Instructions for Weakly Supervised Segmentation with CAM

This document provides detailed steps to run the code and reproduce all reported results.

Additional Packages

Within the comp0197-cw1-pt environment, install the additional required libraries:

```
pip install matplotlib opencv-python pytorch-grad-cam
```

Alternatively, you can use the provided requirements.txt:

```
pip install -r requirements.txt
```

Project Steps

1. Data Preparation and Pre-training Classification Models
2. CAM Evaluation And Extraction
3. Preprocessing CAM files
4. Self-Training to Generate Segmentation Models
5. Evaluation of Segmentation Models
6. Experiments for Potential Open-Ended Questions
    - Experiment 1: Effect of Irrelevant Samples
    - Experiment 2: Effect of Multi-task Training
    - Experiment 3: Effect of Self-training

Step 1: Data Preparation and Pre-training Classification Models

First, prepare the data and train classification models for CAM generation:

```
python pre_training.py
```

This script will:

- Download the Oxford Pet Dataset (if not already downloaded)
- Prepare and split the dataset (split ratio 0.7 : 0.15 : 0.15). Set `use_augmentation=True` if you want to apply data augmentations. Use `target_type=["class", "species", "bbox", "segmentation"]` with the values you would like returned in the target.
- Train various models including CNN and ResNet variants
- Train on different tasks (species classification, breed classification, bounding box regression)
- Train multi-task models with various combinations of tasks
- Save models to the `checkpoints/` directory

Note: Pre-training with non-animal data is also supported

- To use this feature, download the background data using:
  `bg_directory = download_kaggle_dataset("arnaud58/landscape-pictures")`
- Then call:
  `mixed_data.create_mixed_dataloaders(*args, bg_directory=bg_directory, **kwargs)`

Step 2: CAM Evaluation And Extraction

Use trained models to evaluate and generate Class Activation Maps from checkpoints stored in `checkpoints/run_x` (where `run_name` is configurable):

```
python cam_evaluation.py
```

To customize parameters for this step, edit `self_training.py` and modify the following:

- `run_name`: name of the checkpoints sub-folder
- `train_mode`: set to `True` to retrain the model or `False` to use an existing one
- `get_new_cam`: set to `True` to generate new CAMs or `False` to load existing ones

Step 3: Preprocessing CAM files

- Place the best CAMs file in the folder `cam_data/`. Previously we used:
  `res_species_breed_bbox_50_ClassifierHead(2)_GradCAM_idx46_cams.pt`
- If the CAMs are in 256x256, run `resize_CAM.py` with the appropriate directory to produce `resized_64_species_breed_cam_mask_raw.pt`
- If the CAMs contain black pixels between the boundary and the foreground (we experienced this issue in the past, but it did not occur in our most recent runs), run `cleanup_CAM.py` to obtain
  `resized_64_species_breed_cam_mask.pt`

Step 4: Self-Training

The `self_training.py` file includes the complete self-training process. To adjust parameters:

1. Open `self_training.py`
2. Modify the line:
   `resized_data = torch.load("cam_data/resized_64_species_breed_cam_mask_raw.pt")`
   if your CAM file has a different name
3. Set `Skip_first_round = False` (since we don't have a first-round model yet)
4. Adjust `epochs`, `BOOTSTRAP_ROUNDS` and other parameters as needed
5. Run the script:

```
python self_training.py
```

6. Unet Models trained in each round of each experiment will be saved under the folder `checkpoints/Bootstrap/model`. To save time for further experiments, you can set `Skip_first_round = True`, if `first_round_model.pt` is saved under `checkpoints/EVA/` after the first round of training is completed for any experiment. The first round model is our basic model for segmentation.
7. Find the best model with highest IOU score, move it to `checkpoints/EVA/` and rename it as `best_model_selftrain.pt`

This process:

1. Uses CAM as initial pseudo-labels
2. Trains a U-Net segmentation model on these labels
3. Predicts segmentation labels and processes with different filtering strategies
4. Adds these predictions to the training set as new pseudo labels
5. Retrains the model using new pseudo labels
6. Repeats for the specified number of rounds
7. Creates visualisations in the `Bootstrap/evaluation` and `Bootstrap/predicted_masks` directory

Step 5: Evaluation of Segmentation Models

1. Train a baseline model
   1. In `baseline_training.py`, set the desired epochs like `epochs=5`. If you want to train from scratch, set `epochs_previous=0`. Otherwise you can load a previously trained baseline model and continue on training.
   2. baseline model will be saved under `checkpoints/EVA/`
2. Evaluate models
   1. In `final_evaluation_models.py`, modify the file name `model_name=f"first_round_model"` to the model that you want to evaluate under the folder `checkpoints/EVA/`
   2. run `final_evaluation_models.py`, it will evaluate the model both on the small validation set and the testing set.

Step 6: Experiments for Potential Open-Ended Questions

Experiment 1: Effect of Irrelevant Samples

Test if adding irrelevant samples (images without pets) helps improve CAM quality:

```
python mixed_data.py
```

This script:

- Downloads landscape images as background/irrelevant samples
- Creates mixed datasets with pet and landscape images. Need to download the background images with the following code `bg_directory = download_kaggle_dataset("arnaud58/landscape-pictures")`. Then pass `bg_directory=bg_directory` into the `create_mixed_dataloaders` function.
- Creates dataloaders suitable for training

Experiment 2: Effect of Multi-task Training

`pre_training.py` already generates models for multi-task training. The different model variants include:

- Single-task: cnn_species, cnn_breed, cnn_bbox, res_species, res_breed, res_bbox
- Two-task combinations: cnn_breed_species, cnn_breed_bbox, cnn_species_bbox, etc.
- Three-task combinations: cnn_species_breed_bbox, res_species_breed_bbox

Compare results using `pretraining.json` in the `logs/` folder.

Experiment 3: Effect of Self-training

In `self_training.py`, the self-training process generates the basic and many other segmentation models with different strategies of self-training. Variants include:

- Data Iteration: In each iteration, we can either use new labels to replace the current labels, or treat them as a new dataset on top of our original dataset and feed into training loop all together (this will increase size of dataset in training)
- SeedingLoss: A training technique that only calculates loss on part of the pixels (those with pseudo labels generated with high confidence level, namely those with very high predicted probability and very low probability)
- Add-groundtruth: In some experiments we tried to add 100 samples with ground-truth labels, and feed into the training loop

- FilterType: How to process the predicted labels generated from current model to get new pseudo labels
    - Basic: Simple apply a filter (0.2) that filter out predicted pixel labels with value lower than a threshold
    - MixLabel: We invented this algorithm to update labels on top of previous labels. In which predicted pixel labels larger than a high threshold (0.9) will be converted to 1 and that lower than a low threshold (0.1) will be converted to 0, and both be applied to the original labels (compared with basic filter, which only uses predicted labels and not the original labels)
    - GrabCut: A computer vision algorithm that takes the image and the preliminary mask, and generate binary mask based on the features of the image such as shape, colour, etc. We used a 10 percentile threshold, which takes the top 10% pixels and the least 10% pixels from the probability mask, as frontground and background, and feed into the GrabCut algorithm.

Directory Structure

- `oxford_pet_data/` : Dataset directory
- `checkpoints/` : Saved model checkpoints
- `checkpoints/Bootstrap/` : Saved self-training models and images (directory will be automatically created when running self-training)
- `checkpoints/EVA/` : Saved basic and baseline models
- `visualizations/` : Generated visualization outputs
- `logs/` : Training logs
- `cam_data/` : Generated CAMs and tools for CAMs pre-processing

Key Files Description

- `data.py` : Oxford Pet Dataset handling
- `pre_training.py` : Pre-training classification models
- `models.py` : Model architectures (CNN, ResNet, U-Net)
- `self_training.py` : Self-training implementation to train basic and best(self-training) models
- `baseline_training.py` : Training a baseline model
- `final_evaluation_models.py` : Evaluating basic, best and baseline models on the small validation set and testing set
- `evaluation.py` : Evaluation metrics and functions
- `mixed_data.py` : Mixed dataset handling with background images
- `utils.py` : Utility functions