GCN Report

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.96	0.96	135
1	0.97	0.97	0.97	184
2	0.97	0.93	0.95	281
3	0.93	0.98	0.96	154
4	0.97	0.99	0.98	115
5	0.99	0.97	0.98	72
6	0.95	0.97	0.96	145
accuracy			0.96	1086
macro avg	0.96	0.97	0.97	1086
weighted avg	0.96	0.96	0.96	1086

Implementation Approach:

1.Data Loading: The code starts by loading the required data from files (cora.content, cora_train.cites, cora_test.cites). This includes node information, labels, attributes, training and test edge data.

2.Data Preprocessing: The loaded data is processed to extract node indices, labels, attributes, and edges. Labels are converted to numerical form using a predefined dictionary (mydict). Attributes are stored as a numpy array. Edge information is processed to construct an adjacency matrix.

Graph Convolutional Network (GCN) Implementation:

The GraphConvolution class defines a single layer of a graph convolutional operation. The GCN class defines the overall GCN architecture, consisting of two graph convolutional layers with ReLU activation and dropout in between. It implements the forward pass method.
Data Normalization: The adjacency matrix is normalized using the normalize_adj function to add self-loops and apply symmetric normalization.

Training and Evaluation Functions: train_model and evaluate_model functions are defined to train and evaluate the GCN model, respectively. The training function updates the model parameters using backpropagation and the Adam optimizer.

Model Initialization and Optimization: The GCN model is initialized with the required parameters including input feature size, hidden layer size, output class size, dropout rate, and number of nodes. An Adam optimizer and cross-entropy loss function are defined for training.

Training Loop: The model is trained for a fixed number of epochs (2000 in this case). In each epoch, the training data is passed through the model, and the loss is computed and printed.

Evaluation: After training, the model is evaluated using the test data, and a classification report is generated and saved to a file (gcn_metrics.txt).

Training Process:

Model Initialization: The GCN model is initialized with the specified parameters including input feature size, hidden layer size, output class size, dropout rate, and number of nodes.

Optimizer and Loss Function: Adam optimizer is chosen for optimization and cross-entropy loss function is used for computing the loss during training.

Training Loop: The model is trained for 2000 epochs. In each epoch, the training data is passed through the model. The loss is computed based on the predictions and actual labels, and gradients are computed for updating the model parameters.

Evaluation: After training, the model is evaluated using the test data. The test data predictions are compared with the actual labels to generate a classification report, which provides metrics such as precision, recall, and F1-score for each class.

Output: During training, loss values are printed for each epoch. After evaluation, a classification report is saved to a file (gcn_metrics.txt) for further analysis.

This implementation follows the typical pipeline for training a graph convolutional network, including data loading, preprocessing, model definition, training, and evaluation.