Name: **Zahra Soukhtedel**
Student Id: **98105138**

Assignment Number: **5**
Course: **Deep Learning**

Problem 1

1.1

1. SimCLR (Simultaneous Contrastive Learning of Representations) is a self-supervised learning method that uses contrastive learning. It requires negative samples to achieve better outputs for the following reasons:

a. Learning meaningful representations: Negative samples help the model learn more meaningful and discriminative representations by forcing it to distinguish between similar and dissimilar images. This process helps the model capture the essential features and subtle differences between images, leading to more informative representations.

b. Contrastive loss optimization: The contrastive loss function used in SimCLR aims to minimize the distance between positive pairs (similar images) and maximize the distance between negative pairs (dissimilar images). Negative samples provide the necessary contrast for the model to optimize this loss function effectively. Without negative samples, the model would not have a reference point to differentiate between similar and dissimilar images, making it difficult to learn meaningful representations.

c. Robustness and generalization: Including negative samples in the training process helps the model become more robust and generalize better to unseen data. By learning to differentiate between various negative samples, the model can handle a wide range of data and adapt to different scenarios. This improved generalization capability leads to better performance on downstream tasks.

d. Improved performance on downstream tasks: Empirical results have shown that using negative samples in self-supervised learning methods leads to improved performance on downstream tasks, such as classification, object detection, and segmentation. This is because the learned representations capture more meaningful information about the data, which can be leveraged for various tasks.

In conclusion, negative samples are crucial in methods like SimCLR because they help the model learn discriminative representations, enable the contrastive loss function to work effectively, improve generalization, and ultimately lead to better performance on downstream tasks.

1.2

BYOL (Bootstrap Your Own Latent) is another self-supervised learning method that does not rely on negative samples. The main reason for this difference is the approach used to learn representations:

a. Different objective: BYOL uses a different objective function compared to SimCLR. Instead of using contrastive learning with negative samples, BYOL focuses on learning representations by predicting the output of one view of the data (the target) from another view of

the same data (the source) using an online network and a target network. The objective is to minimize the distance between the representations produced by these two networks.
b. Momentum encoder: BYOL employs a momentum encoder, which is a moving average of the online network's weights. This momentum encoder acts as a slowly changing target for the online network to predict, providing a form of consistency regularization. This approach allows BYOL to learn meaningful representations without the need for negative samples.
c. Symmetric loss: BYOL uses a symmetric loss function, which means that it learns representations by predicting the target from the source and vice versa. This bidirectional learning helps the model capture more information about the data, making it less reliant on negative samples.
The main reason why BYOL does not require negative samples is its different approach to learning representations, which focuses on predicting one view of the data from another using an online network and a target network. This method, combined with the momentum encoder and symmetric loss, allows BYOL to learn meaningful representations without the need for negative samples.

1.3

BYOL's robustness to certain hyperparameters, such as batch size and the choice of data augmentations, can be attributed to its unique learning approach. Firstly, BYOL does not need negative samples, making it less sensitive to batch size since the learning signal is not dependent on the number of negative samples in the batch. Contrastive learning methods, on the other hand, may require larger batch sizes to provide enough negative samples for effective learning. Secondly, BYOL uses a momentum encoder, which is a moving average of the online network's weights. This momentum encoder acts as a slowly changing target for the online network to predict, providing a form of consistency regularization. This consistency helps the model to be more stable and less sensitive to the choice of data augmentations, as the learning signal is focused on predicting a consistent target rather than contrasting with negative samples. Additionally, BYOL uses a symmetric loss function that learns representations by predicting the target from the source and vice versa. This bidirectional learning helps the model capture more information about the data and makes it less reliant on specific data augmentations. Finally, BYOL's self-supervised learning objective focuses on minimizing the distance between representations produced by the online network and target network for different views of the same data. This objective is less sensitive to the choice of data augmentations, allowing the model to learn invariant features that are consistent across different views of the data. In conclusion, these factors make BYOL more stable and less sensitive to hyperparameter choices than some other self-supervised learning methods.

1.4

The approach of bringing the representations of large slices (global crops) closer together while keeping the small slices (local crops) close to the global crops but away from each other is based on the idea of capturing both global and local information in the learned representations. The rationale behind this approach can be explained as follows:
1. Global context: Large slices or global crops of an image capture the overall context and structure of the scene. By encouraging the model to bring the representations of global crops closer together, the model learns to recognize and encode the global context and high-level

semantic information present in the images.
2. Local details: Small slices or local crops of an image capture fine-grained details and local patterns. By encouraging the model to keep the representations of local crops close to the global crops, the model learns to recognize and encode the local details and textures present in the images. This helps the model to capture both high-level semantic information and low-level details in the learned representations.
3. Diversity in local crops: Encouraging the model to move the representations of local crops away from each other helps to capture the diversity of local patterns and textures present in the images. This ensures that the model learns a rich and diverse set of features, making it more robust and better at generalizing to unseen data.
4. Hierarchical feature learning: This approach can be seen as a form of hierarchical feature learning, where the model learns to capture both global and local information in a hierarchical manner. By learning to represent both the overall context and fine-grained details, the model can better understand the structure and content of the images, leading to improved performance on downstream tasks.
Overally, to encode high-level semantic information and low-level texture details, while also capturing diversity in local patterns. This approach can be seen as a form of hierarchical feature learning, allowing the model to better understand image structure and content and leading to improved performance on downstream tasks.

2

2.1

If we introduce graphs as follows (y nodes stands for red nodes in each graph):
first graph :

$$G_1(V_1, E_1), V_1 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, y\}$$

$$E_1 = \{\{x_1, x_2\}, \{x_3, x_2\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_7\}, \{x_7, x_8\}, \{x_5, y\}\}$$

second graph :

$$G_2(V_2, E_2), V_2 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y\}$$

$$E_2 = \{\{x_1, x_2\}, \{x_3, x_2\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_7\}, \{x_6, x_8\}, \{x_7, x_9\}, \{x_8, x_9\}\{x_5, y\}\}$$

first we have :

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, y] \rightarrow [1, 1, 1, 1, 1, 1, 1, 1, 1]$$

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y] \rightarrow [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

After applying first layer we have:

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, y] \rightarrow [2, 3, 3, 3, 4, 3, 3, 2, 2]$$

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y] \rightarrow [2, 3, 3, 3, 4, 4, 3, 3, 3, 2]$$

After applying second layer we have:

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, y] \rightarrow [5, 8, 9, 10, 12, 10, 8, 7, 6]$$

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y] \rightarrow [5, 8, 9, 10, 13, 14, 10, 10, 6, 6]$$

After applying third layer we have:

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, y] \rightarrow [13, 22, 27, 31, 38, 30, 25, 21, 13]$$

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y] \rightarrow [13, 22, 27, 32, 43, 47, 30, 30, 20, 19]$$

Thus after at least 3 layers the y node's embedding gets different.

2.2

To find the random walk transition matrix M, we need to look at the probabilities of moving from one node to another. In a random walk, the probability of moving from one node to another is equal to the inverse of the degree of the node (the number of edges connected to the node). Let's find the degree of each node:
x_1: degree 2 (connected to x_2 and x_3)
x_2: degree 2 (connected to x_1 and x_3)
x_3: degree 3 (connected to x_1, x_2, and x_4)
x_4: degree 1 (connected to x_3)
Now, we can create the transition matrix M:

$$M = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 1 \\ 0 & 0 & \frac{1}{3} & 0 \end{bmatrix}$$

To find the steady-state distribution, we need to find the eigenvector of M that has an eigenvalue of 1. We can do this by solving the equation $(M - I)r = 0$, where I is the identity matrix. After solving this equation, we get the eigenvector r:

$$r = \begin{bmatrix} \frac{2}{5} \\ \frac{2}{5} \\ \frac{3}{5} \\ \frac{1}{5} \end{bmatrix}$$

Now, we need to normalize r using the L2-norm. The L2-norm of r is:

$$\|r\| = \sqrt{\left(\frac{2}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{3}{5}\right)^2 + \left(\frac{1}{5}\right)^2} = \frac{3\sqrt{2}}{5}$$

So, the steady-state distribution r is:

$$r = \begin{bmatrix} \frac{2\sqrt{2}}{3} \\ \frac{2\sqrt{2}}{3} \\ \sqrt{2} \\ \frac{\sqrt{2}}{3} \end{bmatrix} = \begin{bmatrix} 0.94 \\ 0.94 \\ 1.42 \\ 0.47 \end{bmatrix}$$

2.3

2.3.1

The transition matrix of the random walk, denoted as P, can be described using the adjacency matrix A and the degree matrix D. The transition matrix P is an n x n matrix, where n is the

number of nodes in the graph. The element P(i, j) represents the probability of transitioning from node i to node j in one step of the random walk.
To compute the transition matrix P, we need to normalize the adjacency matrix A by the degree matrix D. Since D is a diagonal matrix with the degree of each node on the diagonal, we can compute the inverse of D, denoted as D_{inv}. Then, we can multiply D_{inv} by A to obtain the transition matrix P:

$$P = D_{inv} \times A$$

Here, D_{inv} is the inverse of the degree matrix D, where each diagonal element $D_{inv}(i,i) = \frac{1}{D(i,i)} = \frac{1}{|N_i|}$, and $|N_i|$ is the degree of node i.
In this way, the transition matrix P captures the probability of transitioning from one node to another in a single step of the random walk, taking into account the degree of each node.

2.3.2

In this case, the message passing equation has been modified to include a self-loop with a weight of 0.5. Let's define the modified adjacency matrix A' to account for the self-loop:

$$A'(i,j) = \begin{cases} 0.5, & \text{if } i = j \\ 1, & \text{if } j \in N_i \\ 0, & \text{otherwise} \end{cases}$$

Next, we need to normalize A' using the degree matrix D. To do this, we first create a modified degree matrix D' where D'(i, i) = 2D(i, i) + 1, which accounts for the self-loop weight of 0.5. Finally, we can compute the transition matrix P as the product of the inverse of D' and A':

$$P = D'^{-1}A'$$

This transition matrix P captures the modified random walk probabilities, considering the self-loop with a weight of 0.5.

2.4

To show that all node embeddings will converge to the same vector as $l \to \infty$, we can use the Markov Convergence Theorem. The aggregation function can be represented as a Markov process. Let's first define the transition matrix P, where $P_{ij} = \frac{1}{|N_i|}$ if $j \in N_i$ and 0 otherwise. The matrix P is stochastic (all rows sum to 1) and non-negative.
Now, let's denote $H^{(l)}$ as a matrix where each row i contains the node embedding $h_i^{(l)}$. Given the aggregation function $h_i^{(l+1)} = \frac{1}{|N_i|} \sum_{j \in N_i} h_j^{(l)}$, we can rewrite it in matrix form as:

$$H^{(l+1)} = H^{(l)} \times P$$

Now, we need to show that as $l \to \infty$, the rows of $H^{(l)}$ converge to the same vector:

$$H^{(l)} \to 1 \times v^T$$

where 1 is a column vector of all ones, and v is a row vector that all rows of $H^{(l)}$ converge to. According to the Markov Convergence Theorem, a stochastic, non-negative matrix P converges to a matrix with identical rows as $t \to \infty$ if it is irreducible and aperiodic:

$$P^t \to 1 \times \pi^T$$

where π is the stationary distribution of the Markov chain.
In our case, to show that $H^{(l)}$ converges to the same vector for all nodes, we need to show that the graph is connected (irreducible) and has no periodicity (aperiodic). If the graph is connected, then there exists a path between any two nodes, which makes the transition matrix P irreducible. If the graph is not bipartite (has no 2-coloring), it is aperiodic. For the sake of this demonstration, we assume that the graph has these properties.
Under these conditions, the Markov Convergence Theorem states that P^t converges to a matrix with identical rows as $t \to \infty$. Since the aggregation function can be represented as a Markov process ($H^{(l+1)} = H^{(l)} \times P$), the node embeddings $H^{(l)}$ will also converge to a matrix with identical rows. Thus, all node embeddings will converge to the same vector as $l \to \infty$.

2.5

To design a Graph Neural Network (GNN) that learns to execute the Breadth-First Search (BFS) algorithm, we can define the message function, aggregation function, and update rule as follows:
Message function:
The message function, denoted as M, takes the features of a node and its neighboring nodes and generates messages to be passed along the edges. In this case, we can define the message function as:

$$M(h_u, h_v) = h_v,$$

where h_u is the embedding of the source node u and h_v is the embedding of the target node v. This message function simply forwards the embedding of the target node.
Aggregation function:
The aggregation function, denoted as A, accumulates the messages from neighboring nodes. In this case, we can define the aggregation function as the element-wise maximum, which takes the maximum value from the messages received:

$$A(\{m_1, m_2, ..., m_n\}) = \max(m_1, m_2, ..., m_n),$$

where $\{m_1, m_2, ..., m_n\}$ is the set of messages received by a node from its neighbors. The maximum function ensures that if any of the neighbors have an embedding of 1, the aggregated message will also be 1.
Update rule:
The update rule, denoted as U, updates the node embeddings based on the aggregated messages. In this case, we can define the update rule as:

$$U(h_u, a) = \max(h_u, a),$$

where h_u is the current embedding of node u, and a is the aggregated message. The update rule takes the maximum of the current embedding and the aggregated message, ensuring that if the aggregated message is 1, the updated embedding will also be 1.

This combination of message function, aggregation function, and update rule allows the GNN to learn the BFS algorithm perfectly for 1-dimensional embeddings. At each step, visited nodes will propagate their 1 value to their unvisited neighbors, updating their embeddings to 1, and thus marking them as visited in the BFS algorithm.