Assignment 1

This assignment selects the Cora dataset, which is a commonly used citation network dataset in the field of graph neural networks and is highly suitable for GCN training exercises.

https://github.com/kimiyoung/planetoid/tree/master

It can also be used by torch_geometric package.

The Cora dataset contains 2,708 academic papers, categorized into 7 classes: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory. Each paper is represented as a node, and its features are extracted using the bag-of-words model. Since each feature corresponds to the presence or absence of a word, the feature vector of each paper has a dimension of 1,433.

The citation relationships between papers form the edges of the graph. If paper A cites paper B, there is a directed edge from B to A. However, in practical processing, it is usually treated as an undirected edge. The entire dataset forms a graph structure with 2,708 nodes and 5,429 edges.

1. Task Description

The task of this assignment is a node classification task, that is, using the GCN model to classify the paper nodes in the Cora dataset and predict the category to which each paper belongs.

2. Training Requirements

Normalize the node features to eliminate the impact of different feature scales on model training.
Reasonably divide the dataset into training set, validation set, and test set.
Build a GCN model using PyTorch's torch_geometric library. The model should contain at least two GCN convolutional layers. The first layer is used to extract node features, and the second layer is used to output the category prediction results of nodes.
Set appropriate hyperparameters.
Select the cross-entropy loss function as the loss function of the model, and choose an optimizer.
During the training process, record the loss values and accuracies of the training set and validation set every 20 iterations, and plot the loss curve and accuracy curve.

3. Evaluation Metrics

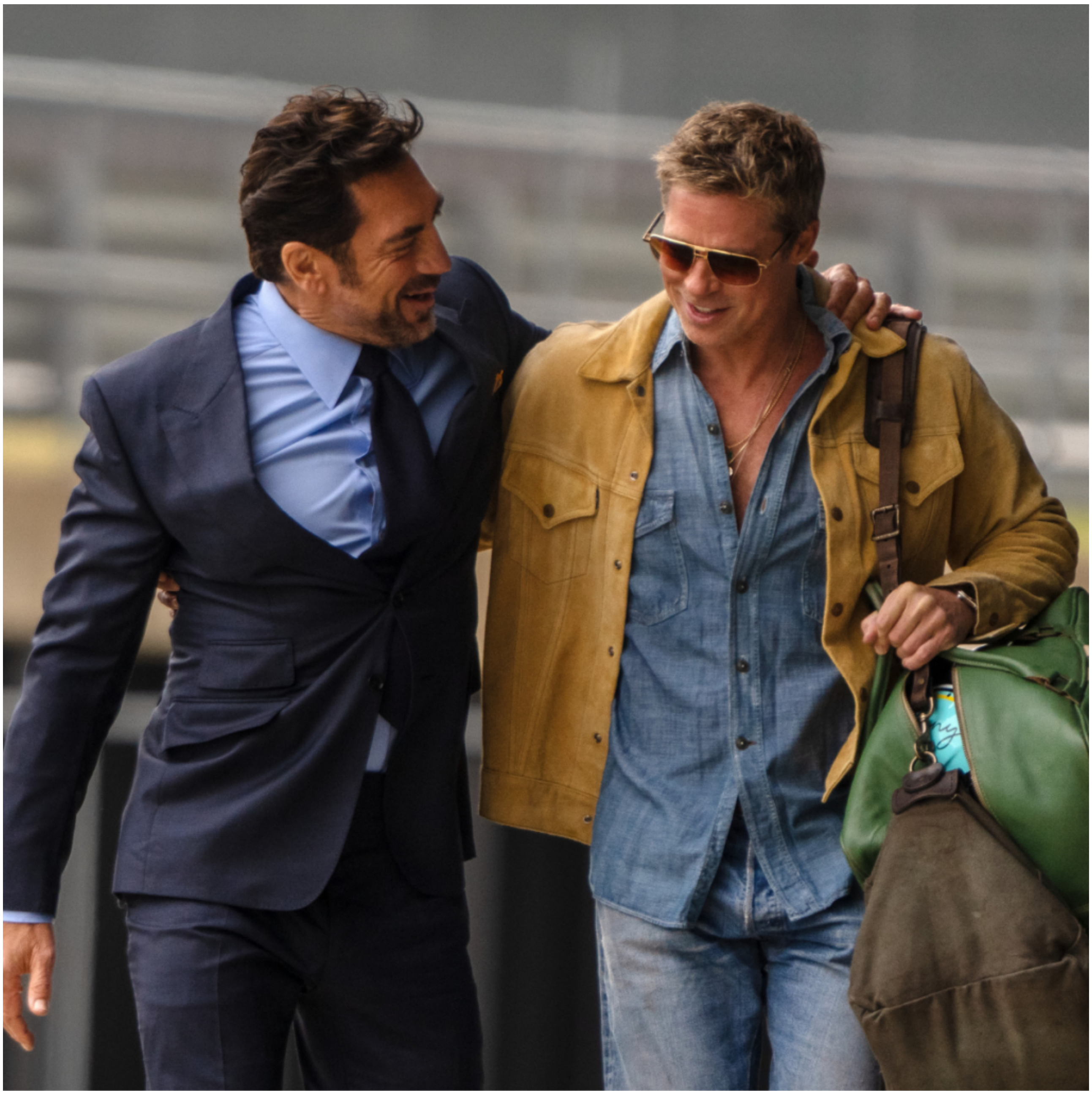
The following metrics are used to evaluate the performance of the model:
Accuracy: The ratio of the number of correctly classified nodes to the total number of nodes.
Precision: The ratio of the number of nodes that truly belong to a certain category among the nodes predicted to be in that category.
Recall: The ratio of the number of nodes correctly predicted in a certain category to the total number of nodes in that category.

F1–Score: The harmonic mean of precision and recall, comprehensively reflecting the classification performance of the model.

4. Submission Requirements
A runnable code file that includes the complete code for data loading, preprocessing, model construction, training, and evaluation, with necessary comments added.
Training logs, including information such as loss values and accuracies of each iteration.
A result analysis report, including the results of various evaluation metrics of the model, analysis of the loss curve and accuracy curve, as well as a summary of the model performance and improvement ideas.

Assignment 2

Use the F1.png image as the input data, along with a 3×3 Sobel convolution kernel.



1. Task Requirements
CPU Serial Convolution Implementation:
Write a C/C++ program to perform serial convolution calculations on the input image. For color images, convolution operations should be carried out separately for the R, G, and B channels. When handling boundary pixels, zero-padding can be used.
Record the time taken for CPU serial computation.

2. CUDA Parallel Convolution Implementation:
Write a program based on CUDA C/C++ to distribute the convolution calculation tasks across multiple threads of the GPU for parallel processing.
Design the division of thread blocks and threads reasonably. For example, each thread block processes 32×32 pixels, and each thread is responsible for the convolution calculation of one pixel.
Optimize memory access, make full use of the GPU's shared memory to reduce the number of accesses to global memory, and improve data access efficiency.
Similarly, process the three channels of the color image, and record the time taken for GPU parallel computation.

3. Performance Comparison and Analysis
Compare the calculation results of the CPU serial implementation and the CUDA parallel implementation to ensure that the error between them is within an acceptable range.
Calculate the speedup ratio of the CUDA parallel implementation relative to the CPU serial implementation.
Analyze the impact of different thread block sizes (such as 16×16, 32×32, 64×64) on the acceleration performance of CUDA convolution, and draw a curve of the speedup ratio changing with the thread block size.
Explore the performance improvement effect of using shared memory, and compare the GPU computation time with and without using shared memory.

4. Submission Requirements
Submit the complete source code for both the CPU serial convolution implementation and the CUDA parallel convolution implementation. The code should include detailed comments explaining the function and implementation ideas of each part.
Submit an experimental report, which should include:
An introduction to the experimental environment, such as CPU model, GPU model, CUDA version, etc.
The algorithm design ideas for the CPU serial implementation and the CUDA parallel implementation.
Performance comparison results, including the computation time of CPU and GPU, speedup ratio, and performance data under different thread block sizes, and attach relevant charts.
Analysis and discussion of the experimental results, summarize the key factors affecting the acceleration performance of CUDA convolution and the optimization methods.
Problems encountered during the experiment and the solutions.

Submit the processed image results, including the results of CPU serial convolution and CUDA parallel convolution, to verify the correctness of the calculations.