Image Segmentation with Supervised and Weakly Supervised Methods

This project provides multiple Python scripts for performing image segmentation on the Oxford-IIIT Pet dataset. The implementation covers fully supervised, weakly supervised, and hybrid (combined weakly and fully supervised) approaches. The scripts use deep learning models (based on variants of ResNet50 and DeepLab) implemented in PyTorch, along with utilities for data preparation, training, evaluation, and visualisation.

Table of Contents

Overview

The project implements segmentation models on the Oxford-IIIT Pet dataset using three different methodologies:

- **Weakly Supervised**: Uses no pixel-level annotations from the dataset and leverages class activation maps (CAM) and self-attention models for segmentation.
- **Fully Supervised**: Uses full pixel-level annotations to train a segmentation network.
- **Hybrid Approach**: Combines both weak and full supervision. Two variants are provided:
  - One using parallel processing of weakly supervised (CAM-based) and fully supervised branches.
  - One that integrates weak and full supervision in a slightly different training regime.

A central script (`main.py`) is provided to sequentially run each segmentation model script.

Project Structure

- **main.py**
  Acts as the entry point for the project. It sequentially calls the other scripts to run each segmentation model.

- **oeq_hybrid_parallel_model.py**
  Implements a hybrid segmentation network that utilises a shared encoder (based on ResNet50) and separate branches for weakly supervised (CAM-based) and fully supervised segmentation. The script contains functions for dataset preparation, dataloader creation, model definition, and training with metrics.

- **oeq_weakly_and_supervised_model.py**
  Provides another variant for combining weak and full supervision to perform segmentation. It focuses on a unified segmentation approach by training with both types of annotations and includes logging, metrics calculation, and a different network architecture.

- **supervised_model.py**
  Contains the implementation of a fully supervised segmentation model. It includes custom dataset classes with data augmentation, a simple encoder–decoder network, training routines, evaluation metrics, and loss/accuracy logging.

- **weakly_supervised_model.py**
  Implements a weakly supervised segmentation method using class activation maps (CAMs), self-attention and refinement techniques. It leverages additional libraries (e.g. Albumentations) for data augmentation and custom transforms. The file also includes functions for generating CAMs and refining them using strategies such as ReCAM and self-attention models.

Installation and Dependencies

To run the scripts, please ensure you have the following dependencies installed:

- Python 3.7 or higher
- PyTorch (https://pytorch.org/) and Torchvision (https://pytorch.org/vision/stable/index.html)
- NumPy
- Pillow
- Matplotlib
- Albumentations (for the weakly supervised model)

- OpenCV (for additional image processing)

You can install most dependencies using `pip`:

```
conda create -n comp0197-cw1-pt python=3.12 pip && conda activate comp0197-cw1-pt && pip install torch==2.5.0 torchvision --index-ur
```

```
conda activate comp0197-cw1-pt
```

```
pip install matplotlib albumentations opencv-python
```

Data Preparation

The scripts expect the Oxford-IIIT Pet dataset to be available with the following structure:

- **images:** Original JPEG images.
- **annotations/trimaps:** Corresponding segmentation masks in PNG format.

If the dataset is not already present, the scripts include functions (such as `prepare_dataset`) that will:

- Create the required folder structure.
- Split the dataset into training, validation, and testing sets.
- Copy the images and annotations to their respective directories.

Before running the training scripts, make sure you have downloaded and extracted the dataset archives (for example, `images.tar.gz` and `annotations.tar.gz`) into your project directory.

Usage

You can run the entire segmentation pipeline by executing the main script:

```
python main.py
```

This central script will sequentially execute:

- The weakly supervised model training and evaluation.
- The fully supervised model training and evaluation.
- The hybrid model using parallel branches.
- The alternate hybrid model integrating weak and full supervision.

Alternatively, you may run each script individually during development or testing.

Script Details

main.py

- Purpose: Orchestrates the running of the different segmentation experiments by calling each script sequentially using a subprocess.
- Error Handling: If any script fails, the error is logged and the execution stops.

oeq_hybrid_parallel_model.py

- Purpose: Implements a hybrid segmentation model with a shared encoder (ResNet50) that splits into parallel weakly and fully supervised branches.
- Key Features:
  - Data preparation functions and custom dataloaders.
  - Separate branches for CAM-based and fully supervised segmentation.
  - Custom metric calculations (accuracy, precision, IoU, Dice) and logging.

oeq_weakly_and_supervised_model.py

- Purpose: Provides an alternate approach for integrating weak and full supervision within a unified network architecture.
- Key Features:
  - Similar dataset preparation to the hybrid model.
  - Integrated architecture that efficiently merges information from both weakly supervised and fully supervised paths.
  - Detailed logging and metrics tracking per training epoch.

supervised_model.py

- Purpose: Trains a fully supervised segmentation network using complete pixel-level annotations.
- Key Features:
  - Custom dataset with on-the-fly data augmentation.
  - An encoder–decoder style network designed for segmentation.
  - Functions for training, evaluation, and plotting of loss/accuracy curves.

weakly_supervised_model.py

- Purpose: Focuses on using minimal annotations (just bounding-boxes) and class activation maps (CAMs) to drive segmentation.
- Key Features:
  - Uses Grad-CAM and multi-scale CAM, ReCAM for refinement of CAM's and self-attention maps for weak supervision.
  - Implements additional data augmentations with Albumentations and Custom Transforms.
  - Contains functions for CAM visualisation and refinement (e.g., ReCAM technique).

Logging and Output

Each script includes robust logging functionality:

- Logging Files: Training progress and metrics (loss, accuracy, precision, recall, IoU, Dice) are saved to timestamped log files.
- Plotting: Training and validation loss curves, accuracy plots, and learning rate progress are generated to aid in monitoring performance.
- Model Checkpoints: Model weights are saved after training is completed, allowing for later evaluation or further training.

Licence