תמונה שמכילה גופן, גרפיקה, טקסט, לוגו

התיאור נוצר באופן אוטומטי

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](mailto:shaharvac95@gmail.com)  
Shachar Dalal - [drgwgwmd@gmail.com](mailto:drgwgwmd@gmail.com)

**1. Introduction**

The project delves into the implementation of advanced techniques in graph-based hierarchical representation learning. Traditionally, generating multi-scale graphs encounters challenges 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. The research aims to overcome these limitations by leveraging the capabilities of the Graph U-Net model to automate the generation of multi-scale graphs, offering a superior solution for understanding complex relationships across diverse datasets. Through a variety of techniques employed in the project, high-quality graphs are endeavored to be produced. At the core of the project is the utilization of Graph U-Nets (GU-Nets), a groundbreaking advancement inspired by the success of U-Nets in image segmentation tasks [2]. This innovative architecture systematically represents graphs hierarchically while faithfully reconstructing their original structures. By strategically employing graph embedding, encoder blocks housing Graph Convolutional Network (GCN) layers, pooling, unpooling, skip connections, and final GCN layers for prediction, GU-Nets showcase a nuanced understanding of graph structures. Additionally, the integration of Misc-GAN presents a versatile approach to capturing graph structure distributions across various granularity levels [1]. Furthermore, the incorporation of the Diffusion model provides a robust solution, seamlessly integrating diffusion models to ensure stable training for graph generation tasks [3]. These advancements mark a significant transformation in hierarchical representation learning and graph modeling, ushering in substantial progress and innovation.

**2. Background and Related Work**

Deep learning, a potent artificial intelligence methodology, allows computers to glean insights from data akin to the human brain's functioning. This technique empowers machines to execute intricate tasks and deliver precise forecasts, finding wide-ranging applications in domains like image and speech recognition, autonomous vehicles, and medical diagnostics. Neural networks, comprising interconnected nodes organized in layers, are adept at discerning intricate patterns. Widely employed in diverse sectors such as image recognition, natural language processing, and speech recognition, these models offer exceptional adaptability and versatility. Their capacity to grasp complex data relationships has transformed industries like healthcare, finance, and autonomous driving. When considering deep learning and neural networks together, it becomes evident that deep learning leverages neural networks as a key component in its architecture. Deep learning utilizes neural networks to process and analyze data, enabling the extraction of meaningful insights and patterns. By integrating these two concepts, researchers and practitioners can harness the power of neural networks within the framework of deep learning to tackle complex problems across various disciplines effectively.

**2.1 Convolutional Neural Networks (CNNs)**

Convolutional Neural Networks (CNNs) are specialized architectures meticulously designed for processing grid-like data, particularly images. 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].

**2.2 Graph Convolutional Networks (GCNs)**

Graph Convolutional Networks (GCNs) are neural networks designed for graph data. 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’re widely used in social network analysis, bioinformatics, and recommendation systems [6].

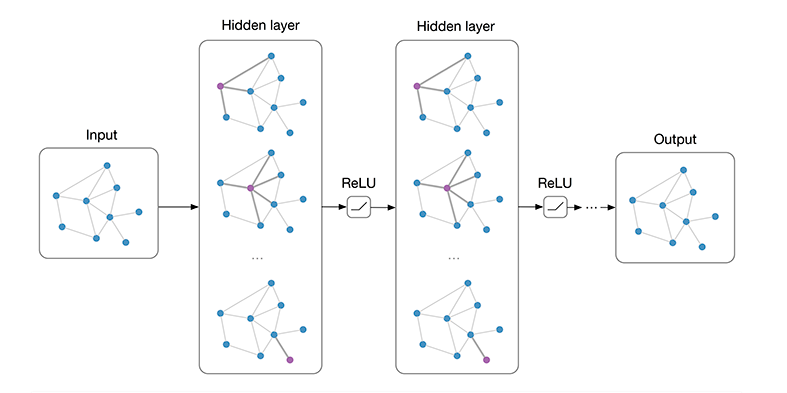


Figure 1: Multi-layer Graph Convolutional Network (GCN) with first-order filters [4].

**2.3 max-pooling**

Max-pooling is a technique utilized in convolutional neural networks (CNNs) to downsample feature maps, reducing their spatial dimensions while retaining important information. 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.

**2.4 Activation Function**

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

**2.4.1 Identity Activation Function**

The identity activation function is a straightforward linear function that outputs the same value as its input. 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.

תמונה שמכילה קו, תרשים, טקסט, עלילה

התיאור נוצר באופן אוטומטי

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

**2.4.2 Softmax Activation Function**

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. 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.

**2.4.3 Sigmoid Activation Function**

The sigmoid function takes a real value as input and yields an output value in the range between 0 and 1. 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.

תמונה שמכילה טקסט, קו, תרשים, צילום מסך

התיאור נוצר באופן אוטומטי

Figure X: Sigmoid Activation function [9].

**2.4.4 Sigmoid vs. Softmax**

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. 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's needs for multi-class classification.

**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- 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 dimension (width and height) decreases while the depth (number of feature maps) increases. Decoder- uses this high-level contextual information to reconstruct the input image detail. It consists of a stack of transposed convolution layers (sometimes called deconvolution), each followed by a concatenation with the correspondingly cropped feature map from the encoder and further convolutions. Also there are skip connection between the encoder and decoder the objective of those skip connections in the U-Net model is to help recover the fine-grained details that are lost during the encoding (downsampling) process. In the encoding stage, the spatial resolution of the input decreases while the semantic complexity increases. However, this process may lose some local, detailed information. When the decoder upsamples the low-resolution encoded features, it might not fully recover the original details[5]. 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, helping it to better reconstruct the original input, especially in tasks like image segmentation where pixel-level detail is important.

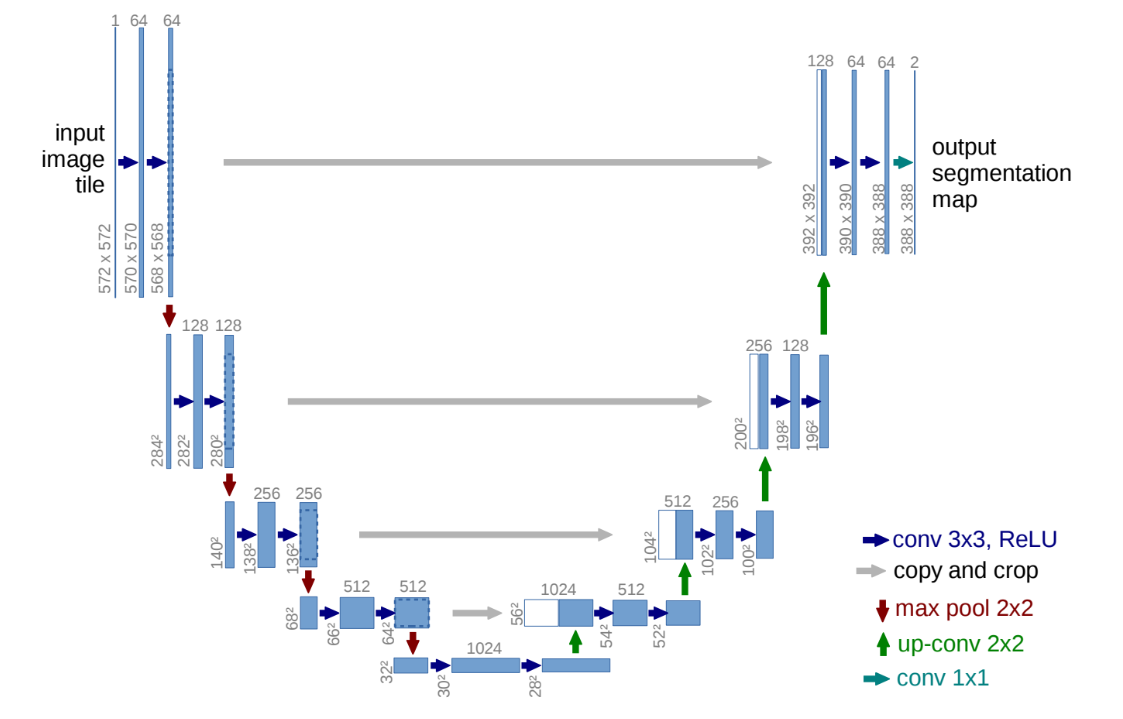


Figure 2: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows 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. 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. Graph Pooling (gPool) and Unpooling (gUnpool) which are operations similar to the encoder-decoder structure of U-Nets are operations that enable to learn hierarchical representations of graphs and reconstruct the original graph structure.

תמונה שמכילה תרשים, צילום מסך, קו

התיאור נוצר באופן אוטומטי

Figure 3: The architecture of gPool and gUnpool in the Graph U-net model.

gPool operation is introduced for down-sampling graph data. 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, . 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 , ensuring consistency in connectivity across nodes in the new graph. Mathematically, the projection formula is represented as Where represents a given node, denotes the node feature vector, signifies the projection vector and measures how much information of node can be retained when projected onto the direction of . The new graph, represented by matrices (represents the adjacency matrix) and (represents the feature matrix), is obtained by selecting the k nodes with the largest scalar projection values on . 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 and the gate vector governs the information of the selected nodes, resulting in the new graph with the most significant features while minimizing information loss.

תמונה שמכילה תרשים, תוכנית, טקסט, צילום מסך

התיאור נוצר באופן אוטומטי

Figure 4: The proposed graph pooling layer operates on a graph with 4 nodes, each having 5 features. It starts by creating an adjacency matrix Aℓ​∈R4×4 and an input feature matrix Xℓ​∈R4×5 for layer ℓ. In the projection stage, a trainable projection vector p∈R5 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 X~ℓ​ and a new adjacency matrix Aℓ+1​. In the gate stage, element-wise multiplication is performed between X~ℓ​ and the selected node scores vector y~​, resulting in Xℓ+1​. The graph pooling layer outputs Aℓ+1​ and Xℓ+1​.

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. 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

Where denotes the feature matrix of the restored graph represents the feature matrix of the current graph is the initially empty feature matrix for the new graph, with dimensions identical to the original graph but filled with zeros and contains the indices of the selected nodes in the corresponding gPool layer, which reduced the graph size from nodes to nodes. The distribution process involves filling the initially empty feature matrix  by placing the feature vectors from the current graph into their corresponding positions based on the indices stored in . In the restored graph , the row vectors corresponding to the selected nodes are updated with the corresponding row vectors from , 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 node's 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.

תמונה שמכילה צילום מסך, תרשים

התיאור נוצר באופן אוטומטי

Figure 5: Architecture of G-U-net

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. 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's 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.

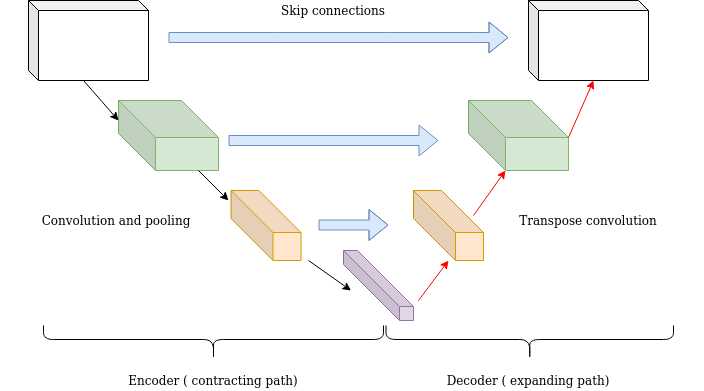


Figure 6: Macro view on the Graph U-net architecture focusing on the skip connection.

**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. 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)[1]  
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.[1] 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 is written as:  
min G max D V(G, D) = Ex∼Pdata(x) [log D(x)] + Ez∼Pz (z) [log(1 − D(G(z)))]"[1].

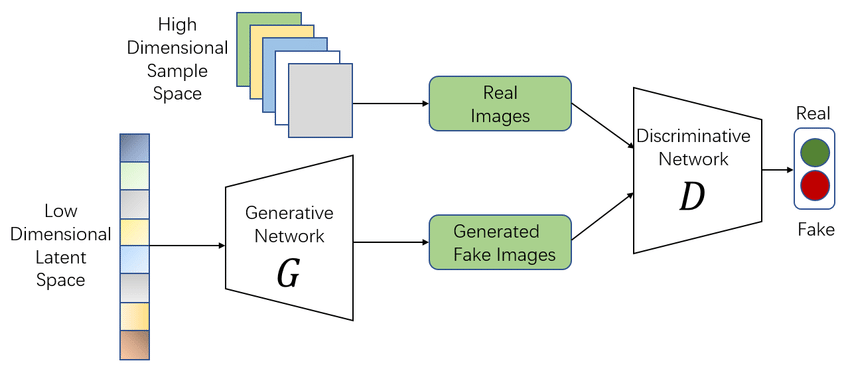


Figure 7: The architecture of a basic GAN [6].

**2.8 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. 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. 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.

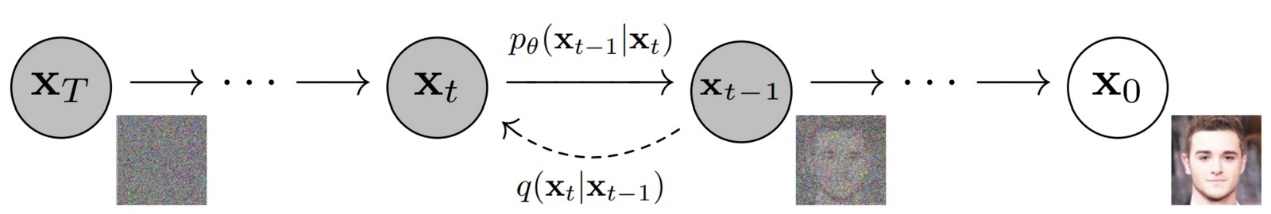


Figure 7: The main idea of diffusion model. By adding noise in the diffusion process the image became more cleaner and by imply reverse diffusion process the image return to the origin.

**2.8.1 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. 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. 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.

**3. Expected Achievements**

The project aims to explore novel methods 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's 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 model input is a graph with hierarchical structures, after pre-processing operations will coarsen the input into representation of different granularity levels, then for each level of granularity generate a corresponding coarsen graph. The output will be a reconstructed graph that retains the same composition of the original input graph.

תמונה שמכילה טקסט, צילום מסך, תרשים, עיצוב

התיאור נוצר באופן אוטומטי

**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 The Graph U-net process**

The Graph U-net process starts after dataset partitioning, aiming to refine the input graphs through coarsening, encoding, and decoding stages. This process adapts dynamically to varying graph complexities, facilitating the extraction of hierarchical features. Ultimately, the output of the Graph U-net is a reconstructed graph that preserves the original composition while serving as input for subsequent model stages.

**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 and the feature matrix . 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. 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 process entails the coarsening of the input graph into representations of varying granularity levels. It achieved by the gPool, gUnpool, and skip connection operations. 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. This generation process entails encoding the hierarchical features of the input graph into a latent representation, which is subsequently decoded to reconstruct the coarsened graph. Leveraging its encoder-decoder architecture, the model integrates graph pooling and unpooling operations to capture and reconstruct the hierarchical structure inherent in the input graph.

**4.2.3 The Graph U-net Output**  
The output of the Graph U-net process constitutes a reconstructed graph that faithfully maintains the composition of the original input graph. This refined graph serves as an augmented representation of the input data, characterized by its absence of noise while crucial structural details remain intact. Furthermore, the Graph U-net Output seamlessly transitions to serve as the input for the subsequent stage in the model architecture, facilitating the continuation of the data processing flow.

**4.3 Generating Multi-scale Graphs**

The next step involves transferring the coarsened graph representations from the Graph U-net output to a model akin to misc-GAN. The model proposed in this study substitutes the coarsened graph generation mechanism from cycleGAN with a diffusion model to assess the potential increase in the overall efficiency of the model. Subsequently, by performing multi-scale reconstruction of the generated coarsened graphs at each granularity level, a new graph is created that maintains the hierarchical topological features of the input graph.

**4.3.1 Generate Coarse Graphs with Diffusion Model**

**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.‏

**O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.‏**

**Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. (2019, May). Simplifying graph convolutional networks. In International conference on machine learning (pp. 6861-6871). PMLR.‏**

[4] https://tkipf.github.io/graph-convolutional-networks/

**Krenker, A., Bešter, J., & Kos, A. (2011). Introduction to the artificial neural networks. Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech, 1-18.‏**

**Zhu, F., Zhang, L., Gao, J., Qin, B., Xu, R., & Yang, H. (2023). A Diffusion Model for Event Skeleton Generation. arXiv preprint arXiv:2305.17458.‏**

[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.‏