Software Engineering Department
Braude College
Capstone Project Phase A – 61998

Graph Generation using Graph U-Nets

24-1-R-4

Yana Raitsin 316086941
Yana.Raitsin@e.braude.ac.il

Hanil Zarbailov 319331419
Hanilz1995@gmail.com

Supervisor:
Dr. Renata Avros

Abstract
In the rapidly evolving landscape of graph representation learning and generative modeling, the quest for effective methodologies to understand and generate complex graph structures remains a pivotal challenge. This project introduces a model that integrates two architectures: MISC-GAN and Graph U-Nets. Our proposed model offers a unified framework for graph generation, leveraging the strengths of both models.

Index Terms: Generative adversarial network; Coarse graphs; Granularity levels; U-Nets; Cycle Consistency; Cycle-GAN; Multi-scale Analysis; Hierarchical Graph Representation.

1. Introduction

1.1 Problem statement

While the Misc-GAN model effectively captures the overall structure of input graphs, its "Multi-Scale Graph Representation Module" has limitations in precision. The proposal involves integrating a trained Graph U-Net into this module after the extraction of the coarse graphs. Graph U-Nets are adept at handling graph-structured data and can enhance the extraction of coarse graphs at various granularity levels. This integration aims to improve the accuracy of graph generation in the Misc-GAN model, leading to better performance in complex graph data analysis.

1.2 Motivation for Empowering Misc-GAN using Graph U-Nets

Graph U-Nets, inspired by the success of U-Nets in image processing, offer a promising direction for enhancing Misc-GAN. Graph U-Nets are adept at capturing hierarchical and topological features of graphs, essential for understanding complex network structures.

Misc-GAN is used to generate synthetic graphs in each granularity layer and then merge them all to construct a synthetic graph that is very similar to the source graph. The accuracy and relevance of these granularity layers heavily depend on the coarse graphs extracted from the original graph in the "Multi-Scale Graph Representation Module". Enhancing this module's outcome with Graph U-Nets is expected to greatly improve the accuracy of the synthetic graphs generated.

Graph U-Nets are particularly adept at considering node features in the source graph while preserving spatial information during graph reconstruction. This capability is pivotal for our enhancement strategy. By utilizing the strengths of Graph U-Nets to enhance and reconstruct coarse graphs, a significant improvement in the quality of synthetic graphs generated by Misc-GAN is anticipated.

1.3 Special uses

The improved Misc-GAN model, augmented with Graph U-Nets, opens up a plethora of applications that were previously challenging or inefficient with the standard Misc-GAN. This enhanced model is particularly valuable in domains where understanding and replicating complex graph structures are crucial.

One prominent area of application is in social network analysis. The enhanced model can effectively replicate and study social networks' dynamics, accounting for intricate community structures and evolving relationships. This is crucial in understanding social phenomena, network growth patterns, and information dissemination.

Another critical application is in biological networks, particularly in protein-protein interaction networks and neural network mapping. The model's ability to capture multi-scale structures aids in unraveling the complexities of biological systems, facilitating advancements in drug discovery and understanding of neural pathways.

Furthermore, the model finds relevance in technological networks, like the Internet and communication networks. It can simulate network behaviors under different scenarios, aiding in robust network design and analysis of network vulnerabilities.

In summary, the integration of Graph U-Nets into Misc-GAN expands its utility for analyzing and synthesizing complex networks across various domains. This enhanced model is not only a step forward in generative models for graphs but also a bridge connecting intricate network structures to practical, real-world applications.

2. Related Works

This section delves into the foundational work underpinning our research, focusing on three key areas: Generative Adversarial Networks (GANs), Multi-scale Graph Analysis, and the evolving field of Graph Neural Networks, particularly Graph U-Nets.

2.1. Graph Neural Networks and Graph U-Nets

Graph Neural Networks (GNNs) have emerged as powerful tools for learning from graph-structured data. Among the advancements in this area, the Graph U-Net(H. Gao and S. Ji., 2019) stands out for its ability to operate on graph data while maintaining hierarchical information. The architecture of Graph U-Nets includes novel components such as G-Pooling and G-Unpooling layers, which enable efficient learning of graph topologies and features. By integrating Graph U-Nets into the Misc-GAN framework, we aim to leverage their superior capability in capturing hierarchical graph structures, thereby enhancing the model's performance in generating and reconstructing complex networks.

2.2. Generative Adversarial Networks (GANs) And Misc-GAN

The concept of GANs(Goodfellow et al., 2014) revolutionized the field of generative modeling. GANs comprise two neural networks, the generator and the discriminator, trained simultaneously through adversarial processes. This framework has been extensively applied and adapted in various domains, including image and text generation. Misc-GAN, a variant of GANs, extends this approach to the generation of graph-structured data, addressing the challenges posed by the complex, high-dimensional nature of graphs (Zhou et al., 2019).

In summary, our work builds upon these foundational concepts, proposing a novel integration of Graph U-Nets into the Misc-GAN framework. This integration aims to harness the strengths of both models, leading to significant improvements in the generation and analysis of complex graph structures.

3. Background

3.1 Neural Network

A neural network represents a paradigm within the field of artificial intelligence, wherein computational systems assimilate data through a mechanism inspired by the neural architecture of the human brain. Specifically categorized as a subset of machine learning known as deep learning, neural networks leverage an arrangement of interconnected nodes or neurons organized in layers, mirroring the organizational structure of the human brain. This configuration engenders an adaptive system, enabling computers to iteratively refine their performance by learning from errors. Consequently, artificial neural networks are applied to address intricate problem domains, exemplified by endeavors such as document summarization and facial recognition, with a heightened emphasis on precision.

The training process of a neural network entails the provision of input data and subsequently adjusting its weights to mitigate the disparity between the anticipated output and the observed actual output.

3.1.1 Backpropagation

Backpropagation, also recognized as the backward propagation of errors, constitutes a method for error evaluation that proceeds in reverse from output nodes to input nodes. It serves as a valuable mathematical instrument for enhancing predictive accuracy in the domains of data mining and machine learning. Neural networks employ backpropagation to calculate a gradient descent with respect to weight values across varied inputs. The systems undergo refinement through the adjustment of connection weights, aiming to minimize the discordance between desired and actual system outputs to the greatest extent possible.

3.1.2 Loss function

The loss function in a neural network calculates the disparity between the current output produced by the algorithm and the expected output. This metric is pivotal in evaluating the algorithm's efficacy in modeling the underlying data.

3.1.3 Activation function

An activation function plays a pivotal role in determining the activation state of a neuron within a neural network, serving as a decision mechanism to ascertain the significance of the neuron's input during the predictive process. By employing straightforward mathematical operations, the activation function derives the output based on a given set of input values supplied to a node or layer.

3.1.4 Bias in Neural Networks

Bias in neural networks is characterized as a constant term added to the product of features and weights, serving the purpose of offsetting the result. This incorporation of bias facilitates the models in adjusting the activation function, thereby influencing its shift towards either the positive or negative side.

3.1.5 Convolutional Neural Network

Convolutional Neural Networks (CNNs) represent a specialized class of neural networks designed for the processing of grid-like data structures, notably applied in contexts such as image analysis and recognition. Primarily employed for tasks involving visual information, CNNs excel in learning intricate visual features and patterns through the examination of pixel correlations, a process referred to as feature extraction. Their applicability extends to picture classification, object detection, and segmentation, where the hierarchical learning of features is achieved by the application of numerous filters to the input data. These filters enable the network to discern patterns and features at varying levels of abstraction, encompassing low-level attributes such as edges and corners, to more complex high-level features, including shapes and objects.

3.1.6 Convolutional Layer

The fundamental computational operation within a Convolutional Neural Network (CNN) involves a series of feature detectors, typically represented by a 3x3 matrix, traversing the input data. This process applies a mathematical operation known as convolution, wherein the feature detectors compute the dot product with localized regions of the input data. The outcome of this convolution is a collection of feature maps or activation maps, which effectively encapsulate distinct local patterns or features inherent in the input data. Convolutional layers play a pivotal role in extracting meaningful features across diverse scales and orientations, contributing to the network's capacity for hierarchical pattern recognition.

3.1.7 Pooling Layer

Pooling layers, recognized as a form of downsampling, orchestrate the reduction of dimensionality, thereby curtailing the number of parameters within the input. Analogous to the convolutional layer, the pooling operation involves the traversal of a filter across the entire input; however, the critical distinction lies in the absence of weights associated with this filter. Instead, the kernel applies an aggregation function to values within the receptive field, culminating in the generation of the output array. Although the pooling layer results in a loss of information, it engenders several advantages within Convolutional Neural Networks (CNNs), including complexity reduction, enhanced computational efficiency, and mitigation of overfitting risks.

3.1.8 Max Pooling

Max Pooling is a pooling operation that calculates the maximum value for patches of a feature map and uses it to create a downsampled (pooled) feature map. It is usually used after a convolutional layer. It adds a small amount of translation invariance - meaning translating the image by a small amount does not significantly affect the values of most pooled outputs.

3.1.9 Fully Connected Layer

The fully connected layer in a Convolutional Neural Network plays a pivotal role in global feature learning by connecting all neurons from the preceding layer to each neuron in the subsequent layer. Unlike convolutional and pooling layers that focus locally, the fully connected layer captures complex relationships across the entire input space. It forms the final stage of the network, learning weights for connections during training and consolidating information for comprehensive predictions or classifications. This layer's adaptability in learning intricate relationships makes it crucial for tasks requiring a holistic understanding of input data.

3.1.10 Graph Convolutional Network

Graph Convolutional Network is a specialized neural network architecture designed for processing graph-structured data, commonly found in applications such as social networks and molecular graphs. Unlike traditional Convolutional Neural Networks designed for regular grids like images, GCNs extend convolutional operations to irregular graph structures. They address the challenge posed by the non-grid topology of graphs by employing a message-passing approach, where each node aggregates information from its neighbors, simulating a convolutional operation. This enables GCNs to effectively capture and integrate information from the graph topology, making them suitable for tasks where understanding relational dependencies within the data is crucial.

3.1.11 Message Passing

Within the architecture of Graph U-Nets, message passing constitutes a fundamental mechanism for information propagation and aggregation across the graph-structured data. In the encoder phase, nodes engage in message passing to exchange information with their neighboring nodes, thereby facilitating the dissemination of local contextual knowledge. Subsequently, the received messages undergo aggregation, commonly accomplished through techniques such as summation or weighted aggregation, resulting in a refined representation of each node's neighborhood that encapsulates both local and global features. This aggregated information is then employed to update the features of individual nodes. The U-Net structure further integrates this process with downsampling in the encoder phase, akin to pooling layers, and upsampling in the decoder phase. The strategic implementation of skip connections ensures the preservation of high-resolution information during the downsampling and upsampling operations. The orchestration of message passing in Graph U-Nets, therefore, enables the model to effectively capture intricate dependencies within the graph, rendering it adept at tasks such as point cloud segmentation or molecular property prediction.

3.1.12 Softmax

The Softmax activation function plays a crucial role in neural networks, particularly in the context of classification tasks. It is commonly used in the output layer of neural networks when dealing ***with*** categorical target variables.

Concept and Functionality of Softmax:

The Softmax function is a generalization of the logistic function.

It converts a vector of raw scores (also known as logits) from the network into a probability distribution.

The output probabilities of Softmax sum up to 1, making it suitable for representing probabilities across multiple classes.

Mathematical Representation:

Given a vector z of raw scores for each class, the Softmax function is defined as:

$$Softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where i is the index of a particular class, K is the total number of classes, z_i is the raw score for class i, and e is the base of the natural logarithm.

The numerator, e^{z_i}, is the exponentiated score for class i.

The denominator $\sum_{j=1}^{K} e^{z_j}$ is the sum of exponentiated scores for all classes, ensuring that the probabilities sum up to 1.

Role in Classification:
In classification problems, Softmax is used to interpret the outputs of a neural network as probabilities.
Each output value represents the probability that the given input belongs to a particular class.
This is particularly useful for tasks where an input can belong to only one of many possible classes (e.g., image classification, where an image might represent one of several different objects).
Advantages:
Softmax provides a probability-based output for multi-class classification problems, offering a measure of confidence or certainty in the classification.
It helps in determining the most likely class for a given input.
In summary, the Softmax activation function is essential for transforming the outputs of a neural network into a probability distribution over predicted output classes. This function is particularly beneficial in scenarios requiring a clear distinction between multiple categorical outcomes.

3.2 U-Nets

U-Nets are a type of neural network architecture mainly used in image processing, particularly for tasks like image segmentation. The main characteristic of U-Nets is their architecture, which is shaped like the letter "U". This structure consists of two main paths: an encoding (or contraction) path and a decoding (or expansion) path.

3.2.1 Encoding and Decoding in U-Nets

The concepts of encoding and decoding are fundamental in the architecture of U-Nets, playing a critical role in their functionality. Encoding in U-Nets refers to the process of compressing the input data into a compact representation, capturing the essential features while reducing dimensionality. This process involves a series of convolutional and pooling layers that progressively downsample the input, abstracting and retaining critical information while minimizing data volume.

Decoding, on the other hand, is the process of reconstructing the data from its encoded form. This step is crucial for generating output that is comparable in dimension and detail to the original input. In U-Nets, decoding involves a series of up-sampling and convolutional layers that gradually upscale the encoded representation back to its original size, while refining details and features. This symmetrical structure enables the network to efficiently learn and generate high-fidelity outputs, making U-Nets particularly effective for tasks requiring precise localization and detailed reconstruction.

3.2.2 Pooling in U-Nets

Pooling layers in U-Nets reduce the spatial dimensions of the input data. This reduction is crucial for minimizing computational complexity and for abstracting the input data into a more manageable form. The pooling operation, typically a max pooling, selects the most prominent features in a region, effectively condensing the information while maintaining the most critical aspects of the data. This process not only reduces the risk of overfitting by providing an abstracted form of the input but also aids in capturing invariant features, which are essential in many applications like image segmentation.

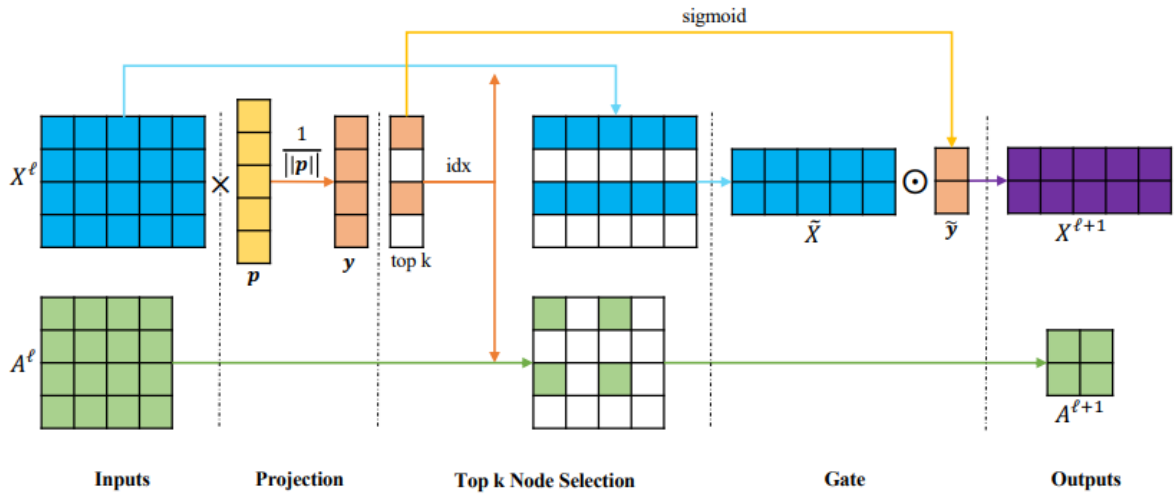
3.2.3 Unpooling in U-Nets

Unpooling layers in U-Nets are designed to reverse the effects of pooling, aiming to reconstruct the detailed structure from the condensed feature representation. This process involves up-sampling the feature maps to a higher resolution, which is crucial for tasks requiring detailed, high-resolution output. The unpooling operation often uses stored indices from the corresponding pooling layers to reconstruct the original structure of the data as accurately as possible. This approach ensures that the critical features lost during pooling are effectively recovered, enabling the U-Net to generate outputs that closely resemble the original input in terms of detail and structure.

3.3 Graph U-Nets

3.3.1 G-Pooling

Pooling is an operation which is used in CNN in order to get better generalization and performance and is best used on grid-like data such as images.
This operation takes the data and partitions it into non-overlapping rectangles, on which certain down-sampling functions are applied.
This operation cannot be applied to graphs, because there is no locality information among nodes in graphs, thus the partition operation is not applicable here. In general, the normal pooling operation outputs inconsistent results, thus, a new algorithm should take its place.
To improve the Pooling operation, the algorithm proposes the graph pooling (gPool) layer to enable down-sampling on graph data as follows:



$$y = \frac{X^l p^l}{||p^l||}$$
$$idx = rank(y, k)$$
$$\tilde{y} = sigmoid(y, k)$$
$$X^{\sim l} = X^l(idx, :)$$
$$A^{l+1} = A^l(idx, idx)$$
$$X^{l+1} = \underline{X}^l \odot (\tilde{y}1_C^T)$$

1. Projection and Scoring ($y = \frac{X^l p^l}{||p^l||}$)
   - X^l is the feature matrix of the nodes in the graph.
   - p^l is a trainable projection vector.
   - The projection $X^l p^l$ computes the scalar projection of each node's features onto p.
   - The division by $||p^l||$ (the norm of p^l) normalizes this projection.
   - The result y is a vector of scores estimating the importance of each node based on its projection value.
2. Node Ranking and Selection ($idx = rank(y, k)$):

- $rank(y, k)$ operation ranks the nodes based on their scores in y.
- It selects the indices (idx) of the top k nodes with the highest scores.
- k is a predefined number representing how many nodes to select.

3. Gating Mechanism ($\tilde{y} = sigmoid(y, k)$):
- $\tilde{y}$ is the gated score vector obtained by applying the sigmoid function to the selected scores $y(idx)$.
- Sigmoid function is used to scale these scores between 0 and 1.

4. Feature Matrix for Selected Nodes ($X^{\sim l} = X^l(idx, :)$):
- $X^{\sim l}$ is the new feature matrix formed by extracting the rows from X^l corresponding to the selected node indices idx.

5. New Adjacency Matrix ($A^{l+1} = A^l(idx, idx)$):
- A^{l+1} is the new adjacency matrix created by selecting the rows and columns from the original adjacency matrix A^l corresponding to the selected nodes.

6. Node Feature Update ($X^{l+1} = \underline{X}^l \odot (\tilde{y}1_C^T)$):
- X^{l+1} is the updated feature matrix for the next layer.
- $\odot$ denotes element-wise multiplication.
- $\tilde{y}1_C^T$ is a vector used in the gating operation, where 1_C^T is a vector of size C (number of features) with all components being 1.
- This step controls the information flow from the selected nodes, emphasizing the features of more important nodes.

3.3.2 G-Unpooling

The gUnpool layer uses the position information extracted from the gPool operation before it to reconstruct the original graph structure by using empty feature vectors for unselected nodes.

The purpose of the operation performs the inverse operation of the gPool layer and restores the graph into its original structure, with empty feature vectors for unselected nodes.

$$X^{l+1} = distribute(0_{NxC}, X^l, idx)$$

It records the locations of nodes selected in the corresponding gPool layer and uses this information to place nodes back to their original positions in the graph.

$idx \, \epsilon \, z^{*k}$ contains the indices of the selected nodes in the corresponding gPool layer that reduces the graph size from N nodes to k nodes.

$X^l \epsilon R^{kxC}$ are the feature matrix of the current graph.

0_{NxC} is the initially empty feature matrix for the new graph.

$distribute(0_{NxC}, X^l, idx)$ is the operation that distributes row vectors in X^l into 0_{NxC} feature matrix according to their corresponding indices stored in idx.

In X^{l+1} row vectors with indices in idx are updated by row vectors in X^l, while other row vectors remain zero.

3.4 GAN

Generative Adversarial Networks (GANs) consist of two neural networks, the **generator** and the **discriminator**, which are trained simultaneously. The generator creates data that is as close as possible to real data, while the discriminator evaluates the authenticity of both real and generated data. The networks engage in a min-max game where the generator tries to fool the discriminator, and the discriminator tries to accurately distinguish between real and generated data.

3.4.1 Generator

The generator serves as the creative component, tasked with generating new data instances that mimic the real data. It aims to produce outputs indistinguishable from actual data, effectively fooling the discriminator. The generator's effectiveness is honed over time through its adversarial relationship with the discriminator, learning to create increasingly convincing fakes.

3.4.2 Discriminator

The discriminator acts as the critic, differentiating between real and generated data. Its primary function is to evaluate the authenticity of the input, whether it is a genuine data instance or a fabrication from the generator. The discriminator's role is crucial in training the generator to produce more realistic data, as it provides feedback that guides the generator's improvements.

The GAN Formula: The objective function of GANs is given by:

$$min_G max_D V(D,G) = E_{x\sim P_{data}(x)}[logD(x)] + E_{z\sim P_z(z)}[log(1-D(G(z)))]$$

In this formula, $D(x)$ is the discriminator's estimate of the probability that real data instance x is real. $G(z)$ is the generator's output when given noise z. P_{data} is the data's distribution and P_z is the noise's distribution.

3.5 Misc-GAN

Misc-GAN (Multi-scale Generative Adversarial Network for Graphs) is a sophisticated neural network model designed for graph data. It stands out for its ability to capture and model the distribution of graph structures at multiple levels of granularity. This multi-scale approach is essential for understanding complex graph organizations, especially in real-world networks that exhibit hierarchical distribution over graph communities.

3.5.1 Granularity Levels

One of the critical aspects of Misc-GAN is its focus on granularity levels. The model recognizes that real-world networks are not flat but have intricate hierarchical structures. By modeling these structures at different granularity levels, Misc-GAN can capture both local and global patterns in the graph, which is vital for a more nuanced and accurate representation of the graph data.

3.5.2 Architecture of Misc-GAN

The architecture of Misc-GAN is built upon three main modules, each contributing to its unique ability to generate and reconstruct graph data:

1) **Multi-Scale Graph Representation Module:** This module is the first stage of Misc-GAN, where the target domain graph is taken as input. The module explores the hierarchical structures within the graph to extract important structures at each granularity layer. It creates a series of coarse graphs, each representing the graph at different levels of granularity.

2) **Graph Generation Module:** At this stage, Misc-GAN works at each granularity level to create synthetic coarse graphs. This process involves dividing the original graph into subgraphs using advanced clustering techniques and then generating block diagonal matrices. These matrices capture community-level graph structures, allowing the model to understand and represent the graph at different levels effectively.

3) **Graph Reconstruction Module:** The final stage involves combining all synthetic coarse graphs created in the previous stage to generate the final graph. This process uses a linear reconstruction function that preserves the hierarchical structures at different granularity levels, ensuring the scalability and efficiency of the model.

3.5.3 Cycle Consistency

Cycle consistency is integral to the Misc-GAN architecture. It ensures the graph generation process is reliable and the reconstructed graph maintains the original graph's essential properties. This concept is borrowed from Cycle-Consistent Adversarial Networks, where a generated graph is expected to be transformed back to its original form, thus ensuring that the generative process preserves critical graph properties.

3.5.4 Multi-Scale Analysis

The multi-scale analysis in Misc-GAN allows the model to capture and process graph data at multiple resolutions. This approach is beneficial for handling large-scale graphs that have numerous representations, making the model versatile and applicable to various types of graph data. The multi-scale approach also aids in reducing computational complexity while preserving important topological features of the graph.

3.5.5 Cycle Gan

(1) This concept relies on GAN but duplicated to achieve more consistent results and better mapping from each domain to the other.

(2) We use 2 discriminators and 2 generators: D_x, D_y, $G: X \to Y$ and $F: X \to Y$ (while X and Y are two domains that depict the source graph and the generated graph domains).

(3) Each set of discriminators and generators transfer from one domain to the other, very similarly to GAN.

To view this concept in action, refer to the next section.

3.5.6 A Generic Joint Learning Framework

The framework is presented as a generic optimization problem with a multi-objective function. This function comprises four key components: multi-scale reconstruction loss, forward adversarial loss, backward adversarial loss, and cycle consistency loss (cycleGAN).

$$L = L_{ms} + L_F + L_B + L_{cyc}$$

$$= KL\left(\sum_{l=1}^{L} w^{(l)} F^{(l)}(G_s^{(l)}) + b, G_t\right)$$

$$+ \alpha \sum_{l=1}^{L} E_{G_t^{(l)} - P_{data}(G_t^{(l)})}[log D_F^{(l)}(G_t^{(l)})] + E_{G_s^{(l)} - P_{data}(G_s^{(l)})}[log(1 - D_F^{(l)}(F^{(l)}(G_s^{(l)})))]$$

$$+ \beta \sum_{l=1}^{L} E_{G_s^{(l)} - P_{data}(G_s^{(l)})}[log D_B^{(l)}(G_s^{(l)})] + E_{G_t^{(l)} - P_{data}(G_t^{(l)})}[log(1 - D_B^{(l)}(B^{(l)}(G_t^{(l)})))]$$

$$+ \gamma \sum_{l=1}^{L} E_{G_s^{(l)} - P_{data}(G_s^{(l)})}[||B^{(l)}(F^{(l)}(G_s^{(l)})) - G_s^{(l)}||_1] + E_{G_t^{(l)} - P_{data}(G_t^{(l)})}[||F^{(l)}(B^{(l)}(G_t^{(l)})) - G_t^{(l)})||_1]$$

Multi-Scale Reconstruction Loss (L_{ms}): This term is designed to minimize the divergence between the target graph (G_t) and the generated graph ($G_t^{\sim}$). It uses a Kullback-Leibler (KL) divergence measure to compare two graphs. The divergence is calculated between the adjacency matrices of G_t and $G_t^{\sim}$.

Forward Adversarial Loss (L_F): This component learns a forward mapping function from the source graph (G_s) to the target graph (G_t). It involves a discriminator ($D_F^{(l)}$) that distinguishes whether a graph is from the target domain or generated by the model.

Backward Adversarial Loss (L_B): Similar to L_F, but in reverse. It defines an adversarial loss that learns the mapping from the target domain to the source domain, with its discriminator ($D_B^{(l)}$).

Cycle Consistency Loss (L_{cyc}): This term ensures that the mapping functions do not contradict each other by maintaining consistency in both directions of the mapping.
Overall Objective: The overall objective (L) is a sum of these four terms, with positive constants (α, β, γ) balancing their impact. The model aims to optimize this objective function using a min-max game typical of GAN frameworks.

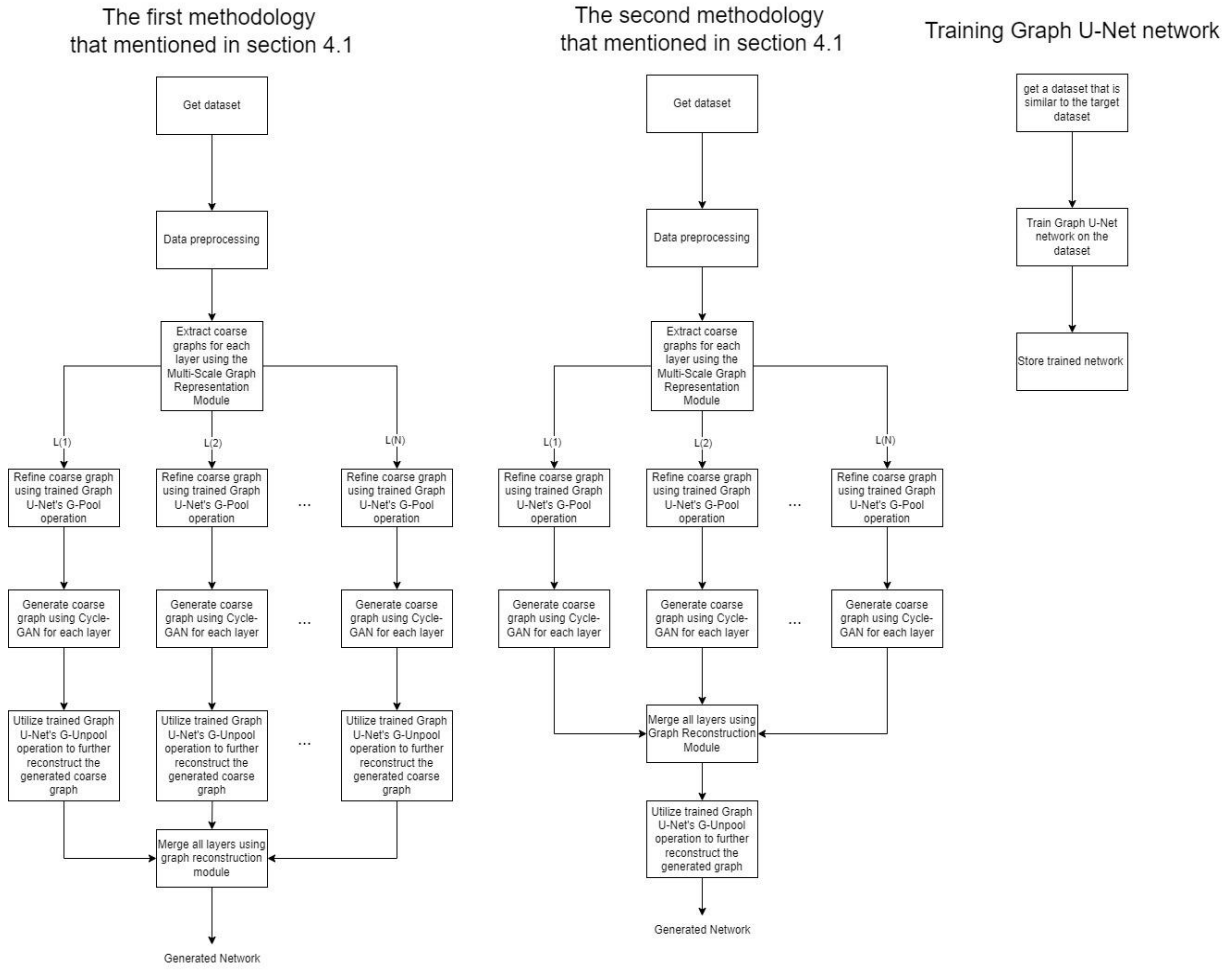
4. The Model

4.1 The Model Overview

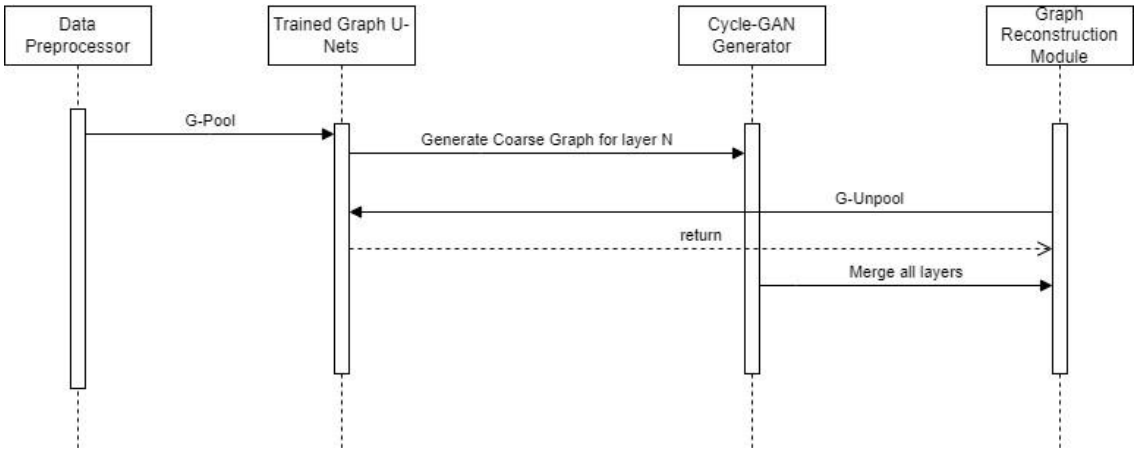
After finding a dataset for generating similar graphs, another dataset resembling the first one is selected for training a Graph U-Net network. Once the training is completed, the Graph U-Net network is saved for future use.
The main model flow proceeds as follows:

1. The original dataset is fed into the data preprocessing module.
2. Coarse graphs corresponding to each granularity layer are derived utilizing the Multi-Scale Graph Representation Module.
3. Subsequently, for each coarse graph obtained, the trained Graph U-Net is deployed by using the G-Pool operation to refine the feature set.
4. The Cycle-GAN is applied to generate coarse graphs that mimic the original coarse graphs for each respective layer.
5. Following the generation of all coarse graphs, two methodologies are evaluated:
    a. The trained network employs the G-Unpool operation on each synthetic coarse graph to replicate the structure of the original graph more accurately using node features, followed by the integration of all final coarse graphs into a final synthetic graph using the Graph Reconstruction Module.
    b. The Graph Reconstruction Module merges all synthetic coarse graphs to form a comprehensive synthetic graph, upon which the trained network's G-Unpool operation is executed to reconstruct the structure of the original graph more faithfully using node features.

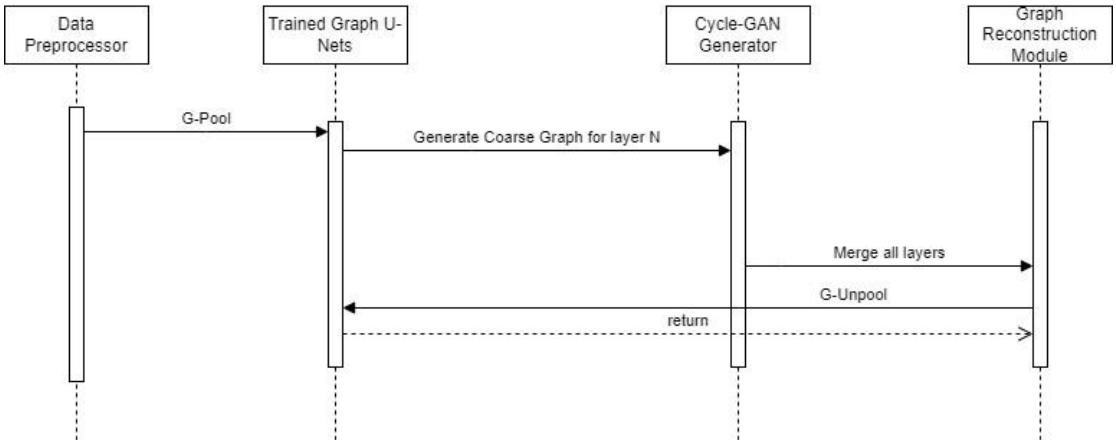
The following Flow Chart displays both flows:



The following Sequence Diagrams represents the Main Model's workflow:
The first methodology that mentioned in section 4.1:



The second methodology that mentioned in section 4.1:



4.2 Training the Graph U-Net Network

A dataset similar to the target dataset is utilized.
The found dataset should contain nodes, edges and features.
This dataset is then fed into the Graph U-Net network for training purposes.
Upon completion of training, the trained network is stored in a suitable format for subsequent use within the main model flow (see section 4.4).

4.3 Data Preprocessing

The data comprises two files: nodes and edges, and a file containing the features of each node. Initially, the nodes and edges file is used. Isolated nodes, which do not contribute to the graph's structure, are eliminated. Subsequently, an adjacency matrix is generated, and various processing modules (such as AMG and DBSCAN) are employed to extract information from it.

4.4 Extracting Coarse Graphs Using Graph U-Nets

The objective is to extract coarse graphs for each granularity layer based on node features. This adjustment represents a fundamental modification to the original MISC-GAN model, as described in the proposed solution.
The rationale for this alteration is the assumption that focusing on node features to refine coarse graphs can potentially lead to enhanced outcomes compared to focusing solely on the structural aspects of the graph.
The proposed method involves using the Multi-Scale Graph Representation Module to obtain the required coarse graphs for subsequent processes.
The G-Pool operation within the trained network, discussed in section 4.2, is then utilized to further refine each coarse graph of each granularity layer. This refinement focuses on extracting the most valuable features while maintaining the integrity of the coarse graph structure.

4.5 Generate coarse graphs of each layer using Cycle-Gan

During this phase, the model operates at each granularity layer to generate synthetic coarse graphs for individual layers.

Comprehending the structure of a graph, particularly when it involves numerous nodes, presents a challenge due to the exponential number of possible representations. To enhance understanding, the model employs coarse graphs for each granularity layer, offering improved time complexity and outcomes.

Utilizing these subgraphs, block diagonal matrices are constructed by rearranging community blocks along the diagonal. These matrices effectively encapsulate the graph's structure at the community level.

The resultant block diagonal matrices are subsequently utilized to generate synthetic graphs for each layer.

4.6 Reconstructing the Final Network Using the Generated Coarse Graphs

In the final phase of the model, two approaches are considered regarding the sequence of operations and the integration of the trained network with Misc-GAN's Graph Reconstruction Module.

4.6.1 Unpooling Each Synthetic Coarse Graph and Merging

During this phase, the trained network's G-Unpool operation is applied to each synthetic coarse graph produced earlier to more closely replicate the structure of the original graph. Subsequently, all synthetic coarse graphs are processed through the Graph Reconstruction Module to combine them into a unified synthetic graph.

The reconstruction of each layer utilizes G˜t, a linear function that includes weights (w) and biases (b), providing users the ability to adjust the significance of each layer within the graph.

This method effectively preserves hierarchical structures across different granularity layers, offering both scalability and efficiency while considering important node features.

4.6.2 Merging All Synthetic Coarse Graphs and G-Unpooling

During this phase, all synthetic coarse graphs created in the previous stage are fed into the Graph Reconstruction Module to form a unified synthetic graph, as detailed in section 4.6.1. As a final step, the G-Unpool operation of the trained network is applied to the synthetic graph to reconstruct it better using the node features.

5. Verification Plan

We will run the following tests to evaluate the model performance and:

Test Number	Test Subject	Expected Result
1	args.which_stage = 'A' Instead of “training” or “testing”.	Our proposed model should throw an exception.
2	Change dataset_A to a path that doesn’t exist.	Our proposed model should throw an exception while trying to load the data.
3	Change starting_layer to be greater than the number of layers (parameter layer)	Our proposed model should throw an exception before starting to train.
4	Change args.epoch = -10	Our proposed model should throw an exception before starting to train.
5	Run the model with the default arguments	Our proposed model should run with default values in the training and testing.
6.	args.gpu toggled between True and False	Our proposed model should verify GPU usage based on the given value.
7.	Generate synthetic network based on the given input	Our proposed model should match the expected network structure.
8.	Save all files in output directory using the args.filename input	Our proposed model should create the directories and save the files correctly.

6. Conclusion

In this capstone project, a new graph representation learning model is introduced. It combines Misc-GAN with Graph U-Nets, thereby potentially enhancing the original Misc-GAN model's ability to generate complex graph structures with greater efficiency and precision. This integration leverages the strengths of both architectures, possibly improving the model's performance in handling graph-structured data across multiple granularity levels.

The synergy between Misc-GAN's generative capabilities and Graph U-Nets' ability to capture hierarchical and topological features has sparked exciting new ideas and possibilities in the field. The model is employed to generate synthetic graphs that aim to resemble real-world graph structures, demonstrating the potential of merging different neural network architectures to tackle complex issues in graph generation and analysis.

In the next phase of the research, the aim is to identify the optimal method for executing the final step of the model. Each of the two methodologies will be evaluated to capture the best possible model configuration.

Looking ahead, the implications of this project extend beyond its immediate contributions. This work establishes a solid foundation for future research, potentially leading to new methods and applications in graph generative models.

7. References

1. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).
2. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2019). Graph neural networks: A review of methods and applications. AI Open, 1, 57-81.
3. Gao, H., & Ji, S. (2019). Graph U-Nets. In Proceedings of the 36th International Conference on Machine Learning (ICML 2019) (Vol. 97, pp. 2083-2092).
4. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In 5th International Conference on Learning Representations (ICLR 2017).
5. Zhang, Z., Cui, P., & Zhu, W. (2020). Deep learning on graphs: A survey. IEEE Transactions on Knowledge and Data Engineering, 32(8), 1485-1509.
6. Zhu, J., Zhou, Y., Ye, J., Xu, Q., & Wang, J. (2019). Graph convolutional networks for graph-structured data: Methods, systems, and applications. ACM Computing Surveys (CSUR), 52(6), 1-35.
7. Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1125-1134).
8. Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: Going beyond Euclidean data. IEEE Signal Processing Magazine, 34(4), 18-42.
9. Zhou, D., Zheng, L., Xu, J., & He, J. (2019). Misc-GAN: A Multi-scale Generative Model for Graphs. Frontiers in Big Data, 2:3. doi: 10.3389/fdata.2019.00003