



Software Engineering Department
Braude College

Capstone Project Phase A 61998

Generating Multi-scale Graphs with Graph U-Net

Project code:24-1-R-9

Supervisor: Prof. Zeev Volkovich
Advisor: Dr. Renata Avros

Shahar Vachiler – shaharvac95@gmail.com
Shachar Dalal - drgwgwmd@gmail.com

Table of Contents

Abstract	3
1. Introduction and Background	3
2. Related Work.....	4
2.1 Convolutional Neural Networks (CNNs)	4
2.1.1 Max-pooling	4
2.1.2 Activation Function	4
2.2 Graph Convolutional Networks (GCNs)	6
2.5 U-Nets	7
2.6 Graph U-Nets.....	8
2.7 Generative adversarial network (GANs).....	12
2.8 Misc-GAN	13
2.9 Diffusion models	14
2.9.1 Diffusion models for graph generation.....	14
3. Expected Achievements	15
4. Process	16
4.1 Pre-processing.....	16
4.2 Multi-Scale Graph Representation Module	16
4.2.1 Graph U-net input	16
4.2.2 The Graph U-net process.....	17
4.3 Graph Generation Module	17
4.3.1 Generate Coarsen Graphs	17
4.4 Graph Reconstruction Module.....	18
5. Product	19
6. Testing and Evaluation	20
6.1 Testing Plan	21
References	22

Abstract

This research investigates advanced techniques in hierarchical representation learning based on graphs, with the aim of addressing the challenges associated with capturing hierarchical relationships within graph structures. By employing the Graph U-Net (GU-Net) model and integrating a diffusion model into a Misc-GAN architecture, the project endeavors to automate the generation of multi-scale graphs to enhance the representation of complex relationships across diverse datasets.

The incorporation of GU-Net and diffusion models offers a unique approach to capturing intricate patterns within graph structures, enabling more effective representation learning. Through thorough testing and validation, the project seeks to establish the proposed model as a state-of-the-art solution for multi scale graph generation, with the objective of achieving greater efficiency and accuracy compared to previous methods.

Keywords: Graph U-Net, Graph Generation, Multi-scale Graphs, Graph neural network, GAN

Github link: <https://github.com/shaharvac95/Capstone-project-2024-generating-multi-scale-graphs>

1. Introduction and Background

Deep learning, a powerful artificial intelligence methodology, enables computers to derive insights from data similar to the human brain functioning [7] . This technique equips machines to perform intricate tasks and provide accurate predictions, finding widespread applications in domains such as image and speech recognition, autonomous vehicles, and medical diagnostics.

One area where deep learning holds significant potential is in graph-based hierarchical representation learning [2] . Traditionally, generating multi-scale graphs has been challenging due to the inherent complexity of capturing hierarchical relationships within graph structures. Existing methods may lack efficiency and struggle to provide a comprehensive representation of multi-scale features [1].

This project aims to address these limitations by leveraging the capabilities of the Graph U-Net (GU-Net) model and a diffusion model to automate the generation of multi-scale graphs, offering a superior solution for understanding complex relationships across diverse datasets [1][2] .

Graph-based representation learning, which is closely tied to deep learning techniques, poses challenges due to the complexity of hierarchical relationships within graph structures. By applying deep learning methodologies, such as the GU-Net model and diffusion model, to the domain of graph-based hierarchical representation learning, this project seeks to overcome these challenges and provide a more efficient and comprehensive approach to capturing multi-scale features and hierarchical relationships within graph structures [2] .

This advancement could potentially enhance the performance of various applications that rely on graph-based representations, ultimately benefiting from the powerful insights derived through deep learning techniques [7] .

2. Related Work

Existing research in graph-based representation learning has explored various techniques, but challenges persist in capturing hierarchical relationships effectively [2] [5] . Methods such as GAN [6] and Graph U-Net [2] separately have shown promise but still lack efficiency and comprehensive representation of multi-scale features [1]. The proposed project aims to build upon this foundation by integrating the Graph U-Net model and a diffusion model within a Misc-GAN framework, offering a novel solution for automated multi-scale graph generation [1][2] .

2.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are specialized architectures meticulously designed for processing grid-like data, particularly images [7] . By leveraging convolutional layers, CNNs adeptly capture hierarchical features and intricate spatial relationships within images, facilitating their superior performance across various computer vision tasks. From precise image classification to nuanced object detection and seamless segmentation, CNNs have become indispensable tools in modern AI applications. Their innate capability to autonomously discern and learn intricate features directly from raw data has catalyzed revolutionary breakthroughs in fields such as healthcare, robotics, and autonomous vehicles [5] .

2.1.1 Max-pooling

Max-pooling is a technique utilized in CNNs to downsample feature maps, reducing their spatial dimensions while retaining important information [5] . It operates by partitioning the input feature map into disjoint windows and selecting the maximum value from each window to form the output feature map. This process effectively reduces computational complexity and memory requirements while preserving the most relevant features for subsequent layers. Max-pooling plays a crucial role in hierarchical feature extraction, enabling CNNs to capture and emphasize significant spatial patterns across multiple scales efficiently.

2.1.2 Activation Function

In neural networks, the activation function determines whether the neuron is activated by performing calculations on a weighted sum with an attached bias [9] . Its purpose is to introduce nonlinearity into the neuron's output, enabling the network to handle complex tasks effectively.

The identity activation function is a straightforward linear function that outputs the same value as its input [9] . It does not introduce nonlinearity into the network but can be useful in scenarios where linear transformations are desired. While it does not add additional complexity, it still plays a role in the overall behavior of the network.

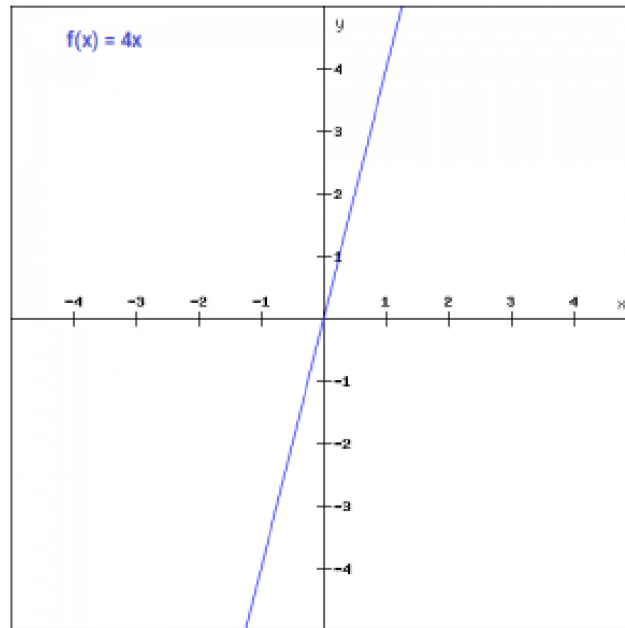


Figure 1: Identity (sometime also called Linear) Activation function [9] .

The softmax function takes a vector of real values, positive, negative, or zero, and transforms them into a vector of real values between 0 and 1 [9] . This transformation allows them to be interpreted as probabilities, with the sum of vector values being 1. Small or negative inputs are turned into low probabilities, while large inputs become high probabilities.

The sigmoid function takes a real value as input and yields an output value in the range between 0 and 1 [9] . It maps small values closer to zero and large values nearer to 1. Commonly used in the output layer of binary classification tasks, where classes are represented by zero and one. Data is classified as 1 if the sigmoid output is greater than 0.5 and as 0 otherwise.

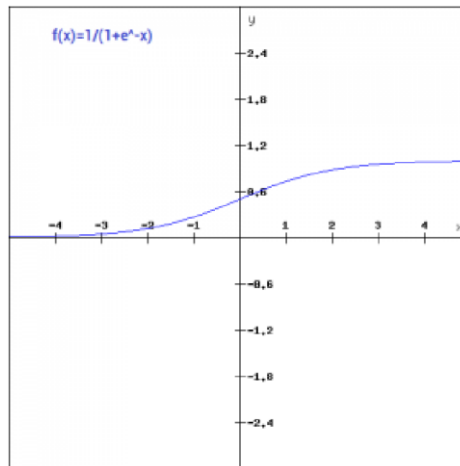


Figure 2: Sigmoid Activation function [9] .

The choice between using softmax and sigmoid in the final layer of a neural network depends on the nature of the problem and task requirements [9] . Softmax is well-suited for scenarios with multiple classes, transforming logits into a probability distribution over the classes. It allows the network to make definitive choices among classes, assuming each input belongs to only one class.

Softmax also encourages global information sharing among classes, capturing complex relationships. On the other hand, sigmoid is commonly used in binary classification tasks, providing independent probabilities for each class. It is preferred in tasks where an input can belong to multiple classes simultaneously, offering flexibility with decision thresholds. Softmax is chosen because it fits the model needs for multi-class classification.

2.2 Graph Convolutional Networks (GCNs)

Graph Convolutional Networks (GCNs) are neural networks designed for graph data [4] . They capture relationships between nodes in a graph, learning features directly from the graph's structure. GCNs use convolutional layers adapted for graphs, enabling them to handle tasks like node classification, link prediction, and graph clustering. They are widely used in social network analysis, bioinformatics, and recommendation systems.

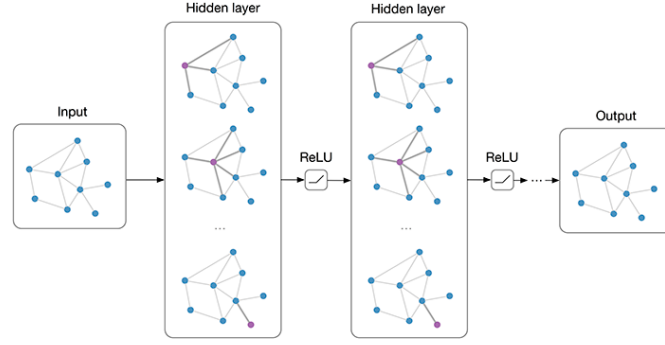


Figure 3: Multi-layer Graph Convolutional Network (GCN) with first-order filters [4] .

2.5 U-Nets

U-Net is a type of CNN designed for fast and precise segmentation of images. It has an encoder-decoder structure:

Encoder: This component captures the context in the image. It consists of a stack of convolutional layers, each followed by a max pooling layer. As we move deeper into the network, the spatial dimensions (width and height) decrease while the depth (number of feature maps) increases [5] .

Decoder: Utilizing the high-level contextual information captured by the encoder, the decoder reconstructs the input image detail. It comprises a stack of transposed convolution layers (also known as deconvolution), each followed by a concatenation with the correspondingly cropped feature map from the encoder and further convolutions. Additionally, there are skip connections between the encoder and decoder. The objective of these skip connections in the U-Net model is to help recover the fine-grained details lost during the encoding (downsampling) process [5] .

During the encoding stage, the spatial resolution of the input decreases while the semantic complexity increases. However, this process may lead to the loss of some local, detailed information. When the decoder upsamples the low-resolution encoded features, it may not fully recover the original details. Skip connections address this issue by directly forwarding the feature maps from the encoder to the corresponding decoder. This provides local, detailed information to the decoder, aiding it in better reconstructing the original input, especially in tasks like image segmentation where pixel-level detail is important [5] .

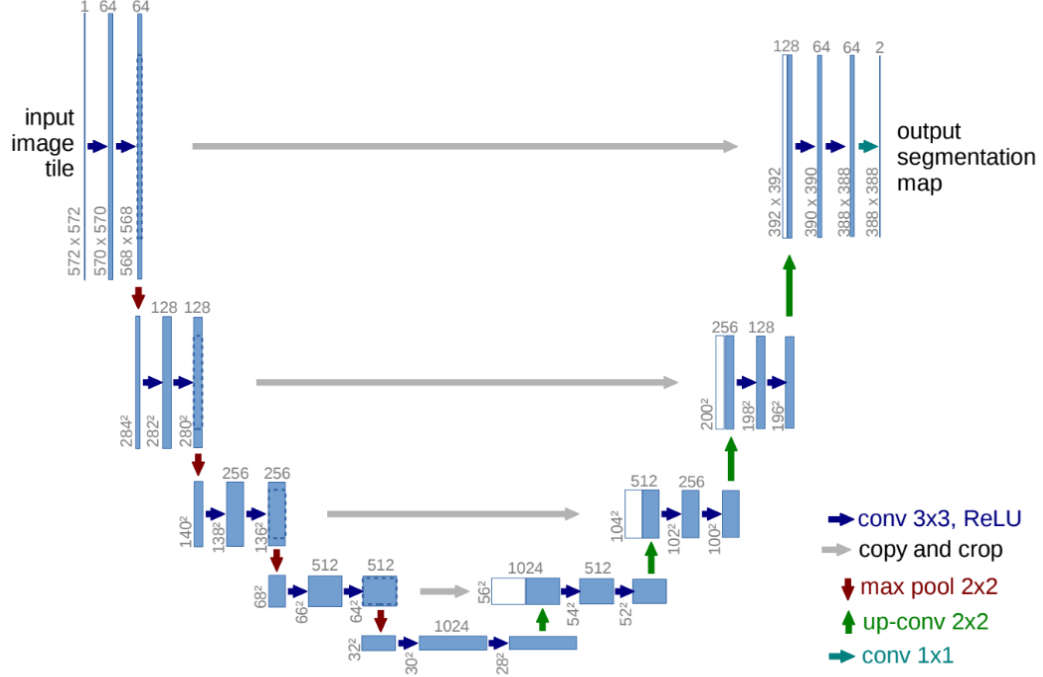


Figure 4: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations [5].

2.6 Graph U-Nets

Graph U-Nets represent a specialized architecture tailored for efficient and accurate graph-based hierarchical representation learning [2]. Similar to U-Nets designed for image segmentation, Graph U-Nets adopt an encoder-decoder structure to capture intricate graph structures and reconstruct the original graph detail.

Two key operations that enable the model to learn hierarchical representations of graphs and reconstruct the original graph structure are Graph Pooling (gPool) and Unpooling (gUnpool) [2]. These operations, similar to the encoder-decoder structure of U-Nets, facilitate learning hierarchical representations of graphs and reconstructing the original graph structure.

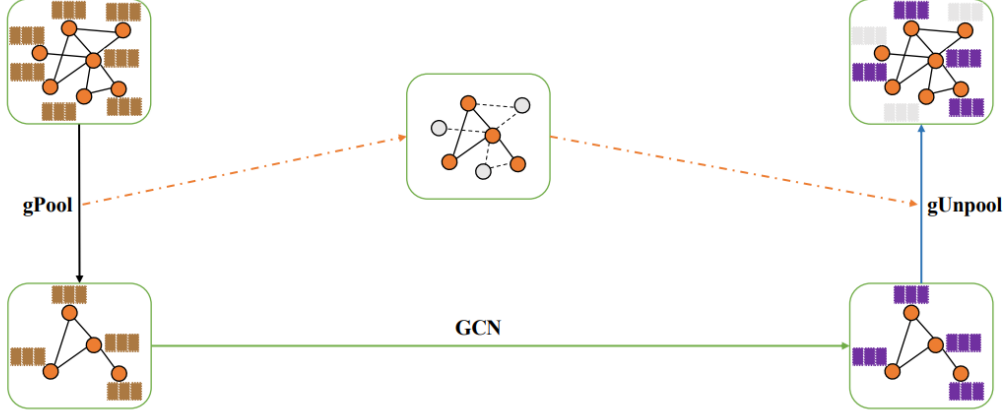


Figure 5: The architecture of gPool and gUnpool in the Graph U-net model [2] .

gPool operation is introduced for down-sampling graph data [2] . In each convolutional layer, a subset of nodes is adaptively selected to form a smaller graph. This selection process is guided by a trainable projection vector, p . By projecting all node features onto a 1D space, k-max pooling is performed for node selection. The selection of nodes is determined by the scalar projection values on p , ensuring consistency in connectivity across nodes in the new graph. Mathematically, the projection formula is represented as:

$$y_i = \frac{x_i * p}{\|p\|}$$

Where i represents a given node, x_i denotes the node feature vector, p signifies the projection vector and y_i measures how much information of node i can be retained when projected onto the direction of p . The new graph, represented by matrices A^φ (represents the adjacency matrix) and X^φ (represents the feature matrix), is obtained by selecting the k nodes with the largest scalar projection values on p .

Further, a gate operation controls the information flow by applying a sigmoid function to each element in the scalar projection vector. The element-wise matrix product of X' and the gate vector Y' governs the information of the selected nodes, resulting in the new graph with the most significant features while minimizing information loss.

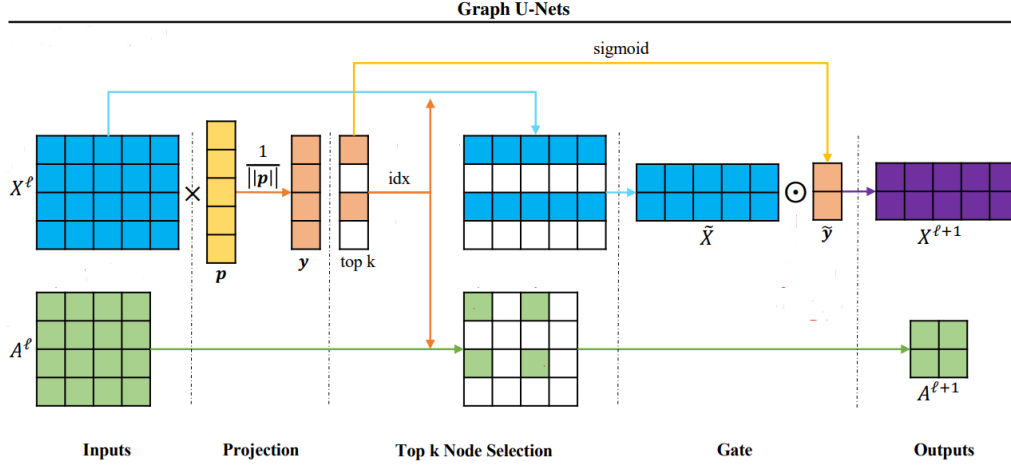


Figure 6: The proposed graph pooling layer operates on a graph with 4 nodes, each having 5 features. It starts by creating an adjacency matrix $A^l \in \mathbb{R}^{4 \times 4}$ and an input feature matrix $X^l \in \mathbb{R}^{4 \times 5}$ for layer l . In the projection stage, a trainable projection vector $p \in \mathbb{R}^5$ is used to estimate scalar projection values for each node, resulting in a score vector y . The top 2 nodes ($k=2$) with the highest scores are selected, and their indices are used to form a new graph with a pooled feature map \tilde{X}^l and a new adjacency matrix A^{l+1} . In the gate stage, element-wise multiplication is performed between \tilde{X}^l and the selected node scores vector \tilde{y} , resulting in X^{l+1} . The graph pooling layer outputs A^{l+1} and X^{l+1} [2].

In contrast to the gPool layer, the gUnpooling layer serves to restore the graph to its original structure, enabling up-sampling operations on graph data [2]. This layer operates in tandem with the gPool layer, utilizing the information recorded during node selection in the gPool layer to reconstruct the original graph. The layer-wise propagation rule of the gUnpool is expressed as:

$$X^{l+1} = \text{distribute}(0_{N \times C}, X^l, \text{idx})$$

Where X^{l+1} denotes the feature matrix of the restored graph X^l represents the feature matrix of the current graph $0_{N \times C}$ is the initially empty feature matrix for the new graph, with dimensions identical to the original graph but filled with zeros and idx contains the indices of the selected nodes in the corresponding gPool layer, which reduced the graph size from N nodes to K nodes.

The distribution process involves filling the initially empty feature matrix $0_{N \times C}$ by placing the feature vectors from the current graph X^l into their corresponding positions based on the indices stored in idx . In the restored graph X^{l+1} , the row vectors corresponding to the selected nodes are updated with the corresponding row vectors from X^l , while the row vectors corresponding to nodes not selected remain zero.

The Graph U-Nets architecture begins with the application of a graph embedding layer, aiming to convert nodes into low-dimensional representations. This step is particularly crucial since some datasets utilize high-dimensional feature vectors. Following this, the encoder is constructed by stacking multiple encoding blocks, with each block comprising a GCN layer and a gPool layer.

The gPool layer plays a pivotal role in reducing the size of the graph to encode higher-order features, while the GCN layers are responsible for aggregating information from each nodes first-order neighbors. On the other hand, the decoder is designed with the

same number of decoding blocks as the encoder. Each decoding block consists of a GCN layer and a gUnpooling layer.

The gUnpooling layer serves to restore the graph to its higher-resolution structure, while the GCN layer continues to aggregate information from the neighborhood.

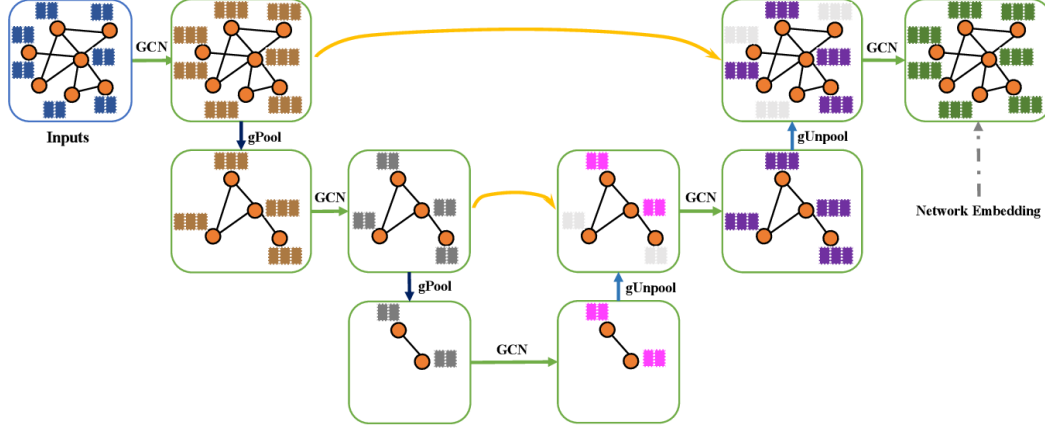


Figure 7: Architecture of G-U-net [2] .

Additionally, the architecture similar to U-nets Incorporating skip connections between the encoder and decoder, Graph U-Nets aim to mitigate information loss during the encoding process [2] . These connections facilitate the direct transmission of feature maps from the encoder to the corresponding decoder layers, providing local, detailed information essential for accurate graph reconstruction.

By preserving fine-grained details that may be lost during downsampling, skip connections enhance the decoder ability to reconstruct the original input graph, particularly in tasks requiring pixel-level accuracy. Finally, a GCN layer is employed for final predictions before applying the soft max function.

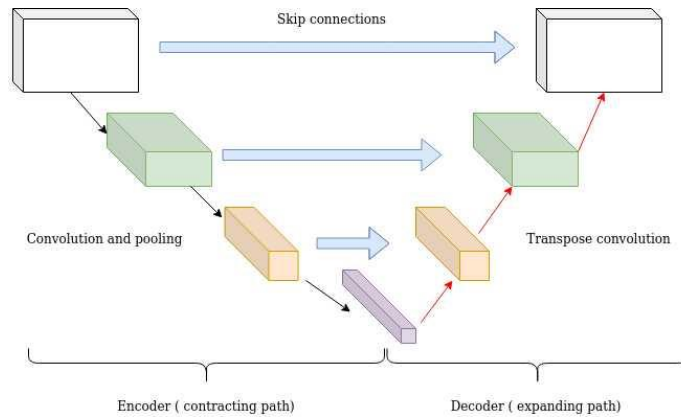


Figure 8: Macro view on the Graph U-net architecture focusing on the skip connection [11].

2.7 Generative adversarial network (GANs)

Generative Adversarial Networks (GANs) are powerful frameworks used for generating synthetic data samples. Comprising a generator responsible for creating synthetic data and a discriminator trained to differentiate between real and generated data, GANs engage in a competitive training process known as adversarial learning. This methodology has propelled their application in diverse fields such as image generation, style transfer, and data augmentation, where they exhibit exceptional performance.

The proficiency of GANs in producing realistic and diverse data has profound implications for content creation, data privacy preservation, and simulation tasks, thereby revolutionizing various industries. Additionally, GANs demonstrate a remarkable capacity to learn intricate data distributions and generate novel instances, offering significant potential for advancing machine learning tasks and addressing real-world challenges effectively. The continuous evolution and refinement of GANs underscore their pivotal role in propelling innovation and expanding the horizons of generative modeling techniques.

A GAN consists of a generator and a discriminator, where each one of the mentioned is a neural network [6]. This network is usually trained using adversarial training. The generator G creates synthetic data samples. It aims to generate data that is indistinguishable from real data by the discriminator. The generator function $G(z)$ takes random noise z as input and outputs synthetic data samples. Mathematically, the generator can be represented as: Generated Data = $G(z)$. The discriminator D is trained to differentiate between real and generated data. It provides feedback to the generator by indicating how convincing its generated samples are.

The discriminator function $D(x)$ takes data samples x as input and outputs the probability that the sample is real. Mathematically, the discriminator can be represented as: Discriminator Output = $D(x)$ Where x is a data sample, and $D(x)$ represents the probability that x is real [6]. The discriminator D aims to maximize the probability of assigning the correct label to both the real samples and the faked samples generated by the generator G , while the generator G aims to minimize the probability that the discriminator D successfully distinguishes the faked samples from the real samples. The objective of this min-max game [6] is written as:

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

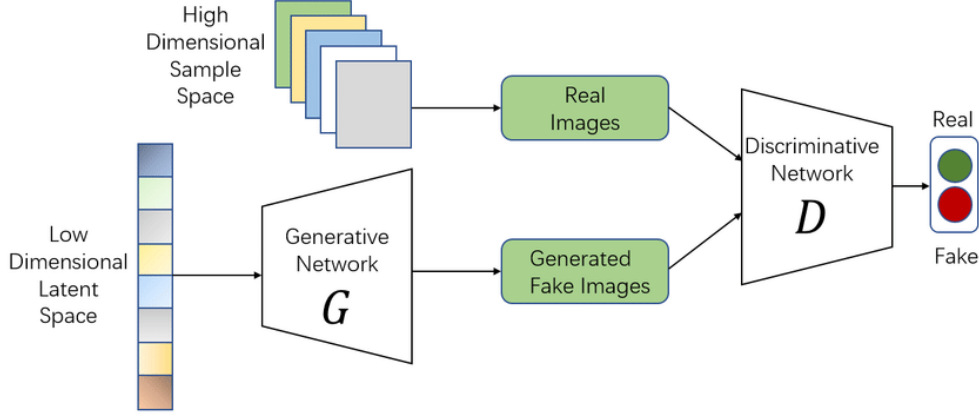


Figure 9: The architecture of a basic GAN [6] .

2.8 Misc-GAN

Misc-GAN framework comprises three stages: the Multi-Scale Graph Representation Module, the Graph Generation Module, and the Graph Reconstruction Module [1]. The Multi-Scale Graph Representation Module, a pivotal element of the Misc-GAN framework, explores hierarchical cluster-within-cluster structures to characterize input graphs. Employing methodologies like hierarchical clustering and algebraic multigrid, this module constructs coarse graphs at various granularity levels to capture the complex organization of graphs [1]. The construction of the coarser graph $G_t^{(1)}$ at the first layer is defined by the following Equation:

$$G_t^{(1)} = P^{(1)'} G_t P^{(1)}$$

Here, $P^{(1)}$ represents the coarsening operator, responsible for generating the coarse graph at the initial layer. This equation outlines the process of generating the coarse graph at the initial layer. For the recursive construction of a multi-scale hierarchy of increasingly coarser graphs at the l -th layer, where $l = 1, \dots, L$ the following Equation is employed:

$$G_t^{(l)} = P^{(l-1)'} \dots P^{(1)'} G_t P^{(1)} \dots P^{(l-1)}$$

These equations explain how coarse graphs are systematically generated at different granularity levels. Moving on to the Graph Generation Module, it serves as another critical component of the Misc-GAN framework. This module is tasked with generating new graphs while upholding the hierarchical structure distribution observed in the target graph. By leveraging deep models and generative adversarial networks, it acquires characteristic topological features to effectively model the complex joint probability of nodes and edges.

The objective is to produce new graphs that mirror the hierarchical structures present in the input graphs [1]. In the Graph Reconstruction Module, which constitutes the final stage of the framework, the graph is reconstructed while preserving significant local structures captured in the Multi-Scale Graph Representation Module. The reconstruction of the fine graph $\tilde{G}_t^{(1)}$ at the first layer is accomplished through the following Equation:

$$\tilde{G}_t^{(1)} = R^{(1)'} G_t^{(2)} R^{(1)}$$

Here, $R^{(1)}$ represents the reconstruction operator responsible for mapping the coarser graph back to the fine graph at the initial layer. Similarly, the recursive construction of a multi-scale hierarchy of increasingly refined graphs at the l -th layer is delineated by the following Equation:

$$\tilde{G}_t^{(l)} = R^{(1)'} \dots R^{(l-1)'} G_t^{(l)} R^{(l-1)} \dots R^{(1)}$$

These modules collectively empower the Misc-GAN framework to model the underlying distribution of graph structures at varying granularity levels and generate new graphs that faithfully maintain these structures.

2.9 Diffusion models

Diffusion probabilistic models, also known as score-based generative models, are a subset of latent variable generative models extensively employed in the realm of machine learning [8]. These models leverage a unique methodology involving the introduction of noise to the original training data, a process named the diffusion process. Subsequently, the reverse diffusion process is harnessed to reconstruct the initial data from its perturbed state.

By mastering this reverse process, these models acquire the ability to generate novel data points. This iterative learning mechanism empowers the model to effectively eliminate noise, unveiling the underlying structure of a dataset by simulating the diffusion of data points across their latent space [8]. With broad applicability, diffusion probabilistic models serve diverse purposes including image denoising, inpainting, super-resolution, and image synthesis tasks.

Rooted in stochastic processes, this modeling paradigm integrates randomness through noise scheduling to enhance the fidelity of the generated data and capture intricate patterns within the dataset.

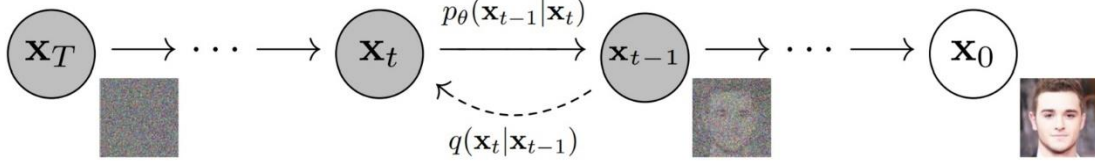


Figure 10: The main idea of diffusion model. By adding noise in the diffusion process the image became more cleaner and by imply reverse diffusion process the image return to the origin [10].

2.9.1 Diffusion models for graph generation

Diffusion models for graph generation represent a class of generative models designed to comprehend the underlying distribution of a given set of graphs and subsequently generate new graphs that adhere to the same distribution [8]. At their core, these models leverage the concept of the diffusion process.

A diffusion process in graph generation is a methodological framework where noise, in the form of nodes and edges, is systematically introduced in a controlled and

incremental manner during a forward process. This process starts with a basic graph, such as an empty one or a graph with a single node, representing the initial simplicity. The controlled addition of noise at each step mimics the stochastic nature of diffusion, gradually transforming the graph into a more complex structure.

This intentional introduction of noise reflects the essence of a diffusion process, capturing the step-by-step evolution of the graph through a forward process. The diffusion process has a reverse diffusion process aims to reverse this process and generate the original data [8] . The reverse diffusion process involves training a denoising network to recursively remove the noise that has been previously added by the forward process.

Instead of removing all noise in a single timestep, a denoising network is trained to iteratively remove the noise between two consecutive timesteps. This process moves backwards on the multi-step chain as the timestep decreases from T (the total number of timesteps) to 0.

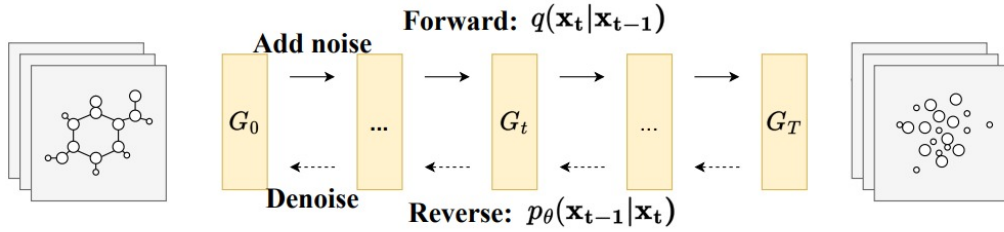


Figure 11: Diffusion model for graph generation [8] .

3. Expected Achievements

The project aims to explore a novel method of graph generation by employing the Graph U-Net and a distinctive diffusion model. The belief is that the approach pursued will surpass existing methods, with the anticipation that the model enhanced capacity to accurately capture intricate dependencies and distributions inherent in graph data will contribute to this improvement.

The effectiveness of the approach will be demonstrated through a series of testing and evaluations conducted across diverse datasets related to graph generation. These assessments aim to showcase the superior performance and effectiveness of the proposed approach.

4. Process

The project model is based on the Misc-GAN framework architecture [1]. Originally, Misc-GAN comprises three components: the Multi-Scale Graph Representation Module, the Graph Generation Module, and the Graph Reconstruction Module.

The first and third component are based on the Graph U-NET [2]. The second component will be implemented using a diffusion model [8] (to be selected at a later stage).

The model input is a graph with hierarchical structures, after pre-processing operations in the Multi-Scale Graph Representation Module the model coarsens the input into representation of different granularity levels, then for each level of granularity in the Graph Generation Module the model generates a corresponding coarsen graph.

Then the Graph Reconstruction Module reconstructs the generated graphs into an output target graph that retains the same composition of the original input graph. The following sections detail the adaptations made for integration into this model.

4.1 Pre-processing

The data gathered from various sources may not be sufficiently clean, necessitating the need for a pre-processing step to obtain the most relevant data possible. The model consistent pre-processing of noise removal, handle missing values outliers and inconsistencies, and normalize features.

The noise removal involves filtering out edges or nodes that do not contribute meaningfully to the overall structure. Handle missing values outliers and inconsistencies can also be achieved by removing unwanted data. Normalize features ensures that all features contribute equally to the analysis and prevents features with larger scales from dominating the learning process.

4.2 Multi-Scale Graph Representation Module

The Graph Representation Module is employed by a graph U-net model [1] that start after dataset partitioning, aiming to refine the input graphs by the encoder process through coarsening stages. This process adapts dynamically to varying graph complexities, facilitating the extraction of hierarchical features. Ultimately, the output of the encoder process is a presentation of the input graph coarsen by various granularity levels this output act as the input to the graph generation module.

4.2.1 Graph U-net input

The data obtained after the pre-processing step is divided into three distinct parts: Training, Validation, and Testing. Each part of the dataset comprises of graphs characterized by hierarchical structures. Every graph is outlined by two matrices: the adjacency matrix $A^\varphi \in \mathbb{R}^{N \times N}$ and the feature matrix $X^\varphi \in \mathbb{R}^{N \times C}$.

These matrices serve as the input for the Graph U-net incorporated into the model architecture. The Graph U-net is fed with two portions of the dataset: Training and Validation, while the Testing segment is reserved for subsequent analysis.

4.2.2 The Graph U-net process

Upon completion of the pre-processing steps and subsequent partitioning of the dataset into Training, Validation, and Testing sets, the Graph U-net process begins [2]. The model accepts a graph with hierarchical structures as its input, where individual nodes signify specific entities, and edges outline relationships between said entities. The complexity of these graphs may vary, necessitating adaptive processing by the model. The initial phase of the Graph U-net encoder process entails the coarsening of the input graph into representations of varying granularity levels. It is achieved by the gPool operation.

This coarsening procedure aims to streamline the graph while retaining its fundamental structural attributes. It enables the model to function across multiple abstraction levels, thereby facilitating the extraction of hierarchical features. At each granularity level, the Graph U-net generates a corresponding coarsened graph, contributing to holding the key features of the input graph without damaging or losing crucial data.

4.3 Graph Generation Module

The next step in the process is to transfer the coarsened graph representations from the Graph U-net output to the Graph Generation Module [1]. The Module consists of 3 steps where at the last step a diffusion model is applied.

This module facilitates the generation of target graphs from each coarsened graph at different granularity levels. As a result, the module generates various target graphs which are the base of the input of the Graph Reconstruction Module.

4.3.1 Generate Coarsen Graphs

The Graph Generation Module applies three steps for each coarsened graph representation. "The first step involves partitioning the graph into multiple non-overlapping subgraphs using state-of-the-art graph clustering methods. Then, based on the detected communities, it generates a set of block diagonal matrices by shuffling community blocks over the diagonals, which are used to characterize the community-level graph structures" [1].

In the last step a diffusion model is applied to generate the target graphs. This model adds noise in a controlled manner to the coarsened graphs, transforming them into a simple noise distribution through a series of steps. It then applies a reverse diffusion process, starting from the simple noise distribution and gradually removing the added noise. This process uses a learned distribution to guide the noise removal, resulting in graphs that closely resemble the original input graphs. The new generated graphs now pass to the Graph Reconstruction Module.

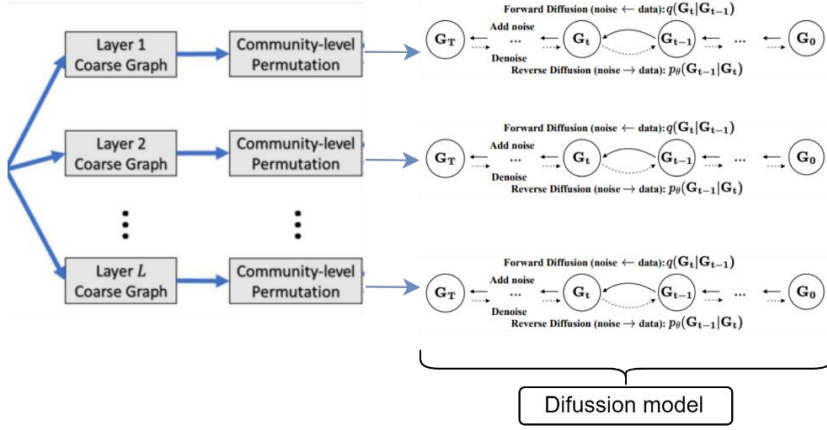


Figure 12: The 3 steps of the Graph Generation Module for each layer.

4.4 Graph Reconstruction Module

The Graph Reconstruction Module is employed by a GU-net model [[1],[2]] The process starts after the coarsened graphs generation operation in the generation module. The primary objective of the Graph Reconstruction Module is to reconstruct a graph close to the original input graph from its coarsened representations, effectively reversing the coarsening operations performed during the encoding phase. This is achieved through a combination of two key operations: gUnpool and skip connections.

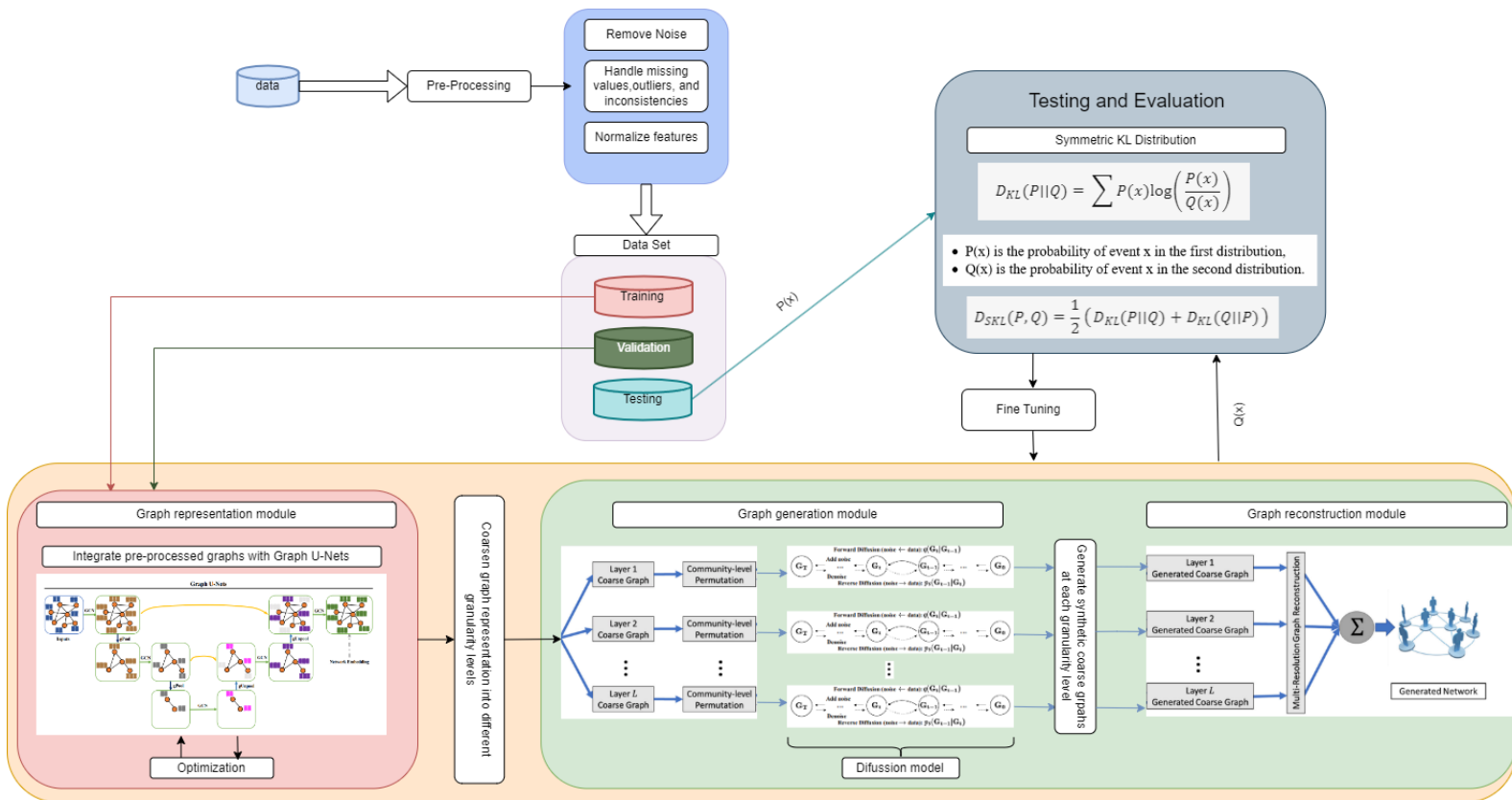
The gUnpool operation is responsible for repositioning nodes within the graph structure, leveraging the positional information accumulated during the corresponding gPool layer in the encoding phase. By tracking the locations of nodes selected for pooling, the gUnpool layer can effectively recover these nodes back to their original positions within the graph, preserving the overall structure and topology.

Complementing the gUnpool operation, skip connections play a vital role in retaining valuable features from the original input graph throughout the reconstruction process. These connections facilitate the direct transfer of information from the encoding phase to the decoding phase, bypassing the coarsening operations. The layer-wise propagation rule of gUnpool layers is defined in the previous section. This process specifies the precise manner in which nodes are repositioned within the graph structure, ensuring an accurate and consistent reconstruction process across multiple layers of the GU-Net model.

5. Product

The product is an interface that will implement the research model, using GU-net and diffusion model for the generation of multi-scale graphs. Below is the flow chart of the research model.

Multi-Scale Graph Generation



6. Testing and Evaluation

There are many methods available to test and evaluate models (e.g. Misc GAN [1]) like the one proposed in the project. Some of those methods are metrics used to examine the model efficiency: Average Degree (AD), Size of the Largest Connected Component (LCC), Exponent of the Power-Law Distribution (EPL), Gini Coefficient (GC), Symmetric Kullback-Leibler (KL) Divergence, and Graph Kernel Similarity.

For example, the Symmetric KL divergence is a measure of how one probability distribution diverges from a second expected probability distribution. It is used to quantify the difference between the actual and observed probability distribution. In the context of evaluating a generated graph against an original graph both graphs are represented as probability distributions.

This is done by treating the degree of each node as a random variable and creating a distribution of these degrees for each graph. The KL divergence between two distributions P and Q is calculated using the following formula:

$$D_{KL}(P \parallel Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Where $P(x)$ is the probability of event x in the first distribution, and $Q(x)$ is the probability of event x in the second distribution. The Symmetric KL Divergence is then calculated by taking the average of the KL divergence of P from Q and the KL divergence of Q from P:

$$D_{SKL}(P, Q) = \frac{1}{2} (D_{KL}(P \parallel Q) + D_{KL}(Q \parallel P)).$$

The result of this calculation represents the divergence between the new generated graph and the original graph. A value of 0 indicates a perfect match between the two graphs, while 1 indicates that the graphs are not identical at all.

In this project, the graphs from the testing dataset are compared to the graphs generated by the model. The goal is to achieve 85% similarity between the graphs. If the result is less than that, the model needs to be fine-tuned and testing continues.

6.1 Testing Plan

Test	Test Subject	Expected Result
1	Graph Generation	The model will generate multi-scale graphs accurately based on the input.
2	Graph Processing	The model will correctly process the generated graphs, capturing hierarchical relationships effectively.
3	Graph Output Validation	The model will produce valid output graphs that accurately represent the hierarchical relationships in the data.
4	Runtime Efficiency	The model will perform the graph generation and processing within an acceptable time frame.
5	Input Size/Graph Size	The model will be able to handle larger graphs without a significant decrease in performance.
6	Empty Graphs	If the input graph is empty, the model will return a specific error message indicating that the input graph cannot be empty. The model will not proceed with further processing until a valid graph is provided.
7	Incomplete Graphs	If the input graph is incomplete (missing nodes or edges), the model will either return an error message or fill in the missing information based on predefined rules or assumptions.
8	Incorrect Graphs	If the input graph is incorrect (contains invalid nodes or edges), the model will return an error message indicating the nature of the error. The model will not proceed with further processing until a valid graph is provided.
9	Incompatible Graphs	If the input graph is incompatible with the model (for example, the graph is too large or the structure is not supported), the model will return an error message indicating the incompatibility. The model will not proceed with further processing until a compatible graph is provided.
10	Generation Similarity Index	Similarity Index > 85%.

References

- [1] Zhou, D., Zheng, L., Xu, J., & He, J. (2019). Misc-GAN: A multi-scale generative model for graphs. *Frontiers in big Data*, 2, 3.
- [2] Gao, H., & Ji, S. (2019, May). Graph u-nets. In *international conference on machine learning* (pp. 2083-2092). PMLR.
- [3] Wang, Z., Zheng, H., He, P., Chen, W., & Zhou, M. (2022). Diffusion-gan: Training gans with diffusion. *arXiv preprint arXiv:2206.02262*.
- [4] <https://tkipf.github.io/graph-convolutional-networks/>
- [5] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III* 18 (pp. 234-241). Springer International Publishing.
- [6] Cai, L., Chen, Y., Cai, N., Cheng, W., & Wang, H. (2020). Utilizing amari-alpha divergence to stabilize the training of generative adversarial networks. *Entropy*, 22(4), 410.
- [7] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
- [8] Liu, C., Fan, W., Liu, Y., Li, J., Li, H., Liu, H., ... & Li, Q. (2023). Generative diffusion models on graphs: Methods and applications. *arXiv preprint arXiv:2302.02591*.
- [9] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, 6(12), 310-316.
- [10] <https://learnopencv.com/denoising-diffusion-probabilistic-models/>
- [11] <https://theaisummer.com/skip-connections/>

ChatGPT

Link: <https://chatgpt.com/>

Prompts: "Explain to me the concept of graph generation and the application of graph generation in real world".

"Can you help me create a Flowchart of every step i need to take in order to collaborate Graph U-Nets, Misc-GAN and Diffusion-GAN in my project?".

"Can you assure if this next sentence is in present simple and 3rd person view, if not correct it?".