**Project – Machine Learning on Graphs 097922**

**Task 2 - Graph Classification using Graph Convolutional Networks**

Roni Fridman – 205517097

Simon Bellilty – 345233563

**1. Introduction**

Graph classification is a fundamental task in geometric deep learning, where the goal is to classify entire graphs rather than individual nodes. This project aimed to train a Graph Neural Network (GNN) model to classify graphs into predefined classes, achieving over 75% accuracy on the validation set. The dataset (processed\_data.pt) contains graphs split into training, validation, and test sets, with labels for the training and validation sets and unlabeled graphs in the test set.

**2. Dataset and Preprocessing**

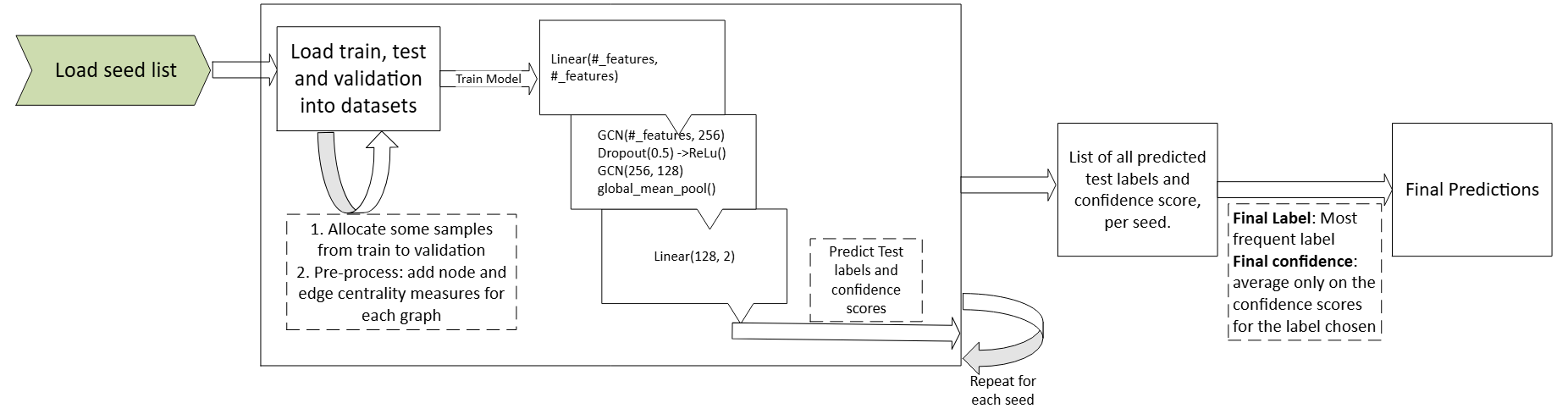
The dataset used in this project consists of PyTorch Geometric (PyG) data objects. Each graph has node features and connectivity information, representing it as a set of nodes connected by edges. Given the structure of the data, I focused on adding additional centrality features, including degree centrality, closeness centrality, betweenness centrality, and eigenvector centrality, to enrich node attributes and better capture structural nuances.

The preprocessing involved:

1. **Node-level Centrality**: Calculating various centrality measures using NetworkX and concatenating them with existing node features to capture graph structure more effectively.
2. **Edge-level Features**: Calculating edge betweenness, load centrality, and Jaccard coefficients for each edge, then concatenating these as edge attributes to provide more information during GCN convolution steps.

**3. Model Architecture**

We chose a custom Graph Convolutional Network (GCN) for this classification task. The model architecture can be seen in this diagram:



Key points in the architecture:

* **The validation set was too small**: with only 19 samples in validation, we decided that since our early stoppage depends on the validation set results, to enrich it by allocating some sample from the training set.
* **Pre-processing**: I noticed a lack of attributes on all nodes and edges. I created a function that loads every sample graph and adds nodes and edges centrality measures. This was the most influential change in bettering the loss and accuracy for the validation set.
* **Two GCN layers**: These layers capture local connectivity patterns in the graph. The first layer expands node features, while the second and third layers refine the embeddings. Dropout and ReLU activations are used for regularization and non-linearity.
* **Global Mean Pooling Layer**: Following the convolutional layers, global mean pooling is applied to aggregate node-level embeddings into a graph-level embedding, which serves as the input for classification.
* **Fully Connected Layers**: Since I used a lot more features then the original ones, I made the model quite bigger (with 256 hidden features) and as we learned in the course, it was successful to add a linear layer **before** the GCN convolution layers.  
  The last layer is of course for predicting the class.

**4. Hyperparameters**

The model’s hyperparameters were set as follows:

* **Batch Size**: 5
* **Learning Rate**: 5e-4
* **Epochs**: 100 (Max epochs without early stoppage)
* **Hidden Channels**: 256
* **Weight Decay**: 1e-4

Early stopping was used to prevent overfitting, with a patience of 3 epochs and a minimum improvement threshold (min\_delta) of 0.001 in validation loss.

**5. Training and Evaluation**

The dataset was split into training, validation, and test sets, with a seed (seed=42) for reproducibility. The training loop included:

* **Training Phase**: Each epoch involved a forward pass of the training graphs through the GCN layers, followed by backpropagation using cross-entropy loss.
* **Validation Phase**: After each epoch, the model was evaluated on the validation set. If there was no improvement in validation loss or accuracy, the early stopping counter was incremented, eventually halting training when further improvements were minimal.

**6. Results and analysis**

The model was reaching a validation accuracy of over 80% after about 4-5 epochs consistently.

But, we noticed the seed we chose varied the results, although the loss and accuracy was rather consistent, it did affect the overall prediction. So we decided to go with the bootstrap method – since the architecture was rather small and fast, we would see the variance between several different seeds. We noticed that most samples would always get the same labels, but some (such as index 12 and index 18) would vary. Thus, we then decided to run 25 different seeds and choose the label as the most frequent.  
using 25 seeds the worst label variance we got were for indexes 12 and 18, which can be seen in their prediction score (about 0.58 each, which are the lowest out of all the samples).

The model achieved a validation accuracy of approximately 0.83-0.86. To create the final predictions for the test set, the model output predictions along with confidence scores calculated using a sigmoid function for each graph in the test set. These predictions, along with their confidence scores, were saved to predictions.csv as required.

**7. Bonus Points Considerations**

In pursuit of bonus points, we enhanced the model by:

1. Implementing centrality-based node and edge feature engineering.
2. Implementing the bootstrap and weak learners approach to determine the label of problematic samples the model predicts with a confidence score that is close to 0.5.
3. Added the confidence scores to the predictions.csv file as requested.

**Conclusion**

This project demonstrated the effectiveness of GCNs in graph classification, achieving high accuracy by leveraging structural properties and enhanced features. The addition of centrality measures improved the model's ability to capture the inherent structure of each graph. Future improvements could involve experimenting with alternative pooling methods, adding attention mechanisms, or trying deeper models like Graph Attention Networks (GAT) to capture more complex dependencies in graphs, although we wanted the architecture to be more general and not constrained by the type or source that the graphs came from.

**Appendix**

* **Result File**: The predictions for the test set are saved in **predictions.csv**
* **main.py**: Code for the second task.