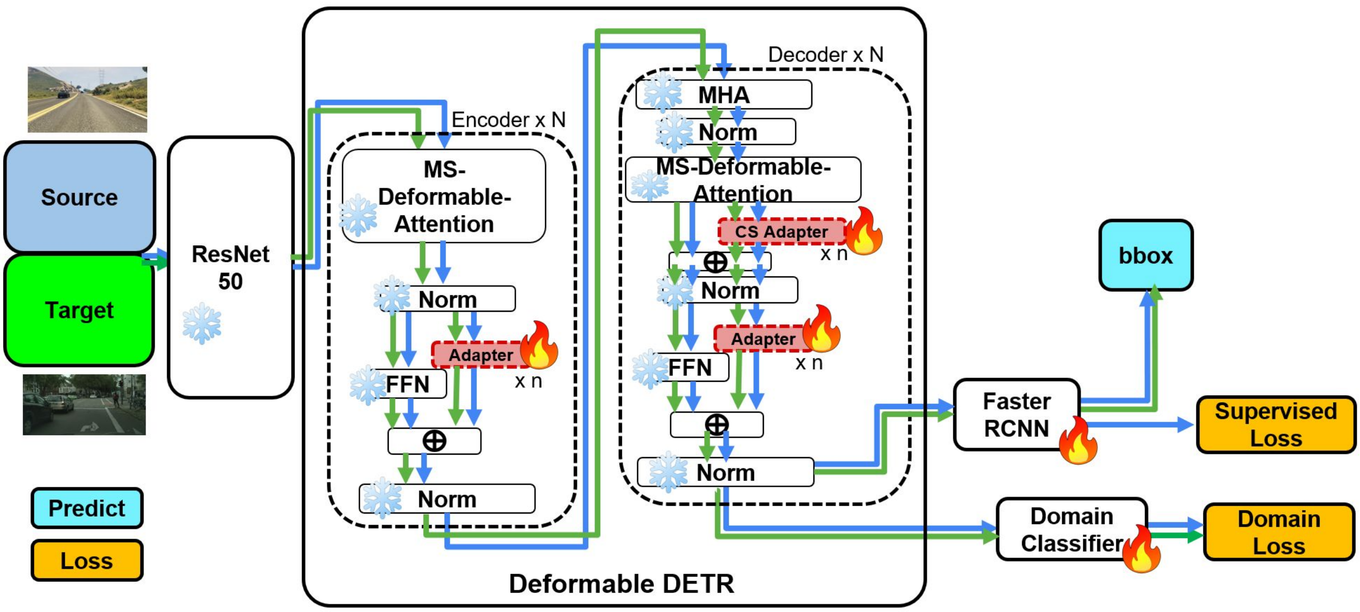
Officail Code for "Efficient Transfer Learning with Spatial Attention and Channel Communication for Unsupervised Domain Adaptation in Object Detection"

▶ Table of Contents

Overview

Code release for the paper: https://drive.google.com/file/d/19ZY15MAgTQxjjQTUtHEn1vZDbd18qiny/view?usp=sharing



Overview of our proposed architecture, which is based on DeformableDETR. The FG Adapter, depicted in the orange dotted box, is integrated into both the encoder and decoder of Deformable-DETR, while the CS Adapter, shown in the red dotted box, is only in the decoder. The domain classifier, indicated by the purple dotted box, is placed after the decoder.

Repository Structure

```
└── DA_stuff/
    ├── README.md
    ├── experiment_saved
    │   ├── CITY2FOGGY_baseline_and_pretrained_on_source
    │   ├── CITY2FOGGY_oracle_and_trained_on_source_and_target
    │   ├── CITY2FOGGY_with_Dcls
    │   ├── CITY2FOGGY_with_Dcls_channel_mixing
    │   ├── CITY2FOGGY_with_Dcls_channel_mixing_spatail_attention
    │   ├── CITY2FOGGY_with_Dcls_spatail_attention
    │   ├── SIM2CITY_baseline_and_pretrained_on_source
    │   ├── SIM2CITY_oracle_and_trained_on_source_and_target
    │   ├── SIM2CITY_with_Dcls
    │   ├── SIM2CITY_with_Dcls_channel_mixing
    │   ├── SIM2CITY_with_Dcls_channel_mixing_spatail_attention
    │   └── SIM2CITY_with_Dcls_spatail_attention
    ├── experiment_saved_future_work
    │   ├── CITY2FOGGY_with_Dcls_channel_mixing_spatail_attention_TINY_GT_LABEL
    │   └── SIM2CITY_with_Dcls_channel_mixing_spatail_attention_TINY_GT_LABEL
    ├── figures
    │   ├── framework.png
    │   └── performance.png
    ├── projects
    │   ├── __init__.py
    │   ├── configs
    │   └── models
    │       ├── co_detr.py # Architecture are in here
    │       └── _transformer.py # Adapters are in here
    ├── requirements.txt
    └── tools
        ├── train.py
        ├── experiment_CITY2FOGGY.sh
        └── experiment_SIM2CITY.sh
```

Getting Started

System Requirements:

- **Python**: `version 3.7.11`
- **GPU**: `NVIDIA GeForce RTX 3090 Ti`

Installation

1. Clone the DA_stuff repository:

```
$ git clone https://github.com/Flame1045/DA_stuff.git
```

2. Change to the project directory:

```
$ cd DA_stuff
```

3. Install the dependencies:

```
$ conda create -n your_repo_name python==3.7.11 -y
$ conda activate your_repo_name
$ conda install pytorch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0
cudatoolkit=11.3 -c pytorch -y
$ pip install -r requirements.txt
$ pip install -U openmim
$ mim install mmengine
$ pip install mmcv-full==1.5.0 -f
https://download.openmmlab.com/mmcv/dist/cu113/torch1.10.0/index.html
$ pip install -v -e .
$ pip install fairscale
$ pip install timm
$ pip3 install natten==0.14.6+torch1110cu113 -f https://shi-
labs.com/natten/wheels
$ pip install tensorboard
```

4. Install pretrained weight:

Cilck here https://drive.google.com/file/d/1ezCc0LeGXj_7uTVBLKknJWufHAQt0TGL/view?usp=sharing to download and extract and merge files in experiment_saved. It will looks like the structure below

```
└── DA_stuff/
    ├── experiment_saved
        ├── [Experiment_name]
            ├── pretrained
                └── XXX.pth # saved pretrained weight
            ├── XXX.log # log
            ├── XXX.py # config
            └── XXX.pth # saved experiment weight
    ...
```

```
$ git clone https://github.com/Flame1045/DA_stuff.git
```

Dataset preparation

Download SIM10k, CITYSCAPES, FOGGY CITYSCAPES from its offical website, convert it to COCO format. Make directory "./data/" and put dataset in below sturcture.

```
└── DA_stuff/
    ├── data
        └── coco
            ├── City2Foggy_source # CITYSCAPES dataset
            ├── City2Foggy_target # FOGGY CITYSCAPES dataset
            ├── SIM2Real_source     # SIM10k dataset
            └── SIM2Real_target     # CITYSCAPES dataset
```

Config Details

▶ experiment_saved.SIM2CITY_baseline_and_pretrained_on_source

File	Summary
custom_sim2city_base.py	`SIM2CITY pretrained on source`

▶ experiment_saved.SIM2CITY_with_Dcls

File	Summary
custom_sim2city_unsupervised_base_wA_woCTBV2_B4.py	`SIM2CITY with domain classifier`

▶ experiment_saved.SIM2CITY_with_Dcls_spatail_attention

File	Summary
custom_sim2city_unsupervised_base_wA_woCTBV2_B4.py	`SIM2CITY with spatail attention`

▶ experiment_saved.SIM2CITY_with_Dcls_channel_mixing

File	Summary
custom_sim2city_unsupervised_base_wA_woCTBV2_B4.py	`SIM2CITY with channel communication`

▶ experiment_saved.SIM2CITY_with_Dcls_channel_mixing_spatail_attention

File	Summary
custom_sim2city_unsupervised_base_wA_woCTBV2_B4.py	`SIM2CITY with channel communication and spatail attention`

▶ experiment_saved.SIM2CITY_oracle_and_trained_on_source_and_target

File	Summary
custom_sim2city_unsupervised_base_wA_woCTBV2_B4_ORALCLE.py	`SIM2CITY oralcle`

▶ experiment_saved.CITY2FOGGY_baseline_and_pretrained_on_source

File	Summary

File	Summary
custom_city2foggy_base.py	`CITY2FOGGY pretrained on source`

▶ experiment_saved.CITY2FOGGY_with_Dcls

File	Summary
custom_city2foggy_unsupervised_base_wA_woCTBV2_B4.py	`CITY2FOGGY with domain classifier`

▶ experiment_saved.CITY2FOGGY_with_Dcls_spatail_attention

File	Summary
custom_city2foggy_unsupervised_base_wA_woCTBV2_B4.py	`CITY2FOGGY with spatail attention`

▶ experiment_saved.CITY2FOGGY_with_Dcls_channel_mixing

File	Summary
custom_city2foggy_unsupervised_base_wA_woCTBV2_B4.py	`CITY2FOGGY with channel communication`

▶ experiment_saved.CITY2FOGGY_with_Dcls_channel_mixing_spatail_attention

File	Summary
custom_city2foggy_unsupervised_base_wA_woCTBV2_B4.py	`CITY2FOGGY with channel communication and spatail attention`

▶ experiment_saved.CITY2FOGGY_oracle_and_trained_on_source_and_target

File	Summary
custom_city2foggy_unsupervised_base_wA_woCTBV2_B4_ORALCLE.py	`CITY2FOGGY oralcle`

Usage

Evaluation & Visualization on `CITY2FOGGY`

We provide six different experimental setups for evaluating the model on the CITY2FOGGY task. These setups are included in the script `tools/experiment_CITY2FOGGY.sh`.

The available experiments are:

- **CITY2FOGGY_baseline_and_pretrained_on_source**
- **CITY2FOGGY_oracle_and_trained_on_source_and_target**
- **CITY2FOGGY_with_Dcls_channel_mixing_spatial_attention**
- **CITY2FOGGY_with_Dcls**
- **CITY2FOGGY_with_Dcls_channel_mixing**
- **CITY2FOGGY_with_Dcls_spatial_attention**

To evaluate a specific experiment, uncomment (# python3 tools/test.py ...) in the desired experiment in the script and run the following command:

```
$ bash tools/experiment_CITY2FOGGY.sh
```

To visualize specific experiment, add below in script behind python3 tools/test.py ... --eval bbox

```
--show --show-score-thr 0.5 --show-dir your_output_dir
```

Evaluation on & Visualization on `SIM2CITY`

We provide six different experimental setups for evaluating the model on the SIM2CITY task. These setups are included in the script `tools/experiment_SIM2CITY.sh`.

The available experiments are:

- **SIM2CITY_baseline_and_pretrained_on_source**
- **SIM2CITY_oracle_and_trained_on_source_and_target**
- **SIM2CITY_with_Dcls_channel_mixing_spatial_attention**
- **SIM2CITY_with_Dcls**
- **SIM2CITY_with_Dcls_channel_mixing**
- **SIM2CITY_with_Dcls_spatial_attention**

To evaluate a specific experiment, uncomment (# python3 tools/test.py ...) the desired experiment in the script and run the following command:

```
$  bash tools/experiment_SIM2CITY.sh
```

To visualize specific experiment, add below in script behind python3 tools/test.py ... --eval bbox

```
--show --show-score-thr 0.5 --show-dir your_output_dir
```

Training on `CITY2FOGGY`

We provide six different experimental setups for training the model on the CITY2FOGGY task. These setups are included in the script `tools/experiment_CITY2FOGGY.sh`.

The available experiments are:

- **CITY2FOGGY_baseline_and_pretrained_on_source**
- **CITY2FOGGY_oracle_and_trained_on_source_and_target**
- **CITY2FOGGY_with_Dcls_channel_mixing_spatial_attention**
- **CITY2FOGGY_with_Dcls**
- **CITY2FOGGY_with_Dcls_channel_mixing**
- **CITY2FOGGY_with_Dcls_spatial_attention**

To train a specific experiment, uncomment (# CONFIG= ... to # done) in the desired experiment in the script and run the following command:

```
$ bash tools/experiment_CITY2FOGGY.sh
```

Training on `SIM2CITY`

We provide six different experimental setups for training the model on the SIM2CITY task. These setups are included in the script `tools/experiment_SIM2CITY.sh`.

The available experiments are:

- **SIM2CITY_baseline_and_pretrained_on_source**
- **SIM2CITY_oracle_and_trained_on_source_and_target**
- **SIM2CITY_with_Dcls_channel_mixing_spatial_attention**
- **SIM2CITY_with_Dcls**
- **SIM2CITY_with_Dcls_channel_mixing**
- **SIM2CITY_with_Dcls_spatial_attention**

To train a specific experiment, uncomment (# CONFIG= ... to # done) the desired experiment in the script and run the following command:

```
$ bash tools/experiment_SIM2CITY.sh
```

How to modify

CS Adapter is in _transformer.py line 235

```
class MLP_Adapter_slide8(nn.Module):
```

FG Adapter is in _transformer.py line 65

```
def build_Adapter(input_dim, hidden_dim, output_dim):
```

Full architecture workflow is in co_detr.py line 290

```
def forward_train(self, ....
```

Future Work

We incorporate two additional experiments:

1. A tiny percentage of target domain labels is used in our proposed method to evaluate whether the adapter requires labeled data for effective fine-tuning in domain adaptation.
2. We explore a self-training strategy to assess whether the model can improve unsupervised domain adaptation with an enhanced training approach.

Pretrained Weight Preparation

Cilck here https://drive.google.com/file/d/1Wst2HzzjMkm4ryjZTI-GEl_RUDbo6FC9/view?usp=sharing to download and extract in experiment_saved_future_work. It will looks like the structure below

```
└── DA_stuff/
    ├── experiment_saved_future_work
        ├── [Experiment_name]
            ├── XXX.log # log
            ├── XXX.py # config
            └── XXX.pth # saved experiment weight
    ...
```

Tiny Percentage of Target Domain label Experiments

Evaluation on `CITY2FOGGY`

We provide one experimental setup for evaluating the model on the CITY2FOGGY task. This is included in the script `tools/experiment_CITY2FOGGY.sh`.

The available experiments are:

- **CITY2FOGGY_with_Dcls_channel_mixing_spatail_attention_TINY_GT_LABEL**

To evaluate a specific experiment, uncomment (# python3 tools/test.py ...) in the desired experiment in the script and run the following command:

```
$ bash tools/experiment_CITY2FOGGY.sh
```

Results of Cityscapes (Ds) → Foggy Cityscapes (Dt)

Percent of target domain label	person	rider	car	truck	bus	train	motorcycle	bicycle	mAP
0 %	42.7	48.5	56.8	32.7	47.0	32.5	33.0	42.6	42.0
1 %	44.2	50.4	62.7	32.6	46.8	34.9	33.3	42.6	43.4

Evaluation on `SIM2CITY`

We provide one experimental setup for evaluating the model on the CITY2FOGGY task. This is included in the script `tools/experiment_SIM2CITY.sh`.

The available experiments are:

- **SIM2CITY_with_Dcls_channel_mixing_spatail_attention_TINY_GT_LABEL**

To evaluate a specific experiment, uncomment (# python3 tools/test.py ...) in the desired experiment in the script and run the following command:

```
$ bash tools/experiment_SIM2CITY.sh
```

Results of Sim10k (Ds) → Foggy Cityscapes (Dt)

Percent of target domain label	AP
0 %	57.7
1 %	64.8

Training on `CITY2FOGGY`

We provide one experimental setup for training the model on the CITY2FOGGY task. This is included in the script `tools/experiment_CITY2FOGGY.sh`.

The available experiments are:

- **CITY2FOGGY_with_Dcls_channel_mixing_spatail_attention_TINY_GT_LABEL**

To train a specific experiment, uncomment (# CONFIG= ... to # done) in the desired experiment in the script and run the following command:

```
$ bash tools/experiment_CITY2FOGGY.sh
```

Training on `SIM2CITY`

We provide one experimental setup for training the model on the CITY2FOGGY task. This is included in the script `tools/experiment_SIM2CITY.sh`.

The available experiments are:

- **SIM2CITY_with_Dcls_channel_mixing_spatail_attention_TINY_GT_LABEL**

To train a specific experiment, uncomment (# CONFIG= ... to # done) the desired experiment in the script and run the following command:

```
$ bash tools/experiment_SIM2CITY.sh
```

Self-training Strategy Implement Thoughts

Guidelines for Converting This Code to Self-Training

Step 1: Understand the Current Codebase

- **Identify Key Components**:
  - Locate the discriminator and generator/feature extractor (found in `projects/models/da_head.py` at line 113).
  - Review how adversarial loss is computed and integrated (see `projects/models/da_head.py` at line 70).
- **Analyze Data Flow**:
  - Trace how source and target domain data are handled (refer to the config file for each experiment).

Step 2: Remove Adversarial Components

- **Discriminator Removal**:
  - Remove the discriminator network and related loss computations.
- **Feature Extractor Adjustment**:
  - Detach the feature extractor from any adversarial dependencies.

Step 3: Implement Pseudo-Labeling

- **Generate Pseudo-Labels**:
  - Mask target data according to the MIC paper (https://arxiv.org/abs/2212.01322).
  - Use the Student networks to predict target domain labels, including confidence scores.
- **Filter Pseudo-Labels**:
  - Apply a confidence threshold to filter pseudo-labels for training, as outlined in the MIC (https://arxiv.org/abs/2212.01322) paper.
- **Assign Pseudo-Labels**:
  - Store the filtered pseudo-labels for training purposes.

Step 4: Modify the Training Loop

- **Combine Data**:

  - Mix labeled source data with pseudo-labeled target data in the data loader.
  - Use the Teacher networks to generate predicted target labels.

- **Update Loss Function**:
  - Modify the loss function to calculate the loss between predicted target labels and pseudo-labels. This loss is used to update the Student networks.
  - Update the Teacher networks using EMA, as described in the MIC (https://arxiv.org/abs/2212.01322) paper.

Acknowledgments

- SAPNetV2
- MMdetection
- NATTEN
- Professor Wen-Hsien Fang and Professor Yie-Tarng Chen

Return
