**Abstract:**

**U-Nets:**

The model is made up of an encoder and a decoder.

The encoder is a series of repeated 3x3 convolutional layers with ReLU activation and 2x2 max pooling, which reduces the spatial dimensions of the features while doubling the number of channels after each downsampling operation.

The decoder is a series of 3x3 convolutional layers with ReLU activation, which upsamples the current set of features and applies a 2x2 convolutional layer to restore the spatial resolution of the features lost during encoding.

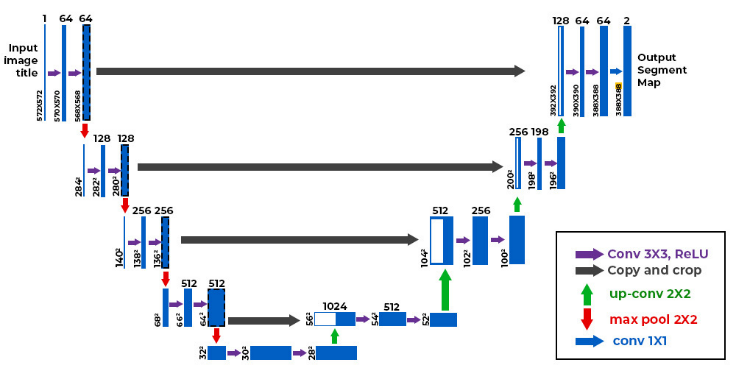
The model uses two types of connections between the encoder and decoder: connecting paths and bottlenecks.

The connecting paths take a copy of the features from the symmetrical part of the encoder and concatenate them onto the corresponding stage in the decoder, to combine the decoded and encoded features together and achieve Pixel Perfect segmentation.

The bottleneck is where the encoder switches into the decoder by downsampling the features, passing them through the recognizable convolutional layers, and then up sampling them to their previous resolution before the bottleneck before passing the features up the decoder up sampling the features as they go and concatenating them via connecting paths.

The U-Net model is a simple architecture that uses ideas similar to that of residual networks and can achieve impressive performance even on small data sets by applying data augmentation techniques.

The U-Net model is used for image segmentation for medical use and diffusion models for iterative image denoising.



**Encoder:**

The encoder starts by taking an input image.

This could be any picture, like a medical image, a photo, or something else.

The job of the encoder is to look at the input image and identify important features.

It does this using a series of filters and convolutional layers.

These filters act like highlighters, emphasizing different patterns and shapes in the image.

After identifying features, the encoder "downsamples" the image.

This means it reduces the size of the image while keeping the essential information.

It's like zooming out, focusing on the bigger picture.

The encoder often uses pooling layers to help with downsampling. Pooling involves taking the average or maximum value in a small region of the image.

This helps to simplify the information and make the network more efficient.

As the encoder goes deeper, it captures more abstract and high-level features.

It's like looking at the image in layers, starting with simple details and gradually understanding more complex structures.

**Decoder:**

The encoded information from the encoder contains a compressed representation of the input image's features.

The decoder's job is to take this compressed information and turn it back into a detailed image.

Just as the encoder downsamples, the decoder "upsamples."

This means it increases the size of the encoded features while maintaining the learned patterns.

It's like zooming back in to see the details.

Here's where U-Net stands out, Skip connections connect equivalent layers from the encoder to the decoder.

These connections provide a shortcut for the network to access detailed information that might have been lost during downsampling.

The information from the skip connections is combined with the upsampled features.

Think of it as merging details from different resolutions. This helps the network to have both a broad understanding and fine details.

The decoder processes the combined information and produces an output.

In the case of U-Net for image segmentation, this output is a map where each pixel indicates the likelihood of belonging to a particular class.

It's essentially saying, "This part of the image is likely to be of class A, that part is likely to be of class B," and so on.

**Downsampling:**

Downsampling in U-Net often involves the use of pooling layers, commonly max pooling.

Pooling layers reduce the spatial dimensions of the input feature maps by selecting the maximum (or average) value in a small window.

תמונה שמכילה טקסט, תרשים, צילום מסך, מקביל

התיאור נוצר באופן אוטומטי

*Max Pooling and Average Pooling being performed*

This helps to retain the most important information while reducing the computational load.

Imagine looking at a detailed map, and you decide to focus on larger regions first.

Downsampling is like zooming out, going from looking at streets to neighborhoods.

This process maintains the general structure but reduces the level of detail.

Despite reducing spatial dimensions, the pooling layers retain important features.

As the network goes through multiple downsampling steps, it captures hierarchical features.

Downsampling is crucial for capturing hierarchical features in an image.

It allows the network to process the image at multiple levels of abstraction.

The pooling or striding operations reduce spatial dimensions, focusing on capturing global information and high-level patterns while discarding fine details.

**Upsampling:**

Upsampling in U-Net often involves transposed convolutional layers (also known as deconvolution or fractionally strided convolution).

These layers increase the spatial dimensions of the feature maps.

תמונה שמכילה קו, עיצוב, אומנות

התיאור נוצר באופן אוטומטי

*The resolution of the output (5x5) is higher than the one of the input (3x3), white is padding.*

תמונה שמכילה טקסט, צילום מסך, תרשים, עיצוב

התיאור נוצר באופן אוטומטי

*The resolution of the output (5x5) is higher than the one of the input (2x2), white is padding.*

If downsampling is like zooming out on a map, then upsampling is like zooming back in.

It's the process of regaining the finer details while still maintaining the broader context.

Upsampling layers contribute to the reconstruction of high-resolution information.

Imagine going from a simplified version of an image back to a more detailed representation.

Skip connections play a crucial role in upsampling.

They provide a shortcut for detailed information to bypass the downsampling bottleneck.

The network can use this detailed information during the upsampling process to better reconstruct the original features.

In U-Net, the combination of upsampled features and information from skip connections ensures that the network produces a final output that balances the broader context with fine details.

Upsampling is equally important as it helps in reconstructing fine-grained details lost during downsampling.

**Skip Connections:**

One key innovation of U-Net is the incorporation of skip connections.

These connections directly connect corresponding layers in the encoder and decoder.

Skip connections help in preserving spatial information and details during the upsampling process, allowing the network to recover fine-grained features.

Skip connections allow the neural network to directly jump from the encoder to the decoder.

They connect equivalent layers in these two parts of the network.

When we downsample in the encoder, some fine details might get lost.

Skip connections act as a recovery point, providing a shortcut for detailed information to travel directly to the decoder

The information from skip connections is combined with the upsampled features in the decoder.

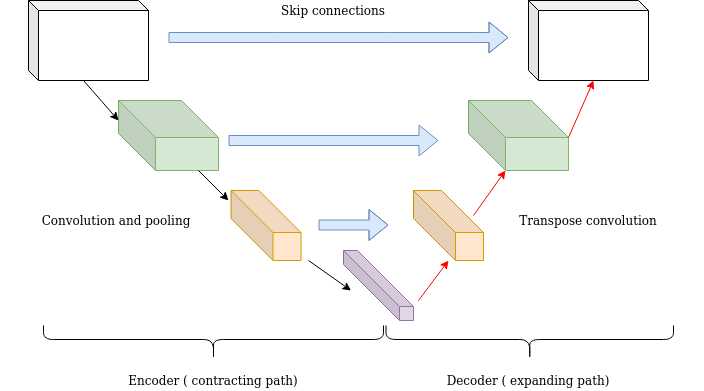
This combination helps the network to have the best of both worlds: the broader context from downsampling and the detailed information from the skip connections.

Without skip connections, the decoder might struggle to recreate intricate details.

By preserving connections to the encoder, the U-Net ensures that the network doesn't lose crucial information during the downsampling process.

The skip connections, which link corresponding layers between the encoder and decoder, play a key role in combining high-level and detailed information.

The decoder, with transposed convolutions or upsampling layers, reconstructs the original spatial resolution.



**Final Layer:**

The final layer of the U-Net is responsible for making pixel-level decisions.

It takes the features processed by the decoder and converts them into a map where each pixel is classified into different categories or classes.

The final layer often uses a mathematical operation called softmax or sigmoid.

The output is a map where each pixel is "colored" based on the class it's most likely to belong to.

This map is what the U-Net uses to tell us where different objects or regions are in the input image.

**Loss Function:**

The loss function calculates the difference between the predicted output and the actual ground truth.

For each pixel in the output map, the loss function checks if the network's prediction matches the true class label.

It penalizes the network more if it makes a big mistake and less if it's close to the truth.

The loss is then used to adjust the internal parameters of the U-Net through a process called backpropagation.

Through multiple iterations of presenting images, predicting, calculating the loss, and adjusting parameters, the U-Net learns to improve its predictions.

The goal is for the loss to become as small as possible, indicating that the network is making accurate pixel-wise predictions.

**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 being solved and the requirements of the task.

softmax is commonly used in the final layer for multi-class classification, contrasting it with sigmoid used in binary classification.

Softmax for Multi-Class Classification:

Softmax is well-suited for scenarios where there are more than two classes.

It transforms the raw output scores (logits) into a probability distribution over multiple classes.

Each class gets a probability score, and the probabilities sum to 1.0.

In tasks with multiple classes, softmax allows the neural network to make a definitive choice among the classes.

The class with the highest probability is selected as the final prediction.

Softmax assumes that each input belongs to one and only one class.

If your task involves mutually exclusive classes (e.g., image classification where an image contains a single object class), softmax is a suitable choice.

Softmax encourages global information sharing among classes.

The probabilities assigned to each class depend not only on the individual class's score but also on the scores of other classes.

This can be beneficial in capturing complex relationships between classes.

**Sigmoid for Binary Classification:**

Sigmoid is commonly used in binary classification tasks where there are only two possible classes (0 or 1).

It produces independent probabilities for each class, representing the likelihood of the input belonging to the positive class.

In tasks where an input can belong to multiple classes simultaneously (multi-label classification), sigmoid is preferred.

Each sigmoid activation independently predicts the probability of an input belonging to its respective class, allowing for the coexistence of multiple positive classes.

Sigmoid provides a probability score for each class separately.

A decision threshold is applied to determine the final prediction.

If the probability surpasses a certain threshold, the class is considered present.

This flexibility is useful when dealing with imbalanced datasets or tasks where classes may overlap.

In conclusion, softmax is suitable for multi-class classification where each input belongs to a single class, while sigmoid is suitable for binary classification and multi-label classification where an input can belong to multiple classes simultaneously.

The choice depends on the problem's characteristics and the desired output format.

**Backpropagation:**

**After the neural network makes predictions, backpropagation provides feedback on how well the network performed by comparing its predictions to the actual (ground truth) values.**

**The first step in backpropagation involves calculating the loss.**

**Loss is a measure of how far off the network's predictions are from the actual values.**

**Backpropagation then attributes this error back through the network.**

**The goal is to understand how each parameter in the network contributed to the overall error.**

**Calculating gradients is a crucial step.**

**Gradients represent how much the loss would change with respect to each parameter in the network.**

**Using the gradients, the network updates its parameters (weights and biases).**

**The adjustments are made in the opposite direction of the gradient to minimize the loss.**

**Backpropagation is an iterative process.**

**The network goes through many cycles of making predictions, receiving feedback, and adjusting its parameters.**

**Backpropagation often works in conjunction with optimization algorithms like stochastic gradient descent (SGD).**

**These algorithms determine the step size and direction for parameter updates.**

**The ultimate goal of backpropagation is for the network to converge, meaning it reaches a state where further adjustments don't significantly improve performance.**

**Training the Algorithm:**

**You train the algorithm using a labeled dataset, where each input (image) is associated with a corresponding ground truth or target output (segmentation map).**

**During training, the algorithm learns to map input images to the correct segmentation maps through an iterative optimization process.**

**To increase the diversity of the training data, data augmentation entails adding numerous transformations to the training images, such as rotations, flips, scaling, and deformations.**

**Regularization techniques such as dropout and batch normalization prevent overfitting and improve model performance on unknown data.**

**Feature Relevance:**

**The algorithm automatically learns which features are relevant during the training process.**

**It adjusts the internal parameters (weights) of the network to assign importance to different patterns and structures in the input images.**

**The relevance of features emerges from the learning process, guided by the loss function, which measures the dissimilarity between predicted and actual segmentation maps.**

**Pixel-wise Classification:**

**The reason for not directly scanning pixels in the image is tied to the goal of pixel-wise precision in semantic segmentation.**

**Directly scanning pixels might not capture complex relationships and context. The U-Net architecture, with its encoder-decoder structure and skip connections, excels in capturing both local and global information, making it more effective for tasks like semantic segmentation.**

**Graph U-Nets:**

Two key operations that enable the model to learn hierarchical representations of graphs and reconstruct the original graph structure.

Graph Pooling and Unpooling are operations that enable the encoder and decoder to learn hierarchical representations of graphs and reconstruct the original graph structure.

Encoder and decoder are general terms that refer to the parts of a neural network model that transform the input into a latent representation and the latent representation into an output, respectively.

Graph Pooling and Unpooling are specific methods that can be used in the encoder and decoder of a graph neural network model, such as Graph U-Nets.

Graph Pooling and Unpooling are needed because graph data is irregular and variable-sized, unlike image data that can be easily processed by convolutional neural networks.

Graph Pooling and Unpooling allow the model to reduce the size of the graph, capture the most important features, and restore the graph into its original structure.

תמונה שמכילה צילום מסך, תרשים

התיאור נוצר באופן אוטומטי

**gPool – Graph Pooling Layer:**

**pooling layer play important role in CNN on grid-like data.  
They can reduce sizes of feature maps and enlarge respective fields thereby get a much better performance when trying to down-sample a 2D structure like grid, feature map etc.**

**The problem to apply pooling methods like max pool or average pool on graph results that in graphs there is no locality information among nodes, so partition operation is not applicable on them.  
Also global pooling operation will reduce all nodes to one single node, witch restricts the flexibility of networks.  
  
The gPool operation is propose to overcome the problems resulting from pooling in graphs for down-sampling data, and here how it works:**

**In each convolution layer we adaptively select a subset of nodes to form a new but smaller graph.  
We employ a trainable projection vector P.  
by projecting all nodes features to 1D, we can perform k-max pooling for node selection.**

**Since the selection is based on 1D footprint of each node, the connectivity in the new graph is consistent across nodes.  
The selection of the Kth largest nodes are is determined by the projection formula:**

**Where:**

**i – a given node.**

**–the node feature vector.**

**P – the projection vector.**

– measures how much information of node I can be retained when projected onto the direction of p.

To preserve as much information as possible from the original graph we select nodes with the largest scalar projection values on p to form a new graph.

The new graph G with N nodes that each node contains C features can be represented by two matrices:

– the adjacency matrix.

- the feature matrix.

The layer-wise propagation rule involves selecting k nodes for the new graph.  
The operation ranks the scalar projection of each node with its features and return the indices of the k largest values.  
From these chosen indices we extract the new adjacency matrix A' and feature matrix X' for the new graph.  
Then a gate operation controls the information flow by applying a sigmoid function to each element in the extracted scaler projection vector.  
The element-wise matrix product of X' and the gate vector y' controls the information of the selected nodes.

Then we get the new graph with the most important feature with as much information loss as possible.

**An illustration of the gPool layer:**  
  
 תמונה שמכילה תרשים, תוכנית, טקסט, צילום מסך

התיאור נוצר באופן אוטומטי

**gUnpool – Graph Unpooling Layer:**

To enable up-sampling operations on graph data, the proposed a graph unpooling (gUnpool) layer, which performs the inverse operation of the gPool layer and restores the graph into its original structure. To achieve this, the locations of nodes selected in the corresponding gPool layer been recorded and used this information to place nodes back to their original positions in the graph. **The layer-wise propagation rule of the gUnpool is as follow:**

**Where:**

**- is the feature matrix of the restored graph.**

**- is the feature matrix of the current graph.**

**– is initially empty feature matrix for the new graph it has the same size as the original graph but filled with zeros.**

**Idx - contains the indices of the selected nodes in the corresponding gPool layer that reduced the graph size from N nodes to K nodes.**

**The distribution process:**

**The distribute operation aims to fill the initially empty feature matrix by placing the feature vectors from the current graph into their corresponding positions based on the indices stored in idx.  
In , the row vectors corresponding to the selected nodes (as indicated by idx) are updated with the corresponding row vectors from , while the row vectors corresponding to nodes not selected remain zero.**

**An illustration of the gUnpool layer:**

**תמונה שמכילה תרשים, צילום מסך, קו

התיאור נוצר באופן אוטומטי**

**Graph U-Nets Architecture:**

**To implement the G-U-net architecture first is been apply a graph embedding layer to convert nodes into low-dimensional representation, since some of the datasets use high- dimensional feature vectors.  
After that the encoder is build by stacking several encoding blocks, each block made of GCN layer and a gPool layer.  
The gPool reduce the size of the graph to encode higher order features, while GCN layers responsible for aggregating information from each node first-order neighbors'** information.  
The decoder part stacked with the same number of decoding blocks as in the encoder part, each decoder block made of GCN layer and gUnpooling layer.  
The gUnpool layer restores the graph into its higher resolution structure, and the GCN layer aggregates information from the neighborhood.  
Another part of the architecture is the skip-connection between the corresponding blocks of encoder and decoder, witch transmit spatial information to decoders for better performance.  
Finally a GCN layer employed for final predictions before soft-max function.

**Graph Connectivity Augmentation via Graph Power:**

**While using the gPool layer for sampling important nodes to form a new graph with high level feature encoding, there is a possibility for nodes to become isolated from the gPool removal of related edges, and this may influence the information propagation in subsequent layers.  
Especially when GCN layers are used to aggregate information from** neighboring nodes**.  
To increase connectivity among nodes, the proposed method is to use kth graph power .  
This operation builds links between nodes whose distances are at most k hops in this work is been employes when k=2.  
Now, the graph sampling is performed on the augmented graph with better connectivity.**

**Improved GCN Layer:**

In regular GCN layer the adjacency matrix before normalization is computed as:   
in which a self-loop is added to each node in the graph.  
In this work, they tried to give a higher weight to node’s own feature vector, since its own feature should be more important for prediction.  
To this end, we change the calculation to by imposing larger weights on self-loops in the graph, which is common in graph processing. All experiments in this work use this modified version of GCN layer for better performance.

**Introduction:**

**Convolutional neural networks:**

Convolutional Neural Networks (CNNs) are a type of deep neural network designed for processing structured grid data, such as images.

They use convolutional layers that apply filters to input data, enabling them to learn hierarchical features and spatial hierarchies.

CNNs excel at capturing local patterns and spatial hierarchies in data.

They are widely used in computer vision tasks, including image classification, object detection, and segmentation.

While CNNs are designed for grid-structured data, Graph U-Nets are tailored for graph-structured data.

Both leverage convolutional operations, but Graph U-Nets extend these operations to graph structures, capturing dependencies in non-grid data.

**feature maps:**

Feature maps are representations of learned features within a neural network.

In CNNs, a feature map is created by applying filters to the input data, highlighting specific patterns and structures**.**

Feature maps provide a visual representation of what the network has learned at different levels of abstraction.

In image processing, feature maps highlight edges, textures, and higher-level features.

Feature maps in CNNs are specific to grid-structured data, such as images.

Graph U-Nets operate on graph-structured data and use graph convolutional layers to create feature representations that capture dependencies between nodes.

**computer vision:**

Computer vision is a field of study that focuses on enabling machines to interpret and understand visual information from the world.

It involves tasks such as image recognition, object detection, and scene understanding.

Computer vision aims to replicate human vision capabilities by developing algorithms and models that can make sense of visual data.

CNNs are a foundational technology in computer vision, allowing machines to learn and recognize patterns in images.

Computer vision often deals with grid-structured image data, where CNNs are effective.

However, as the field expands to non-grid data, such as graphs, Graph U-Nets become relevant for tasks like graph-based image segmentation or scene understanding.

**link prediction:**

Link prediction is a task in graph theory where the goal is to predict missing or potential edges in a graph.

It is commonly applied in social network analysis, recommendation systems, and biological networks.

Link prediction algorithms aim to identify relationships that may exist between entities in a graph but are not explicitly present.

This task is valuable in understanding and uncovering hidden connections in various domains.

Graph U-Nets can be adapted for link prediction tasks by learning representations of nodes and capturing complex relationships within the graph.

The hierarchical feature learning and graph-based operations make Graph U-Nets suitable for tasks involving graph structures.

**node classification:**

Node classification is a task in graph analysis where the goal is to assign labels to nodes in a graph.

Each node is assigned a category or class label based on its properties or connections.

Node classification is essential for understanding the roles and characteristics of individual nodes within a graph.

It has applications in various domains, including social network analysis, biology, and citation networks.

Graph U-Nets can be applied to node classification tasks by learning node representations that capture both local and global information in the graph.

The graph convolutional layers in Graph U-Nets facilitate the extraction of features for accurate node classification.

**node representations:**

Node representations refer to the learned embeddings or features associated with each node in a graph.

These representations capture the characteristics and relationships of nodes within the graph.

Node representations are crucial for tasks like node classification, link prediction, and graph-based machine learning.

They provide a compact and meaningful representation of the nodes in a graph.

Graph U-Nets contribute to learning effective node representations by leveraging graph convolutional layers.

These layers enable the model to capture dependencies and relationships between nodes, resulting in meaningful node representations.

**Graph neural network:**

A Graph Neural Network (GNN) is a type of neural network designed for processing graph-structured data.

GNNs aim to learn representations of nodes and capture dependencies within the graph.

GNNs are powerful for tasks involving graph-structured data, such as node classification, link prediction, and graph classification.

They use graph convolutional layers to perform operations on nodes based on their neighborhood relationships.

Graph U-Nets can be considered a specific architecture within the broader category of GNNs.

While GNNs encompass various architectures, Graph U-Nets specifically focus on leveraging a U-Net structure for hierarchical feature learning in graph-structured data.

**scalar projection:**

Scalar projection is the process of projecting a vector onto another vector, resulting in a scalar value. It represents the component of one vector in the direction of another.

Scalar projection is useful for understanding the contribution of one vector along a specific direction. It helps quantify the influence or magnitude of a vector in a particular dimension.

Scalar projection might not have a direct connection to Graph U-Nets.

**network embedding:**

Network embedding involves transforming nodes or edges in a network into continuous vector representations (embeddings).

These embeddings capture the structural and relational information of nodes or edges in a lower-dimensional space.

Network embeddings are valuable for various tasks, such as node classification, link prediction, and graph visualization.

They provide a compact and meaningful representation of nodes and edges, facilitating downstream machine learning tasks on graphs.

While Graph U-Nets focus on hierarchical feature learning and segmentation in graph-structured data, network embeddings can be seen as a complementary concept.

Graph U-Nets may contribute to learning embeddings by capturing hierarchical features, and embeddings, in turn, can be used for downstream tasks like node classification or link prediction.

the network embedding method used in this code is a graph convolutional network (GCN).

The GCN layers contribute to learning embeddings for nodes in the graph by performing graph convolution operations and applying linear projections, activation functions, and dropout for regularization.

**Transductive Learning:**

Transductive learning is a learning paradigm where the model aims to make predictions for specific examples that are observed during the training phase.

The model is trained on a partially labeled dataset, and the goal is to predict the labels for the remaining unlabeled instances in the same dataset.

The focus is on making predictions for the instances observed during training, especially the ones that are unlabeled. It assumes that the training and testing instances come from the same distribution.

Transductive learning is particularly useful when there is a limited amount of labeled data but a larger amount of unlabeled data available.

Commonly applied in scenarios where the goal is to improve predictions on the specific examples present in the training set.

**Inductive Learning:**

Inductive learning is the traditional learning paradigm where the model is trained on a labeled dataset, and the goal is to make predictions on new, unseen instances.

The model generalizes patterns learned from the training data to make predictions on completely new and previously unseen data points.

The focus is on learning general patterns and relationships from the labeled training data to make predictions on any new instances, not just those observed during training.

It assumes that the model needs to perform well on unseen data that may come from a slightly different distribution than the training data.

Inductive learning is the standard approach for most machine learning tasks where the goal is to build a model that can make accurate predictions on new, unseen examples.

It is suitable when the intention is to develop a general model applicable to a broader range of instances beyond the training set.

**Graph power:**

The k-th graph power, denoted as Gk, is an operation that builds links between nodes based on their distances, ensuring that nodes within a certain number of hops (k hops) are connected. It enhances the connectivity of the graph.

Building Links Between Nodes:

The operation creates edges between nodes whose distances are at most k hops apart. This helps improve the flow of information between nodes that are closer in terms of graph structure.

Choice of k:

In this context, they propose to use k = 2, meaning they are considering nodes within 2 hops. The choice of k is influenced by the presence of a GCN (Graph Convolutional Network) layer before each graph pooling (gPool) layer.

Graph Sampling:

The enhanced graph, obtained by applying the k-th graph power operation, is used for graph sampling. Graph sampling is a technique where a subset of nodes is selected from the graph, often used in pooling operations.

spatial information:

Spatial information refers to the arrangement or location of elements in space.

In the context of graphs, spatial information can represent the geometric or topological relationships between nodes and their positions within the graph structure.

Spatial information is crucial in graph-based tasks.

Graph U-Nets, being a type of neural network designed for graph-structured data, aim to capture and utilize spatial relationships between nodes in a graph for tasks such as node classification or graph-level prediction.

citation networks:

Citation networks represent relationships between academic papers where each paper cites other papers.

Nodes typically represent papers, and edges represent citations.

Graph U-Nets can be applied to citation networks for tasks such as node classification or link prediction.

The spatial dependencies among papers, captured through graph convolutions, can help in understanding the context of each paper within the citation network.

identity activation function:

The identity activation function returns the input as is.

It is often used when no non-linearity is desired in a neural network layer.

The identity activation function can be used in certain layers of Graph U-Nets.

For example, in skip connections, where the original node features are combined with the output of a graph convolution, the identity activation may be employed to maintain the original information without introducing non-linear transformations.

L2 regularization:

L2 regularization, also known as weight decay, is a regularization technique that adds a penalty term to the loss function based on the squared magnitude of model weights.

It helps prevent overfitting by discouraging overly complex models.

L2 regularization can be applied to the parameters of Graph U-Nets to control the model's complexity and mitigate overfitting, especially when dealing with limited training data or large model capacities.

The dropout keep rate:

Dropout is a regularization technique where randomly selected neurons are omitted during training.

The "dropout keep rate" is the probability of keeping a neuron active during dropout.

Dropout can be applied in Graph U-Nets to prevent overfitting by randomly dropping out nodes during training.

The dropout keep rate is a hyperparameter that determines the proportion of nodes to retain.

It introduces a form of ensemble learning, making the model more robust.

over-fitting:

Overfitting occurs when a machine learning model learns the training data too well, capturing noise and details that do not generalize to unseen data.

Overfitting is a concern in any machine learning model, including Graph U-Nets.

Techniques such as dropout, L2 regularization, and careful hyperparameter tuning are employed in Graph U-Nets to mitigate overfitting and improve generalization performance.