Experiment and Technical Report

Method

The model used in this experiment is a DeepLabV3 segmentation model with a ResNet-50 backbone, pretrained on a standard dataset. The model's final classifier layer has been modified to fit the specific number of output classes required for the task. The dataset comprises semantic and instance segmentation masks derived from remote sensing images. A custom dataset class (RemoteSensingDataset) is defined to handle loading and preprocessing of these masks, including resizing, transforming, and simulating point labels.

Experiment

Purpose/Hypothesis

The purpose of this experiment is to evaluate the performance of a modified DeepLabV3 model on the task of semantic and instance segmentation of remote sensing images. The hypothesis is that the pretrained DeepLabV3 model, after fine-tuning with a custom loss function and dataset, will achieve high accuracy and produce meaningful segmentation results.

Experimental Process

1. **Data Preparation**: The semantic and instance segmentation masks are loaded from specified directories. The dataset is split into training (80%) and validation (20%) sets. Data augmentation and transformations (e.g., converting images to tensors) are applied.
2. **Model Initialization**: A pretrained DeepLabV3 model with a ResNet-50 backbone is loaded, and its final classifier layer is adjusted to match the number of output classes required for the segmentation task.
3. **Loss Function and Optimizer**: A custom loss function, PartialCrossEntropyLoss, is defined. The optimizer used is Adam with a learning rate of 1e-4.
4. **Training Loop**: The model is trained for a specified number of epochs. During each epoch, the following steps are performed:
    - The model is set to training mode.
    - For each batch in the training dataset:
        - Inputs (semantic masks, instance masks, and points) are passed through the model.
        - The model outputs are processed using softmax activation.
        - The custom loss function calculates the loss, which is then backpropagated.
        - The optimizer updates the model parameters.
    - Training loss, accuracy, precision, recall, and F1 score are computed and logged.

5. **Validation Loop**: After each training epoch, the model is evaluated on the validation dataset:
    - The model is set to evaluation mode.
    - For each batch in the validation dataset:
        - Inputs are passed through the model, and outputs are processed similarly to the training loop.
        - Validation loss, accuracy, precision, recall, and F1 score are computed and logged.
6. **Metrics and Visualization**: The training and validation losses and accuracies are plotted to visualize the training progress. Precision, recall, and F1 scores are printed after each epoch to evaluate the model's performance on individual classes.

Results

The experiment was conducted over 2 epochs, and the following results were observed:

1. **Training Metrics**:
    - Training Accuracy: 0.92
    - Training Loss: 9.0588
    - Training Precision: 0.8809
    - Training Recall: 0.9195
    - Training F1 Score: 0.8998
2. **Validation Metrics**:
    - Validation Accuracy: 0.94
    - Validation Loss: 8.7794
    - Validation Precision: 0.8893
    - Validation Recall: 0.9430
    - Validation F1 Score: 0.9153
3. **Plots**:
    - **Bar Graph of Training Metrics**
    - **Bar Graph of Validation Metrics**

The results indicate that the model achieved satisfactory performance, with metrics showing trends of improvement over the epochs. However, more epochs may be needed to fully evaluate the

model's capability, and additional fine-tuning of the hyperparameters could further enhance the model's performance.