# COMP0197 CW2

This file contains instruction for executing code for COMP0197 Applied Deep Learning coursework 2.

## Setting up Environment

### Prerequisites

* Python: Version 3.x recommended.
* pip: Python package installer.
* Git: For cloning the repository (if not already downloaded).
* (Optional): CUDA-enabled GPU for significantly faster training and evaluation.

### Base Environment

To create a base conda environment, please execute the below script.

conda create -n comp0197-cw2-pt python=3.12 pip && conda activate comp0197-cw2-pt && pip install torch==2.5.0 torchvision --index-url https://download.pytorch.org/whl/cpu

### Extra Dependencies

In addition to above base packages, this codebase also requires below packages to be installed.

pip install scikit-image torchmetrics

## Script Execution Instructions

### Fully Supervised Model

Fully supervised models are trained to provide comparison with weakly supervised models. To train a fully supervised segmentation model:

python -m supervised.train --model=segnet

If you're using a MacBook with an M1, M2, or M3 chip, we suggest enabling CPU fallback for better compatibility:

PYTORCH\_ENABLE\_MPS\_FALLBACK=1 python -m supervised.train --model=segnet

Available segmentation model options are: [segnet, segnext, effunet, unet]

All available command argument for ***train***:

* **model**: Segmentation model name, allowed value ['segnet', 'segnext', 'effunet', 'unet']
* **pseudo**: boolean, whether to use pseudo mask
* **pseudo\_path**: path where pseudo mask is saved
* **verbose**: boolean, whether to print verbose message
* **collate\_contour**: boolean, whether to collapse contour class into foreground

### Weakly Supervised Model

NOTE: Below command are designed to be executed in sequence. Breaking the sequence may result in receiving errors due to saved model weights not found.

#### Finetune classifier

Run without regularisation: python -m cam.finetune

Run with regularisation:

* python -m crm.train\_cam\_with\_crm
* python -m crm.evaluate\_with\_crm

All available command argument for ***train\_cam\_with\_crm***:

* **model**: Classification model name, allowed value ['resnet', 'resnet\_drs']
* **cls\_lr**: Classifier learning rate
* **rec\_lr**: Reconstruction net learning rate
* **vgg\_weight**: Weight for VGG loss
* **align\_weight**: Weight for alignment loss
* **epochs**: Number of training iteration

All available command argument for ***evaluate\_with\_crm***:

* **model**: Classification model name, allowed value ['resnet', 'resnet\_drs']

Get pseudo masks generated

Run without regularisation: python -m cam.postprocessing --model=resnet

Run with regularisation: python -m cam.postprocessing --model=resnet\_crm

All available command argument for ***postprocessing***:

* **mode**: Classification model name, allowed value ['resnet', 'resnet\_crm', 'resnet\_drs']

Sample heatmap images are generated in the `cam/output/cam\_grid.jpg`

#### Train the SegNet segmentation model with pseudo masks

If classifier used in previous steps is simple resnet, model and pseudo masks are save in cam/saved\_models folder:

python -m supervised.train --model=segnet --pseudo --pseudo\_path=cam/saved\_models/resnet\_pet\_cam\_pseudo.pt

If classifier used earlier is with regularisation, i.e. resnet\_crm or resnet\_drs, model and pseudo masks are save in crm\_models/ folder:

python -m supervised.train --model=segnet --pseudo --pseudo\_path=cam/saved\_models/resnet\_pet\_gradcampp\_crm\_pseudo.pt

All available command argument for ***train***:

* **model**: Segmentation model name, allowed value ['segnet', 'segnext', 'effunet', 'unet']
* **pseudo**: boolean, whether to use pseudo mask
* **pseudo\_path**: path where pseudo mask is saved
* **verbose**: boolean, whether to print verbose message
* **collate\_contour**: boolean, whether to collapse contour class into foreground

### Ablation Experiments

Ensure the classification model is saved under cam/saved\_models prior to running below code.

To perform grid search: python -m ablation.grid\_search

All available command argument for ***grid\_search***:

* **model**: Segmentation model name, allowed value ['segnet', 'segnext', 'effunet', 'unet']
* **model\_path**: Path where trained classifier model is saved
* **result\_path**: Path to search results.

### Open-ended Section

This section details the steps to download data, prepare labels (if needed), train models with different supervision strategies, and evaluate them.

#### Step 1: Download Data

This script checks if the Oxford-IIIT Pet Dataset has been downloaded already. If not, download it.

python -m open\_ended.download\_data

Note: This command will download the dataset into a directory (likely `./data` based on subsequent commands). Ensure you have sufficient disk space and an internet connection.

#### Step 2: Generate Weak Labels (Optional, Attached in submission)

This script generates the sparse weak annotations (points, scribbles, boxes) from the ground-truth segmentation masks provided by the original dataset.

python -m open\_ended.weak\_label\_generator --data\_dir ./data/oxford-iiit-pet --output\_file ./open\_ended/weak\_labels/weak\_labels\_train.pkl

* **data\_dir** ./data/oxford-iiit-pet: Specifies the directory where the Oxford-IIIT Pet dataset was downloaded (input).
* **output\_file** ./open\_ended/weak\_labels/weak\_labels\_train.pkl: Specifies the path to save the generated weak labels as a Python pickle file (output).

**Important Note:** Pre-generated weak labels (***weak\_labels\_train.pkl***) are already included in the ***./open\_ended/weak\_labels/*** directory within the repository. Therefore, running this step is generally NOT necessary unless you want to regenerate the labels with different parameters or settings.

#### Step 3: Train Segmentation Models

This is the core step where the segmentation model is trained using different weak supervision configurations. The script ***open\_ended/train.py*** is used repeatedly with different arguments.

##### Common Training Arguments:

* **supervision\_mode** [mode]: \*\*Crucial argument.\*\* Specifies the type of weak supervision to use. Examples below use `points`, `scribbles`, `boxes`, and various `hybrid\_...` combinations.
* **run\_name** [name]: A unique name for this specific training run. Used for logging and naming checkpoint files (e.g., `segnet\_points\_run1`).
* **data\_dir** ./data/oxford-iiit-pet: Path to the dataset directory.
* **weak\_label\_path** ./open\_ended/weak\_labels/weak\_labels\_train.pkl: Path to the file containing the weak labels.
* **batch\_size** 64: Number of samples per batch during training. Adjust based on GPU memory.
* **lr** 2e-4: Learning rate for the optimizer.
* **epochs** 25: Number of training epochs.
* **num\_workers** 8: Number of worker processes for data loading. Adjust based on your system's CPU cores.
* **img\_size** 256: Resize input images to this square dimension.
* **checkpoint\_dir** [path]: Directory where model checkpoints (saved model weights) will be stored.

##### Training with Single Weak Supervision Types:

These commands train separate models, each using only one type of weak annotation. Checkpoints are saved in ***./checkpoints\_single.***

###### Points:

python -m open\_ended.train \

--supervision\_mode points \

--run\_name segnet\_points\_run1 \

--data\_dir ./data/oxford-iiit-pet \

--weak\_label\_path ./open\_ended/weak\_labels/weak\_labels\_train.pkl \

--batch\_size 64 \

--lr 2e-4 \

--epochs 25 \

--num\_workers 8 \

--img\_size 256 \

--checkpoint\_dir ./checkpoints\_single

###### Scribbles:

python -m open\_ended.train \

--supervision\_mode scribbles \

--run\_name segnet\_scribbles\_run1 \

--data\_dir ./data/oxford-iiit-pet \

--weak\_label\_path ./open\_ended/weak\_labels/weak\_labels\_train.pkl \

--batch\_size 64 \

--lr 2e-4 \

--epochs 25 \

--num\_workers 8 \

--img\_size 256 \

--checkpoint\_dir ./checkpoints\_single

###### Bounding Boxes:

python -m open\_ended.train \

--supervision\_mode boxes \

--run\_name segnet\_boxes\_run1 \

--data\_dir ./data/oxford-iiit-pet \

--weak\_label\_path ./open\_ended/weak\_labels/weak\_labels\_train.pkl \

--batch\_size 64 \

--lr 2e-4 \

--epochs 25 \

--num\_workers 8 \

--img\_size 256 \

--checkpoint\_dir ./checkpoints\_single

Training with Hybrid Weak Supervision Types:

These commands train models using combinations of weak annotation types. Checkpoints are saved in ***./checkpoints\_hybrid***.

###### Points + Scribbles:

python -m open\_ended.train \

--supervision\_mode hybrid\_points\_scribbles \

--run\_name segnet\_hybrid\_points\_scribbles\_run1 \

--data\_dir ./data/oxford-iiit-pet \

--weak\_label\_path ./open\_ended/weak\_labels/weak\_labels\_train.pkl \

--batch\_size 64 \

--lr 2e-4 \

--epochs 25 \

--num\_workers 8 \

--img\_size 256 \

--checkpoint\_dir ./checkpoints\_hybrid

###### Points + Boxes:

python -m open\_ended.train \

--supervision\_mode hybrid\_points\_boxes \

--run\_name segnet\_hybrid\_points\_boxes\_run1 \

--data\_dir ./data/oxford-iiit-pet \

--weak\_label\_path ./open\_ended/weak\_labels/weak\_labels\_train.pkl \

--batch\_size 64 \

--lr 2e-4 \

--epochs 25 \

--num\_workers 8 \

--img\_size 256 \

--checkpoint\_dir ./checkpoints\_hybrid

###### Scribbles + Boxes:

python -m open\_ended.train \

--supervision\_mode hybrid\_scribbles\_boxes \

--run\_name segnet\_hybrid\_scribbles\_boxes\_run1 \

--data\_dir ./data/oxford-iiit-pet \

--weak\_label\_path ./open\_ended/weak\_labels/weak\_labels\_train.pkl \

--batch\_size 64 \

--lr 2e-4 \

--epochs 25 \

--num\_workers 8 \

--img\_size 256 \

--checkpoint\_dir ./checkpoints\_hybrid

###### Points + Scribbles + Boxes:

python -m open\_ended.train \

--supervision\_mode hybrid\_points\_scribbles\_boxes \

--run\_name segnet\_hybrid\_points\_scribbles\_boxes\_run1 \

--data\_dir ./data/oxford-iiit-pet \

--weak\_label\_path ./open\_ended/weak\_labels/weak\_labels\_train.pkl \

--batch\_size 64 \

--lr 2e-4 \

--epochs 25 \

--num\_workers 8 \

--img\_size 256 \

--checkpoint\_dir ./checkpoints\_hybrid

#### Step 4: Evaluate Trained Models

After training, use the ***open\_ended/evaluate.py*** script to evaluate the performance of the saved model checkpoints on the test set.

python -m open\_ended.evaluate \

--data\_dir ./data/oxford-iiit-pet \

--model\_paths

checkpoints\_single/segnet\_boxes\_run1\_best\_acc.pth \

checkpoints\_single/segnet\_points\_run1\_best\_acc.pth \

checkpoints\_single/segnet\_scribbles\_run1\_best\_acc.pth \

checkpoints\_hybrid/segnet\_hybrid\_points\_boxes\_run1\_best\_acc.pth \

checkpoints\_hybrid/segnet\_hybrid\_points\_scribbles\_boxes\_run1\_best\_acc.pth \

checkpoints\_hybrid/segnet\_hybrid\_points\_scribbles\_run1\_best\_acc.pth \

checkpoints\_hybrid/segnet\_hybrid\_scribbles\_boxes\_run1\_best\_acc.pth \

--batch\_size 8 \

--device cuda

##### Evaluation Arguments:

* **data\_dir** ./data/oxford-iiit-pet: Path to the dataset directory.
* **model\_paths** [path1] [path2] ...: \*\*Important:\*\* List the paths to the specific checkpoint files (`.pth`) you want to evaluate. These should typically be the checkpoints saved based on the best validation performance during training (e.g., `\_best\_acc.pth` or similar, as indicated by the filenames). Ensure these paths correctly point to the files generated in Step 4.
* **batch\_size** 8: Batch size for evaluation. Can often be larger than training batch size depending on GPU memory.
* **device** cuda: Specifies the device for evaluation. Use `cuda` for GPU or `cpu` for CPU.

Note: The second `evaluate.py` command listed in the original README under a separate "Evaluate" heading seems potentially inconsistent or refers to different models/runs not shown in the main training sequence. This rewritten guide focuses on evaluating the models trained in Step 4 above.