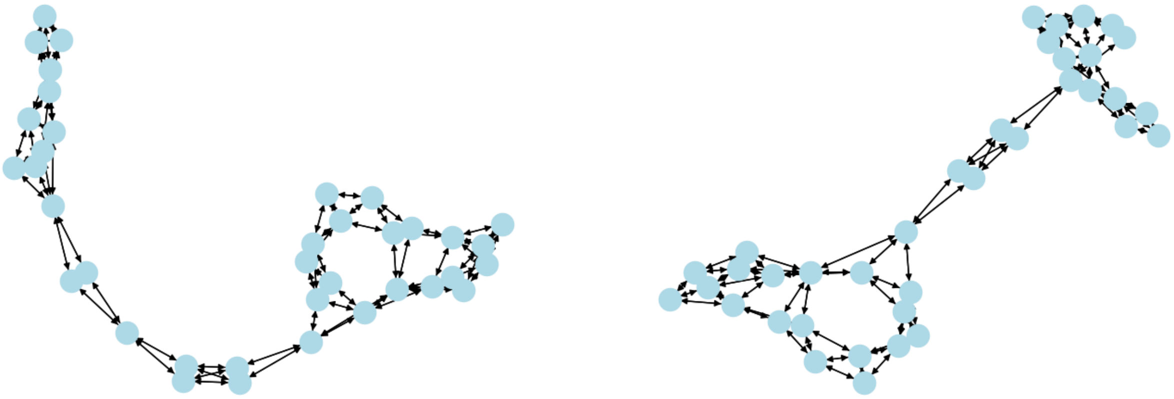
Report

> Student Name: Alhim Adonai Vera Gonzalez - UC: M16155349

Introduction

This project is submitted for **CSE6073 - Deep Learning**, taught by Dr. Jun Bai, and is part of **Homework 4**, due on 11**/18/2024**.

Graph classification is a key task in deep learning for structured data, aiming to assign labels to entire graphs based on their structural and feature properties. In this homework, the **ENZYMES dataset** is used to classify graph-structured data. Each graph in the dataset represents a protein, and the classification task involves predicting the enzyme class it belongs to. The project utilizes **PyTorch Geometric** for data handling and model implementation. The steps include data preprocessing, model development, and performance evaluation.



Steps of the Homework:

Step 1: Exploratory Data Analysis (EDA)

Before training, the dataset was analyzed to understand its structure:

1. **Number of Samples**:
    - The dataset has **600 graphs**, each representing a protein.
2. **Task**:
    - This is a **graph classification problem** to predict the enzyme class.
3. **Graph Details**:
    - Each graph has **37 nodes**, **168 edges**, and **3 features per node**.
4. **Missing Information**:
    - No missing data was found; all graphs are complete.
5. **Labels**:
    - There are **6 classes** in this multi-class problem.
6. **Data Split**:
    - The data is divided into:
        - **Training**: 70%
        - **Validation**: 15%
        - **Testing**: 15%
    - Random seeds ensure consistent splits.
7. **Preprocessing**:
    - The node features are normalized, data is shuffled, and batches are created for training and evaluation.

These steps prepare the dataset for effective training and testing of graph models.

Step 2: Model

The results of different graph neural network architectures, including GCN with varying layers and the GAT model with varying configurations, are summarized in the table below:

Model	**F1 Score**	**Accuracy**

GCN (1 Layer)	0.3733	0.3667
GCN (2 Layers)	0.5692	0.5667
GCN (5 Layers)	0.5249	0.5222
GAT (2 Layers, 2 Heads)	0.4317	0.4556
GAT (3 Layers, 4 Heads)	0.5622	0.5556

GCN (Graph Convolutional Network):

- The GCN model aggregates information from neighboring nodes to learn graph representations. It uses simple convolutional layers with ReLU activation and dropout for regularization.
- Despite its simplicity, GCN effectively captures graph structures but may struggle with deeper architectures due to over-smoothing or vanishing gradients.
    - **1 Layer GCN** shows the lowest performance in both F1 score and accuracy, as its shallow structure may not capture the graph's complexity effectively.
    - **2 Layers GCN** achieves the best results among GCN models, striking a balance between depth and overfitting.
    - **5 Layers GCN** shows slightly reduced performance, likely due to overfitting or gradient vanishing in deeper networks.

GAT (Graph Attention Network):

- The GAT model introduces attention mechanisms to assign different weights to neighboring nodes, allowing it to focus on the most relevant neighbors during message passing.
- Its multi-head attention enhances feature representation, making it more robust in handling complex graph structures compared to traditional GCNs.
    - **2 Layers, 2 Heads GAT** has a moderate performance but does not outperform the 2-layer GCN.
    - **3 Layers, 4 Heads GAT** achieves competitive results, with accuracy and F1 scores comparable to the best GCN model.

Step 3: Objective

To train the models, the **Cross-Entropy Loss** function is used. This loss function is well-suited for multi-class classification tasks, as it calculates the difference between the predicted probabilities and the true class labels.

Why Cross-Entropy Loss?

1. **Multi-Class Capability**:

   The dataset involves **6 classes**, and Cross-Entropy Loss effectively measures the performance of the model's predictions for each class.

2. **Probability-Based**:

   The model outputs probabilities for each class (via `softmax`), and Cross-Entropy Loss penalizes predictions that are far from the true class.

3. **Efficiency**:

   It is computationally efficient and works well with the gradient-based optimization techniques used in GCN and GAT.

This loss function ensures that the models learn to correctly classify the graphs into their respective enzyme classes.

Step 4: Optimization

The **Adam Optimizer** was selected to train the models.

Why Adam Optimizer?

1. **Adaptive Learning Rate**:

   - Adam adjusts the learning rate for each parameter during training, making it effective for handling sparse gradients or features with different scales.

2. **Efficiency**:

   - It combines the benefits of **momentum-based** optimization (like SGD with momentum) and **adaptive learning rates** (like Adagrad), resulting in faster convergence.

3. **Robustness**:
   - Adam performs well across a wide range of tasks and architectures without requiring extensive hyperparameter tuning.
4. **Suitability for GNNs**:
   - Graph Neural Networks often involve complex gradient landscapes, and Adam's adaptive nature ensures stable and efficient updates for the graph parameters.

By using Adam, the training process becomes more efficient and reliable, especially for models like GCN and GAT, which involve multiple layers and attention mechanisms.

Step 5: Model Selection

Based on the training results, the **GCN with 2 layers** achieved the best performance among all models, with:

- **F1 Score**: 0.5692
- **Accuracy**: 0.5667

Learning Rate Experiments with best model:

Learning Rate	F1 Score	Accuracy
0.001	0.5136	0.5222
0.01	0.4627	0.4556
0.1	0.3652	0.3889

The results show that a **learning rate of 0.001** provided the most stable and accurate performance, highlighting the importance of selecting an appropriate learning rate to prevent underfitting or overfitting.

Comparison with GAT:

The **GAT with 3 layers and 4 attention heads** achieved competitive results:

- **F1 Score**: 0.5622

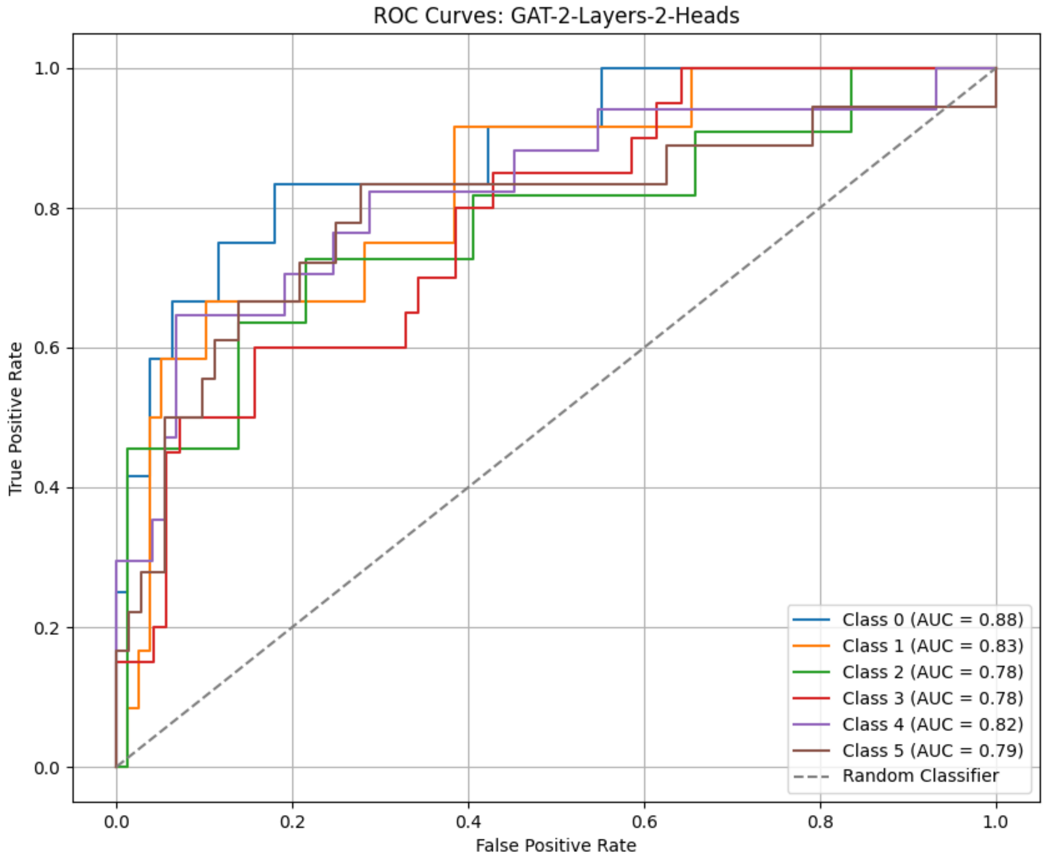
- **Accuracy**: 0.5556

While the GAT model demonstrates strong performance, the simpler architecture of the 2-layer GCN makes it more efficient and easier to train for this task. Thus, the GCN (2 layers) is selected as the best-performing model.

Step 6 Model Performance:

Below are the AUC curves for different models and configurations.

1. GAT with 2 Layers and 2 Attention Heads

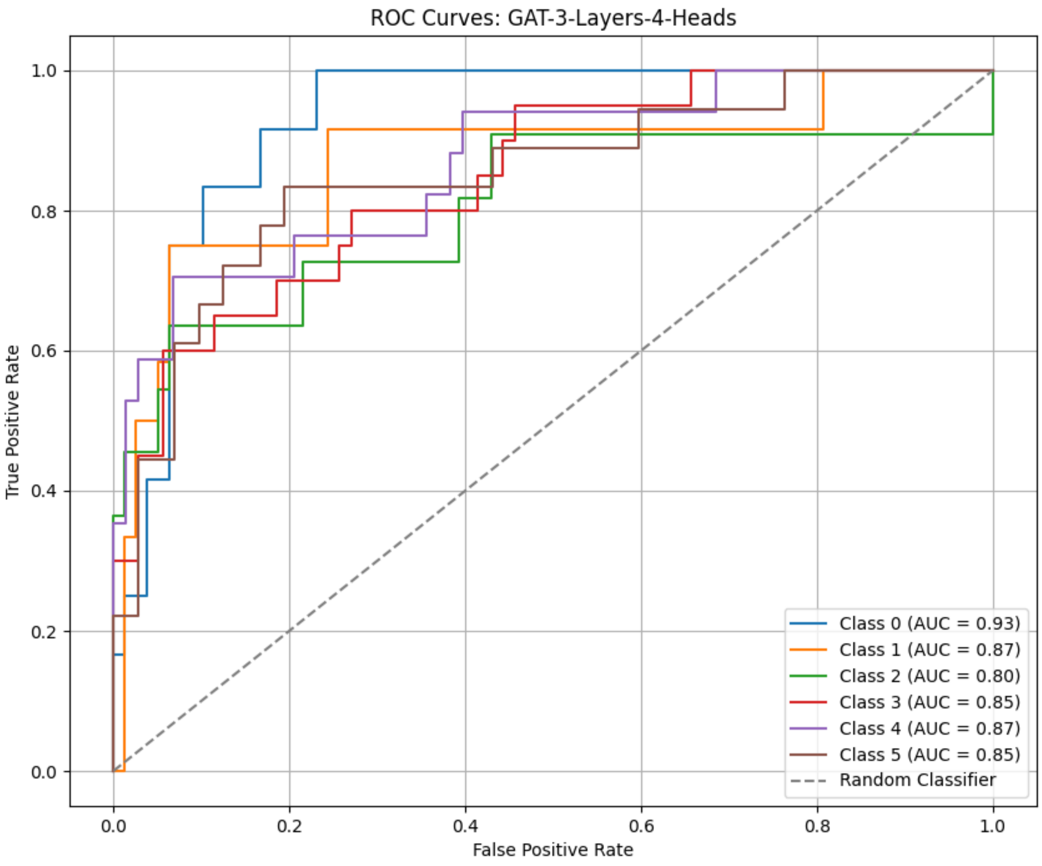


AUC Curve for GAT with 2 Layers and 2 Attention Heads

Observation:

- The AUC values for each class range between **0.78 and 0.88**.
- Class 0 shows the highest AUC score of **0.88**, indicating strong performance in this class.
- Performance is consistent across classes but slightly lower compared to models with more layers.

2. GAT with 3 Layers and 4 Attention Heads

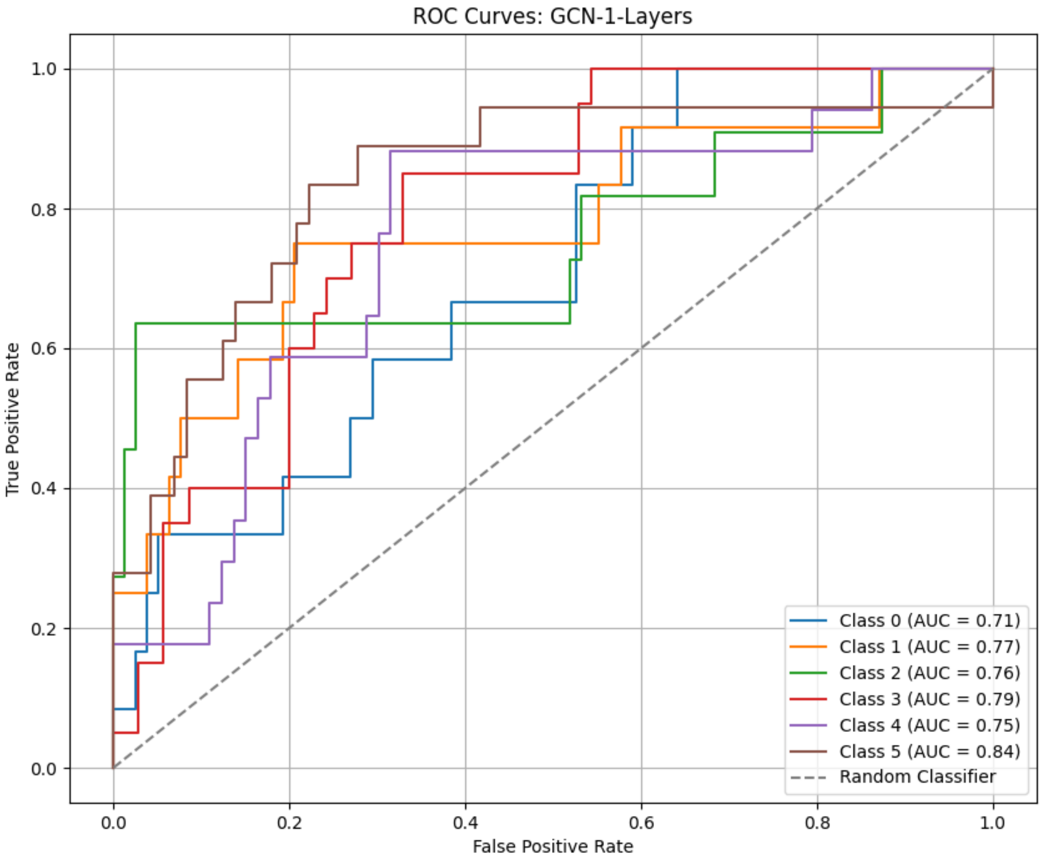


AUC Curve for GAT with 3 Layers and 4 Attention Heads

Observation:

- AUC scores improve with deeper architecture and more attention heads, ranging from **0.80 to 0.93**.
- Class 0 achieves the highest AUC of **0.93**, showing significant improvement in classification performance for this class.
- Overall, this configuration demonstrates better class separability than the GAT with 2 layers.

3. GCN with 1 Layer



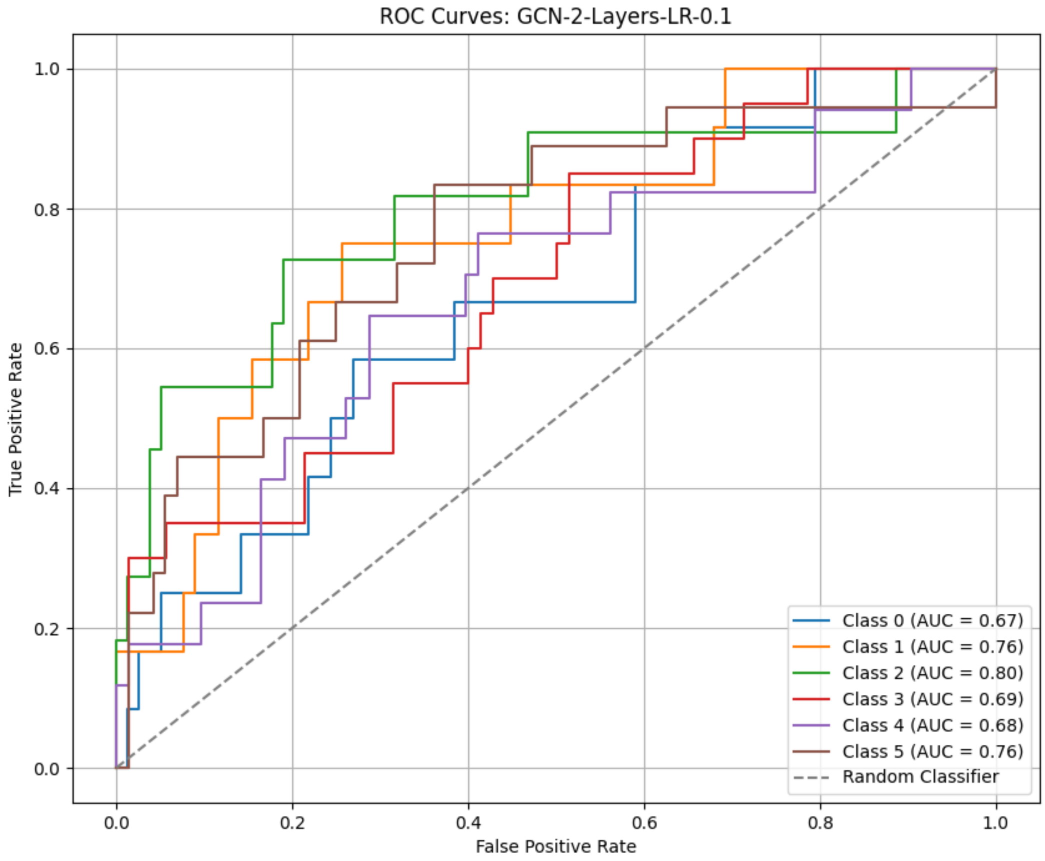
AUC Curve for GCN with 1 Layer

Observation:

- AUC scores range from **0.71 to 0.84**, with Class 5 achieving the highest score.

- The model performs reasonably well for a shallow architecture, but its performance is limited compared to deeper models.

4. GCN with 2 Layers and Learning Rate 0.1

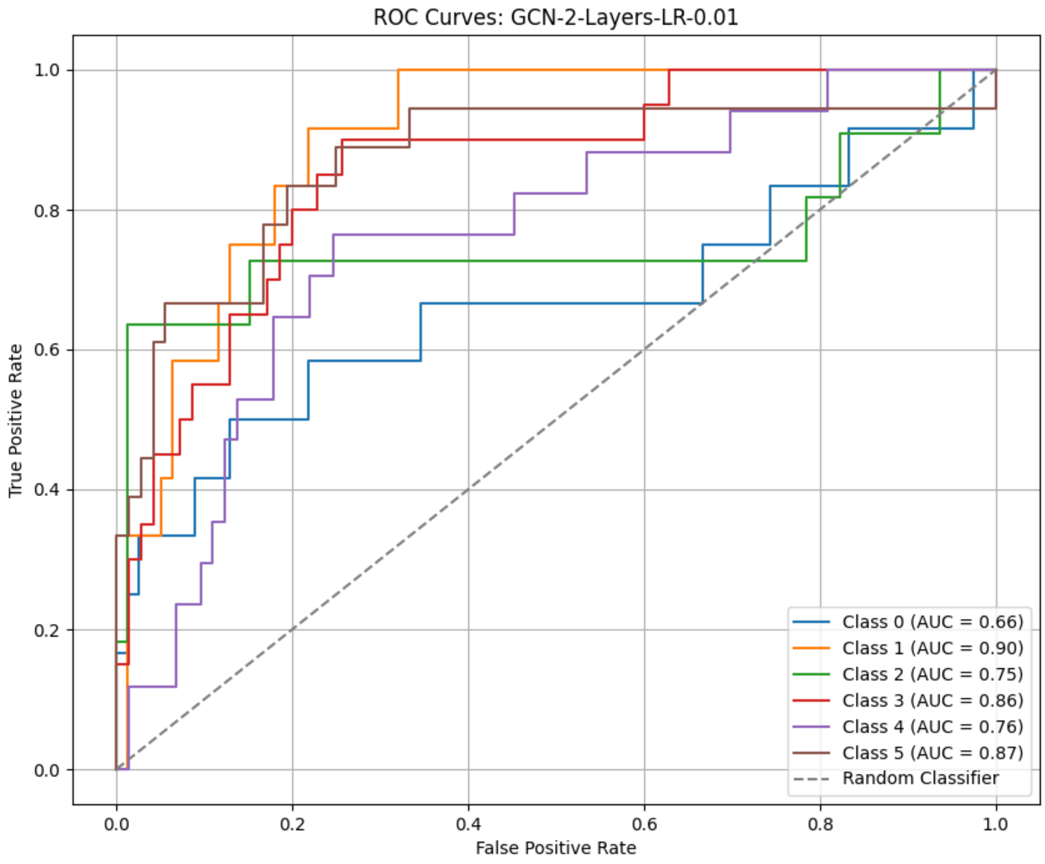


AUC Curve for GCN with 2 Layers and Learning Rate 0.1

Observation:

- AUC values range between **0.67 and 0.80**, with moderate performance across all classes.
- High learning rate seems to lead to instability and lower performance compared to smaller learning rates.

5. GCN with 2 Layers and Learning Rate 0.01

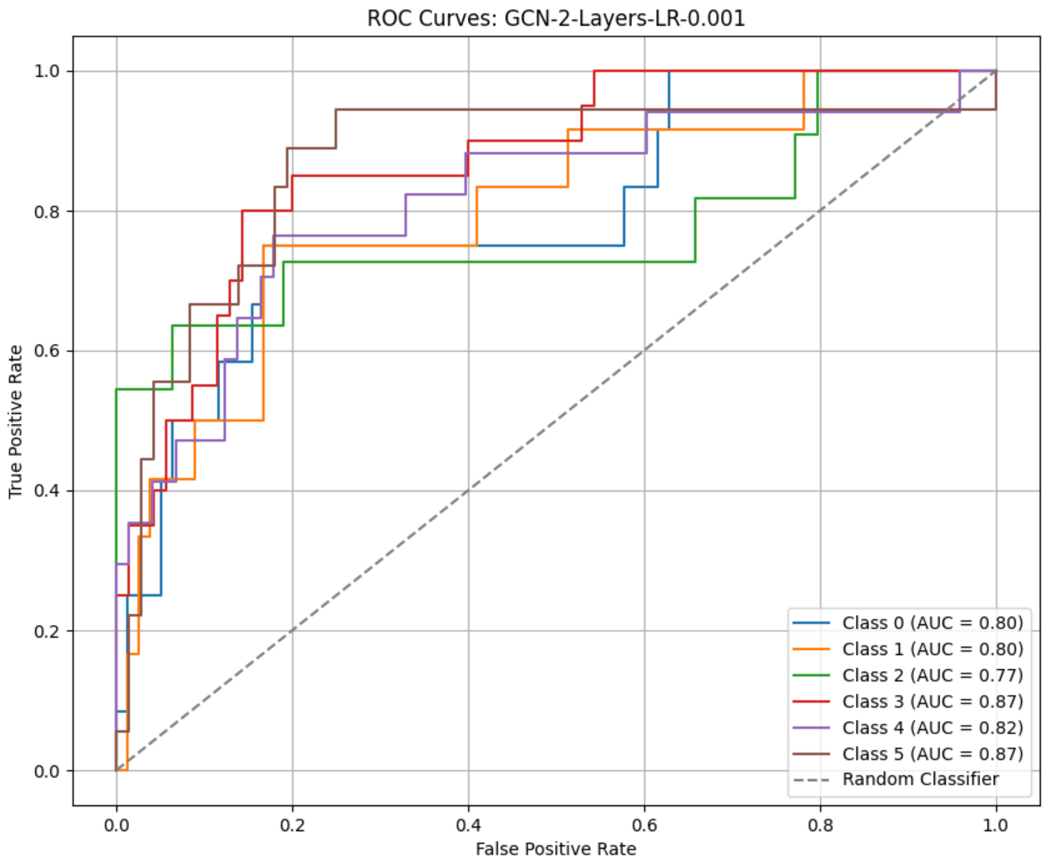


AUC Curve for GCN with 2 Layers and Learning Rate 0.01

Observation:

- AUC values range from **0.66 to 0.90**, with Class 1 achieving the highest score of **0.90**.
- Moderate improvement over a higher learning rate but still not as stable as smaller learning rates.

6. GCN with 2 Layers and Learning Rate 0.001

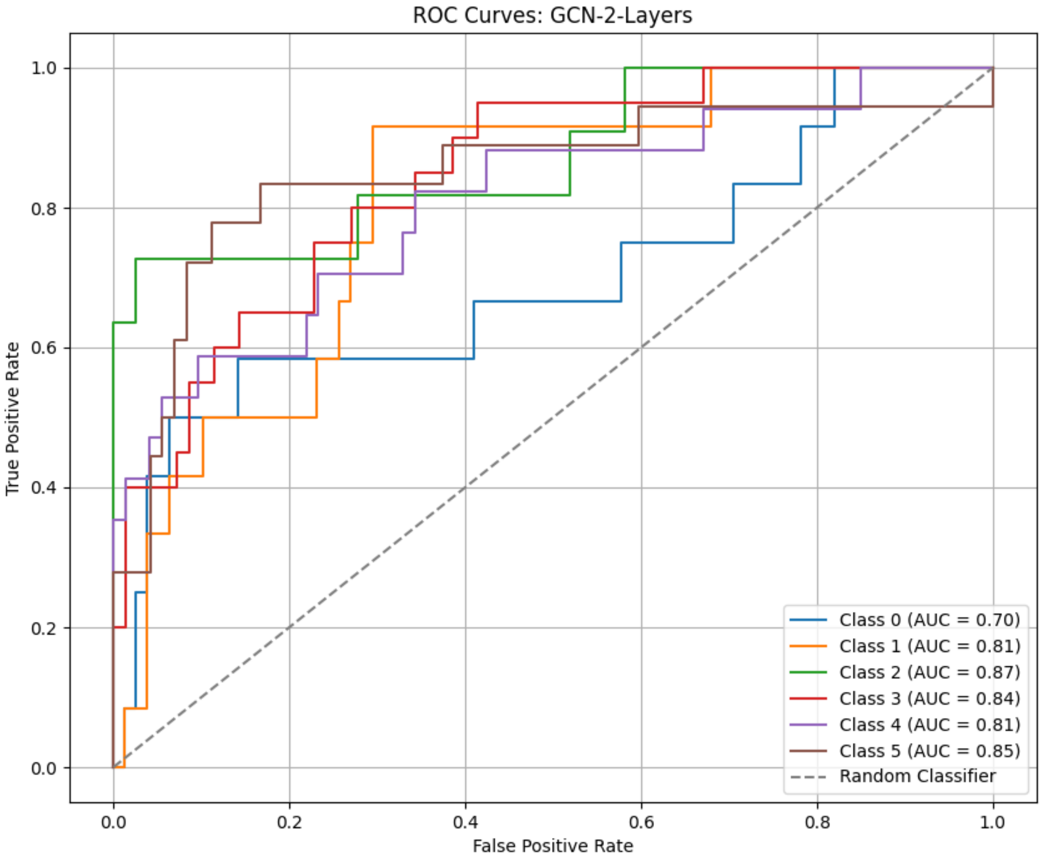


AUC Curve for GCN with 2 Layers and Learning Rate 0.001

Observation:

- AUC scores are consistent and range between **0.77 and 0.87**.
- Class 3 achieves the highest score of **0.87**, demonstrating stability and balanced class separability with this learning rate.

7. GCN with 2 Layers

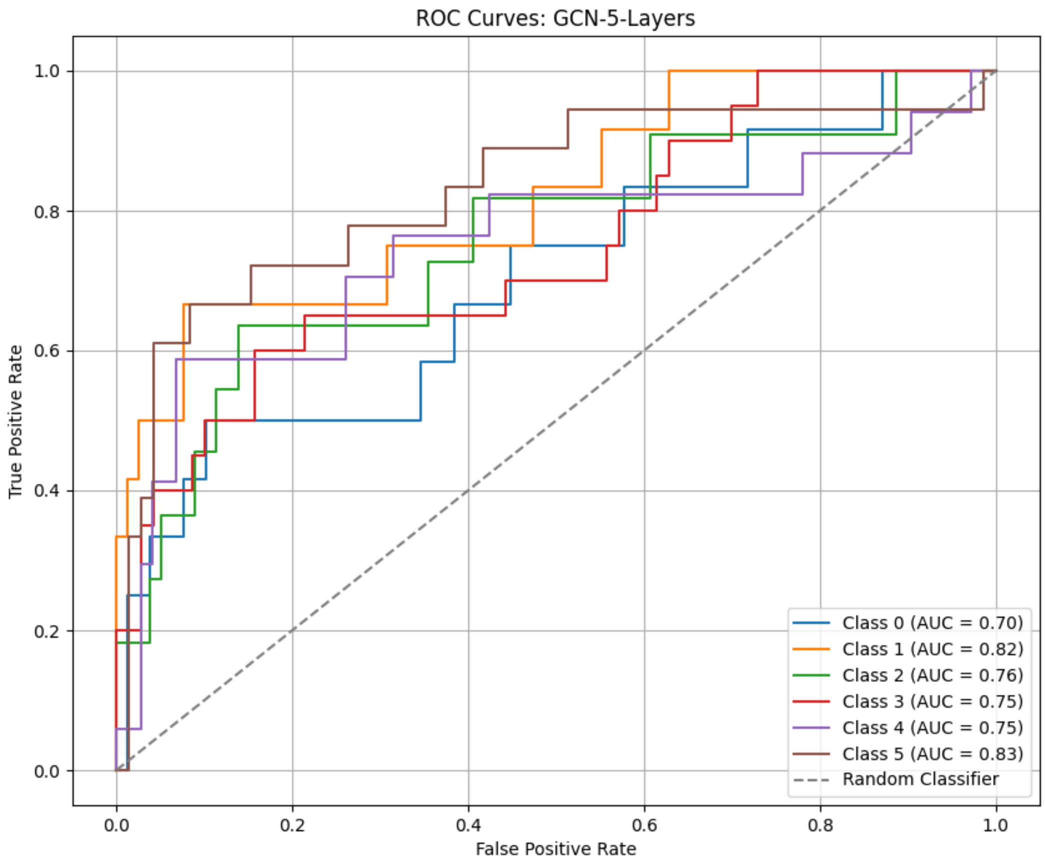


AUC Curve for GCN with 2 Layers

Observation:

- AUC values range between **0.70 and 0.87**, showing strong and consistent performance across classes.
- Class 2 has the highest AUC score of **0.87**, highlighting the effectiveness of this configuration.

8. GCN with 5 Layers



AUC Curve for GCN with 5 Layers

Observation:

- AUC values range between **0.70 and 0.83**.
- Deeper architecture does not necessarily improve performance, with slightly reduced AUC values compared to the 2-layer model.
- Class 1 achieves the highest AUC score of **0.83**.

Summary of AUC Curve Analysis

1. **Best Overall Model:**

The **GAT with 3 Layers and 4 Attention Heads** achieves the highest overall AUC values, indicating better class separability.

2. **Effect of Learning Rate:**

   For the GCN with 2 layers, a learning rate of **0.001** offers the best balance between performance and stability.

3. **Depth and Performance:**

   Deeper models like GCN with 5 layers show diminishing returns, while the 2-layer models strike a balance between complexity and performance.

Conclusion

In this study, we evaluated the performance of GCN and GAT models for graph classification tasks using the ENZYMES dataset. Among the tested configurations, the **GAT model with 3 layers and 4 attention heads** achieved the best overall performance, demonstrating superior AUC scores and class separability. For GCN models, the **2-layer configuration with a learning rate of 0.001** provided a stable and balanced performance. Increasing model depth did not always lead to better results, highlighting the importance of selecting the right architecture and hyperparameters. These findings emphasize the effectiveness of attention mechanisms in graph learning and the need for careful model selection based on task requirements.

Code Structure:

The main code for this project allows you to run all model configurations by simply executing `python main.py`. Before running the script, ensure all dependencies are installed by using `pip install -r requirements.txt`. The requirements file is structured to include everything needed for training and evaluating segmentation models.

Additionally, the `test_model.py` script is available for evaluating the best-performing model on the task. This script loads and preprocesses the dataset, applies the saved model weights, and calculates metrics. The code is designed for easy testing and performance comparison, making it straightforward to reproduce results or explore further model adjustments.

Repo github: https://github.com/AdonaiVera/graph-classification-enzymes

References:

K. M. Borgwardt, C. S. Ong, S. Schoenauer, S. V. N. Vishwanathan, A. J. Smola, and H. P. Kriegel. Protein function prediction via graph kernels. Bioinformatics, 21(Suppl 1):i47–i56, Jun 2005.

I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. Nucleic Acids Research, 32D:431 433, 2004.