자율주행 **DNA:** 차량용 신호등 인식 매뉴얼

삼육구
류한국(팀장), 강경수, 지현동

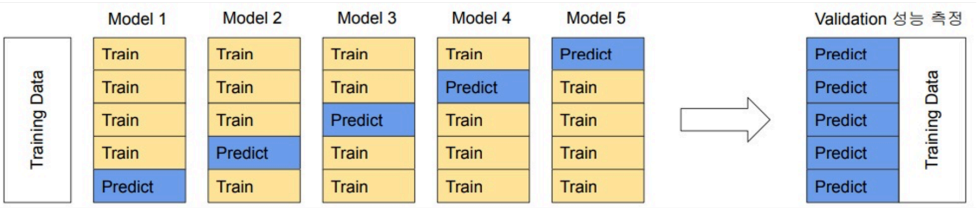
<목차>

1. 신호등 인식 모형 학습 및 추론 전략

A. 데이터셋

i. k=3의 Out-of-fold (OOF) 전략을 선택. 총 세 개의 학습-검증 데이터셋을 만듦. 세 개의 검증 데이터셋은 각각 학습 데이터셋의 10%, 10% 5%로 구성
ii. train dataset의 비슷한 주행 환경에서의 이미지가 100~200 프레임으로 연속됨을 확인
iii. 전체 학습 데이터를 100구간으로 나눈 뒤 그 중 랜덤으로 36구간을 검증 데이터셋의 후보군으로 선정
iv. 12구간씩 36 구간을 각각의 검증데이터셋의 후보군으로 다시 나눔.
v. 각 검증데이터셋은 12구간의 후보군 중에서 연속된 이미지를 검증 데이터셋으로 취함

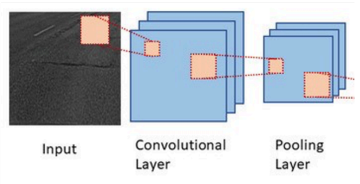
Out-of-fold 전략

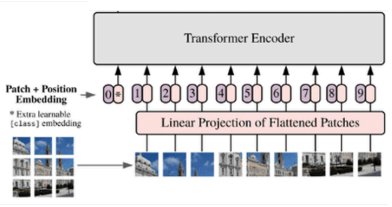


B. 신호등 인식 모형 학습 전략

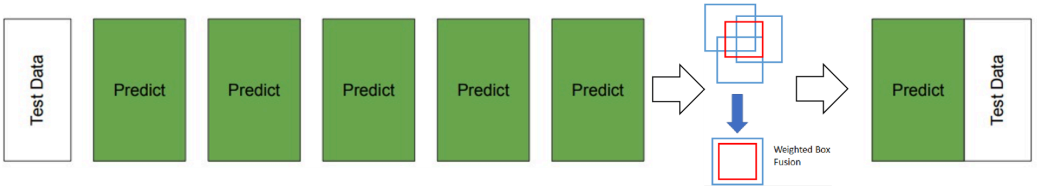
i. 주최측 베이스라인(baseline) 소스코드를 기반으로 재현성을 위해 seed를 고정하고 학습. CNN과 Transformer 기반 모형을 선택하여 앙상블 기법으로 일반화 성능을 높이는 방법 선택
ii. 보유한 학습셋에 기반하여 학습된 결과물은 CNN이 Transformer 아키텍처보다 잘 잡으나, Transformer의 일반화성능이 좋아 학습 데이터셋에 없는 특징. 예를 들어 야간 사진에서 좀 더 강건하게 작동함
iii. 따라서 본 참여팀은 CNN과 Transformer 모형 앙상블을 통해 베이스라인보다 성능 좋은 모형을 개발하고자 함

CNN과 Transformer 아키텍처





WBF 전략을 적용한 추론 성능 향상



C. 신호등 인식 모형 추론 전략

i. 다른 데이터셋 조합으로 학습된 모형들을 앙상블하여 추론 후 예측된 바운딩 박스를 하나의 결과로 합치기 위해 Weight Box Fusion(WBF) 방법 도입

D. 소스코드 다운로드

i. 소스 코드 **URL: http://gofile.me/71Q8k/GFhfqHnfc**

```
mm_log_pre_trained.zip    # mmdetection 로그 파일 및 pretrained weight
split_dataset_src.zip     # 데이터셋 분할 코드
traffic_transformer.zip   # mmdetection 학습 및 추론, WBF
traffic_cnn.zip           # ultralytics 학습 및 추론
ulralytics_log.zip        # ultralytics 로그 파일 및 아티팩트
```

2. 신호등 인식 모형 테스트 환경

A. CNN 기반 탐지기 학습 및 추론 환경

i. 본 참여팀이 신호등 인식 모형 개발을 위해 사용한 학습 및 추론 환경은 다음과 같음

구분	상세 스펙
CPU	Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz
RAM	394GB
GPU	NVIDIA A100 (80GB)
OS	Ubuntu 20.04.6 LTS
SW	CUDA 11.8, PyTorch 2.3.0

B. Transformer 기반 탐지기 학습 및 추론 환경

구분	상세 스펙
CPU	AMD Ryzen 9 3900X 12-Core Processor
RAM	128GB
GPU	NVIDIA RTX Titan (24GB), A6000 (48GB)
OS	Ubuntu 22.04.4 LTS (docker)
SW	CUDA 11.8, PyTorch 2.4.1(conda)

3. 신호등 인식 모형 개발 환경 설정

A. 공통 환경 설정

i. 본 참여팀은 자율주행 DNA 차량용 신호등 인식 과제 모형 개발을 위해 도커 컨테이너 환경을 사용함

ii. NVIDIA GPU용 도커 설치는 nvidia-container-toolkit 문서 참고: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html

```
$ docker pull pytorch/pytorch:2.3.0-cuda11.8-cudnn8-devel

$ docker run -it --ipc=host --gpus=all --name {container_name} -p
  {local_port_no}:22 {image_id} bash

$ docker exec -it {container_name} bash

$ cd workspace


$ mkdir traffic_cnn  # YOLO

$ mkdir traffic_transformer  # Co-DETR, DINO-SWIN
```

B. 학습(train)/검증(val)/평가(test) 데이터 셋 설정

i. 데이터 증강을 위해 albumentations 라이브러리를 활용하여 오프라인으로 학습 데이터의 5% 증강(학습 이미지의 명도와 채도를 조절함)

ii. pip install albumentations로 라이브러리 설치 후 아래 스크립트 실행하여 데이터 증강. 증강한 images와 lables를 아래 3~4 과정 후 생기는 train1/ 폴더에 추가(YOLO 모델에만 데이터 증강 적용함)

```
$ cd traffic_transformer/

$ python data_augmentation.py
```

iii. YOLO 모델에 경우, k=3의 OOF 전략 채택. 다만 동영상에서 프레임을 저장해 연속된(비슷한) 데이터로 구성되어 있지만 개별적

모델링을 위해 `split_3_dataset_traffic_light.py`를 실행하여 아래와 같이 세 세트의 학습/검증 데이터셋을 구성함

iv. 세 개의 YOLO 모델에 세 개의 데이터셋을 각각 사용함(최종 선정 데이터셋은 train1, train2로 학습한 모형만 선택하였음)

```
/workspace/traffic_cnn
 ↳datasets/  # 학습, 검증, 테스트 데이터셋 경로
   ↳data_traffic/
     ↳test/
     ↳train1/
     ↳train2/
     ↳val1/
     ↳val2/
```

v. `mmdetection`의 `transformer` 모델에 경우, OOF 전략 채택 안 함

vi. `traffic_transformer/utils/split_dataset_traffic_light.py` 실행(주의: 복사가 아닌 파일 이동)

```
/workspace/traffic_transformer
 ↳datasets/  # 학습, 검증, 테스트 데이터셋 경로
   ↳data_traffic/
     ↳test/
     ↳train/
     ↳val/
```

C. CNN 기반 모델 개발 환경 설정

i. 본 팀의 CNN 기반 탐지기 모델 개발을 위해 `conda` 가상환경과 `ultralytics` 내 YOLO 모형들을 사용하였음

ii. 설치된 conda package는 `environments.yml` 파일 참조

iii. 첨부한 traffic_cnn.zip 파일을 ./ 현재 경로에 압축 풀어서 위의 파일 및 폴더 구조와 일치하는 지 확인

```
$ pwd # 경로확인
workspace/traffic_cnn
$ vim /root/.config/Ultralytics/settings.json
"datasets_dir": "/workspace/traffic_cnn/datasets" # 확인
$ conda create -n {env_name} python=3.11
$ conda activate {env_name}
$ conda install pytorch==2.3.0 torchvision==0.18.0 pytorch-cuda=11.8
  -c pytorch -c nvidia
$ pip install ultralytics tqdm … # environments.yml 설치하면 됨
```

```
datasets/  # 학습, 검증, 테스트 데이터셋 경로
  ↳data_traffic/
    ↳test/
    ↳train/
    ↳train1/
    ↳train2/
    ↳val/
    ↳val1/
    ↳val2/
runs/  # 학습 모형 아티팩트
  ↳detect/
    ↳tld_yolov11x_d1/
    ↳tld_yolov11x_d2_2/
    ...
results/  # 추론 결과 .txt 파일들
  ↳tld_yolov11x_d1
  ↳tld_yolov11x_d2_2
```

```
  ...
train_traffic_1.py
train_trafiic_2.py
traffic_2024_1.yaml
traffic_2024_2.yaml
train.sh
test_traffic.py
yolov10x.pt  # pretrained weights
yolov11x.pt  # pretrained weights
...
```

D. Transformer 기반 모델 개발 환경 설정

i. `mmdetection` 설치는 공식 문서 참고 (https://mmdetection.readthedocs.io/en/latest/get_started.html)

ii. **AssertionError: MMCV==2.2.0 is used but incompatible. Please install mmcv>=2.0.0rc4, <2.2.0.**
위와 같은 Error 발생 시 mmdetection/mmdet/__init__.py 의 9번 줄을 mmcv_maximum_version = '2.2.1'로 변경.
(ref: https://github.com/open-mmlab/mmcv/issues/3096)

iii. 설치된 conda package는 conda_list.yml, conda_list.txt 파일 참조

iv. mmdetection 공식 소스 코드를 git clone하여 일부 파일을 수정하거나, 제출한 압축파일의 traffic_transformer 소스 코드 다운로드 후 압축 풀기

```
$ conda create -n {env_name} python=3.8
$ conda activate {env_name}
$ conda install pytorch==2.4.1 torchvision==0.19.1 torchaudio==2.4.1
```

```
  pytorch-cuda=11.8 -c pytorch -c nvidia
$ pip install -U openmim
$ mim install mmengine
$ mim install "mmcv>=2.0.0"
$ git clone https://github.com/open-mmlab/mmdetection.git
$ cd mmdetetction
$ pip install -v -e .
```

4. 신호등 검출 모델 학습

A. ultralytics 모델 학습

i. 아래는 CNN 기반 신호등 인식 모형 학습을 위한 ultralytics 프로젝트 파일 및 폴더 구조

ii. 두 모형에 대한 하이퍼 파라미터 관리를 위해 .py 스크립트를 두 개와 데이터셋 config.yaml 파일 두 개로 관리. 최종 학습은 train.sh 쉘 스크립트 파일로 학습 돌리면 됨

1. train_traffic_1.py: YOLOv10x 기반 pretrained yolov10x.pt 사용
2. train_traffic_2.py: YOLOv11x 기반 pretrained yolov11x.pt 사용
   a. train_traffic_2.py 데이터셋을 train1과 train2로 교차 선택해서 학습함

```python
if __name__ == "__main__":
    model = YOLO('yolov11x.pt')
    model.train(
        # name="tld_yolov11x_d1_cos",
        name="tld_yolov11x_d2_",
        # data='./traffic_2024_1.yaml',
        data='./traffic_2024_2.yaml',
        epochs=70,
        batch=64,
```

vi. 여러 실험(다양한 YOLO 모형 학습)을 통해서 가장 좋은 성능을 달성한 yolov11x_d2_2와 yolov11x_d1_cos 모형 사용

1. yolov11x_d1_cos: OOF 2번째 데이터셋에 코사인 어닐링 적용한 모형이라는 뜻임
2. yolov11x_d2_2: OOF 3번째 데이터셋을 학습한 모형을 뜻함

```
$ pwd # 경로확인
workspace/traffic_cnn
$ sh train.sh  # 모형 학습 실행

# 학습 후 제출파일 생성 및 시각화
$ python test_traffic.py
```

```
$ python visualize.py
```

B. mmdetection coco dataset 수정

i. mmdetection/mmdet/datasets/coco.py 의 METAINFO를 신호등 인식 데이터셋에 맞게 아래와 같이 수정.

```
METAINFO = {

'classes' : (

       "Veh_go", "veh_goLeft", "veh_noSign", "veh_stop",

       "veh_stopLeft", "veh_stopWarning", "veh_warning",

       "ped_go","ped_noSign", "ped_stop", "bus_go",

       "bus_noSign", "bus_stop", "bus_warning",

       ),

'palette':

        [(220, 20, 60), (119, 11, 32), (0, 0, 142), (0, 0, 230), (106, 0,
  228),

         (0, 60, 100), (0, 80, 100), (0, 0, 70), (0, 0, 192), (250, 170,
  30),

        (100, 170, 30), (220, 220, 0), (175, 116, 175), (250, 0, 30),]

}
```

C. mmdetection DINO-SWIN 모델 학습

iii. mmdetection/configs/dino/dino-4scale_r50_8xb2-12e_coco.py 의 62번째 줄 **num_classes=14** 로 변경

iv. 아래 경로에서 dino_swin.py 파일 생성 및 config 작성(부록 **[1]** 참조).
(config 수정 시 경로 유의: data_root, ann_file, data_prefix)

```
$ cd mmdetection/configs
```

```
$ mkdir traffic_light
$ cd traffic_light  # dino_swin.py 파일 생성 및 작성.
```

v. 아래와 같이 config인자에 dino_swin.py 경로를 입력하여 tain.py 실행
vi. traffic_transformer/work_dirs에 log 파일 생성됨.
vii. 아래 pre_trained 모델에 경우 train.py 실행 시 자동으로 받음.
pre_trained checkpoint:
https://github.com/SwinTransformer/storage/releases/download/v1.0.0/swin_large_patch4_window12_384_22k.pth

```
$ cd ../  # mmdetection/
$ python tools/train.py configs/traffic_light/dino_swin.py
```

D. mmdetection CO-DETR 모델 학습

i. co_dino_5scale_r50_lsj_8xb2_1x_coco.py의
9번째 줄 num_classes=14로 변경
ii. 아래의 경로에서
co_dino_5scale_swin_l_16xb1_16e_o365tococo.py를 파일 경로 주의하며 config 수정(부록 [2] 참조)
(config 수정 시 경로 유의: data_root, ann_file, data_prefix)

```
$ cd mmdetection/projects/CO-DETR/configs/codino
```

iii. 아래와 같이 config인자에 co_dino…py경로를 입력하여 tain.py 실행
iv. traffic_transformer/work_dirs에 log 파일 생성됨.
v. 아래 pre_trained 모델에 경우 train.py 실행 시 자동으로 받음.
pre_trained checkpoint:
https://download.openmmlab.com/mmdetection/v3.0/codetr/co_dino_5scale_swin_large_16e_o365tococo-614254c9.pth
https://github.com/SwinTransformer/storage/releases/download/v1.0.0/swin_large_patch4_window12_384_22k.pth

```
$ cd ../  # mmdetection/

$                          python                          tools/train.py
  projects/CO-DETR/configs/codino/co_dino_5scale_r50_lsj_8xb2_1x_coco
  .py
```

5. 신호등 인식 모델 추론 및 앙상블 전략

A. `ultralytics` 모델 추론

i. 학습된 모형은 내부 평가를 통해 yolovx11_d1_cos와 yolov11_d2_2로 선정함

ii. 모델의 상대경로는 다음과 같음:
"runs/detect/tld_yolov11x_d1_cos/weights/best.pt"`
"runs/detect/tld_yolov11x_d2_2/weights/best.pt"

iii. `test_traffic.py`의 `main()`함수 내 `model_filename` 변수를 변경하면서 추론함

```
$ pwd  # 현재 경로 확인
/workspace/traffic_cnn
$ python test_traffic.py # best.pt 상대경로를 바꿔가면서 추론
```

B. `mmdetection` DINO-SWIN 모델 추론

i. test_dino_swin.py 실행. 경로 문제시 --config, --checkpoint, --out 의 default 값을 변경하거나, 파이썬 스크립트 실행시 각각의 경로를 입력하여 실행

ii. 추론 완료 후 traffc_transformer/outputs/epoch_4_dino_swin.pkl 생성됨.

iii. pkl2txt_traffic_light.py 를 실행하여 pkl파일을 대회 제출 포맷인 txt 파일로 변경
(nms_iou_threshold=0.01, score_threshold=0.0 으로 설정)

```
$ python tools/test_dino_swin.py
$ python pkl2txt_traffic_light.py
```

C. mmdetection CO-DETR 모델 추론

i. test_co_detr.py 실행. 경로 문제시 --config, --checkpoint, --out의 default 값을 변경하거나, 파이썬 스크립트 실행시 각각의 경로를 입력하여 실행

ii. 추론 완료 시 traffc_transformer/outputs/epoch_3_co_detr.pkl 생성됨.

iii. pkl2txt_traffic_light.py를 실행하여 pkl파일을 대회 제출 포맷인 txt 파일로 변경.

```
$ python tools/test_co_detr.py
$ python pkl2txt_traffic_light.py
```

D. Weighted Boxes Fusion

i. WBF은 모든 경계 박스의 신뢰도 점수를 활용하여 평균화된 박스를 구성하는 앙상블 방법. 앙상블시 모형별로 서로 다른 가중치 입력 가능. 가중치를 통해 각 모델의 영향력 조절 가능.

ii. Ultralytics conda 가상환경에 pip install ensemble_boxes 라이브러리 설치.

```
$ conda activate {env_name}  # activate the ultralytics env
$ pip install ensemble_boxes
```

iii. 각기 다른 모형으로 추론한 파일들을 아래와 같은 디렉토리 구조로 구성. 각 컨테이너에서 추론한 결과 파일들을 해당 폴더로 옮긴 다음에 WBF 실행해야 함

```
traffic_transformer/datasets/
 results/
   ↳tld_yolov11x_d1_cos/
     ↳10000000.txt
     ...
   ↳tld_yolov11x_d2_2/
     ↳10000000.txt
```

```
    ...
  ↳co_detr/
    ↳10000000.txt
    ...
  ↳dino_swin/
    ↳10000000.txt
    ...

ensemble.py
test_traffic.py
image_sizes.pkl   # 앙상블 시 필요. ensemble.py에서 save_image_sizes()로 생성.
```

iv. ensemble.py 내의 pred_dirs에 추론 txt 파일들의 경로를 아래 순서대로 입력.
WBF 가중치 적용시 인덱스 번호로 매칭되므로 인덱스 순서가 중요.

```
def ensemble_wbf_boxes():
   pred_dirs = [
       "datasets/results/tld_yolov11x_d2_2",
       "datasets/results/tld_yolov11x_d1_cos",
       "datasets/results/co_detr_3e_nms_0.001",
       "datasets/results/dino_swin_4e_nms_0.001",
   ]
```

v. weighted_boxes_fusion 함수의 인자는 다음과 같이 설정함
(…, weights=[1, 1, 2, 1], iou_thr=0.25, skip_box_thr=0)

```
       boxes, scores, labels = \
              weighted_boxes_fusion(boxes, scores, labels, weights=[1, 1, 2, 1], iou_thr=0.25, skip_box_thr=0)
```

vi. `./pred_ensemble`에 앙상블을 수행한 최종 결과 제출 파일(.txt)이 생성됨

Appendix

[1] /workspace/traffic_transformer/configs/traffic_light/dino_swin.py

```
_base_ = [
    '../dino/dino-4scale_r50_8xb2-12e_coco.py',
]


########################## dataset ###############################

dataset_type = 'CocoDataset'

data_root    =    './datasets/data_traffic/'        #    path/to/dataset    e.g)
./datasets/data_traffic/ <- 마지막에 슬래쉬 꼭 넣을 것.


metainfo = {
    'classes' : (
        "veh_go",
        "veh_goLeft",
        "veh_noSign",
        "veh_stop",
        "veh_stopLeft",
        "veh_stopWarning",
        "veh_warning",
        "ped_go",
        "ped_noSign",
        "ped_stop",
        "bus_go",
        "bus_noSign",
        "bus_stop",
        "bus_warning",
    )
}
```

```
backend_args = None
# train_pipeline, NOTE the img_scale and the Pad's size_divisor is different
# from the default setting in mmdet.
train_pipeline = [
   dict(type='LoadImageFromFile', backend_args={{_base_.backend_args}}),
   dict(type='LoadAnnotations', with_bbox=True),
   dict(type='RandomFlip', prob=0.5),
   dict(
       type='RandomChoice',
       transforms=[
           [
               dict(
                   type='RandomChoiceResize',
                       scales=[(480, 1333), (512, 1333), (544, 1333), (576,
1333),
                               (608, 1333), (640, 1333), (672, 1333), (704,
1333),
                           (736, 1333), (768, 1333), (800, 1333)],
                   keep_ratio=True)
           ],
           [
               dict(
                   type='RandomChoiceResize',
                   # The radio of all image in train dataset < 7
                   # follow the original implement
                   scales=[(400, 4200), (500, 4200), (600, 4200)],
                   keep_ratio=True),
               dict(
                   type='RandomCrop',
                   crop_type='absolute_range',
                   crop_size=(384, 600),
```

```
                    allow_negative_crop=True),
                dict(
                    type='RandomChoiceResize',
                        scales=[(480, 1333), (512, 1333), (544, 1333), (576,
1333),
                                (608, 1333), (640, 1333), (672, 1333), (704,
1333),
                            (736, 1333), (768, 1333), (800, 1333)],
                    keep_ratio=True)
            ]
        ]),
    dict(type='PackDetInputs')
]
train_dataloader = dict(
    batch_size=2,
    num_workers=2,
    dataset=dict(
        data_root=data_root,
        metainfo=metainfo,
        ann_file='train/train.json',  # 경로 주의
        data_prefix=dict(img='train/images/'),  # 경로 주의
        ))


val_dataloader = dict(
    dataset=dict(
        data_root=data_root,
        metainfo=metainfo,
        ann_file='val/val.json',  # 경로 주의
        data_prefix=dict(img='val/images/'),))  # 경로 주의


test_pipeline = [
```

```
    dict(type='LoadImageFromFile', backend_args=backend_args),
    dict(type='Resize', scale=(1333, 800), keep_ratio=True),
    # If you don't have a gt annotation, delete the pipeline
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='PackDetInputs')]

test_dataloader = dict(
    batch_size=1,
    num_workers=2,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        ann_file='test/test.json',  # 경로 주의
        data_prefix=dict(img='test/images/'),  # 경로 주의
        test_mode=True,
        pipeline=test_pipeline))

val_evaluator = dict(
    ann_file=data_root + 'val/val.json',)  # 경로 주의

test_evaluator = dict(
    type='CocoMetric',
    metric='bbox',
    format_only=True,
    ann_file=data_root + 'test/test.json',  # 경로 주의
    outfile_prefix='./work_dirs/dino_swin_test')
```

```
######################## model ##############################

pretrained                                                                =
'https://github.com/SwinTransformer/storage/releases/download/v1.0.0/swin_la
rge_patch4_window12_384_22k.pth'  # noqa

num_levels = 5

model = dict(

   num_feature_levels=num_levels,

   backbone=dict(

       _delete_=True,

       type='SwinTransformer',

       pretrain_img_size=384,

       embed_dims=192,

       depths=[2, 2, 18, 2],

       num_heads=[6, 12, 24, 48],

       window_size=12,

       mlp_ratio=4,

       qkv_bias=True,

       qk_scale=None,

       drop_rate=0.,

       attn_drop_rate=0.,

       drop_path_rate=0.2,

       patch_norm=True,

       out_indices=(0, 1, 2, 3),

       # Please only add indices that would be used

       # in FPN, otherwise some parameter will not be used

       with_cp=True,

       convert_weights=True,

       init_cfg=dict(type='Pretrained', checkpoint=pretrained)),

   neck=dict(in_channels=[192, 384, 768, 1536], num_outs=num_levels),

   encoder=dict(layer_cfg=dict(self_attn_cfg=dict(num_levels=num_levels))),

   decoder=dict(layer_cfg=dict(cross_attn_cfg=dict(num_levels=num_levels))),
```

```
    bbox_head=dict(
        num_classes=14,))

# optimizer
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(
        type='AdamW',
        lr=0.0001,  # 0.0002 for DeformDETR
        weight_decay=0.0001),
    clip_grad=dict(max_norm=0.1, norm_type=2),
    paramwise_cfg=dict(custom_keys={'backbone': dict(lr_mult=0.1)})
)     #  custom_keys  contains  sampling_offsets  and  reference_points  in
DeformDETR  # noqa

# learning policy
max_epochs = 12
train_cfg = dict(
    type='EpochBasedTrainLoop', max_epochs=max_epochs, val_interval=1)

val_cfg = dict(type='ValLoop')
test_cfg = dict(type='TestLoop')

param_scheduler = [
    dict(
        type='MultiStepLR',
        begin=0,
        end=max_epochs,
        by_epoch=True,
        milestones=[11],
```

```
        gamma=0.1)
]


# NOTE: `auto_scale_lr` is for automatically scaling LR,
# USER SHOULD NOT CHANGE ITS VALUES.
# base_batch_size = (8 GPUs) x (2 samples per GPU)
auto_scale_lr = dict(base_batch_size=16)
```

[2] mmdetection custom config: **co_dino_5scale_swin_l_16xb1_16e_o365tococo.py**

```
_base_ = ['co_dino_5scale_r50_8xb2_1x_coco.py']


pretrained = 'https://github.com/SwinTransformer/storage/releases/download/v1.0.0/swin_large_patch4_window12_384_22k.pth'  # noqa
load_from = 'https://download.openmmlab.com/mmdetection/v3.0/codetr/co_dino_5scale_swin_large_16e_o365tococo-614254c9.pth'  # noqa


data_root = './datasets/data_traffic/'       # path/to/dataset    e.g) ./datasets/data_traffic/ <- 마지막에 슬래쉬 꼭 넣을 것.


metainfo = {
    'classes' : (
        "veh_go",
        "veh_goLeft",
        "veh_noSign",
        "veh_stop",
        "veh_stopLeft",
        "veh_stopWarning",
        "veh_warning",
        "ped_go",
        "ped_noSign",
        "ped_stop",
        "bus_go",
        "bus_noSign",
        "bus_stop",
        "bus_warning",
    )
}
```

```
# model settings
model = dict(
    backbone=dict(
        _delete_=True,
        type='SwinTransformer',
        pretrain_img_size=384,
        embed_dims=192,
        depths=[2, 2, 18, 2],
        num_heads=[6, 12, 24, 48],
        window_size=12,
        mlp_ratio=4,
        qkv_bias=True,
        qk_scale=None,
        drop_rate=0.,
        attn_drop_rate=0.,
        drop_path_rate=0.3,
        patch_norm=True,
        out_indices=(0, 1, 2, 3),
        # Please only add indices that would be used
        # in FPN, otherwise some parameter will not be used
        with_cp=True,
        convert_weights=True,
        init_cfg=dict(type='Pretrained', checkpoint=pretrained)),
    neck=dict(in_channels=[192, 384, 768, 1536]),
    query_head=dict(
        dn_cfg=dict(box_noise_scale=0.4, group_cfg=dict(num_dn_queries=500)),
        transformer=dict(encoder=dict(with_cp=6))))

train_pipeline = [
```

```
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='RandomFlip', prob=0.5),
    dict(
        type='RandomChoice',
        transforms=[
            [
                dict(
                    type='RandomChoiceResize',
                        scales=[(480, 2048), (512, 2048), (544, 2048), (576,
2048),
                                (608, 2048), (640, 2048), (672, 2048), (704,
2048),
                                (736, 2048), (768, 2048), (800, 2048), (832,
2048),
                                (864, 2048), (896, 2048), (928, 2048), (960,
2048),
                            (992, 2048), (1024, 2048), (1056, 2048),
                            (1088, 2048), (1120, 2048), (1152, 2048),
                            (1184, 2048), (1216, 2048), (1248, 2048),
                            (1280, 2048), (1312, 2048), (1344, 2048),
                            (1376, 2048), (1408, 2048), (1440, 2048),
                            (1472, 2048), (1504, 2048), (1536, 2048)],
                    keep_ratio=True)
            ],
            [
                dict(
                    type='RandomChoiceResize',
                    # The radio of all image in train dataset < 7
                    # follow the original implement
                    scales=[(400, 4200), (500, 4200), (600, 4200)],
```

```
                keep_ratio=True),
            dict(
                type='RandomCrop',
                crop_type='absolute_range',
                crop_size=(384, 600),
                allow_negative_crop=True),
            dict(
                type='RandomChoiceResize',
                    scales=[(480, 2048), (512, 2048), (544, 2048), (576,
2048),
                            (608, 2048), (640, 2048), (672, 2048), (704,
2048),
                            (736, 2048), (768, 2048), (800, 2048), (832,
2048),
                            (864, 2048), (896, 2048), (928, 2048), (960,
2048),
                        (992, 2048), (1024, 2048), (1056, 2048),
                        (1088, 2048), (1120, 2048), (1152, 2048),
                        (1184, 2048), (1216, 2048), (1248, 2048),
                        (1280, 2048), (1312, 2048), (1344, 2048),
                        (1376, 2048), (1408, 2048), (1440, 2048),
                        (1472, 2048), (1504, 2048), (1536, 2048)],
                keep_ratio=True)
        ]
    ]),
    dict(type='PackDetInputs')
]

test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', scale=(2048, 1280), keep_ratio=True),
```

```
    dict(type='LoadAnnotations', with_bbox=True),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor'))
]

train_dataloader = dict(
    batch_size=1,
    num_workers=2,
    dataset=dict(
        data_root=data_root,
        ann_file='train/train.json',  # 경로 주의
        data_prefix=dict(img='train/images/'),  # 경로 주의
        pipeline=train_pipeline,))

val_dataloader = dict(
    dataset=dict(
        data_root=data_root,
        metainfo=metainfo,
        ann_file='val/val.json',  # 경로 주의
        data_prefix=dict(img='val/images/'),  # 경로 주의
        pipeline=test_pipeline,))

# test_dataloader = val_dataloader
test_dataloader = dict(
    dataset=dict(
        data_root=data_root,
        ann_file='test/test.json',  # 경로 주의
        data_prefix=dict(img='test/images/'),  # 경로 주의
```

```
        pipeline=test_pipeline))


val_evaluator = dict(
    ann_file=data_root + 'val/val.json',)  # 경로 주의


test_evaluator = dict(
    ann_file=data_root + 'test/test.json',  # 경로 주의
    outfile_prefix='./work_dirs/co_detr_test')  # 경로 주의


optim_wrapper = dict(optimizer=dict(lr=1e-4))


max_epochs = 16
train_cfg = dict(max_epochs=max_epochs)


param_scheduler = [
    dict(
        type='MultiStepLR',
        begin=0,
        end=max_epochs,
        by_epoch=True,
        milestones=[8],
        gamma=0.1)
]
```