Wrap-up report

Contents

1. 프로젝트 개요

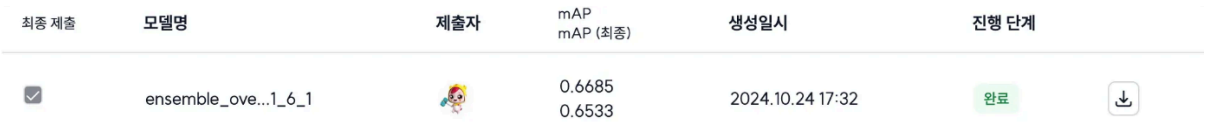
분리수거는 이러한 환경 부담을 줄일 수 있는 방법 중 하나입니다. 잘 분리배출 된 쓰레기는 자원으로서 가치를 인정받아 재활용되지만, 잘못 분리배출 되면 그대로 폐기물로 분류되어 매립 또는 소각되기 때문입니다.
따라서 우리는 사진에서 **쓰레기를 Detection 하는 모델**을 만들어 이러한 문제점을 해결해보고자 합니다. 문제 해결을 위한 데이터셋으로는 일반 쓰레기, 플라스틱, 종이, 유리 등 10 종류의 쓰레기가 찍힌 사진 데이터셋이 제공됩니다.
여러분에 의해 만들어진 우수한 성능의 모델은 쓰레기장에 설치되어 정확한 분리수거를 돕거나, 어린아이들의 분리수거 교육 등에 사용될 수 있을 것입니다. 부디 지구를 위기로부터 구해주세요! 🌎

2. 프로젝트 팀 구성 및 역할

이름_캠퍼아이디	역할
김민솔	모델링(MMDetection)
김준현	Data augmentationDetectron2), Ensemble
김현진_T7140	EDA, 모델링(MMDetction)
정권희	모델링(Detectron2), Ensemble
윤영서_T7172	Data Augmentation, Project Manager
이재건_T7227	Yolo 모델 적용

3. 프로젝트 수행 결과

3.1 최종 제출 목록



ensemble_over_60_with_ensemble_1111_6_1ensemble_over_60_with_ensemble_1111_6_1

1. cascade_rcnn_swin_L_test
    - 가중치 : 1
2. ensemble_best4_4622_6_2_5_5_2
    - 가중치 : 1
        - cascade_rcnn_swin_L_test → 가중치 4
        - cascade_swin_input:1024 → 가중치 6
        - dino_resnet50 → 가중치 2
        - DINO_base(12epoch) → 가중치 2
3. best5
    - 가중치 : 1
        - cascade_rcnn_swin_L_aug_test → 가중치 1
        - cascade_rcnn_swin_L_test → 가중치 1
        - cascade_swin_input:1024 → 가중치 1
        - DINO_base(12epoch) → 가중치 1

    - dino_resnet50 → 가중치 1
4. DINO_base
    - 가중치 1
5. over_50_models_plus_1_ensemble
    - 가중치 1
        - 0.5 넘는 모델들 전부 + best5
6. 파라미터 설정값
    - WBF IoU Threshold : 0.6
    - WBF skip_box_thr : 0.1

ensemble_over_60_with_ensemble_1111_7_1ensemble_over_60_with_ensemble_1111_7_1

최종 제출	모델명	제출자	mAP mAP (최종)	생성일시	진행 단계	
☑	ensemble_ove...1_7_1		0.6714 0.6558	2024.10.24 17:36	완료	⬇

- 위와 같은 모델들을 가지고 ensemble
- WBF IoU Threshold : 0.7 → (0.1 up)
- WBF skip_box_thr : 0.1

3.2 Private Dataset

내등수 20	CV_02조		0.6714	55	22h

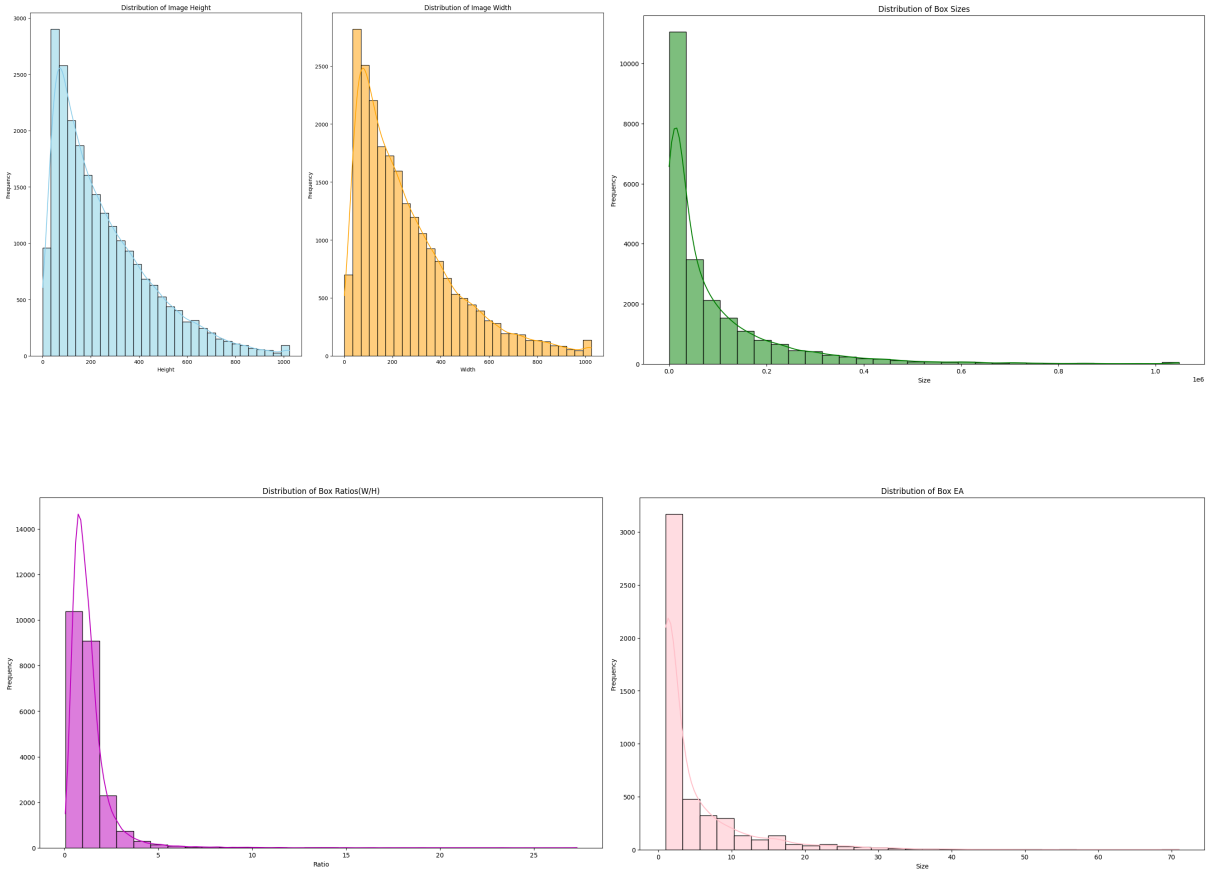
3.3 Public Dataset

내등수 21	CV_02조		0.6558	55	22h

4. 프로젝트 수행 절차 및 방법

4.1 EDA

4.1.1 Box 높이, 너비, 넓이, 비율 분포

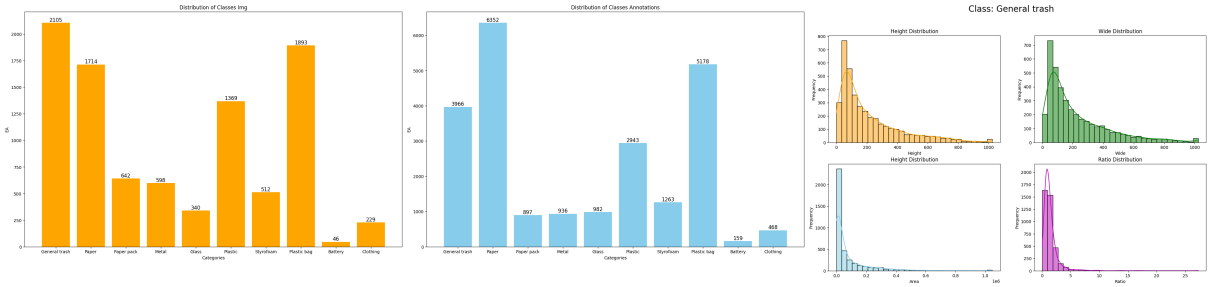


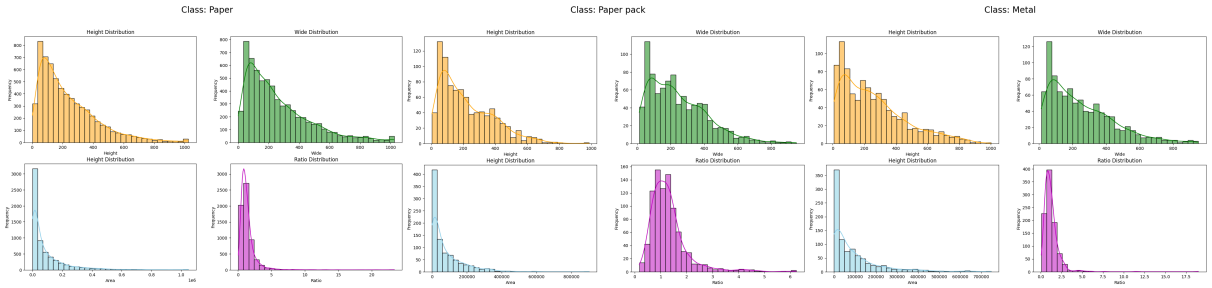
높이와 너비는 대부분 100-300 사의 값을 가졌으나 큰 box의 경우 이미지 높이/너비와 같았다.

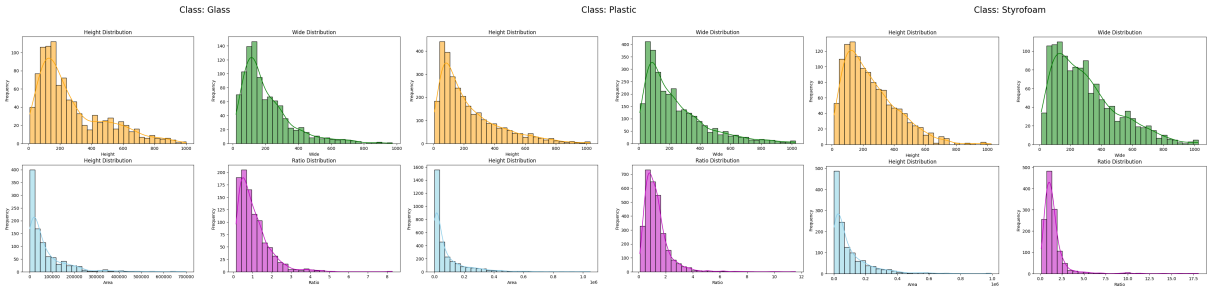
박스의 크기는 작은건 1x1보다 작은 거 부터 1024x1024에 가까운 크기까지 존재했다. 작은 이미지들이 많이 구성됨을 확인했다.

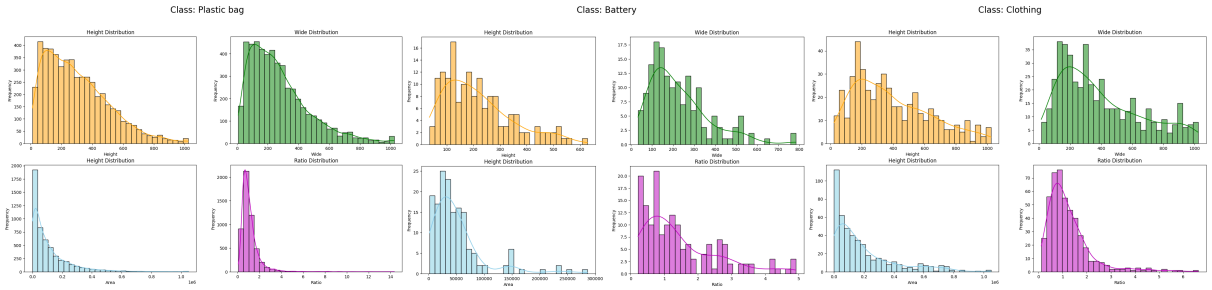
이미지 내의 박스 수는 평균적으로 4개였으나 많이 가진 이미지는 71개까지 가지고 있었다.

4.1.2 클래스별 이미지 개수, Annotation 분포





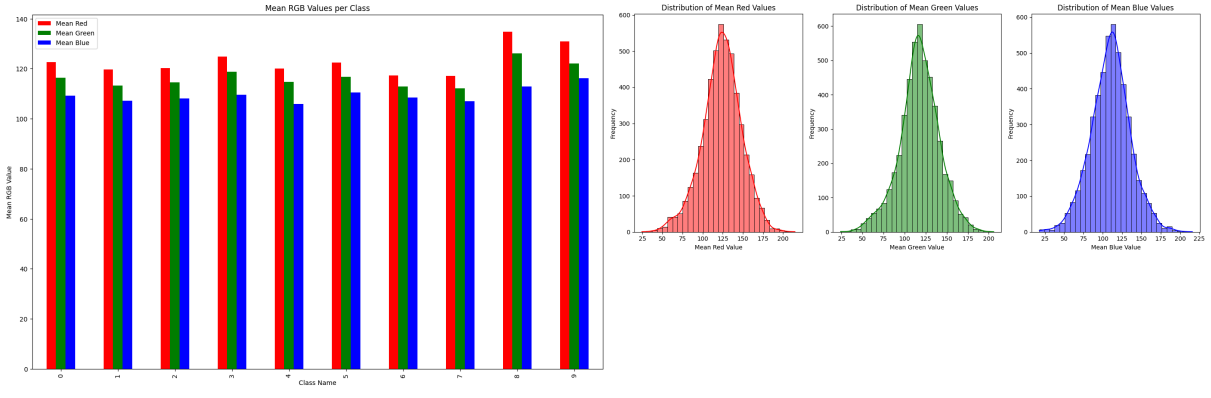




클래스별로 보았을때 배터리와 옷 클래스 수가 적었다.

클래스별로 높이, 너비, 넓이, 비율을 비교해봤을때 다른 클래스보다 특징이 두드러지는 편이었다.

4.1.3 색상에 대한 정보



색상은 모든 클래스, 이미지에서 치우치는 값 없이 일정한 편이었다.

4.1.4 랜덤한 이미지를 출력

랜덤하게 이미지를 출력했을때, 스티로폼 관련된 이미지가 Annotation이 이상한 것이 꽤 많았다.

4.2 Model & Data Augmentation (1)

팀 내에서 다양한 Object Detection Library를 경험해보는 것을 프로젝트의 목표로 삼았다.

이에 따라, 각각 MMdetection, Detectron2, Ultralytics 라이브러리를 다양하게 실험해보는 것에 집중했다.

4.2.1 Base model: DETR

- ResNet를 backbone으로 사용
- input image size: 512 × 512

📌 제출 결과

Detr_base		0.3608	-

📌 Evaluation 분석

```
2024-10-17 07:15:03,570 - mmdet - INFO -
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.469
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.629
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.514
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.021
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.086
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.562
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.641
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.641
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.641
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.028
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.273
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.735

2024-10-17 07:15:03,571 - mmdet - INFO -
+--------------+-------+------------+-------+------------+-------+
| category     | AP    | category   | AP    | category   | AP    |
+--------------+-------+------------+-------+------------+-------+
| General trash | 0.311 | Paper      | 0.349 | Paper pack | 0.463 |
| Metal        | 0.482 | Glass      | 0.503 | Plastic    | 0.417 |
| Styrofoam    | 0.368 | Plastic bag | 0.577 | Battery    | 0.653 |
| Clothing     | 0.568 | None       | None  | None       | None  |
+--------------+-------+------------+-------+------------+-------+
```

💡

- Evaluation 시, train dataset의 일부로 평가했다.
  - 라이브러리에 적응하는 시간이 길어지다보니, 공통의 과제였던 (Stratified) K-Fold 구현이 미뤄졌다.
- General Trash, Paper AP이 낮은 이유:
  - General Trash는 작은 객체가 많다.
  - Paper와 Paper pack을 제대로 구분하지 못했다.
- Styrofoam AP 값이 낮은 이유:
  - EDA 시, Styrofoam box의 annotation이 정확하지 않았다.
- Clothing, Battery AP 값이 높은 이유:
  - DATA 수는 적지만, 특징이 확실하기 때문으로 분석했다.
- AP small, AP medium에서 급격하게 value가 낮아지는 것을 확인하여, DETR이 작은 크기의 물체를 거의 잡아내지 못한다고 판단했다.
- Recall 값이 AP에 비해 높은 것을 보아, Precision이 낮은 것으로 파악했다.
  - 즉, negative estimation(잘못 추정하는 박스들)이 많은 것으로 예상하고, 이를 해결하기 위해 input image를 두 배씩 늘려 학습했다.

4.2.2 Base model: CascadeRCNN

⚙️

- FPN 사용 → Small objects에 대한 feature 잡아내기
- Swin Transformer를 backbone으로 사용
- input image size:
  - 1024 × 1024
  - 2048 × 2048

📌 제출 결과

	cascade_swin...:1024		0.5317 -	2024.10.17 14:38	완료	

	cascade_swin...:2048		0.5074 0.4914	2024.10.20 00:14	완료	

📌 Evaluation 분석

```
2024-10-17 09:55:31,962 - mmdet - INFO -
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.872
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=1000 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=1000 ] = 0.927
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.604
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.797
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.890
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.887
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.887
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.887
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.628
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.817
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.904

2024-10-17 09:55:31,964 - mmdet - INFO -
+---------------+-------+-------------+-------+------------+-------+
| category      | AP    | category    | AP    | category   | AP    |
+---------------+-------+-------------+-------+------------+-------+
| General trash | 0.852 | Paper       | 0.766 | Paper pack | 0.867 |
| Metal         | 0.905 | Glass       | 0.857 | Plastic    | 0.851 |
| Styrofoam     | 0.825 | Plastic bag | 0.880 | Battery    | 1.000 |
| Clothing      | 0.916 | None        | None  | None       | None  |
+---------------+-------+-------------+-------+------------+-------+
```

💡

- DETR과 비교하였을 때, mAP가 높아진 결과를 얻었다.
- 특히, AP small 및 AP medium이 향상되었다.
- 2048 × 2048로 Resizing 시 성능이 하락하였다.
  → 이미지의 정보가 손실되었기 때문으로 파악된다.

4.2.3 Mosaic augmentation

```python
train_dataset = dict(
    type='MultiImageMixDataset',
    dataset=dict(
        type=dataset_type,
        ann_file=data_root + '/train.json',
        img_prefix=data_root,
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations', with_bbox=True)
        ],
        filter_empty_gt=False,
        classes=classes,
    ),
    # classes=classes,
    pipeline=train_pipeline,
)
```

- MultiImageMixDataset 기반으로 config를 수정했다.
  - inner dataset: COCODataset
  - 기타 cfg 설정은 동일
- train_pipeline에 mosaic augmentation을 추가했다.

 Trouble Shooting About Mosaic augmentation

1. **1차 에러**

```
2024-10-21 11:37:29,179 - mmdet - INFO -
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.360
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.530
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.394
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.006
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.063
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.418
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.501
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.501
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.501
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.024
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.164
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.568

2024-10-21 11:37:29,180 - mmdet - INFO -
+--------------+-------+--------------+-------+--------------+-------+
| category     | AP    | category     | AP    | category     | AP    |
+--------------+-------+--------------+-------+--------------+-------+
| General trash | 0.198 | Paper        | 0.209 | Paper pack   | 0.343 |
| Metal        | 0.396 | Glass        | 0.349 | Plastic      | 0.304 |
| Styrofoam    | 0.223 | Plastic bag  | 0.433 | Battery      | 0.748 |
| Clothing     | 0.401 | None         | None  | None         | None  |
+--------------+-------+--------------+-------+--------------+-------+
```

```
py", line 316, in __call__
    self._resize_img(results)
  File "/opt/conda/lib/python3
py", line 220, in _resize_img
    results[key],
KeyError: 'img'
```

results에서 img key에러가 발생하였다. 원인 파악을 위해 여러 문서를 참고하였고, 파악이 되지 않는 상황이 지속되어, 코드를 MMDetection Tutorial에 기재된 Reference code 기반으로 재작성하였다.

2. **2차 에러**

```
  File "/opt/conda/lib/python3.10/site-package
py", line 2042, in __call__
    results = self._mosaic_transform(results)
  File "/opt/conda/lib/python3.10/site-package
py", line 2068, in _mosaic_transform
    assert 'mix_results' in results
AssertionError
```

results에 mix_results가 없다는 에러가 발생하였다. 해당 에러에도 여러 원인을 찾아보려고 하였지만, 해결에 사용할 수 있는 자료가 나오지 않았다. mmdection 버전을 3.x로 업그레이드 하여도 같은 문제가 발생하였다.

3. **최종**

config 내의 _base_를 모두 제거하여 tree 구조를 사용하지 않고, 한 config 파일에 dataset, model, runtime, scheduler 구성해서 사용하였다. 코드 실행은 정상적으로 되었지만, aistage 제출 시 결과 값이 0이 나오는 상황이 발생하였다. 프로젝트 마감까지 시간이 얼마 남지 않아, mosaic 적용을 포기하게 됐다.

⚠️ 작은 객체들을 잘 탐지하지 못해서, 작은 객체를 잘 탐지한다고 알려진 모델들을 더 알아보았다.

4.2.4 Base model: EfficientNet

- RetinaNet을 backbone으로 사용
- input image size: 1024 × 1024

📌 제출 결과

📌 Evaluation 분석

category	mAP	mAP_50	mAP_75	mAP_s	mAP_m	mAP_l
General trash	0.033	0.059	0.035	0.002	0.012	0.046
Paper	0.025	0.065	0.015	0.001	0.01	0.031
Paper pack	0.047	0.069	0.055	0.0	0.005	0.062
Metal	0.028	0.047	0.03	0.0	0.0	0.038
Glass	0.008	0.021	0.003	0.0	0.0	0.009
Plastic	0.019	0.038	0.017	0.0	0.003	0.026
Styrofoam	0.025	0.049	0.022	0.0	0.001	0.028
Plastic bag	0.089	0.194	0.074	0.0	0.002	0.105
Battery	0.001	0.003	0.0	nan	0.004	0.001
Clothing	0.031	0.058	0.034	0.0	0.0	0.033

```
2024/10/18 20:32:10 - mmengine - INFO - bbox_mAP_copypaste: 0.031 0.060 0.029 0.000 0.004 0.038
2024/10/18 20:32:10 - mmengine - INFO - Epoch(val) [12][4883/4883]
coco/General trash_precision: 0.0330  coco/Paper_precision: 0.0250  coco/Paper pack_precision: 0.0470
coco/Metal_precision: 0.0280  coco/Glass_precision: 0.0080  coco/Plastic_precision: 0.0190
coco/Styrofoam_precision: 0.0250  coco/Plastic bag_precision: 0.0890  coco/Battery_precision: 0.0010
coco/Clothing_precision: 0.0310
coco/bbox_mAP: 0.0310  coco/bbox_mAP_50: 0.0600  coco/bbox_mAP_75: 0.0290
coco/bbox_mAP_s: 0.0000  coco/bbox_mAP_m: 0.0040  coco/bbox_mAP_l: 0.0380
data_time: 0.0023  time: 0.0492
```

→ Train dataset의 일부로 평가하였음에도 전반적으로 성능이 잘 나오지 않았다.

4.2.5 Base model: EfficientDet

- EfficientNet를 backbone으로 사용
- input image size: 1024 × 1024

📌 제출 결과

```
index created!
  PredictionString       image_id
0                  test/0000.jpg
1                  test/0001.jpg
2                  test/0002.jpg
3                  test/0003.jpg
4                  test/0004.jpg
```

→ Test 결과를 미리보기 했을 때, 예측된 박스가 없어 학습이 되지 않는다 판단했다.

📌 Evaluation 분석

```
+---------------+-------+--------+--------+-------+-------+-------+
| category      | mAP   | mAP_50 | mAP_75 | mAP_s | mAP_m | mAP_l |
+---------------+-------+--------+--------+-------+-------+-------+
| General trash | 0.052 | 0.082  | 0.057  | 0.001 | 0.032 | 0.069 |
| Paper         | 0.061 | 0.13   | 0.051  | 0.016 | 0.018 | 0.075 |
| Paper pack    | 0.079 | 0.105  | 0.091  | 0.0   | 0.013 | 0.101 |
| Metal         | 0.052 | 0.068  | 0.058  | 0.015 | 0.002 | 0.068 |
| Glass         | 0.014 | 0.025  | 0.014  | 0.0   | 0.0   | 0.016 |
| Plastic       | 0.039 | 0.064  | 0.041  | 0.005 | 0.009 | 0.051 |
| Styrofoam     | 0.019 | 0.034  | 0.018  | 0.0   | 0.001 | 0.022 |
| Plastic bag   | 0.208 | 0.338  | 0.225  | 0.001 | 0.009 | 0.243 |
| Battery       | 0.005 | 0.009  | 0.006  | nan   | 0.007 | 0.006 |
| Clothing      | 0.01  | 0.019  | 0.011  | 0.0   | 0.0   | 0.011 |
+---------------+-------+--------+--------+-------+-------+-------+
2024/10/18 16:25:26 - mmengine - INFO - bbox_mAP_copypaste: 0.054 0.087 0.057 0.004 0.009 0.066
2024/10/18 16:25:27 - mmengine - INFO - Epoch(val) [12][4883/4883]
coco/General trash_precision: 0.0520  coco/Paper_precision: 0.0610  coco/Paper pack_precision: 0.0790
coco/Metal_precision: 0.0520  coco/Glass_precision: 0.0140  coco/Plastic_precision: 0.0390
coco/Styrofoam_precision: 0.0190  coco/Plastic bag_precision: 0.2080  coco/Battery_precision: 0.0050
coco/Clothing_precision: 0.0100
coco/bbox_mAP: 0.0540  coco/bbox_mAP_50: 0.0870  coco/bbox_mAP_75: 0.0570
coco/bbox_mAP_s: 0.0040  coco/bbox_mAP_m: 0.0090  coco/bbox_mAP_l: 0.0660
data_time: 0.0024  time: 0.0626
```

→ Train dataset의 일부로 평가하였음에도 전반적으로 성능이 잘 나오지 않았다.

4.2.6 Base model: DINO

- Swin-L를 backbone으로 사용
- input image size: 1024 × 1024

📌 제출 결과

DINO_base(12epoch)		0.6057	2024.10.21 00:51	완료	

📌 Evaluation 분석

```
+--------------+-------+--------+--------+-------+-------+-------+
| category     | mAP   | mAP_50 | mAP_75 | mAP_s | mAP_m | mAP_l |
+--------------+-------+--------+--------+-------+-------+-------+
| General trash | 0.51  | 0.603  | 0.528  | 0.093 | 0.274 | 0.636 |
| Paper        | 0.502 | 0.661  | 0.52   | 0.041 | 0.185 | 0.596 |
| Paper pack   | 0.734 | 0.816  | 0.759  | 0.032 | 0.356 | 0.846 |
| Metal        | 0.7   | 0.788  | 0.716  | 0.13  | 0.272 | 0.835 |
| Glass        | 0.669 | 0.798  | 0.7    | 0.053 | 0.256 | 0.749 |
| Plastic      | 0.587 | 0.706  | 0.611  | 0.046 | 0.244 | 0.709 |
| Styrofoam    | 0.6   | 0.727  | 0.625  | 0.002 | 0.216 | 0.66  |
| Plastic bag  | 0.715 | 0.829  | 0.749  | 0.035 | 0.233 | 0.791 |
| Battery      | 0.854 | 0.933  | 0.878  | nan   | 0.51  | 0.894 |
| Clothing     | 0.756 | 0.838  | 0.77   | 0.454 | 0.195 | 0.785 |
+--------------+-------+--------+--------+-------+-------+-------+
2024/10/20 09:16:12 - mmengine - INFO - bbox_mAP_copypaste: 0.661 0.770 0.686 0.098 0.274 0.750
2024/10/20 09:16:13 - mmengine - INFO - Epoch(val) [15][4883/4883]
coco/General trash_precision: 0.5100  coco/Paper_precision: 0.5020  coco/Paper pack_precision: 0.7340
coco/Metal_precision: 0.7000  coco/Glass_precision: 0.6690  coco/Plastic_precision: 0.5870
coco/Styrofoam_precision: 0.6000  coco/Plastic bag_precision: 0.7150  coco/Battery_precision: 0.8540
coco/Clothing_precision: 0.7560
coco/bbox_mAP: 0.6610  coco/bbox_mAP_50: 0.7700  coco/bbox_mAP_75: 0.6860
coco/bbox_mAP_s: 0.0980  coco/bbox_mAP_m: 0.2740  coco/bbox_mAP_l: 0.7500
data_time: 0.0027  time: 0.3775
```

💡 Train dataset의 일부로 평가해서 제출결과보단 높은 점수를 얻었다.
특히 작은 객체에 대한 성능이 다른 모델들보다 좋았다.
DINO모델을 베이스모델로 다양한 Augmentation을 적용해보고자 했다.

4.2.7 Dino Augmentation

1. Mosaic augmentation

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.664
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.774
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.688
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.046
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.269
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.754
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.817
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.829
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.829
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.288
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.619
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.883
10/22 02:23:38 - mmengine - INFO -
+--------------+-------+--------+--------+-------+-------+-------+
| category     | mAP   | mAP_50 | mAP_75 | mAP_s | mAP_m | mAP_l |
+--------------+-------+--------+--------+-------+-------+-------+
| General trash | 0.51  | 0.602  | 0.529  | 0.095 | 0.274 | 0.638 |
| Paper        | 0.512 | 0.672  | 0.53   | 0.049 | 0.187 | 0.608 |
| Paper pack   | 0.731 | 0.811  | 0.762  | 0.0   | 0.328 | 0.849 |
| Metal        | 0.696 | 0.79   | 0.719  | 0.098 | 0.275 | 0.825 |
| Glass        | 0.659 | 0.794  | 0.678  | 0.061 | 0.258 | 0.739 |
| Plastic      | 0.597 | 0.717  | 0.621  | 0.056 | 0.259 | 0.72  |
| Styrofoam    | 0.618 | 0.758  | 0.644  | 0.009 | 0.225 | 0.678 |
| Plastic bag  | 0.718 | 0.831  | 0.753  | 0.033 | 0.227 | 0.797 |
| Battery      | 0.834 | 0.909  | 0.851  | nan   | 0.493 | 0.877 |
| Clothing     | 0.775 | 0.853  | 0.792  | 0.008 | 0.161 | 0.807 |
+--------------+-------+--------+--------+-------+-------+-------+
```

Mosaic를 처리했을 때 아무것도 처리하지않은 DINO와 비슷한 성능을 가졌다.

2. Random Crop Agmentation

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.335
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.443
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.342
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.022
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.100
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.399
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.640
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.658
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.658
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.199
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.372
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.728
10/23 06:48:01 - mmengine - INFO -
+---------------+-------+--------+--------+-------+-------+-------+
| category      | mAP   | mAP_50 | mAP_75 | mAP_s | mAP_m | mAP_l |
+---------------+-------+--------+--------+-------+-------+-------+
| General trash | 0.214 | 0.276  | 0.219  | 0.012 | 0.09  | 0.287 |
| Paper         | 0.206 | 0.346  | 0.196  | 0.009 | 0.07  | 0.25  |
| Paper pack    | 0.451 | 0.539  | 0.474  | 0.0   | 0.12  | 0.561 |
| Metal         | 0.416 | 0.502  | 0.429  | 0.014 | 0.067 | 0.536 |
| Glass         | 0.326 | 0.46   | 0.331  | 0.101 | 0.085 | 0.38  |
| Plastic       | 0.269 | 0.365  | 0.275  | 0.029 | 0.073 | 0.344 |
| Styrofoam     | 0.223 | 0.352  | 0.213  | 0.0   | 0.04  | 0.254 |
| Plastic bag   | 0.336 | 0.479  | 0.344  | 0.005 | 0.094 | 0.383 |
| Battery       | 0.579 | 0.67   | 0.602  | nan   | 0.331 | 0.635 |
| Clothing      | 0.335 | 0.442  | 0.338  | 0.025 | 0.035 | 0.356 |
+---------------+-------+--------+--------+-------+-------+-------+
```

아무것도 처리하지 않은 모델 성능의 반으로 떨어져서 사용하지 않았다.

4.2.8 Custom Augmentation

1. **데이터셋의 구조 및 분석**

- **소형 객체 문제**: 이미지 내 작은 객체의 비율이 높아 모델이 특정 객체를 인식하기 어려울 가능성이 있었다. 객체의 크기 분포를 시각화한 결과, 약 60% 이상이 작거나 중간 크기의 객체임을 확인했다. 이러한 객체들은 확대하거나 이미지 합성 등을 통해 모델 학습에 중요한 요소로 다루기로 했다.

2. **Augmentation 적용**

기본 augmentation 선택

초기에는 Resize, RandomFlip, Normalize 등의 기초적인 전처리만 사용했으나, 작은 객체 비율을 높이기 위해 추가 기법을 고려했다. 특히, Mosaic, MixUp 같은 multi-image 기법을 활용하여 작은 객체를 더욱 두드러지게 할 수 있도록 했다.

Custom Augmentation 개발

작은 객체에 특화된 custom augmentation을 추가하여 작은 객체가 잘 학습되도록 했다.

이 모듈을 config에서 채택하는 방식으로 필요한 augmentation을 적용시켰다.

- **RandomBrightness**: 이미지의 밝기를 무작위로 조정하여 조명 변화에 강인한 모델을 만들고자 했다.
- **RandomCropWithMinIoU**: 작은 객체들이 잘려나가지 않도록, IoU 비율이 최소한의 임계값을 충족하는 경우에만 크롭이 적용되도록 했다.
- **ResizeWithAspectRatioClipping**: 비율을 유지하면서 작은 객체를 포함한 이미지를 확대하도록 하여 모델이 작은 객체를 더 잘 학습할 수 있도록 했다.

```
# custom augmentation 예시
from mmdet.datasets.pipelines import PIPELINES

@PIPELINES.register_module()
class RandomCropWithMinIoU(object):
    def __init__(self, min_iou=0.5):
        self.min_iou = min_iou

    def __call__(self, results):
        # 이미지 일부를 자르되 객체가 최소 IoU 조건을 충족할 때만 진행
        ...
        return results
```

4.3 Model & Data Augmentation (2)

4.3.1 About Detectron2 Library

- 학습 및 추론 속도가 빠름
- Pytorch기반, 사용 용이성
- mmdetection보다 적은 모델, 좋지 않은 성능

4.3.2 Faster_RCNN

- 2-stage-detector 중 가장 기본적인 형태
- 베이스라인 코드를 이해하기 위해 기본적인 모델을 선정
- Object Detection을 공부하는 입장에서 기초적인 모델을 튜닝해가며 공부하는 것이 좋을 것이라 판단

- Faster R-CNN은 객체 탐지에서 강력한 성능을 보여주는 대표적인 모델이며, ResNeXt-101은 더 넓은 표현력을 제공해 성능을 강화합니다. FPN이 추가되어 다양한 크기의 객체를 잘 인식
- ResNeXt-101의 그룹 컨볼루션 구조가 다중 객체 인식에 강점을 제공하여 쓰레기 객체 탐지에 유리
- Detectron2의 COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml 사용

Faster R-CNN:

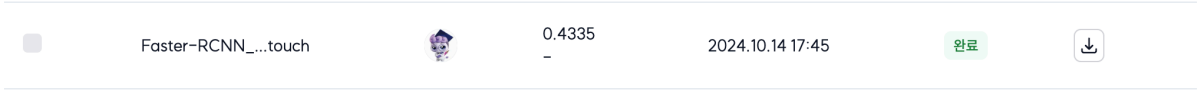
Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model | metrics
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model | metrics
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model | metrics
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model | metrics
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model | metrics
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model | metrics
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model | metrics

- 가장 기본이 되는 Augmentation 적용

```
    transform_list = [
        T.ResizeShortestEdge(short_edge_length=(640, 672, 704, 736, 768, 8
00), max_size=1333, sample_style='choice'),
        T.RandomFlip(prob=0.5, horizontal=True, vertical=False),  # 수평 뒤집기
        T.RandomRotation(angle=[0, 90, 180, 270]),  # 이미지 회전
        T.RandomBrightness(0.8, 1.8),  # 밝기 조정
        T.RandomContrast(0.6, 1.3)  # 대비 조정
    ]
```

Faster_RCNN 리더보드 결과

- 기초가 되는 모델 자체의 성능이 낮기 때문에 낮을 것으로 예상



4.3.3 Retinanet

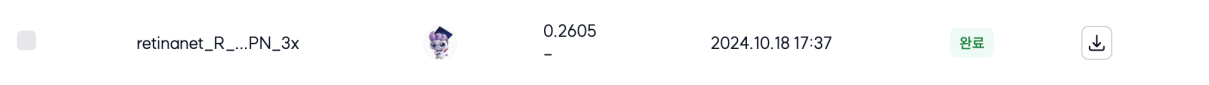
- RetinaNet은 Focal Loss를 사용하여 작은 객체나 불균형한 데이터에 대해 강력한 성능을 보여준다
- 쓰레기 객체는 종종 크기가 작거나 다양한 모양을 가질 수 있어 RetinaNet의 Focal Loss가 성능을 크게 향상
- 불균형 클래스 분포나 작은 객체에 특화된 성능을 보여, 쓰레기와 같은 다양한 크기의 객체 탐지에 적합
- 회의 중 mmdetection 모델에서 작은 객체를 탐지하지 못한다는 것을 공유받고 더 작은 객체를 잘 탐지할 수 있는 retinanet모델을 선정
- Detectron2의 COCO-Detection/retinanet_R_101_FPN_3x.yaml 사용
- Augmentation은 faster-rcnn과 동일하게 기본적인 것들 적용

RetinaNet:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50	1x	0.205	0.041	4.1	37.4	190397773	model | metrics
R50	3x	0.205	0.041	4.1	38.7	190397829	model | metrics
R101	3x	0.291	0.054	5.2	40.4	190397697	model | metrics

Retinanet 리더보드 결과

- sweep과 wandb를 적용해서 진행했지만 낮은 성능을 보임



4.3.4 Cascade-mask-rcnn

- Cascade Mask R-CNN은 여러 스테이지로 객체 탐지를 개선하는 구조를 가지고 있어, 더 정교한 바운딩 박스와 객체 분할을 제공

- ResNet-101 백본을 사용하여 높은 성능을 보장
- 다단계 예측으로 인해 작은 객체 탐지나 복잡한 형태의 객체에 대해 높은 성능을 기대
- Detectron2의 Misc/cascade_mask_rcnn_R_101_FPN_3x.yaml 모델 사용

Ablations for Deformable Conv and Cascade R-CNN:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	mask AP	model id	download
Baseline R50-FPN	1x	0.261	0.043	3.4	38.6	35.2	137260431	model | metrics
Deformable Conv	1x	0.342	0.048	3.5	41.5	37.5	138602867	model | metrics
Cascade R-CNN	1x	0.317	0.052	4.0	42.1	36.4	138602847	model | metrics
Baseline R50-FPN	3x	0.261	0.043	3.4	41.0	37.2	137849600	model | metrics
Deformable Conv	3x	0.349	0.047	3.5	42.7	38.5	144998336	model | metrics
Cascade R-CNN	3x	0.328	0.053	4.0	44.3	38.5	144998488	model | metrics

Cascade-mask-rcnn 리더보드 결과

	misk-cascade...0-fpn		0.2976 -	2024.10.19 20:25	완료	
	cascade_rcnn..._test		0.3360 -	2024.10.20 16:30	완료	

4.3.5 Cascade-rcnn

- detectron2가 제공하는 백본들에 대한 성능이 좋지 않아 백본을 변경하기로 함
- swin-L 백본을 사용하기 위해 detectron2에서 노력했지만 지원을 하지 않아 복잡함
- 해서 mmdetection으로 넘어가서 수행
- object detection에서 좋은 성능을 보이는 cascade-rcnn으로 선정
- 백본을 swin-L로 가져가서 좋은 성능을 노림

Cascade-rcnn code - cascade_rcnn_swin_L.py

- 기본적인 augmentation만 적용한 부분과 mosaic, cutmix등 두가지 방법 모두 실험

```
# cascade_rcnn_swin_L.py

_base_ = [
    '../_base_/models/cascade_rcnn_r50_fpn.py'
]

pretrained = 'https://github.com/SwinTransformer/storage/releases/download/v1.0.0/swin_large_patch4_window7_224_22k.pth'  # Swin-L pretrained

model = dict(
    backbone=dict(
        _delete_=True,
        type='SwinTransformer',
        embed_dims=192,  # Swin-L의 임베딩 차원
        depths=[2, 2, 18, 2],  # Swin-L 각 스테이지 깊이
        num_heads=[6, 12, 24, 48],  # Swin-L의 헤드 수
        window_size=7,
        mlp_ratio=4,
        qkv_bias=True,
        qk_scale=None,
        drop_rate=0.,
        attn_drop_rate=0.,
        drop_path_rate=0.3,  # Swin-L의 Drop path rate
        patch_norm=True,
        out_indices=(0, 1, 2, 3),
        with_cp=False,
        convert_weights=True,
        init_cfg=dict(type='Pretrained', checkpoint=pretrained)),  # Swin-L 가중치 경로
    neck=dict(in_channels=[192, 384, 768, 1536]),  # Swin-L에서 neck으로 전달하는 각 레이어 채널 수
    roi_head=dict(
        bbox_head=[
            dict(
                type='Shared2FCBBoxHead',
                in_channels=256,
                fc_out_channels=1024,
                roi_feat_size=7,
```

```
            num_classes=10,
            bbox_coder=dict(
                type='DeltaXYWHBBoxCoder',
                target_means=[0., 0., 0., 0.],
                target_stds=[0.1, 0.1, 0.2, 0.2]),
            reg_class_agnostic=True,
            loss_cls=dict(
                type='CrossEntropyLoss',
                use_sigmoid=False,
                loss_weight=1.0),
            loss_bbox=dict(type='SmoothL1Loss', beta=1.0,
                           loss_weight=1.0)),
        dict(
            type='Shared2FCBBoxHead',
            in_channels=256,
            fc_out_channels=1024,
            roi_feat_size=7,
            num_classes=10,
            bbox_coder=dict(
                type='DeltaXYWHBBoxCoder',
                target_means=[0., 0., 0., 0.],
                target_stds=[0.05, 0.05, 0.1, 0.1]),
            reg_class_agnostic=True,
            loss_cls=dict(
                type='CrossEntropyLoss',
                use_sigmoid=False,
                loss_weight=1.0),
            loss_bbox=dict(type='SmoothL1Loss', beta=1.0,
                           loss_weight=1.0)),
        dict(
            type='Shared2FCBBoxHead',
            in_channels=256,
            fc_out_channels=1024,
            roi_feat_size=7,
            num_classes=10,
            bbox_coder=dict(
                type='DeltaXYWHBBoxCoder',
                target_means=[0., 0., 0., 0.],
```

```
                target_stds=[0.033, 0.033, 0.067, 0.067]),
            reg_class_agnostic=True,
            loss_cls=dict(
                type='CrossEntropyLoss',
                use_sigmoid=False,
                loss_weight=1.0),
            loss_bbox=dict(type='SmoothL1Loss', beta=1.0, loss_weight=1.0))
    ]))

# Optimizer remains the same
optimizer = dict(
    type='AdamW',
    lr=0.0001,
    betas=(0.9, 0.999),
    weight_decay=0.05,
    paramwise_cfg=dict(
        custom_keys={
            'absolute_pos_embed': dict(decay_mult=0.),
            'relative_position_bias_table': dict(decay_mult=0.),
            'norm': dict(decay_mult=0.)
        }))

# 학습률 감소 스케줄 설정: milestones = [2, 8, 11], gamma = 0.5
lr_config = dict(
    policy='step',
    step=[8, 11],  # 2, 8, 11 에포크에서 학습률 감소
    gamma=0.5  # 학습률을 0.5배로 감소
)

# custom augmentation이 정의된 파일을 임포트
custom_imports = dict(
    imports=['mmdet.datasets.pipelines.custom_augmentation'],  # custom_
augmentation.py의 경로
    allow_failed_imports=False
)

# 이 부분은 기본 aug만 적용
data = dict(
```

```
    train=dict(
        type='CocoDataset',
        classes=("General trash", "Paper", "Paper pack", "Metal", "Glass",
            "Plastic", "Styrofoam", "Plastic bag", "Battery", "Clothing"),
        img_prefix='/data/ephemeral/home/level2-objectdetection-cv-02/data
set/',
        ann_file='/data/ephemeral/home/level2-objectdetection-cv-02/datase
t/train.json',
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations', with_bbox=True),
            dict(type='RandomFlip', flip_ratio=0.5),
            dict(type='Resize', img_scale=(1024, 1024), keep_ratio=True),  # 이
미지 크기 조정 단계 추가
            dict(type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])
        ]
    ),

    test=dict(
        type='CocoDataset',
        img_prefix='/data/ephemeral/home/level2-objectdetection-cv-02/data
set/',
        ann_file='/data/ephemeral/home/level2-objectdetection-cv-02/datase
t/test.json',
        pipeline=[
            dict(type='LoadImageFromFile'),
            # dict(type='RandomFlip', flip_ratio=None),
            dict(type='Resize', img_scale=(1024, 1024), keep_ratio=True),  # 이
미지 크기 조정 단계 추가
            dict(type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
```

```
            to_rgb=True),
        dict(type='Pad', size_divisor=32),
        dict(type='Collect', keys=['img'],
            meta_keys=('filename', 'ori_shape', 'img_shape', 'pad_shape',
              'scale_factor', 'img_norm_cfg'))
    ])
)

# Mixup과 Mosaic을 포함한 Train Pipeline 설정
data = dict(
   train=dict(
      type='MultiImageMixDataset',
      dataset=dict(
         type='CocoDataset',
         classes=("General trash", "Paper", "Paper pack", "Metal", "Glass",
                  "Plastic", "Styrofoam", "Plastic bag", "Battery", "Clothing"),
         img_prefix='/data/ephemeral/home/level2-objectdetection-cv-02/d
ataset/',
         ann_file='/data/ephemeral/home/level2-objectdetection-cv-02/data
set/train.json',
         pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations', with_bbox=True)
         ]
      ),
      pipeline=[
         dict(type='Mosaic', img_scale=(1024, 1024), pad_val=114.0),  # Mos
aic 증강 추가
         dict(type='RandomAffine', scaling_ratio_range=(0.1, 2), border=(-51
2, -512)),  # Affine 변환
         dict(type='MixUp', img_scale=(1024, 1024), ratio_range=(0.8, 1.6), p
ad_val=114.0),  # Mixup 추가
         dict(type='RandomFlip', flip_ratio=0.5),  # RandomFlip 추가
         dict(type='RandomBrightness', brightness_delta=32),  # 커스텀 증강
추가
         dict(type='ResizeWithAspectRatioClipping', img_scale=(1024, 1024),
keep_ratio=True),  # 커스텀 증강 추가
         dict(type='Normalize', mean=[123.675, 116.28, 103.53], std=[58.395,
```

```
57.12, 57.375], to_rgb=True),
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])
        ]
    ),

    test=dict(
        type='CocoDataset',
        img_prefix='/data/ephemeral/home/level2-objectdetection-cv-02/data
set/',
        ann_file='/data/ephemeral/home/level2-objectdetection-cv-02/datase
t/test.json',
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='Resize', img_scale=(1024, 1024), keep_ratio=True),
            dict(type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img'],
                meta_keys=('filename', 'ori_shape', 'img_shape', 'pad_shape',
                        'scale_factor', 'img_norm_cfg'))
        ]
    )
)

# 14 에포크 설정
runner = dict(type='EpochBasedRunner', max_epochs=14)

# 체크포인트 설정
checkpoint_config = dict(max_keep_ckpts=3, interval=1)

# 로그 설정
log_config = dict(
    interval=100,  # 50 iterations마다 로그를 기록
```

```
    hooks=[
        dict(type='TextLoggerHook'),  # 터미널에 출력
        # dict(type='TensorboardLoggerHook'),  # TensorBoard 사용 시 추가
    ]
)

# 학습만 적용
workflow = [('train', 1)]
```

Cascade-rcnn-swin-L 결과 확인

	cascade_rcnn..._test		0.6041 -	2024.10.23 12:27	완료	

성능 좋은 백본을 활용하니 눈에 띄게 증가한 성능

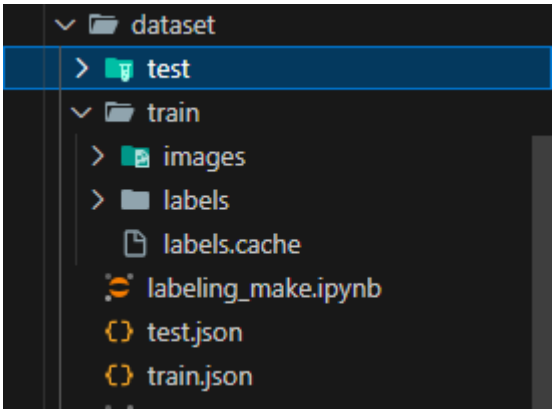
	cascade_rcnn..._test		0.5835 -	2024.10.23 23:24	완료	

augmentation 적용을 했지만 성능이 감소함

4.4 Model & Data Augmentation (3)

4.4.1 Yolo 모델 적용 방법

1. **yolo 데이터 형식에 맞추기**



```
dataset
│   ├── train
│   │   ├── images
│   │   ├── labels
│   │   └── labels.cache
```

위와 같은 폴더 형식이어야 하며, images 에 `0000.jpg` 라면 labels 에는 `0000.txt` 로 고쳐주었다.

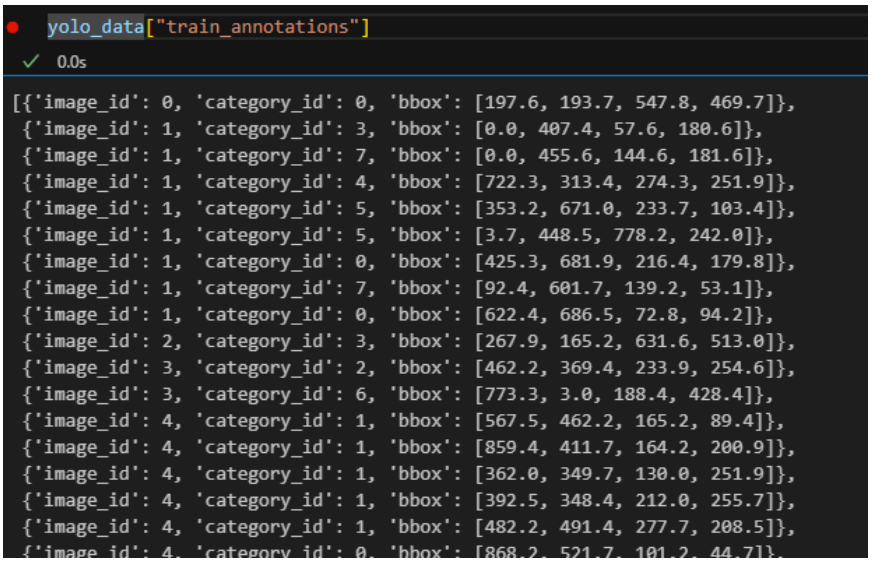
2. 라벨 내용 yolo에 맞추기

위 내용을 맞추기 위해서는 아래의 2가지 task를 풀어야 한다.

`1) 라벨링문제`

Yolo는 <class> <x_center> <y_center> <width> <height> .txt 이렇게 있어야 하기 때문에, train_json으로 있는 것에는 문제가 있었다.

→ 이는 json 파일을 뜯어보고 하면,

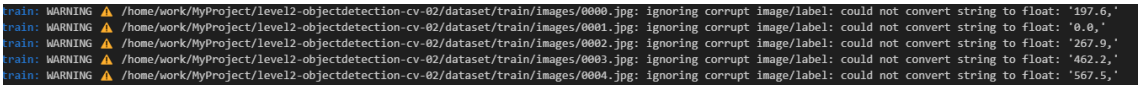


이렇게 규칙적으로 되어 있기 때문에 코드를 짜서 txt 파일로 만들 수 있었다.

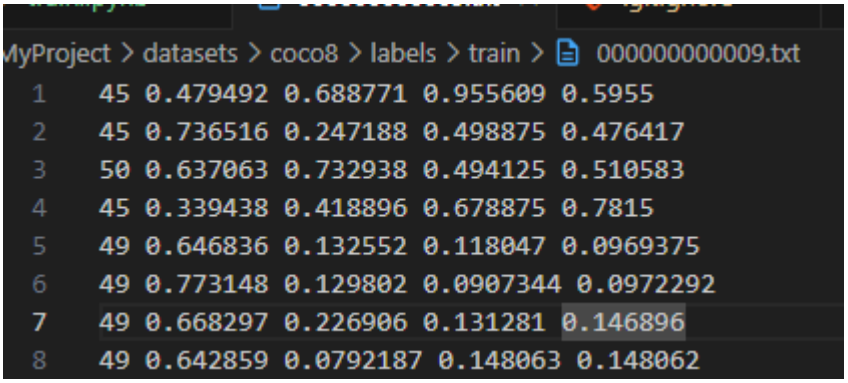
(cf. train파일이 0000.jpg 이면 label파일은 0000.txt로 이름이 같아야함)

`2) bbox 정규화 문제`

위에 같이 이름을 맞춤에도 불구하고



이런 WARNING 이 계속 뜰 것인데, 이는 Yolo가 bbox를 0~1 사이에서만 인식하기 때문이다..



coco 에서는 모델이 돌아가길래 뜯어보니까 label 부분에 이런식으로 되어 있음을 알 수 있었다. 즉 우리도 정규화를 해야 하는 것이었다.

위 두가지 task (라벨링문제와 bbox정규화문제)를 해결하니 모델이 동작함을 알 수 있었다.

3. config 설정

`yolo train cfg=config.yaml` 를 통해서 매개변수들을 조절할 수 있다.

```
# config.yaml

task: detect # (str) YOLO task, i.e. detect, segment, classify, pose, obb
mode: train # (str) YOLO mode, i.e. train, val, predict, export, track, benchm
ark

# Train settigs -------------------------------------------------------------
----------------------------------------
model: yolo11n.pt # (str, optional) path to model file, i.e. yolov11n.pt,yolov11
s.pt,yolov11m.pt,yolov11l.pt.yolov11x.pt
data: data.yaml # (str, optional) path to data file, i.e. coco8.yaml
epochs: 100 # (int) number of epochs to train for
time: # (float, optional) number of hours to train for, overrides epochs if sup
plied
```

```
patience: 10 # (int) epochs to wait for no observable improvement for early stopping of training
batch: 16 # (int) number of images per batch (-1 for AutoBatch)
imgsz: 640 # (int | list) input images size as int for train and val modes, or list[h,w] for predict and export modes
save: True # (bool) save train checkpoints and predict results
save_period: -1 # (int) Save checkpoint every x epochs (disabled if < 1)
cache: False # (bool) True/ram, disk or False. Use cache for data loading
device: 0,1 # (int | str | list, optional) device to run on, i.e. cuda device=0 or device=0,1,2,3 or device=cpu
workers: 32 # (int) number of worker threads for data loading (per RANK if DDP)
project: recycling # (str, optional) project name
name: experiment # (str, optional) experiment name, results saved to 'project/name' directory


# Val/Test settings ----------------------------------------------------------------------------------------
val: True # (bool) validate/test during training
split: val # (str) dataset split to use for validation, i.e. 'val', 'test' or 'train'
save_json: False # (bool) save results to JSON file
save_hybrid: False # (bool) save hybrid version of labels (labels + additional predictions)
conf: # (float, optional) object confidence threshold for detection (default 0.25 predict, 0.001 val)
iou: 0.8 # (float) intersection over union (IoU) threshold for NMS
max_det: 100 # (int) maximum number of detections per image
half: False # (bool) use half precision (FP16)
dnn: False # (bool) use OpenCV DNN for ONNX inference
plots: True # (bool) save plots and images during train/val
```

이렇게 조절할 수 있었으며, 그외에 것은 `default.yaml` 로 확인해볼 수 있다.

4. **다양한 모델 적용**

Yolov11 부터는 **n,s,m,l,x 이렇게 5가지 모델**을 사용할 수 있다.

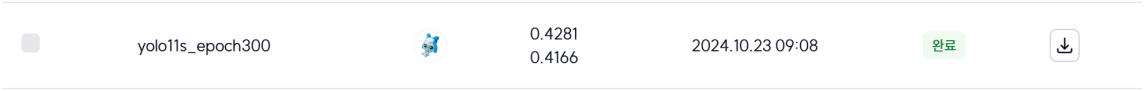
Model	Filenames	Task	Inference	Validation	Training	Export
YOLO11	yolo11n.pt yolo11s.pt yolo11m.pt yolo11l.pt yolo11x.pt	Detection	✅	✅	✅	✅

n,s,m,l,x는 파라미터 수의 차이이다.

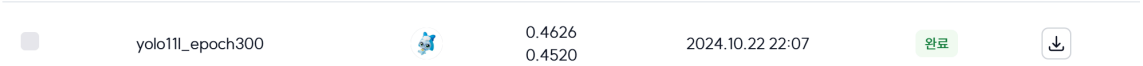
```
# Parameters
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=yolo11n.yaml' will call yolo11.yaml with scale 'n'
  # [depth, width, max_channels]
  n: [0.50, 0.25, 1024] # summary: 319 layers, 2624080 parameters, 2624064 gradients, 6.6 GFLOPs
  s: [0.50, 0.50, 1024] # summary: 319 layers, 9458752 parameters, 9458736 gradients, 21.7 GFLOPs
  m: [0.50, 1.00, 512] # summary: 409 layers, 20114688 parameters, 20114672 gradients, 68.5 GFLOPs
  l: [1.00, 1.00, 512] # summary: 631 layers, 25372160 parameters, 25372144 gradients, 87.6 GFLOPs
  x: [1.00, 1.50, 512] # summary: 631 layers, 56966176 parameters, 56966160 gradients, 196.0 GFLOPs
```

4.4.2 Yolo 모델 제출결과

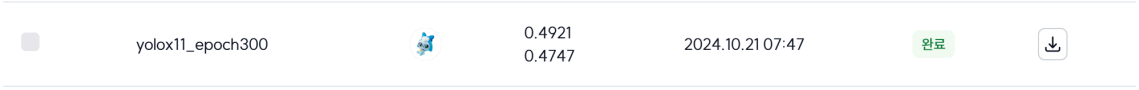
- **yolo11s.pt**



- yolo11l



- yolo11x



4.4.3 Evaluation 분석

- **모델 규모와 성능 상관관계**: 실험 결과, 모델의 크기가 증가함에 따라 성능이 뚜렷이 향상되는 경향을 보였다. YOLOv11s에서 YOLOv11l, 그리고 YOLOv11x로 갈수록 mAP, 정밀도, 재현율 모두 상승하였다.

- **모델 최적화 가능성**: 현재 결과는 YOLOv11의 잠재력을 완전히 발휘하지 못했을 가능성을 시사한다. 또한 하이퍼파라미터 튜닝, 데이터 증강, 또는 학습 전략 개선을 통해 성능을 더욱 향상시킬 여지가 있다.
- **실시간 응용 가능성**: 모델 파일(pt)의 크기가 상대적으로 작다는 점은 주목할 만하며 이는 실시간 응용 프로그램에 적용할 수 있는 잠재력을 제시하고 특히 엣지 디바이스나 모바일 환경에서의 활용 가능성을 높인다.
- **스케일링 효과**: YOLOv11l에서 YOLOv11x로의 성능 향상이 상대적으로 미미한 점은 모델 크기 증가에 따른 수확 체감 법칙을 보여주며. 이는 단순히 모델 크기를 키우는 것 이상의 최적화 전략이 필요함을 시사한다.

4.5 Ensemble

4.5.1 Ensemble_boxes library

- Object Detection 결과를 개선학기 위해 다양한 모델의 예측을 병합하는 ensemble 기법 제공
- 각기 다른 모델에서 생성된 Bounding Boxes를 결합하여 보다 신뢰성 있는 예측을 할 수 있게 함
- https://github.com/ZFTurbo/Weighted-Boxes-Fusion
- 이번 프로젝트에서는 Soft-NMS와 WBF를 사용

4.5.2 Non-Max Suppression(NMS)

- 입력 이미지에 Object Detection 알고리즘에 의해 bounding box regression이 적용되면, 객체에 여러 개의 bbox가 그려지며 물체의 확률 값을 가지게 됨
- 여러 개의 bbox 중 가장 스코어가 높은 박스만 남기고 나머지를 제거하여 모델을 간결하게 하고 중복된 결과를 제거함으로써 정확한 객체 탐지 수행

NMS 알고리즘의 동작 원리

1. **Object Detection 후보군 추출**
    a. 객체탐지 모델을 사용하여 입력 이미지에서 객체 후보군을 추출
    b. 각 후보 객체에 대한 추상적인 위치와 해당 객체가 어떤 클래스에 속하는지에 대한 Confidenc 값을 반환

2. **IoU(Intersection over Union) 계산 및 정렬**
    a. 계산된 IoU 값에 따라 후보 객체들을 높은 순서대로 정렬
    b. 주로 IoU가 가장 높은 객체를 선택하고, 최종 객체로 설정
3. **임계값(threshold) 적용**
    a. 일정 IoU 임계값 이상으로 겹치는 객체들을 제거
4. **반복 및 최종 객체 결정**
    a. 다중 클래스가 존재한다면, (1)~(3) 단계를 반복

4.5.3 Soft-NMS

- Soft-NMS는 **NMS(Non-Maximum Suppression)**의 변형으로, 특정 객체에 대해 모델이 예측한 여러 겹치는 박스를 단순히 제거하는 대신, 박스의 점수를 점진적으로 감소시키는 방식을 사용
- 원본 NMS가 겹친 박스 중 하나만 선택하는 과정에서 정보를 손실하는 문제를 보완하여, 보다 유연한 예측을 가능하게 함

Soft - NMS 알고리즘 동작 원리

1. 예측 박스들은 confidence score를 기준으로 내림차순 정렬
2. 가장 높은 score를 가진 박스를 기준으로 나머지 박스들과의 IoU를 계산
3. 겹치는 박스들이 IoU threshold를 넘을 경우, 단순히 제거하는 대신, IoU의 크기에 따라 score를 감소
    - score 감소는 다음과 같은 식으로 이루어짐
        - Linear 방식 : $s_j = s_j \times (1 - IoU(i, j))$
        - Gaussian 방식 : $s_j = s_j \times \exp(-\frac{IoU(i,j)^2}{\sigma})$

Soft-NMS의 주요 파라미터

- IoU threshold
    - 두 박스가 동일 객체로 간주될 기준이 되는 IoU 임계값
    - 겹치는 정도가 이 임계값 이상인 경우에만 score를 감소
- sigma

    - Gaussian 방식에서 박스 score 감소를 제어하는 파라미터
    - 이 값이 높을수록 score 감소폭이 작아지며, 낮을수록 score가 더 크게 감소
- thresh
    - 특정 점수 이하의 박스는 Soft-NMS에서 제거

Soft-NMS의 장단점

- 장점
    - 겹치는 예측이 많을 때 개별 객체를 놓치지 않고 보존 가능, 예측의 다양성 유지
- 단점
    - IoU threshold 및 점수 감소 방식을 잘못 설정할 경우 성능 저하를 초래할 수 있음

4.5.4 Weighted Box Fusion (WBF)

- 박스의 점수를 줄이거나 제거하는 대신, 가중치를 부여하여 하나의 새로운 박스를 생성하는 방식
- 여러 모델의 예측을 결합하여 보다 신회성 있는 예측을 생성할 때 유리

WBF 알고리즘 동작 원리

1. 모든 박스를 confidence score 기준으로 내림차순 정렬
2. 동일 객체에 해당한다고 판단되는 박스들은 IoU가 일정 threshold 이상인 경우 그룹으로 묶임
3. 그룹으로 묶인 박스들의 중심좌표 및 크기는 각 모델의 score를 기반으로 가중평균을 통해 새로운 박스 생성

WBF의 주요 파라미터

- weights
    - 예측에 대한 가중치 조정
- iou_thresh
    - 병합할 박스의 IoU 임계값 설정, 이 값을 초과하는 박스들만 동일한 객체로 간주하여 병합

- skip_box_thr
    - 특정 점수 이하의 박스를 병합 과정에서 제외시키는 임계값

Library usage examples

```
from ensemble_boxes import *

boxes_list = [[
    [0.00, 0.51, 0.81, 0.91],
    [0.10, 0.31, 0.71, 0.61],
    [0.01, 0.32, 0.83, 0.93],
    [0.02, 0.53, 0.11, 0.94],
    [0.03, 0.24, 0.12, 0.35],
],[
    [0.04, 0.56, 0.84, 0.92],
    [0.12, 0.33, 0.72, 0.64],
    [0.38, 0.66, 0.79, 0.95],
    [0.08, 0.49, 0.21, 0.89],
]]
scores_list = [[0.9, 0.8, 0.2, 0.4, 0.7], [0.5, 0.8, 0.7, 0.3]]
labels_list = [[0, 1, 0, 1, 1], [1, 1, 1, 0]]
weights = [2, 1]

iou_thr = 0.5
skip_box_thr = 0.0001
sigma = 0.1

boxes, scores, labels = nms(boxes_list, scores_list, labels_list, weights=wei
ghts, iou_thr=iou_thr)
boxes, scores, labels = soft_nms(boxes_list, scores_list, labels_list, weights
=weights, iou_thr=iou_thr, sigma=sigma, thresh=skip_box_thr)
boxes, scores, labels = weighted_boxes_fusion(boxes_list, scores_list, label
s_list, weights=weights, iou_thr=iou_thr, skip_box_thr=skip_box_thr)
```

- boxes_list
    - 3차원 리스트 형식
- scores_list, labels_list

- 2차원 형식

코드 구현

```
# ensemble.py

import pandas as pd
import os
from ensemble_boxes import weighted_boxes_fusion, soft_nms
import ast
import numpy as np

# 특정 경로에서 CSV 파일을 불러오는 함수
def load_csv_files_from_directory(directory_path):
    csv_files = [f for f in os.listdir(directory_path) if f.endswith('.csv') and not
f.startswith('._')]
    dfs = []
    for file in csv_files:
        file_path = os.path.join(directory_path, file)
        df = pd.read_csv(file_path, encoding='utf-8')
        df['PredictionString'] = df['PredictionString'].fillna('')
        dfs.append((file, df))  # 파일명과 데이터프레임을 함께 저장
    return dfs

# 예측 문자열을 파싱하여 필요한 데이터를 추출하는 함수
def parse_prediction_string(pred_string):
    if not pred_string:
        return []

    pred_values = list(map(float, pred_string.split()))

    parsed = []
    for i in range(0, len(pred_values), 6):
        if i + 5 < len(pred_values):
            parsed.append({
                'class_id': int(pred_values[i]),
                'score': pred_values[i + 1],
                'xmin': pred_values[i + 2],
```

```
            'ymin': pred_values[i + 3],
            'xmax': pred_values[i + 4],
            'ymax': pred_values[i + 5],
        })
    return parsed

# WBF를 적용할 수 있는 형식으로 데이터를 변환하는 함수
def prepare_wbf_input(df):
    boxes = []
    scores = []
    labels = []
    image_ids = df['image_id'].unique()

    for img_id in image_ids:
        img_boxes = []
        img_scores = []
        img_labels = []
        predictions = df[df['image_id'] == img_id]['parsed_predictions'].values
[0]

        for pred in predictions:
            img_boxes.append([pred['xmin'] / 1024, pred['ymin'] / 1024, pred['x
max'] / 1024, pred['ymax'] / 1024])
            img_scores.append(pred['score'])
            img_labels.append(pred['class_id'])

        boxes.append(img_boxes)
        scores.append(img_scores)
        labels.append(img_labels)

    return boxes, scores, labels, image_ids

# 최종 WBF 결과를 PredictionString 형식으로 변환하는 함수
def convert_to_prediction_string(boxes_nms, scores_nms, labels_nms):
    prediction_string = ''

    # 박스가 없거나 값이 없는 경우 빈 PredictionString 반환
    if len(boxes_nms) == 0 or len(scores_nms) == 0 or len(labels_nms) == 0:
```

```
        return prediction_string  # 빈 문자열 반환

    for i in range(len(boxes_nms)):
        box = boxes_nms[i]
        score = scores_nms[i]
        label = labels_nms[i]
        # box, score, label이 비어있지 않은지 확인 후 처리
        if len(box) == 4 and isinstance(label, (int, float)):
            prediction_string += f"{int(label)} {score:.6f} {box[0] * 1024:.1f} {box
[1] * 1024:.1f} {box[2] * 1024:.1f} {box[3] * 1024:.1f} "

    return prediction_string.strip()

# Soft-NMS 적용 함수
def apply_soft_nms(boxes, scores, labels, sigma=0.5, thresh=0.001, iou_thr
=0.5):

    unique_classes = np.unique(labels)  # 모든 클래스 값 추출
    boxes_nms_list, scores_nms_list, labels_nms_list = [], [], []

    for cls in unique_classes:
        # 각 클래스에 해당하는 바운딩 박스, 점수, 라벨만 추출
        cls_indices = [i for i, label in enumerate(labels) if label == cls]
        cls_boxes = [boxes[i] for i in cls_indices]
        cls_scores = [scores[i] for i in cls_indices]
        cls_labels = [labels[i] for i in cls_indices]

        # 각 클래스별로 Soft-NMS 적용
        boxes_nms, scores_nms, labels_nms = soft_nms(
            np.array(cls_boxes), np.array(cls_scores), np.array(cls_labels), iou_t
hr=iou_thr, sigma=sigma, thresh=thresh
        )

        # 빈 리스트 확인 후 확장
        boxes_nms_list.extend(boxes_nms)  # 리스트 확장
        scores_nms_list.extend(scores_nms)
        labels_nms_list.extend(labels_nms)
```

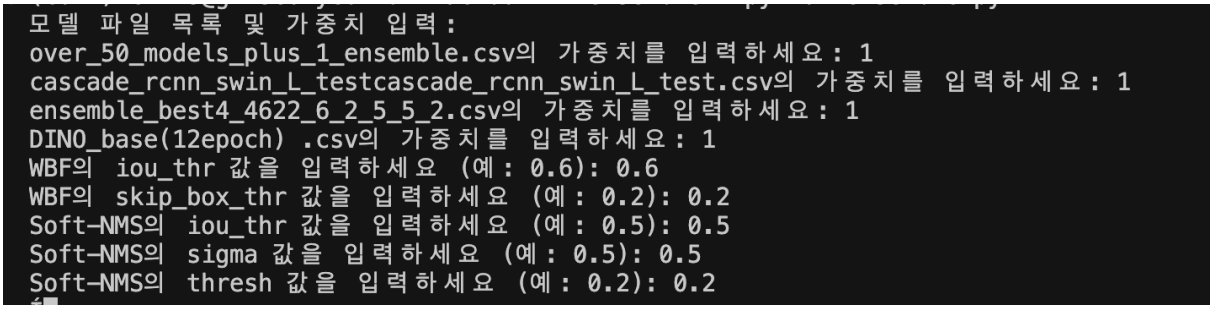
```
    return boxes_nms_list, scores_nms_list, labels_nms_list

# WBF와 Soft-NMS를 적용하는 함수
def ensemble_wbf_with_soft_nms(df_list, weights, iou_thr_wbf, skip_box_thr
_wbf, iou_thr_nms=0.5, sigma_nms=0.5, thresh_nms=0.001):
    for _, df in df_list:
        df['parsed_predictions'] = df['PredictionString'].apply(parse_predictio
n_string)

    # 여러 모델 예측을 결합할 준비 (각 모델의 예측을 결합)
    boxes_list = []
    scores_list = []
    labels_list = []
    image_ids = df_list[0][1]['image_id'].unique()

    for _, df in df_list:
        boxes, scores, labels, _ = prepare_wbf_input(df)
        boxes_list.append(boxes)
        scores_list.append(scores)
        labels_list.append(labels)

    # WBF 적용 및 결과 저장
    results = []
    for idx, image_id in enumerate(image_ids):
        model_boxes = []
        model_scores = []
        model_labels = []

        # 각 모델에 대한 예측을 확인하고 예측이 없는 경우 빈 리스트를 추가
        for i in range(len(df_list)):
            if idx < len(boxes_list[i]):
                model_boxes.append(boxes_list[i][idx])
                model_scores.append(scores_list[i][idx])
                model_labels.append(labels_list[i][idx])
            else:
                # 예측이 없는 경우 빈 리스트 추가
                model_boxes.append([])
                model_scores.append([])
```

```
            model_labels.append([])

        if model_boxes and any(model_boxes):
            # ----------------------------------------------------------------
------------------------
            # WBF 적용
            boxes_ens, scores_ens, labels_ens = weighted_boxes_fusion(
                model_boxes, model_scores, model_labels, weights=weights, iou_
thr=iou_thr_wbf, skip_box_thr=skip_box_thr_wbf
            )

            # ----------------------------------------------------------------
------------------------
            # Soft-NMS 적용
            # boxes_ens, scores_ens, labels_ens = apply_soft_nms(
            #     model_boxes, model_scores, model_labels, sigma=sigma_nms,
thresh=thresh_nms, iou_thr=iou_thr_nms
            #     )

            # model_boxes = np.array(model_boxes)
            # model_scores = np.array(model_scores)
            # model_labels = np.array(model_labels)

            # print('model_boxes:', model_boxes)
            # print('model_scores:', model_scores)
            # print('model_labels:', model_labels)

            # boxes_ens, scores_ens, labels_ens = soft_nms(
            #     model_boxes, model_scores, model_labels, sigma=sigma_nms,
thresh=thresh_nms, iou_thr=iou_thr_nms
            # )

            # print('boxes_ens:', boxes_ens)
            # print('scores_ens:', scores_ens)
            # print('labels_ens:', labels_ens)
            # ----------------------------------------------------------------
------------------------
```

```
            # 결과를 PredictionString 형식으로 변환
            prediction_string = convert_to_prediction_string(boxes_ens, scores_
ens, labels_ens)
            results.append({'image_id': image_id, 'PredictionString': prediction_
string})
        else:
            # 빈 결과 처리
            results.append({'image_id': image_id, 'PredictionString': ''})

    return pd.DataFrame(results, columns=['image_id', 'PredictionString'])

# 특정 디렉토리에서 모든 CSV 파일을 불러와 앙상블 실행
directory_path = './models'
model_dfs = load_csv_files_from_directory(directory_path)

# 모델 파일명 출력 및 가중치 입력 받기
weights = []
print("모델 파일 목록 및 가중치 입력:")
for i, (file_name, _) in enumerate(model_dfs):
    weight = float(input(f"{file_name}의 가중치를 입력하세요: "))
    weights.append(weight)

# WBF 및 Soft-NMS 관련 파라미터 입력 받기
iou_thr_wbf = float(input("WBF의 iou_thr 값을 입력하세요 (예: 0.6): "))
skip_box_thr_wbf = float(input("WBF의 skip_box_thr 값을 입력하세요 (예: 0.
2): "))
iou_thr_nms = float(input("Soft-NMS의 iou_thr 값을 입력하세요 (예: 0.5): "))
sigma_nms = float(input("Soft-NMS의 sigma 값을 입력하세요 (예: 0.5): "))
thresh_nms = float(input("Soft-NMS의 thresh 값을 입력하세요 (예: 0.2): "))

# 앙상블 실행
ensemble_df = ensemble_wbf_with_soft_nms(model_dfs, weights, iou_thr_
wbf, skip_box_thr_wbf, iou_thr_nms, sigma_nms, thresh_nms)

# 결과를 CSV 파일로 저장
ensemble_df[['PredictionString', 'image_id']].to_csv('ensemble_predictions.
csv', index=False)
```

- 주석을 이용하여 WBF, Soft-NMS를 한 코드에서 사용할 수 있도록 함
- 앙상블 하는 모델의 이름을 출력하도록 함
- 각 파라미터들을 input 값으로 설정할 수 있게 함

코드 실행 예시



4.5.5 앙상블 전략

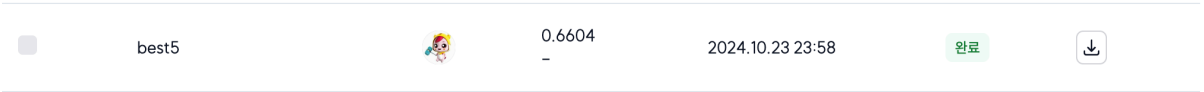
앙상블에 대한 EDA를 위한 코드

```
import pandas as pd
import cv2
import matplotlib.pyplot as plt
import numpy as np

# CSV 파일 경로 설정
csv_path = '/data/ephemeral/home/level2-objectdetection-cv-02/baseline/
mmdetection/work_dirs/cascade_rcnn_aug/ensemble_predictions_soft_nm
s.csv'  # 여기에 CSV 파일의 경로를 설정

# CSV 파일을 판다스로 읽어오기
df = pd.read_csv(csv_path)

# 필요한 이미지 목록 정의
selected_images = [f"test/000{i}.jpg" for i in range(10)]

# 클래스별 고유 색상 정의 (10개)
class_colors = {
    0: (255, 0, 0),    # 강한 빨강
```

```
    1: (0, 255, 0),    # 강한 초록
    2: (0, 0, 255),    # 강한 파랑
    3: (255, 255, 0),  # 밝은 노랑
    4: (255, 0, 255),  # 자홍 (마젠타)
    5: (0, 255, 255),  # 밝은 청록 (시안)
    6: (128, 0, 128),  # 보라
    7: (255, 165, 0),  # 주황
    8: (75, 0, 130),   # 인디고 (깊은 보라)
    9: (0, 128, 0)     # 진한 초록 (포레스트 그린)
}

# 각 이미지에 대해 바운딩 박스 정보 추출
def process_and_draw(image_name, data_row):
    # 이미지 경로 설정
    image_path = f"/data/ephemeral/home/level2-objectdetection-cv-02/dat
aset/test/{image_name.split('/')[-1]}"  # 이미지 폴더의 경로로 수정

    # 이미지 불러오기
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: {image_path} not found.")
        return

    # PredictionString의 값 분리
    predictions = data_row.split()

    # 바운딩 박스 정보를 하나씩 가져오기 (클래스, 스코어, xmin, ymin, xmax, ymax
순서로)
    num_predictions = len(predictions) // 6
    for i in range(num_predictions):
        class_id = int(predictions[i * 6])
        score = float(predictions[i * 6 + 1])
        xmin = int(float(predictions[i * 6 + 2]))
        ymin = int(float(predictions[i * 6 + 3]))
        xmax = int(float(predictions[i * 6 + 4]))
        ymax = int(float(predictions[i * 6 + 5]))

        # 클래스별 색상 가져오기
```

```
        color = class_colors.get(class_id, (255, 255, 255))  # 기본은 흰색

        # 바운딩 박스 그리기
        cv2.rectangle(image, (xmin, ymin), (xmax, ymax), color, 2)
        label = f"Class {class_id}: {score:.2f}"
        cv2.putText(image, label, (xmin, ymin - 10), cv2.FONT_HERSHEY_SIMP
LEX, 0.5, color, 1)

    # 결과 이미지 표시
    plt.figure(figsize=(8, 8))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(image_name)
    plt.axis('off')
    plt.show()

# 선택된 이미지들에 대해 처리
for image_name in selected_images:
    row_data = df[df['image_id'] == image_name]['PredictionString'].values
    if row_data.size > 0:
        process_and_draw(image_name, row_data[0])
    else:
        print(f"No data found for {image_name}")
```

📌 WBF + Soft NMS

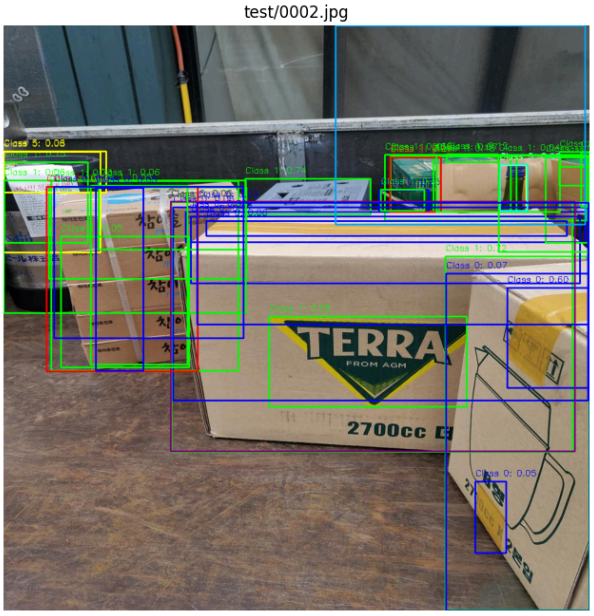


- 제출 모델 중 최고 성능 csv확인
- 0000.jpg ~ 0003.jpg 까지 확인 중

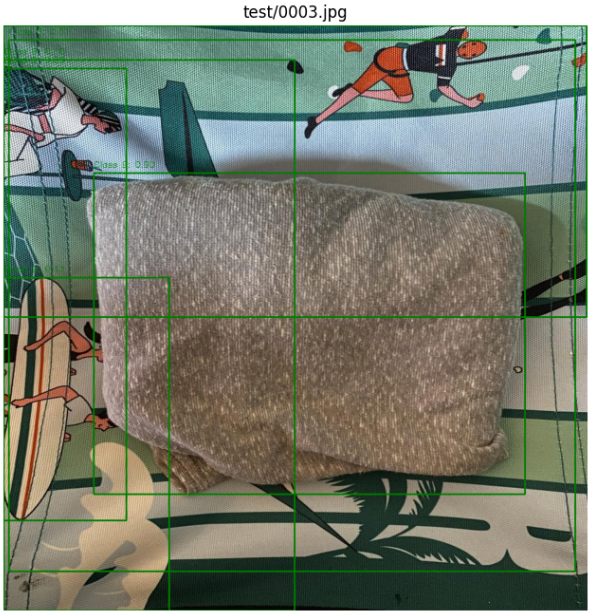
test/0000.jpg

test/0001.jpg

test/0002.jpg



test/0003.jpg



- 대략 0.2 ~ 0.3 아래는 무시해도 될 예측값들임
- 0.25 아래는 무시하기로 결정
- 무시하고 다시 확인

	ensemble_0.25_test		0.5846 -	2024.10.24 00:51	완료	

- 불필요한 바운딩 박스들이 줄어들었지만 리더보드 제출 시 점수가 떨어짐

test/0000.jpg

test/0001.jpg

test/0002.jpg

test/0003.jpg

📌 WBF

- WBF만 사용하여 진행해봄
- 모든 모델의 가중치는 1, 0.5가 넘는 모델들만 사용

	over_50_mode...emble		0.6634 -	2024.10.24 01:41	완료	

📌 Soft-NMS

- Sofg-NMS만 사용하여 진행해봄
- WBF와 동일한 조건

	soft_nms_5_7_1		0.5747 -	2024.10.24 15:55	완료	

- 성능 저하를 확인하여 WBF만 사용하기로 결정
- 또한 리더보드에서 평가 기준이 Bbox가 많으면 많을 수록 좋을 것 같다고 생각하여 시각화 진행

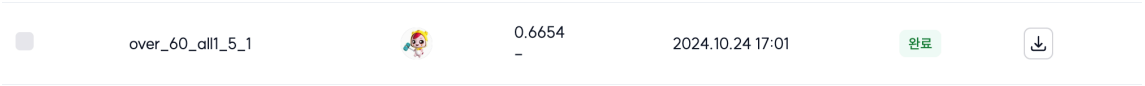
→ WBF

→ Soft-NMS

확실히 WBF가 박스가 더 많다

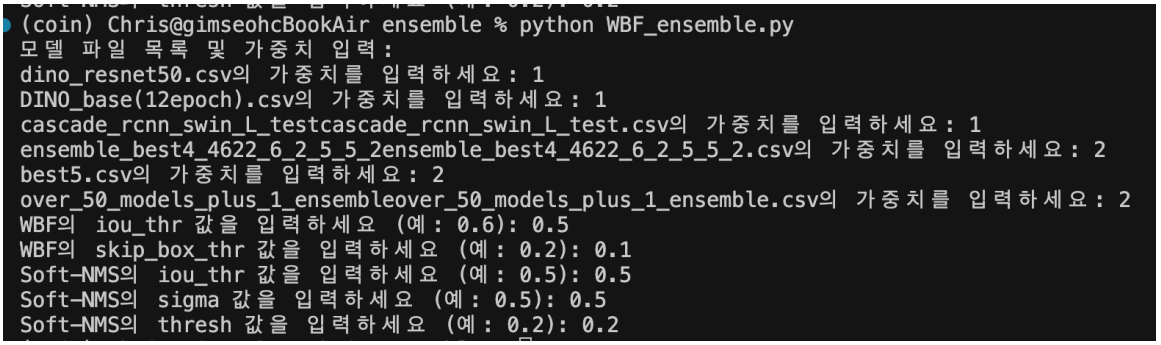
WBF로 최종 output 뽑기

- 0.6 넘는 csv만 추출하여 진행
- 앙상블 한 csv도 포함



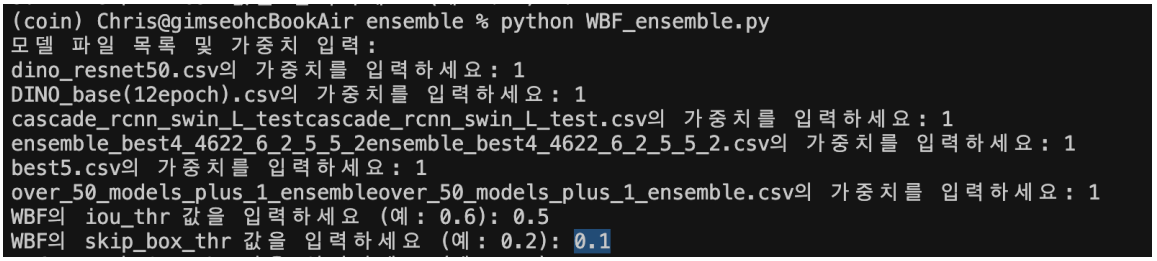
= 소량 상승

- cascade-rcnn-resent 모델 추가
- 앙상블 csv에는 가중치를 2, 기본 모델에는 1 적용하여 진행



= 소량 감소

- 가중치 모두 1로 진행



= 최고점보다 소량 감소

- 앙상블 후 0.66이 넘는 csv만 진행
- 가중치 모두 1로 진행

```
(coin) Chris@gimseohcBookAir ensemble % python WBF_ensemble.py
모델 파일 목록 및 가중치 입력:
over_60_all1_5_1.csv의 가중치를 입력하세요: 1
origin1_ensemble1.csv의 가중치를 입력하세요: 1
origin1_ensemble2.csv의 가중치를 입력하세요: 1
best5.csv의 가중치를 입력하세요: 1
over_50_models_plus_1_ensembleover_50_models_plus_1_ensemble.csv의 가중치를 입력하세요: 1
WBF의 iou_thr 값을 입력하세요 (예: 0.6): 0.5
WBF의 skip_box_thr 값을 입력하세요 (예: 0.2): 0.1
```

	ensemble_over_66		0.6511 -	2024.10.24 17:22	완료	

= 감소

- 이제까지 가장 좋았던 실험에서 파라미터 수정 후 진행

```
(coin) Chris@gimseohcBookAir ensemble % python WBF_ensemble.py
모델 파일 목록 및 가중치 입력:
cascade_rcnn_swin_L_testcascade_rcnn_swin_L_test.csv의 가중치를 입력하세요: 1
ensemble_best4_4622_6_2_5_5_2ensemble_best4_4622_6_2_5_5_2.csv의 가중치를 입력하세요: 1
best5.csv의 가중치를 입력하세요: 1
DINO_base(12epoch) .csv의 가중치를 입력하세요: 1
over_50_models_plus_1_ensembleover_50_models_plus_1_ensemble.csv의 가중치를 입력하세요: 1
WBF의 iou_thr 값을 입력하세요 (예: 0.6): 0.4
WBF의 skip_box_thr 값을 입력하세요 (예: 0.2): 0.04
```

	ensemble_ove...emble		0.6360 -	2024.10.24 17:29	완료	

= 대량 감소

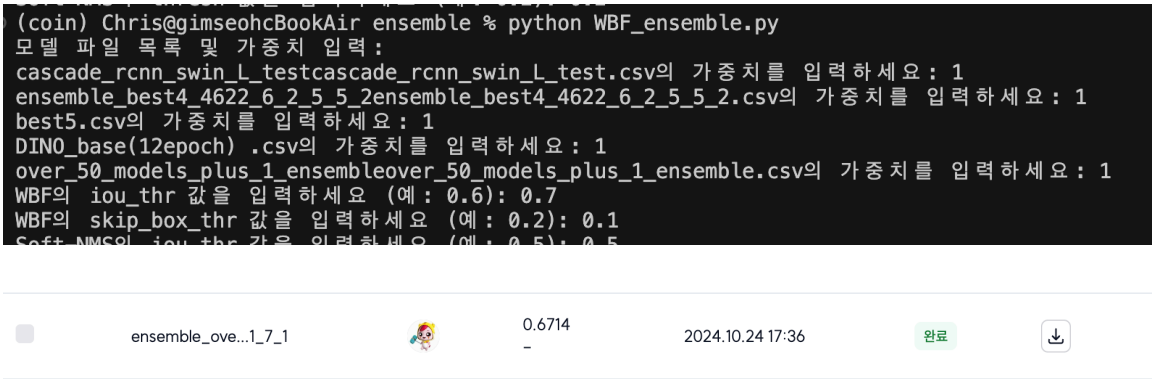
- 결과를 바탕으로 수정 후 진행

```
(coin) Chris@gimseohcBookAir ensemble % python WBF_ensemble.py
모델 파일 목록 및 가중치 입력:
cascade_rcnn_swin_L_testcascade_rcnn_swin_L_test.csv의 가중치를 입력하세요: 1
ensemble_best4_4622_6_2_5_5_2ensemble_best4_4622_6_2_5_5_2.csv의 가중치를 입력하세요: 1
best5.csv의 가중치를 입력하세요: 1
DINO_base(12epoch) .csv의 가중치를 입력하세요: 1
over_50_models_plus_1_ensembleover_50_models_plus_1_ensemble.csv의 가중치를 입력하세요: 1
WBF의 iou_thr 값을 입력하세요 (예: 0.6): 0.6
WBF의 skip_box_thr 값을 입력하세요 (예: 0.2): 0.1
```

	ensemble_ove...1_6_1		0.6685 -	2024.10.24 17:32	완료	

= 상승

- 마지막 실험



0.6714로 상승

5. 자체 평가 의견

- 민솔 : Stratified K-Fold를 적용하지 못하여, Training 과정에서 제대로 된 validation이 이루어지지 않았던 점이 가장 아쉬웠습니다. 또한, Co-DETR과 같은 detector 모델의 스펙업이 이루어지지 못한 점도 아쉬운 것 같습니다. 마지막으로, 권희 캠퍼의 YoloV4 논문 리뷰에서 Mosaic augmentation을 활용하였을 때의 성능 향상 정도를 파악할 수 있었는데, 코드로 구현하지 못한 점이 아쉬웠습니다.
- 현진 : 좋았던 점은 첫번째로, 기록을 잘 남긴 것. 두번째로, 데일리 미팅을 통해 진행도가 공유되었던 것이었습니다. 반면, 아쉬웠던 점으로는 첫번 째로 제대로된 Validation Set을 구현하지 못한 것. 두번째로, OOM문제를 많이 겪었는데 모델 실험에 바빠 Gradient Accumulation과 같은 기법을 적용하지 못한 것. 세번째로, 시간이 부족해서 CO-DETR을 끝까지 학습 못해본 것 입니다.
- 준현
  1. 서버가 견고하지 못했음
     - 서버 기본 세팅을 할 때부터 무거운 모델을 돌릴 때까지 서버가 시도때도 없이 터졌음
  2. Detctron2 Library의 아쉬운 사용
     - 내장된 함수가 부족하여, 특정 기능을 구현할 때 대부분의 코드를 직접 작성해야 했음
       - Trouble Shooting 과정에서 시간을 많이 투자함
     - 심지어 성능이 잘 나오는 것도 아님..
  3. Competition 에서의 아쉬운 평가 지표(mAP-50)

        - Object Detection model를 하나의 평가 지표로만 순위를 내다보니 박스가 많아지면 score가 오르는 현상 발생
            - Ensemble EDA 과정에서 혼선을 겪음

- 영서
    - EDA를 다같이 진행하지 못한 것에 대한 아쉬움이 있음. 각자 맡은 바를 수행하면서 당장 궁금한 점이나 필요한 부분을 해소하는 형태로 가져간게 아쉽다. 다음에는 EDA를 우선적으로 다같이 진행하고 데이터셋의 특이점을 모두가 같이 말로써 공유하는 시간이 반드시 필요할 것 같다.
    - Validationi Set을 따로 구현하지 못한 점이 아쉽다. 주기적으로 성능을 파악하고 하는 데에 어려움을 겪었다. 결과적으로는, 쓰지 않아도 될 시간을 더 많이 할애한 것 같다.
    - 서버를 보다 더 효과적으로 사용할 방법이 필요한 것 같다. 쉬지 않는 학습을 돌릴 수 있도록 고안해야할 것 같다.

- 재건
    - YOLO 모델을 다양한 크기로 실험하여 성능변화를 분석했고 데이터전처리와 하이퍼파라미터 조정으로 모델 최적화를 이끌어냈다. 다만, 다른 탐지 라이브러리를 시도하지 못해 경험하지 못하였고, 프로젝트 후반에는 집중도가 낮아져 온전히 집중하지 못한 것이 아쉽다. 향후에는 앙상블과 다양한 모델 실험으로 탐지 성능을 높이고 모델에 대한 이해도도 높이고자 한다.

- 권희
    - Validation set 구성 못했다. Stratified K-Fold를 적용하지 못한 부분이 아쉽다.
    - Ensemble 과정을 조금 더 구체적으로 가져갔어야 한다.
    - Detectron2의 모델들과 코드를 조금 더 이해하고 진행했어야한다고 생각한다.
    - 데일리 미팅과 작업 기록으로 level1보다 더 협업에 대한 구조를 잘 가져갔다.

---

6. 개인 회고

김민솔_T7111

- 나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

단순한 SOTA 모델을 사용하기 보다는, Competition의 목적을 파악하고, 이를 바탕으로 한 실험들을 설계하는 목표를 설정하였습니다. EDA 결과를 바탕으로 모델을 설정하는 것에 집중하였고, 강의 및 논문들을 통해 detector 모델들의 발전 방향들을 토대로 detector 및 backbone을 설정하였습니다. 추가적으로, small objects를 효과적으로 잡을 수 있는 기법들을 찾아보고, input image size를 늘리는 등의 실험들을 적용해보았습니다.

또한, Github 협업 뿐만 아니라, notion을 활용하여 실험 기록들을 체계적으로 남기자는 목표를 두었습니다. 팀 내에서 이름과 기록 종류(model, docs 등)와 함께 각자의 실험 결과들을 기록하고, 공유하는 과정과 함께 프로젝트를 진행하였습니다. 현진 캠퍼의 작업 기록에 결과 분석에 대한 피드백을 댓글로 작성하는 등, 실험에 대한 소통을 활발히 하였습니다.

- 전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

저번 프로젝트 발표에서, 마스터님이 실험에 대한 가정을 설계하고 적용하는 과정이 체계적이지 못했다고 언급해주셨습니다. 따라서, 이번 프로젝트에서는 체계적인 실험을 만들어가는 것에 집중하였습니다. 크기가 작은 피사체가 많다는 Dataset의 특성에 맞게, base 모델을 설정하고, 한 단계씩 data augmentation을 적용하여 실험에 대한 결과 분석을 하나씩 기록하였습니다. 이를 통해, 원인 분석을 더 효과적으로 이끌어낼 수 있었고, 작업 기록들을 더 꼼꼼하게 적는 효과를 얻을 수 있었습니다.

- 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

Mosaic augmentation에 대한 트러블 슈팅에서 많은 시간을 소요한 점이 아쉬웠습니다. 또한, 라이브러리에 관계 없는 공통의 과제였던 Stratified K-Fold 구현을 모두가 미루다 보니, 마지막에도 구현하지 못한 점이 아쉬웠습니다.

- 한계/교훈을 바탕으로 다음 프로젝트에서 시도해보고 싶은 점은 무엇인가?

작은 기능들을 구현하는 것보다, 모두가 필요로 하는 코드를 나서서 먼저 구현하는 것이 팀원들을 위한 것이라는 교훈을 얻었습니다. 또한, 너무 자율적으로 역할을 부여하기 보다는, 각자 고른 양의 업무를 확실하게 정하는 것이 더 발전적인 협업으로 이뤄질 것이라는 실험도 적용해보고 싶습니다.

김현진_T7140

- 나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

첫번째로, 이번에 맡은 부분이 모델이기 때문에 성능지표나 캐글 등에서 정보를 찾아 근거있는 학습을 진행하고 싶었습니다. 두번째로, 프로젝트가 끝나고 나면 내용이 금방 휘발되게 때문에 작은 거라도 기록해두려고 했습니다. 마지막으로, 좀 더 적극적으로 나서서 다양한 도전을 해보고자했습니다.

- 전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

지난 프로젝트에서 아쉬웠던 점 세가지를 뽑는다면, 근거, 기록, 시간부족이었습니다.

근거를 위해서는 첫번째로는 성능지표를 바탕으로 모델을 고랐습니다. 두번째로는 모델의 학습결과와 EDA를 통해 얻은 사실을 바탕으로 이에 맞는 모델을 찾았습니다. 그 결과 비교적 짧은 시간 안에 다양한 모델을 실험하고 좋은 성능을 가진 모델을 빨리 찾을 수 있었습니다.

기록을 위해서는 우선 팀전체가 매일 아침 데일리 스크럼을 작성하고 피어세션 때 어디까지 어떻게 진행되었는지 이야기를 나누었고, 작업기록 페이지를 만들어 라이브러리 설명부터 실험결과, 트러블 슈팅까지 작성해 다른 사람들도 읽을 수 있고 의견을 나눌 수 있게 만들었습니다. 덕분에 모델의 결과를 더 잘 이해할 수 있었고, 비슷한 문제가 발생했을 때 이전 글을 참고해서 해결할 수 있었습니다.

지난번의 시간부족은 강의를 듣느냐 미뤄두고 진행했던것이 문제였다 생각을 해서, 이번에는 조금 더 일찍 시작을 했습니다. 다만, 전보다 사용한 모델들의 학습이 오래걸려서 이부분은 성공적이지 못했던거같습니다.

- 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

MMDetction을 3.3으로 업데이트 하는 것, Mosaic와 Mixup을 사용하려고했을 때 에러 발생한 것, 갑자기 모델의 학습이 0이 되버린 것 이 세가지를 해결하는 과정에서 한계를 느꼈습니다.

모델의 구조가 복잡하다보니 타고들어가도 원인을 잘 찾지못하는 경우가 많았고, 트리구조로 되어 있을때는 오류가 나다가 한개의 config 파일로 구성하면 괜찮아지는 등의 해결방식이 많아 아쉬웠습니다.

그리고 여전히 시간문제에서 한계를 느꼈습니다. 지난 번의 학습시간을 생각하고 접근했는데. Base로 사용한 모델은 기본적으로 하루는 학습을 시켜야했고 트러블슈팅과정을 겪으며 다양한 Augmentation을 적용해보지 못했습니다. 결국 학습을 다 마치지 못해 사용하지 못한 모델들도 있었습니다. 서버가 쉬지않고 학습되도록 짜야한다는 걸 다시금 느꼈습니다.

- 한계/교훈을 바탕으로 다음 프로젝트에서 시도해보고 싶은 점은 무엇인가?

다음 프로젝트 때에는 최대한 로컬에서 할 수 있는 작업은 로컬에서 작업하고, 서버는 쉬지않고 학습을 할 수 있도록 하고 싶습니다.

정권희_T7243

- 잘했던 점

    - 사실 detectron2에 대한 이해도가 지금도 많이 낮아서 성능이 안 좋았던 거라고 생각하지만 그래도 데이터도 눈으로 확인하고 문제에 대한 고찰을 해보려 노력하였고 모델을 테스트할 때 백본의 성능, 팀원들의 진행사항 등등을 확인하며 진행했던 부분이 잘 했다고 생각하고 또한 노션에 작업 기록을 순간순간 많이 남기지는 못했지만 그래도 실험에 대한 부분은 모두 기록하여 프로젝트 기간 중 했던 과정들이 눈에 보이게 남아 잘했던 것 같습니다. 또한 앙상블 시 예측된 바운딩 박스를 직접 눈으로 보며 성능을 높이기 위한 분석을 진행했던 부분이 잘했던 부분이라 생각합니다.
- 시도 했으나 잘 되지 않았던 점
    - detectron2의 모델의 성능을 높이기 위해 많은 노력을 했지만 성능이 잘 나오지 않았던 부분이 아쉽고 augmentation을 적용할 때마다 성능이 떨어졌던 부분이 잘되지 않았던 것 같습니다.
- 아쉬웠던 점들
    - detectron2의 모델을 조금 더 잘 사용하고 초반 코드를 뜯어볼 때 조금 더 집중해서 몰입해서 했다면 더 나은 성능을 가지는 모델 하나 정도는 가져오지 않았을까 아쉽고 또한 augmentation 적용에 대한 이해도 부족했기에 적용 시 모델의 성능이 개선되지 않았던 부분이 아직 이 프로젝트 데이터를 파악하지 못했다는 생각이 들어 아쉽습니다. 앙상블 과정에서도 파라미터를 수정하고 직접 그려보는 것 까지는 좋았지만 시간과 리더보드 제출 횟수 제한으로 인해 더 많은 실험으로 최고의 성능을 가지고 갈 수 있는 앙상블을 하지 못했던 게 너무 아쉽습니다.
- 프로젝트를 통해 배운 점 또는 시사점
    - 결국 라이브러리와 pre-trained 모델을 사용한다고 해도 그 기초가 되는 코드 이해와 정확한 사용법을 알아야 더 나은 성능, 더 나은 프로젝트 과정을 가지고 갈 수 있는 것을 깨닫게 되었고 EDA와 앙상블, augmentation 모두 적절한 사용 방식과 과정이 아직 많이 부족하다는 것, 즉 인공지능 프로젝트를 아직 많이 경험해 보지 못한 경험에 대한 부족함을 많이 느끼게 되었고 이번 프로젝트를 통해 조금은 더 인공지능에 대한 세세한 부분을 배우게 되었습니다. 다음 프로젝트, 그다음 프로젝트로 갈수록 더 나은 식견을 가질 수 있다는 믿음을 가지고 나아가겠습니다.

김준현_T7130

- 잘했던 점
    - Detectron2 모델들의 augmentation code를 직접 작성하며 Cutmix, Mosaic 등의 각종 augmentation 기법들의 구동방식을 익힐 수 있었음

    - Prediction들의 EDA를 streamlit으로 구현하며 streamlit의 코드들을 조금이나마 익힐 수 있었음
    - Model Ensemble을 수행하며 비교적 미약한 Github baseline code를 모델들이 예측한 csv 파일들에 적용하는 방법을 채택함
        - 다양한 종류의 모델들을 개수에 상관 없이 Ensemble을 할 수 있었음
        - 파라미터 조절이나 모델명, 가중치들을 확인하기 쉽게 코드를 짬
- 시도 했으나 잘 되지 않았던 것들
    - 시간이 촉박하여 Ensemble 이후의 Prediction에 대해 streamlit을 이용한 EDA를 구현하지 못함
    - Ensemble_boxes library의 사용에서 내장함수의 output의 형태 에러를 해결하는데 시간이 오래 걸림
        - 인풋은 [1,1,1] 형태, 아웃풋은 [1 1 1] 형태
        - 하필 WBF와 Soft_NMS를 섞는 시도를 하느라 형태에 관한 에러가 끊이질 않았음
    - Detectron2에 Mosaic를 적용할 때 random crop, center crop를 적용하려했지만, 성능이 반토막..
- 아쉬웠던 점들
    - mAP-50 하나만으로의 score 계산
        - 최종 bbox EDA의 신뢰도가 매우 낮았음
    - Detectron2 모델을 빨리 결단내리지 못함
        - 투자한 시간 대비 성능이 안나왔음
- 프로젝트를 통해 배운 점 또는 시사점
    - 호환성에 주안점을 두고 코드를 작성하니 코드 작성시간은 오래 걸렸어도, 구현 이후 짧은 시간에 많고 다양한 모델을 ensemble 할 수 있었음
    - 숫자에 기반한 근거를 갖고 전략을 잘 짜야한다는 것을 느낌
        - bbox와 score 사이의 간극, 각종 파라미터들의 튜닝 과정에서의 전략적 비교군 형성
    - score가 수렴하는 징조를 보인다면, 소수점 아래의 값 상승에 연연하지 말고, 다른 모델이나 다른 기법들을 적용을 빠르게 해야 할 듯

윤영서_T7172

- **좋았던 점**:
  - augmentation을 통해 작은 객체를 효과적으로 부각시키고, 다양한 기법을 실험하면서 적절한 조합을 찾아낼 수 있었던 점이 의미 있었습니다.
  - 특히, EDA 단계에서 데이터셋의 특징을 명확히 파악한 것이 성능 향상에 도움이 되었습니다.
- **아쉬운 점**:
  - 실험의 다양성에 비해 시간이 부족하여 일부 설정의 조정이나 추가 검증이 미흡했습니다.
  - 특히 중간 크기 객체에 대한 추가 검토가 필요했으며, 시간 제약으로 최적화 과정을 간소화한 부분이 아쉬움으로 남았습니다.
- **향후 개선 방향**:
  - augmentation 설정에 대한 실험을 자동화하고, 다양한 배치 크기와 학습률에 따른 성능 변화를 모니터링하여 더 안정적인 성능 개선을 목표로 삼고자 합니다.
  - 또한, 로컬과 서버를 구분해서 최대한 학습 시도를 많이 해보고 싶습니다.
  - 차후에 채택이나 디벨롭시킬 수 있을 것 같은 바식은 다음과 같습니다.

    1. Grid Search 또는 Hyperparameter Optimization Framework (Optuna 등)를 사용해 augmentation 파라미터를 조합하고 자동화된 실험 진행
    2. Scheduler 또는 Orchestration Tool을 통해 실험을 병렬 실행하여 시간을 절약하고 효율적으로 실험
    3. 결과 모니터링 툴 (MLflow 등)을 통해 최적의 augmentation 설정을 시각적으로 분석

이재건_T7227

- 좋았던 점
  - YOLO 모델을 다양한 크기(n,s,m,l,x)로 실험하여 크기에 따른 성능 변화를 분석하고 이해할 수 있었다.
  - 데이터 라벨링 및 bbox 정규화 문제를 해결하며 데이터 전처리의 중요성을 실감하였고, 이를 통해 모델이 원활하게 학습할 수 있는 환경을 마련했다

    - YOLO의 config 설정을 통해 하이퍼파라미터를 세부적으로 조정하며 모델을 최적화할 수 있었다. 특히, 실시간 응용 가능성을 고려해 작은 크기의 모델(yolov11s)로도 만족스러운 결과를 도출할 수 있음을 확인했다.
- 아쉬운 점
    - YOLO 외 다른 객체 탐지 라이브러리를 경험할 시간이 부족하여 실험의 다양성을 확보하지 못했다. 이를 통해 YOLO의 성능을 다른 모델과 비교하여 더욱 깊이 있는 분석을 할 기회를 놓친 점이 아쉽다.
    - 모델 앙상블을 다양하게 시도하지 못한 점이 아쉬웠다.
    - 마지막에는 새로운 팀을 구성하는데 시간을 많이 소모하여 이번 프로젝트에 집중하지 못한게 아쉽다.
- 향후 개선방향
    - **모델 앙상블**: 여러 YOLO 모델을 앙상블하여 성능을 높이고, 특히 다양한 크기의 모델을 결합해 작은 객체와 큰 객체에 대한 탐지 성능을 동시에 강화하는 전략을 고려할 계획이다
    - **다른 라이브러리 실험**: YOLO 외 다른 탐지 모델(DINO, DETR 등)과 비교하여 다양한 탐지 방식과 성능을 비교 분석함으로써 YOLO의 장단점을 더 잘 이해하고 최적의 선택을 할 수 있도록 할 것이다-
    - **하이퍼파라미터 튜닝**: 학습 전략 개선과 데이터 증강 기법을 추가 적용하여 성능 향상 가능성을 탐구하고, 성능과 처리 속도 간의 균형을 맞출 수 있는 방안을 모색하겠다