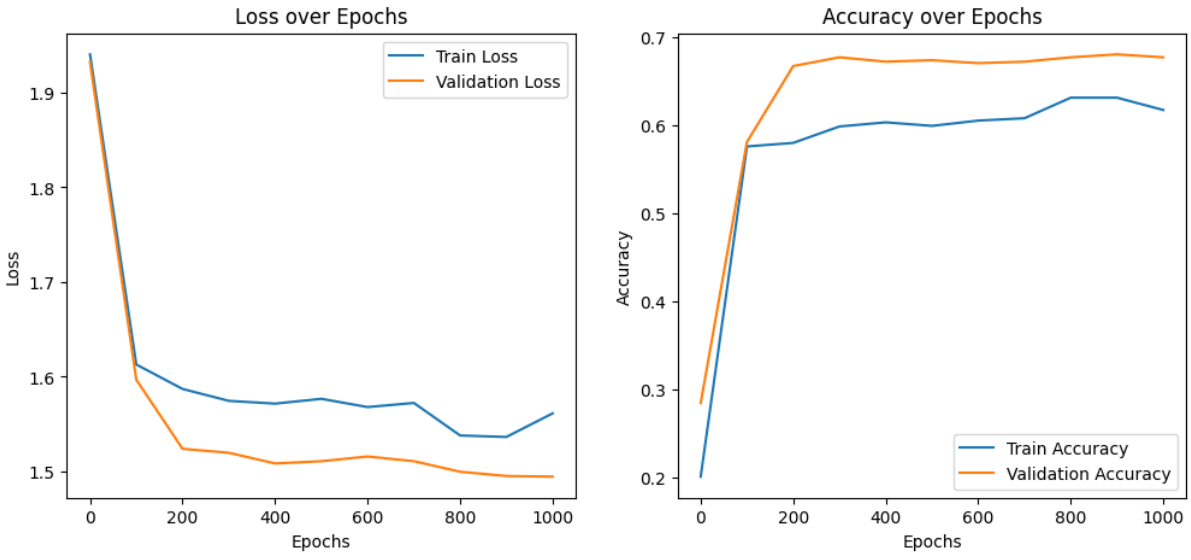

Figure 14: Loss and Accuracy chart on training and validation over a thousand epochs of ablation part GCN model

In the new graphs, both the training and validation losses begin with a fast fall, as in the previous graphs, but the validation loss ultimately diverges significantly higher after roughly 600 epochs, indicating probable overfitting as the model trains. Previously, the loss curves had been well matched throughout, suggesting improved generalization. In terms of accuracy, the current graph shows a rapid increase in both training and validation accuracy, with training accuracy plateauing around 0.65 and validation accuracy slightly lower after an initial cross-over, as opposed to the previous scenario, in which validation accuracy occasionally surpassed training accuracy.

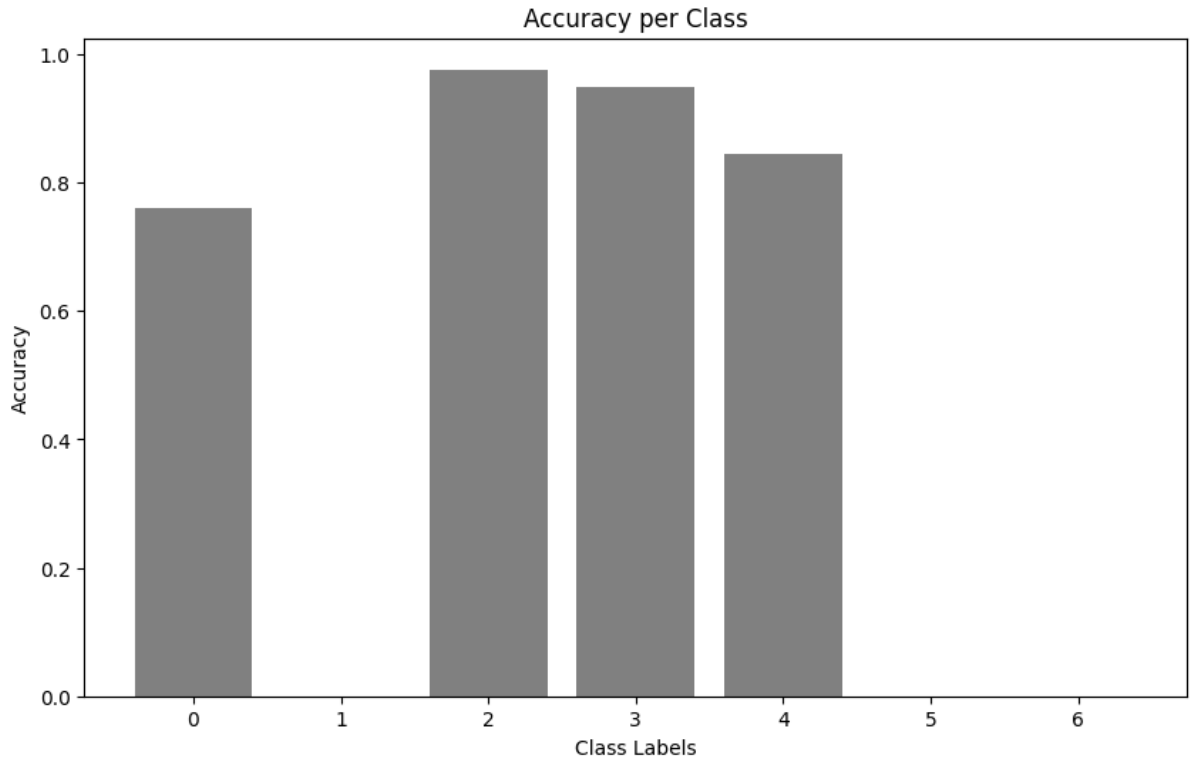

Figure 15: Class prediction accuracy of ablation study GCN model

We can see in Figure 15 that the model's performance on overall prediction of classes also decreased. In the first model, we only had poor results for class 6. Here, the model predicted poorly for class – 1, 5, and 6. Although the prediction accuracy for class 2 remains higher suggesting good feature generalization for the specific class. For more information, we will observe the graph embeddings in 2D space and try to explain this poor performance.

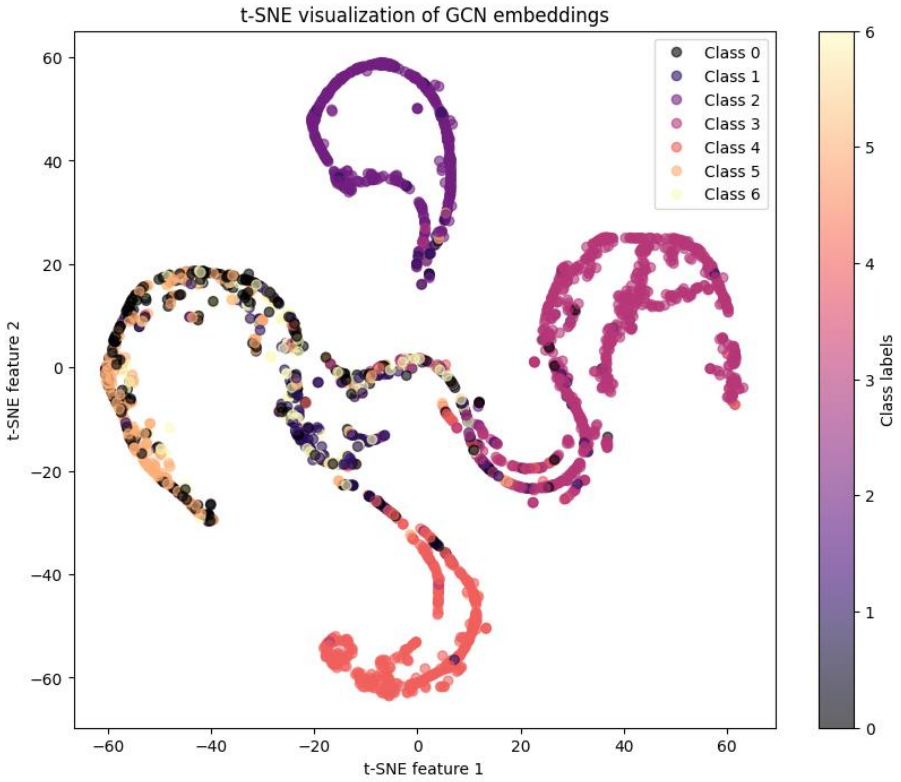

Figure 16: Visualization of graph embeddings of ablation study GCN model

In Figure 16 we can see more overlapping between different classes than the previous model embeddings. This happened due to the reduced layer and reduced size of layers. The model could not propagate feature embeddings for good generalization due to the lack of perceptron. It is visually proven that the model performs well for generalizing class 2 as it has less overlapping and quite distinguishable in this graph than the other classes.

References

Kipf, T. N., & Welling, M. (2017). *SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS*. ICLR.