1. Increase the N value from 20 (original value) to 200 with multiple N values in between and observe the change of graph density and degree distribution (i.e., histogram plot). Explain what you observe and write the answer in a word file

**Graph Density**:

* + As you increase the number of vertices from 20 to 200, the density of the graph decreases. This is because the number of possible connections (edges) increases much faster than the number of actual connections you are adding. So, the graph becomes relatively sparser as it gets larger.

**Degree Distribution**:

* + For a smaller graph (N=20), the degrees (number of connections each node has) vary but tend to be around the middle values.
  + For a larger graph (N=200), despite having more nodes, most nodes have a smaller number of connections, and the degrees are more tightly grouped around lower values.
  + This suggests that as the graph gets bigger, the nodes tend to have a more uniform number of connections, and extreme values (very high or very low degrees) are less common.

In simple terms, as you make the graph bigger but only double the connections, the graph becomes less dense, and nodes generally have fewer connections compared to the total possible, leading to more uniformity in how connected the nodes are.

1. Explain the differences between supervised learning, self-supervised learning and semi-supervised learning methods

**Supervised Learning:**

* **Definition**: Supervised learning involves training a model on a labelled dataset, meaning that each input data point has a corresponding output label.
* **Usage**: This method is used when a sufficient amount of labelled data is available.
* **Examples**: Classifying emails into spam and non-spam, or predicting the price of a house based on its features.

**Self-Supervised Learning:**

* **Definition**: Self-supervised learning is a type of unsupervised learning where the data itself provides the supervision. Here, the model is trained with data that has not been explicitly labelled, but where labels are derived from the data itself.
* **Usage**: It's particularly useful when labels are scarce or expensive to obtain, but there is a lot of unlabelled data.
* **Examples**: Generating a sentence with a missing word and having the model predict the missing word; using part of an image to predict another part of the same image.

**Semi-Supervised Learning:**

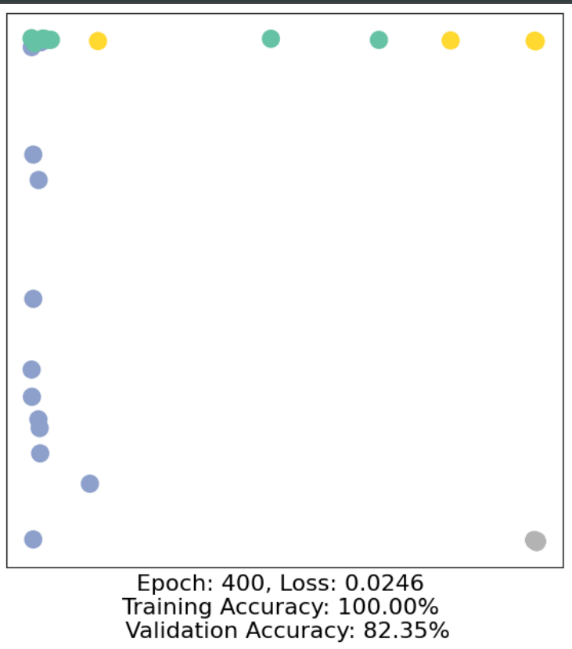
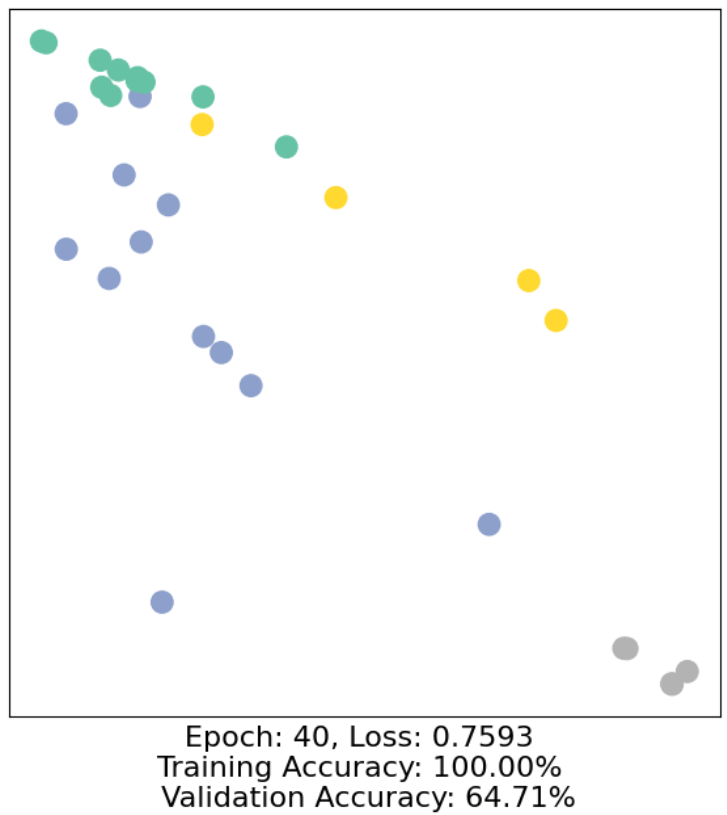
* **Definition**: Semi-supervised learning uses a small amount of labelled data combined with a large amount of unlabelled data. This method leverages the labelled data to guide the learning process in the unlabelled dataset.
* **Usage**: It is used when acquiring a fully labelled dataset is costly or impractical, but some labelled data can enhance learning significantly.
* **Examples**: A small number of labelled photos are used to train a model that then categorizes a larger set of unlabelled photos.

1. Explain the differences between transductive learning and inductive learning.  
     
   **Transductive Learning:**

* **Definition**: Transductive learning specifically aims to predict labels for a given, fixed set of unlabelled data. The model learns from both the labelled and unlabelled data but is only concerned with making predictions for the specific unlabelled data it has seen during training.
* **Usage**: It is used when the task is to make predictions only on the specific set of data available during the training phase, and not on unseen data.
* **Example**: If a model is trained to classify sentiments of specific tweets collected during an election period, it is not expected to generalize beyond these tweets.

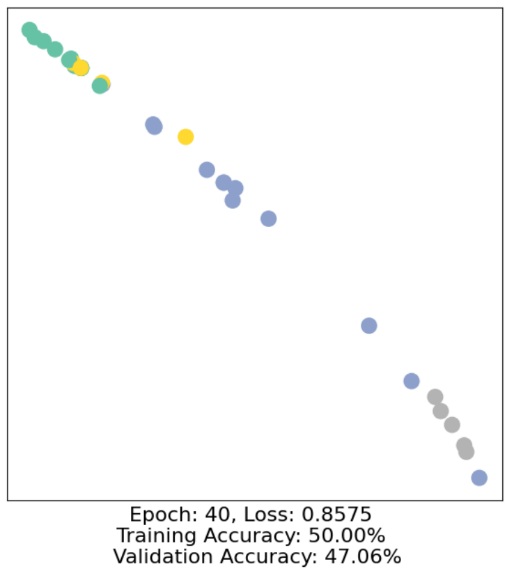
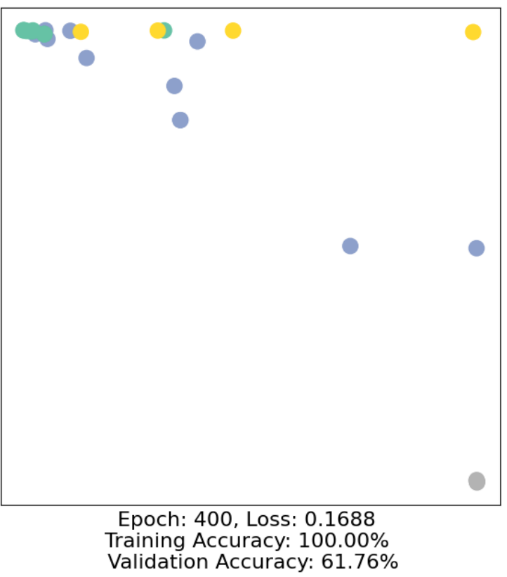
**Inductive Learning:**

* **Definition**: Inductive learning, on the other hand, focuses on building a general model that can make predictions on any new, unseen data based on the patterns learned from a training dataset.
* **Usage**: This is the most common goal in machine learning, aimed at creating models that generalize well to new, unseen situations.
* **Examples**: A spam detection model trained on some emails should be able to detect spam in future emails not seen during the training.

1. Increase the number of epochs from 50 to 500 and observe the change in validation accuracy and write what you observe in the word file.  
     
   

**Improvement in Validation Accuracy**: Increasing the number of epochs leads to an improvement in validation accuracy, suggesting that the model is gradually learning to generalize better. However, the persistence of 100% training accuracy from early in the training process suggests that the model might benefit from techniques to reduce overfitting.

1. Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease in the word file

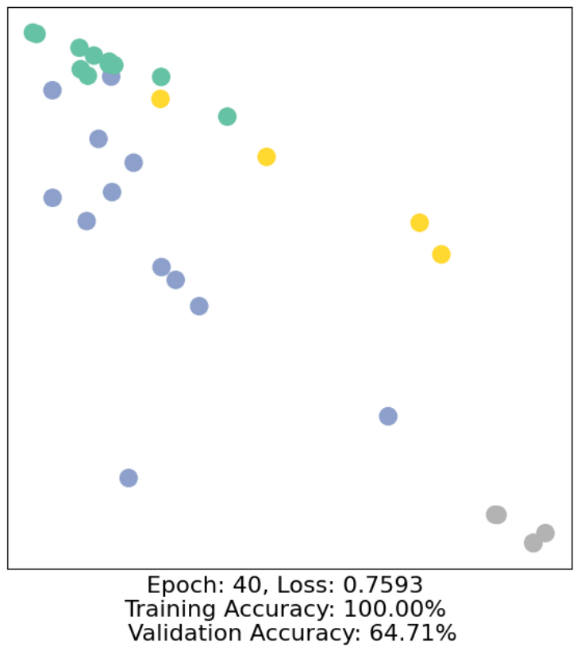
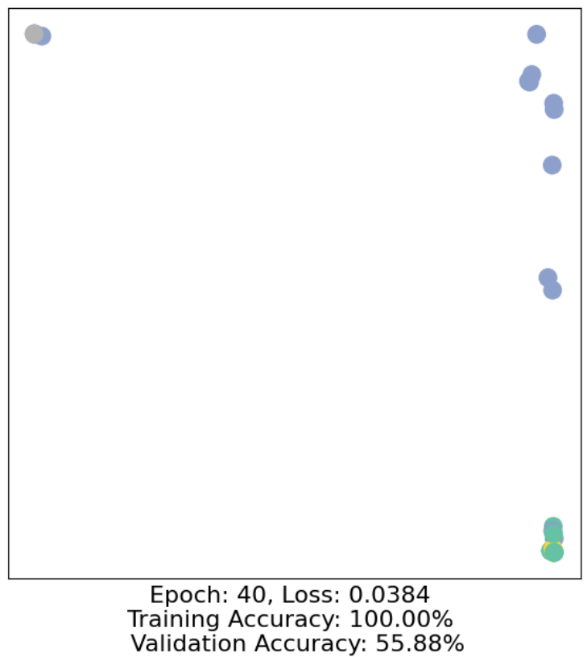


**With Self-Loops:**

* **Early Training (Epoch 40)**:
  + Validation Accuracy: 64.71%
* **Later Training (Epoch 400)**:
  + Validation Accuracy: 82.35%
* **Observation**: Higher validation accuracies suggest better learning and generalization.

**Without Self-Loops:**

* **Early Training (Epoch 40)**:
  + Validation Accuracy: 47.06%
* **Later Training (Epoch 400)**:
  + Validation Accuracy: 61.76%
* **Observation**: Lower validation accuracies indicate struggles in model generalization.
* Self-loops help improve model performance, showing higher validation accuracies and better generalization from training to unseen data.

1. Increase the number of GCNConv() layers in the GCN() model upto 8 layers from original 3 layers. Detail the accuracy increase/decrease

* **3-Layer Model (Epoch 40)**:
  + **Training Accuracy**: 100%
  + **Validation Accuracy**: 64.71%
  + **Loss**: 0.7593
* **8-Layer Model (Epoch 40)**:
  + **Training Accuracy**: 100%
  + **Validation Accuracy**: 55.88%
  + **Loss**: 0.0384

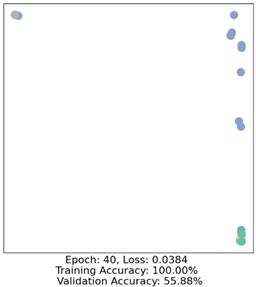
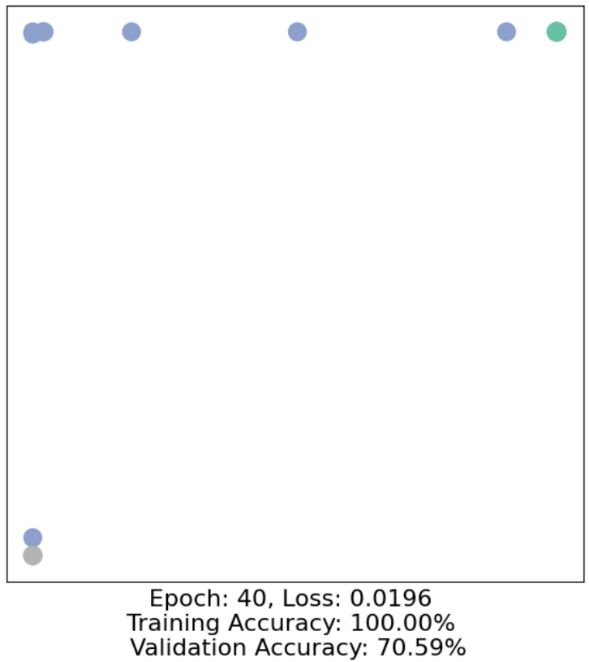
**Observations**

1. **Training Accuracy**: Both models reach 100% training accuracy, indicating they can perfectly fit the training data.
2. **Validation Accuracy**: The 3-layer model outperforms the 8-layer model (64.71% vs. 55.88%), suggesting better generalization to unseen data with fewer layers.
3. **Loss**: The 8-layer model shows a much lower loss than the 3-layer model, but this does not translate into better validation accuracy, indicating potential overfitting with more layers.

**Conclusion**

Increasing the number of layers to 8 from 3 did not improve model performance on unseen data; instead, it likely increased overfitting. The simpler 3-layer model generalizes better despite having a higher loss. This suggests that more complexity (more layers) in this scenario leads to fitting the training data's noise rather than underlying patterns beneficial for generalizing to new data.

1. Add skip connections between some of the GCNConv() layers and try to see if that can improve the model performance



**8 Layers with Skip Connections:**

* **Loss**: 0.0196
* **Training Accuracy**: 100%
* **Validation Accuracy**: 70.59%

**8 Layers without Skip Connections:**

* **Loss**: 0.0384
* **Training Accuracy**: 100%
* **Validation Accuracy**: 55.88%

**Key Takeaways:**

1. **Better Validation Accuracy with Skip Connections**:
   * The model with skip connections has a higher validation accuracy (70.59%) compared to the one without (55.88%). This means the model generalizes better to new data when using skip connections.
2. **Lower Loss with Skip Connections**:
   * The model with skip connections also has a lower loss (0.0196 vs. 0.0384), showing it fits the data better.
3. **Both Models Fit Training Data Well**:
   * Both models reach 100% training accuracy, but the model with skip connections performs better on unseen data.

**Conclusion:**

* Skip connections help improve the model's performance, making it more effective at generalizing to new data.

1. Explain the differences between Message Passing GNN, graph convolution network (GCN), graph attention network (GAT) and GraphSAGE.

**1. Message Passing GNN:**

* **Core Idea**: Message Passing Neural Networks (MPNNs) are a general framework for learning on graphs. They work by nodes passing messages (i.e., information) to their neighbours. Each node updates its representation by aggregating messages from its neighbors.
* **How it Works**: For each node, it receives "messages" (information) from its neighbors, aggregates these messages, and updates its node features.
* **Key Steps**:
  + **Message Generation**: Nodes send their current feature information to neighbors.
  + **Message Aggregation**: Each node aggregates the messages received from its neighbors.
  + **Update**: The node updates its own feature based on the aggregated messages.

**2. Graph Convolution Network (GCN):**

* **Core Idea**: GCN is a specific type of MPNN where the aggregation of messages from neighbors is done using a weighted sum. These weights are typically based on the degree of the nodes and their connections.
* **How it Works**: Each node updates its representation by taking the weighted average of its neighbors' features. The weights are determined by the structure of the graph.
* **Key Features**:
  + **Convolutional Operation**: Similar to how CNNs operate on images, GCNs apply a convolution-like operation to aggregate neighbor information.
  + **Graph Structure Dependent**: The aggregation uses the graph's adjacency matrix.
* **Pros**: Simple and effective for semi-supervised learning on graph data.
* **Limitations**: GCN assigns equal importance to all neighbors, which may not be optimal for complex graphs.

**3. Graph Attention Network (GAT):**

* **Core Idea**: GAT introduces attention mechanisms to assign different importance (weights) to different neighbors during the message aggregation step.
* **How it Works**: Each node doesn't just average its neighbors' information; it learns to "attend" to the most important neighbors by computing attention scores for each edge.
* **Key Features**:
  + **Attention Mechanism**: Attention weights are learned based on how important each neighbor is for a given node.
  + **Adaptive**: GAT can give more weight to important neighbors and less to unimportant ones.
  + **Edge-Specific**: The attention mechanism is edge-specific, allowing GAT to capture complex node relationships better.
* **Pros**: GAT can capture which neighbors are more important for a node, leading to better performance in many cases.
* **Limitations**: More computationally expensive due to the attention mechanism.

**4. GraphSAGE:**

* **Core Idea**: GraphSAGE is another MPNN variant that focuses on efficient, scalable representation learning by sampling a fixed-size neighborhood rather than using the full set of neighbors for each node.
* **How it Works**: Instead of aggregating information from all neighbors, GraphSAGE samples a subset of neighbors and learns an aggregation function (e.g., mean, LSTM, pooling).
* **Key Features**:
  + **Neighborhood Sampling**: Rather than using all neighbors, GraphSAGE samples a fixed number of neighbors to aggregate information, making it scalable to large graphs.
  + **Inductive Learning**: GraphSAGE is designed for inductive learning, meaning it can generalize to unseen nodes (e.g., in dynamic graphs).
  + **Learnable Aggregators**: GraphSAGE allows for more flexibility in how to aggregate information (mean, max-pooling, etc.).
* **Pros**: Scalable to large graphs and can generalize to unseen data.
* **Limitations**: Performance depends on the choice of aggregation method and the sampling process.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Key Feature | Aggregation Method | Main Benefit |
| Message Passing GNN | General framework for message passing | Varies (depends on model) | General-purpose, flexible |
| GCN | Weighted average aggregation | Weighted sum | Simple and effective |
| GAT | Attention mechanism for neighbors | Attention-weighted aggregation | Handles complex relationships |
| GraphSAGE | Samples fixed-size neighborhood, scalable | Learnable aggregation (mean, max) | Scalable to large graphs |

# **GITHUB REPO LINK :** [**https://github.com/IT21194726/DL\_Lab6.git**](https://github.com/IT21194726/DL_Lab6.git)