Domain Adaptation for 3D Object Detectors

Version: 22/03/2023 (Christoph Staubmann)

Bachelor's project

Institute of Computer Graphics and Vision, Graz University of Technology, Austria

Student: **Christoph Staubmann**

Advisor: **Dipl.-Ing. BSc Christian Fruhwirth-Reisinger**

Supervisor: **Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof**

This file contains a brief overview of the most important commands we used in the bachelor's project **Domain Adaptation for 3D Object Detectors**. A `.pdf` version of this file can be found here: `OpenPCDet/docs/IMPORTANT_COMMANDS.pdf`.

Please refer to the paper for more information!

Important commands

1. Create Dataset Infos

First, we need to extract the data from our desired dataset and transform it into a common format, the info-files. We do this for the respective datasets **ONCE** and **CADC** with the following commands:

ONCE:

```
python -m pcdet.datasets.once.once_dataset
  --func create_once_infos
  --cfg_file tools/cfgs/dataset_configs/once_dataset.yaml
```

CADC:

```
python -m pcdet.datasets.cadc.cadc_dataset
  --func create_cadc_infos
  --cfg_file tools/cfgs/dataset_configs/cadc_dataset.yaml
```

2. Extract Split Information

Similary, by passing a different function argument, we can extract data split information. For us, the most metrics are **Average data points in groundtruth** and **Average 3d bounding box sizes** for each class.

ONCE:

```
python -m pcdet.datasets.once.once_dataset
  --func extract_split_infos
  --cfg_file tools/cfgs/dataset_configs/once_dataset.yaml
```

CADC:

```
python -m pcdet.datasets.cadc.cadc_dataset
  --func extract_split_infos
  --cfg_file tools/cfgs/dataset_configs/cadc_dataset.yaml
```

3. Training

Refer to the paper for concrete training parameters used for training the models! For training, we run **tools/train.py** as follows:

ONCE:

```
python train.py --cfg_file ./cfgs/once_models/sup_models/second.yaml
python train.py --cfg_file ./cfgs/once_models/sup_models/pointpillar.yaml --batch_size 1
python train.py --cfg_file ./cfgs/once_models/sup_models/centerpoints.yaml --batch_size 2
```

CADC:

```
python train.py --cfg_file ./cfgs/cadc_models/second.yaml
python train.py --cfg_file ./cfgs/cadc_models/pointpillar.yaml
python train.py --cfg_file ./cfgs/cadc_models/centerpoints.yaml --batch_size 2
```

4. Evaluation

Similarly to training, we can run **tools/test.py** with the desired configuration to evaluate the last N saved checkpoints from training (defined by the configuration):

Here, we the argument **`--eval_all`** performs the evaluation for the last N saved checkpoints, defined by the configuration. By default, only the last 30 training epochs will be saved. Using **`--eval_all`**, we can further define the starting epoch for evaluation with the command **`--starting_epoch X`**, where X is the desired checkpoint number.

ONCE:

```
python test.py --cfg_file ./cfgs/once_models/sup_models/second.yaml --eval_all
python test.py --cfg_file ./cfgs/once_models/sup_models/pointpillar.yaml --eval_all
python test.py --cfg_file ./cfgs/once_models/sup_models/centerpoints.yaml --eval_all
```

CADC:

```
python test.py --cfg_file ./cfgs/cadc_models/second.yaml --eval_all
python test.py --cfg_file ./cfgs/cadc_models/pointpillar.yaml --eval_all
python test.py --cfg_file ./cfgs/cadc_models/centerpoints.yaml --eval_all
```

In case we only want to evaluate a single checkpoint, e.g., the last training epoch, we can pass the desired checkpoint via the parameter **`--ckpt`**:

```
python test.py
  --cfg_file ./cfgs/once_models/sup_models/second.yaml
  --ckpt ../output/cfgs/once_models/sup_models/second/default/ckpt/checkpoint_epoch_80.pth

python test.py
  --cfg_file ./cfgs/cadc_models/second.yaml
  --ckpt ../output/cfgs/cadc_models/second/default/ckpt/checkpoint_epoch_80.pth
```

5. Evaluation On Target Dataset

Similarly to the normal evaluation, we can evaluate the checkpoints on the target dataset. For this, we use different configuration files, e.g. **second_once_to_cadc.yaml**. These altered configuration files then use the respective other dataset and define the required mapping of class names between the datasets and further parameters relevant for DA:

```
python test.py
  --cfg_file ./cfgs/once_models/second_once_to_cadc.yaml
  --ckpt ../output/cfgs/once_models/sup_models/second/default/ckpt/checkpoint_epoch_80.pth
  --extra_tag "eval DA"
```

As we adjusted the configuration for each experiment, we can save them in different output folder by running the command with a different **`--extra_tag [TAG]`** or **`--eval_tag [TAG]`**:

```
--extra_tag "eval DA shifted coordinates"
--extra_tag "adjusted voxel settings" --eval_tag "60k voxels"
--extra_tag "adjusted voxel settings" --eval_tag "150k voxels"
```

6. Demo / Visualization

For visualizing the point cloud, we can use the adjusted demo file **tools/demo_plane3d.py** with the following minimal syntax:

```
python demo_plane3d.py --cfg_file [CONFIGURATION] --ckpt [CKPT] --data_path [DATA]
```

ONCE:

```
python demo_plane3d.py
  --cfg_file ./cfgs/once_models/sup_models/second.yaml
  --ckpt ../output/cfgs/once_models/sup_models/second/default/ckpt/checkpoint_epoch_80.pth
  --data_path ../data/once
```

Replace the respective arguments to run the demo with different configuration and datasets. Note, that the default paths for ONCE and CADC are **OpenPCDet/data/once** and **OpenPCDet/data/cadc**, or **../data/once** and **../data/cadc** as relative paths from the demo file.

Furthermore, we can add the following optional arguments to the above command:

- **`dont_draw_scenes`**: dont draw scenes from the dataset (e.g., skip visualization when computing ground planes)
- **`draw_gt_boxes`**: draw groundtruth boxes in scene
- **`fit_plane`**: compute and draw best fitting plane for points pointcloud
- **`log_file`**: filename (without extension) to save logger output to disk (OpenPCDet/output directory)

7. Few-Shot Training & Evaluation

For the few-shot training DA approach we need to create a small train split with N labeled target dataset frames first. We do this by the same process explained in **1. Create Dataset Infos**, but with an additional argument.

For this, we can pass the parameter **`--few_shot_frames N`**, where N is the number of target dataset frames. The labeled target dataset frames are sampled from the entire dataset at equal distances between frames. In our case, this is only supported for the ONCE dataset (i.e., the task **CADC -> ONCE**), as few-shot DA was not tested for CADC, but can be implemented similarly.

ONCE (only):

```
python -m pcdet.datasets.once.once_dataset
  --func create_once_infos
  --cfg_file tools/cfgs/dataset_configs/once_dataset.yaml
  --few_shot_frames 100
```

After generating the small target dataset train split, we can train the detector as explained in **3. Training**.

Important: Note that you need to change the DATA_CONFIG field in the respective dataset config file (e.g., **`tools/cfgs/cadc_models.centerpoints.yaml`**). Within this file, change **`cfgs/dataset_configs/cadc_dataset.yaml`** to **`cfgs/dataset_configs/once_dataset.yaml`**, as we now train the detector on the respective other dataset (ONCE):

```
...
DATA_CONFIG:
    _BASE_CONFIG_: cfgs/dataset_configs/once_dataset.yaml
...
```

For example, for our few-shot training task **CADC -> ONCE**, we:

- create a small train split with N frames on **ONCE** (create_once_infos)!
- train a detector (e.g., SECOND) with the source config from CADC on the new data split from ONCE. Defined epochs are the total epochs, i.e. the model is trained for only 20 few-shot epochs. Note, that you need **skip_eval** here, as we will use a different config for that:

  ```
  python train.py
    --cfg_file ./cfgs/cadc_models/centerpoints.yaml
    --ckpt ../output/cfgs/cadc_models/centerpoints/default/ckpt/checkpoint_epoch_80.pth
    --extra_tag "ONCE few-shot/100 frames"
    --epochs 100 --skip_eval
  ```

After a short training time, we get the saved checkpoints for this model. In our case, the output is saved to **`output/cfgs/cadc_models/centerpoints/ONCE few-shot/`**. Using our DA dataset configuration files, i.e. **`centerpoints_cadc_to_once.yaml`** in this case, we can evaluate the performance of our few-shot trained model on ONCE:

```
python test.py
  --cfg_file ./cfgs/cadc_models/centerpoints_cadc_to_once.yaml
  --ckpt_dir ../output/cfgs/cadc_models/centerpoints/ONCE\ few-shot/100\ frames/ckpt/
  --extra_tag "ONCE few-shot/100 frames" --eval_tag "eval on ONCE"
  --eval_all
```

Changelog

[2023-03-22] Initial version.

Acknowledgements

- OpenPCDet is an open source project for LiDAR-based 3D scene perception that supports multiple LiDAR-based perception models as shown above. Some parts of PCDet are learned from the official released codes of the above supported methods.
- ONCE dataset
- ONCE devkit (OpenPCDet)
- CADC dataset
- CADC devkit (OpenPCDet)