Car Segmentation and Part Tagging

This project involves implementing a semantic segmentation code using the UNet architecture to segment different parts of cars. The main components of the code include data preparation, model definition, training loop with early stopping, and inferencing. The project consists of training the model on a custom dataset, validating its performance, and performing inference to demonstrate its effectiveness.

The code is divided into 3 python files:

1. dataset.py : this file handles the data preparation process and its required modules.
2. train.py : it contains the code for training of the UNET model and validation.
3. inference.py : this file contains code to perform inference on the test images supplied in a directory.

Key Components:

1. Dataset Preparation (dataset.py):

The **dataset.py** file defines the **SegmentDataset** class, which is a custom PyTorch Dataset for loading and preprocessing images and masks for segmentation tasks. Key components include:

- **Initialization (__init__)**: Initializes paths to images and masks directories, reads filenames, and sorts them. The dataset can be split into training and testing modes using the train flag.
- **Length (__len__)**: Returns the total number of images in the dataset.
- **Get Item (__getitem__)**: Loads an image and its corresponding mask (if in training mode), applies transformations, and returns the processed image, mask, original image dimensions, and filename.

Transformations:

- The **albumentations** library is used to perform transformations like resizing, normalization, and conversion to tensors.

This setup ensures that the input data is consistently preprocessed, enhancing the model's learning capability and generalization.

2. Model Training and Validation (train.py):

The train.py script is responsible for training the UNet model using the prepared dataset. It includes the following key sections:

- **Model Initialization**: The UNet model is initialized with 5 classes and a sigmoid activation function. The encoder of UNET is initialized with the resnet34 model trained on Imagenet. This was done as the number of training images were limited.
- **Data Preparation (get_train_val_data)**: The dataset is divided into training and validation subsets based on the specified proportion. DataLoaders are created for efficient data batching and loading.
- **Training Loop (train)**: Pytorch Lightning framework is used for training the model. The model is trained over a specified number of epochs, optimizing using the AdamW optimizer and monitoring learning rate with CosineAnnealingLR scheduler.
    - **Loss Function**: CrossEntropyLoss is used for multi-class segmentation.
    - **Metric**: Dice coefficient is used as the evaluation metric to measure overlap between predicted and actual segmentation.

Early Stopping: Implemented to halt training if no improvement in validation loss is observed for a set number of epochs.

Overall, this script manages the full training cycle, including model evaluation and early stopping for efficient training.

3. Inference (inference.py)

The inference.py script performs segmentation on new images using the trained model. Key components include:

- **Model Loading**: The trained model is loaded from disk.
- **Data Loading**: A DataLoader is created for the test images.
- **Inference Loop**: For each image, the script:
    - Performs forward pass through the model to get predicted segmentation masks.
    - Processes masks to generate visual outputs including segmentation masks and bounding boxes for detected objects.
    - Saves the output images with segmentation masks and bounding boxes.

- **Visualization (show)**: Utility function to display or save images with overlaid segmentation masks and bounding boxes.

  This script allows the user to visualize the model's performance on unseen data, providing insights into its segmentation accuracy and robustness.

Further Improvements:

1. **Data Augmentation:**
   a. **Advanced Augmentations**: Include more diverse augmentations like random cropping, flipping, rotation, color jittering, elastic transformations, and cutout to improve model robustness and generalization.
   b. **Augmentation Strategies**: Apply augmentation techniques not only during training but also in a way that simulates real-world conditions, such as weather effects (rain, fog), varying lighting conditions, and occlusions.

2. **Model Architecture Improvements:**
   a. Currently, the default model selected as a feature extractor is Resnet34. Experiment with different encoders available such as Resnet50 and analyze the performance.
   b. **Model Complexity**: Explore using more complex UNet variants (e.g., UNet++, Attention UNet) that can capture finer details and context for more accurate segmentation.

3. **Loss Function Optimization:**
   a. **Custom Loss Functions:** Implement and experiment with different loss functions like Dice Loss to improve segmentation accuracy for smaller object classes.

4. **Post-Processing Techniques:**
   a. **CRFs (Conditional Random Fields)**: Apply CRFs as a post-processing step to refine the segmentation boundaries and reduce noise in the predictions.
   b. **Morphological Operations**: Use morphological operations such as dilation and erosion on the predicted masks to enhance the segmentation results.

5. Expand the Dataset:

a. **Diverse Dataset:** Expand the dataset to include images with different car types, colors, angles, and environmental conditions to improve model generalization.

b. **Synthetic Data**: Generate synthetic data to augment the dataset, providing additional scenarios that might not be well-represented in the current data