# 도메인 프로젝트 1 Object Detection 랩업 리포트 (김해찬_T8058)

## 1. 프로젝트 개요

### 1-1. 문제 정의

- 대회명: 재활용 품목 분류를 위한 Object Detection
- 목표: 10종 쓰레기 객체(일반, 플라스틱, 종이, 유리, 비닐 등)를 사진에서 탐지하고 클래스별로 분류
- 입력 데이터
  - 이미지: 주로 1024×1024 해상도, 한 장에 여러 개의 쓰레기 객체가 포함
  - 학습 어노테이션: COCO format `train.json` (10개 class, bbox 좌표 및 category id 포함)
  - 테스트: bbox 없이 이미지 파일만 제공
- 출력 형식
  - 제출 csv의 `PredictionString` 컬럼에

    `class score x y w h ...` 형식으로 bbox들을 공백으로 이어서 기록
- 평가지표
  - mAP50 (IoU 0.5에서의 mean Average Precision)

데이터 특성상 작은 객체(배터리, 병뚜껑)와 큰 객체(박스, 의류)가 공존하고,

하나의 이미지에 bbox가 많이 겹쳐 있는 cluttered scene이 많다.

또한 클래스별 등장 빈도가 달라 class imbalance도 존재한다.

이 특성 때문에

1. 작은 객체에 강한 구조와 multi-scale augmentation,
2. 과적합을 막기 위한 적절한 regularization,
3. 박스가 많이 겹칠 때도 안정적인 NMS/앙상블 전략이 중요하다고 판단했다.

## 1-2. 팀 구성과 내 역할

- 팀 전체
  - DDQ-DETR Swin-L, CODetr Swin-L, Cascade R-CNN 등 다양한 모델을 분담해 학습
- 김해찬 (본인) 역할
  - MMDetection 기반 추가 모델링
    - Faster R-CNN, Cascade R-CNN, VFNet, Deformable DETR, TOOD, Sparse R-CNN, RTMDet 등
  - DETA 4-fold 실험과 inference 스크립트 설계·디버깅
  - submission csv를 입력으로 사용하는 박스 레벨 앙상블 파이프라인 구현
    - `ensemble_from_submissions.py`
  - 여러 번의 앙상블 실험(ensemble, ensemble_2, ensemble_4, ensemble_5) 설계 및 튜닝
  - 실패한 실험(RTMDet, DETA, 초기 앙상블 등)의 원인 분석과 정리

---

# 2. 단일 모델 실험 정리

## 2-1. 내가 직접 학습한 모델과 성능

(리더보드 기준, mAP50)

- mmdet_faster_rnn: 0.4177
- DETR: 0.2520
- Cascade R-CNN (Swin-L 기반): 0.4642
- VFNet (ResNeXt101-64×4d): 0.3751
- VFNet + Cascade 간 간단 앙상블(vfnet_cascade_ensemble): 0.4401
- rcnn_sin (RCNN 변형 실험): 0.4760

> infer_single_cascade.ipynb

train_single_cascade_full.ipynb

- TOOD: 0.2686
- Sparse R-CNN: 0.2873

Sparse R-CNN_train.ipynb

Sparse R-CNN_inference.ipynb

- RTMDet: 0.0468

RTMDet_train.ipynb

RTMDet_utill.ipynb

RTMDet_inference.ipynb

- DETA / DETA_nms: 0.0031 / 0.0025

train_deta_4fold_v3.py

infer_deta_4fold.py

infer_deta_4fold_ensemble_csv.py

make_4fold_coco.py

이 중 single 기준 내 최고 성능 모델은

Swin-L backbone을 사용한 rcnn_sin 계열(약 0.4760)과 Cascade R-CNN(0.4642)였다.

## 2-2. 팀원 모델 (앙상블에 활용된 것만 정리)

- DDQ-DETR Swin-L 계열
  - ddq_swinl_1024_18e_full 또는 ddq_swinl_10_full: 약 0.6185
  - ddq_swinl_1024_12e: 0.5986
- Cascade R-CNN 1024 12e (팀원): 0.5320
- CODetr Swin-L 1024 7e: 0.6910 (팀 단일 최고 성능)

최종 앙상블은 이들 high-perform 모델들을 조합해서 만들었다.

ensemble_from_submissions.py

▼ **전체 모델 선정 전략 개요 및 정리**

# 전체 모델 선정 전략 개요

다음과 같은 단계적 전략에 따라 모델을 선정하고 실험을 진행했다.

1단계

검증된 two-stage 계열(Faster R-CNN, Cascade R-CNN)로 파이프라인을 안정화하고, 대략적인 성능 상한선을 파악한다.

2단계

데이터 특성(클래스 불균형, 작은 객체, cluttered scene)을 고려하여 one-stage 계열(VFNet, TOOD, RTMDet)과 query 기반(Sparse R-CNN)을 추가로 실험한다.

3단계

Transformer 계열(Deformable DETR, DETA, DDQ-DETR, CODetr)을 도입해 구조적으로 다른 모델 확보하고, 이후 앙상블 후보를 확장한다.

4단계

단일 모델 성능이 일정 수준에 도달하면, submission csv를 활용한 박스 레벨 앙상블로 최종 성능을 향상시킨다.

아래는 각 구조별 선정 의도, 실험 결과, 관련 용어를 정리한 내용이다.

---

# 1. Faster R-CNN / Cascade R-CNN

## 1.1 선정 배경

- Faster R-CNN, Cascade R-CNN은 COCO·VOC 등에서 장기간 검증된 안정적인 two-stage detector이다.
- 새로운 대회 환경에서는 다음 요소를 먼저 확인할 필요가 있다.
  - 데이터 파이프라인 정상 동작 여부
  - augmentation 적용 상태
  - 학습 루프 및 평가 코드의 정상 동작 여부
- 구현 난이도가 낮고 결과 해석이 용이한 모델을 baseline으로 사용하여, 전체 파이프라인의 기준선 성능을 확보하는 것을 목표로 했다.
- 본 대회 데이터는
  - 객체 크기 편차(작은 배터리부터 큰 박스까지)가 크고
  - 한 이미지에 다수 객체가 혼재된 cluttered scene이 많기 때문에

    안정적인 two-stage 계열이 합리적인 초기 선택이라고 판단했다.

## 1.2 관련 용어 정리

- two-stage detector
  - 1단계: RPN 등으로 물체 후보 박스(proposal)를 생성
  - 2단계: 해당 후보에 대해 클래스 및 박스를 정교하게 예측
  - 장점: 정밀도 및 박스 위치 정확도가 높음

    - 단점: 상대적으로 느리고 구조가 복잡함
- backbone (ResNet, Swin-L 등)
    - 원본 이미지를 feature map으로 변환하는 기본 CNN/Transformer 네트워크
    - 검출기의 표현력과 성능을 크게 좌우하는 핵심 요소
- Swin-L (Swin Transformer Large)
    - 윈도우 단위 self-attention을 사용하는 비전 Transformer
    - 높은 해상도를 효율적으로 처리하며,

      작은 객체 및 장거리 문맥 정보를 함께 고려할 수 있다는 장점이 있음
- Cascade head
    - IoU threshold를 단계적으로 증가(예: 0.5 → 0.6 → 0.7)시키며

      여러 단계에 걸쳐 박스를 반복적으로 refine하는 구조
    - 초기 단계에서 대략 맞춘 박스를 후단에서 점진적으로 정교하게 보정

## 1.3 실험 결과

- Faster R-CNN: mAP ≈ 0.4177

  → 파이프라인이 정상 동작함을 확인하고, 최소 기준선 성능을 확보하는 역할
- Cascade R-CNN (Swin-L backbone): mAP ≈ 0.4642

  → backbone을 Swin-L로 교체하고 cascade head를 사용함으로써

  ResNet 기반 Faster R-CNN 대비 뚜렷한 성능 향상 확인
- rcnn_sin (증강/스케줄 튜닝): mAP ≈ 0.4760

  → 구조는 유사하게 유지하되, multi-scale, color jitter, scheduler 조정 등을 통해

  기존 Cascade 대비 추가적인 성능 향상 달성

## 1.4 요약

- 초기 단계에서 “검증된 two-stage + 안정적인 backbone” 조합으로 baseline을 구축함으로써,

  이후 실험들의 상대 비교 기준을 명확히 설정했다.
- Swin-L + Cascade 조합은 작은 객체에 대한 완벽한 대응은 아니었지만,

제한된 시간 대비 안정적으로 0.46~0.47대 mAP를 제공하는 신뢰 가능한 모델로 자리 잡았고,

이후 앙상블에서 핵심 축 중 하나로 활용 가능한 후보가 되었다.

---

# 2. VFNet 및 VFNet + Cascade 앙상블

## 2.1 VFNet 선정 배경

- 데이터 특성:
  - 클래스 불균형(일반/플라스틱/종이 비중이 높고 희소 클래스 존재)
  - 박스 위치 정확도와 classification을 동시에 고려할 필요
- VFNet은 IoU-aware Varifocal Loss를 사용하는 one-stage detector로,

  박스가 정확할수록 높은 score를 부여하고, 애매한 박스는 score를 낮추는 방향으로 학습된다.
- “박스 품질이 반영된 confidence”를 제공할 수 있다는 점에서

  특히 높은 IoU 구간의 mAP 개선 가능성을 기대하고 도입했다.

## 2.2 관련 용어 정리

- one-stage detector
  - 별도 proposal 단계 없이 feature map에서 클래스와 박스를 직접 예측
  - 예: RetinaNet, YOLO 계열, VFNet
  - 장점: 구조가 단순하고 추론 속도가 빠름
  - 단점: refinement 단계가 없어 튜닝 난이도가 높을 수 있음
- anchor-based vs anchor-free
  - anchor-based:
    - 사전에 정의된 다양한 비율·크기의 anchor box를 격자 단위로 배치한 후 anchor 기준으로 박스를 보정
    - 예: Faster R-CNN, RetinaNet
  - anchor-free:
    - 중심점, 거리 등 직접적인 geometric parameter를 예측하여 박스를 생성

        - 예: FCOS, CenterNet, 일부 TOOD 등
- Varifocal Loss
    - classification loss에 IoU 정보를 함께 반영해

      정확한 박스의 score를 높이고, 부정확한 박스의 score를 낮추도록 유도
    - 결과적으로 confidence가 “클래스 확률 × 박스 품질”에 가까운 의미를 갖도록 설계

### 2.3 실험 결과

- VFNet 단독: mAP ≈ 0.3751
- VFNet + Cascade 간 단순 앙상블: mAP ≈ 0.4401

  (Cascade 단독 0.4642보다 낮은 결과)

### 2.4 관찰 사항

- 단독 성능 저하 원인:
    - 클래스 불균형 대응(re-balancing), scale jitter, 긴 학습 스케줄, warmup 등 VFNet 특성을 살릴 수 있는 튜닝이 충분히 진행되지 못함.
- 앙상블 성능 저하 원인:
    - VFNet이 상대적으로 약한 모델임에도 Cascade와 동등 비중으로 결합하면서 강한 예측이 약한 예측에 의해 희석되는 효과 발생.
- 요약:
    - 구조가 다르다는 이유만으로 단순 앙상블을 적용할 경우,

      성능이 반드시 향상된다고 볼 수 없음을 확인했다.

---

## 3. Deformable DETR (ResNet-50)

### 3.1 선정 배경

- Transformer 기반 detector는
    - cluttered scene에서 global 문맥을 잘 포착하고
    - multi-scale feature를 자연스럽게 처리할 수 있다는 장점이 있다.
- 이번 데이터처럼 한 이미지에 다수 객체가 겹쳐 있는 상황에서

Transformer 계열이 실제로 어떤 성능을 보이는지 확인하기 위해 시도했다.

- Deformable DETR는
  - Deformable Attention(희소 multi-scale attention)을 도입하여

    vanilla DETR 대비 학습 속도를 향상시키고 작은 객체에 더 유리한 구조로 알려져 있어 선택했다.

## 3.2 관련 용어 정리

- DETR
  - object detection을 “set prediction” 문제로 정식화
  - 고정 개수의 query가 각 객체를 담당하도록 학습
  - anchor 및 NMS 의존도를 줄이는 구조
- Deformable Attention
  - 모든 위치에 대해 모든 위치를 보는 full attention 대신,

    소수의 중요한 sampling point에 대해서만 attention을 수행하는 방식
  - 여러 해상도 feature map을 동시에 참조하면서 연산량을 줄이고

    작은 객체에 대한 표현을 강화하는 것을 목표로 함

## 3.3 실험 결과 및 해석

- Deformable DETR (ResNet-50): mAP ≈ 0.252

  → two-stage 계열 대비 성능이 크게 낮게 측정됨.
- 원인:
  - 논문에서 제안하는 충분한 학습 스케줄보다 훨씬 짧은 epoch 및 기본 lr schedule 사용
  - augmentation, optimizer, lr decay 등 세부 설정을 거의 손대지 않고 기본 config 수준에 머무름
- 의미:
  - Transformer detector는 구조적 장점에도 불구하고

    짧은 스케줄과 최소 수정만으로는 two-stage보다 성능이 떨어질 수 있음을 확인했다.
  - 이를 계기로 Swin-L 기반 DETA를 4-fold로 본격 도입하고,

보다 체계적인 스케줄/구조 설계를 고려하는 방향으로 전환했다.

---

# 4. TOOD / Sparse R-CNN / RTMDet (실패 실험군)

## 4.1 공통 선정 배경

- two-stage 및 일부 Transformer 계열 외에도
  - one-stage (TOOD, RTMDet)
  - query-based (Sparse R-CNN)

    등 다양한 구조를 실제 데이터에 적용해 보고,

    이후 앙상블 시 구조적 다양성을 확보하는 것을 목표로 했다.
- 각 모델의 주요 관심 포인트:
  - TOOD: anchor-free + task-aligned head의 효과
  - Sparse R-CNN: learnable proposal 개념의 유효성
  - RTMDet: “실전형” 최신 one-stage 구조의 실제 성능 수준

## 4.2 관련 용어 정리

- task-aligned head (TOOD)
  - classification과 localization을 별개로 보지 않고,

    두 신호를 함께 고려해 task에 맞는 feature 위치를 학습하도록 설계된 head 구조
- learnable proposal (Sparse R-CNN)
  - RPN에서 proposal을 생성하는 대신,

    proposal 자체를 학습 대상 파라미터로 두고 반복 refine하는 방식
- RTMDet
  - 실시간/실전(real-time) 환경을 고려하여 설계된 one-stage detector
  - backbone, neck, head 모두 경량성과 성능의 균형을 목표로 함

## 4.3 실험 결과

- TOOD: mAP ≈ 0.2686
- Sparse R-CNN: mAP ≈ 0.2873

- RTMDet: mAP ≈ 0.0468 (사실상 학습 실패 수준)

### 4.4 관찰 사항

- 공통적인 한계:
  - 세 모델 모두 “기본 config + 짧은 학습 시간”에 가까운 설정으로 실험되었으며,

    각 구조 특성에 맞는 세부 튜닝이 거의 이루어지지 못했다.
- RTMDet의 경우:
  - COCO config에서 class 수, augmentation, lr schedule 등을 충분히 조정하지 못했으며,

    고정 해상도(1024), 클래스 불균형 등 데이터 특성을 반영한 커스터마이징이 부족했다.
- 요약:
  - 논문 상 SOTA 모델이라도,

    제안 세팅에 근접한 튜닝 없이 사용하면 baseline보다 낮은 성능이 나올 수 있음을 확인했다.
  - 이후 전략은 “검증된 two-stage + 일부 Transformer + 팀원이 튜닝한 DDQ/CODetr”를 중심 축으로 삼고,

    이에 기반한 앙상블 전략에 집중하는 방향으로 정리되었다.

---

## 5. DETA 4-Fold 선정 이유 및 한계

### 5.1 DETA 4-Fold 선정 배경

- 팀 내에서 Swin-L 기반 DDQ-DETR, CODetr가 이미 mAP 0.59~0.69 수준의 높은 성능을 기록하고 있었다.
- 비슷한 계열의 Swin-L 기반 Transformer detector(DETA)를 추가로 확보하면:
  - DDQ / CODetr / DETA를 조합한 강력한 Transformer ensemble 구성이 가능하고
  - 동일 backbone·유사 구조이더라도 head/학습 방식 차이로 인한 예측 패턴 다양성을 기대할 수 있었다.
- 4-fold 학습을 선택한 이유:

- 전체 데이터를 보다 효율적으로 사용하면서

  fold마다 약간씩 다른 데이터 분포를 학습시켜

  inference 단계에서 fold ensemble을 통해 generalization 향상을 도모하기 위함.

### 5.2 관련 용어 정리

- k-fold cross validation
  - 데이터를 k개로 분할하여,

    k-1개를 학습, 1개를 검증에 사용하는 과정을 k번 반복하는 방식
  - 본 프로젝트에서는 4-fold로 train json을 4개로 분할하여

    fold1~4 각각에 대해 별도 모델을 학습했다.
- fold ensemble
  - 동일 구조의 모델을 fold별로 학습한 뒤,

    추론 단계에서 각 fold 예측 결과를 합쳐 사용하는 방식
  - 일반적으로 overfitting 완화 및 성능 안정성을 기대할 수 있다.

### 5.3 결과적으로 드러난 한계

- DETA 4-fold의 학습 자체는 겉보기에는 정상적으로 진행되었으나,
  - num_classes 설정,
  - label mapping,
  - DetInferencer 버전 호환성,
  - WBF score 평균 계산 방식

    등이 동시에 얽히면서 리더보드 점수는 약 0.003 수준에 머물렀다.
- 정리하면:
  - DETA를 선택한 전략적 판단 자체는 합리적이었으나,
  - inference 및 ensemble 파이프라인을 충분히 검증하기 전에

    대규모 fold ensemble을 적용한 부분이 가장 큰 리스크로 작용했다.

---

## 2-3. 주요 구조별 의도와 결과

## Faster R-CNN / Cascade R-CNN

- 의도
  - 검증된 two-stage 구조를 baseline으로 사용해 파이프라인을 안정화
- 결과
  - Faster R-CNN: 0.4177
  - Cascade R-CNN (Swin-L): 0.4642
  - rcnn_sin (증강/스케줄 튜닝 버전): 0.4760
- 해석
  - Swin-L backbone + Cascade head 조합이
    기존 ResNet 기반 Faster R-CNN보다 mAP가 확실히 상승
  - 작은 객체에 대해서는 여전히 recall 부족이 있었지만,
    제한된 시간 대비 안정적인 성능을 보여줌

## VFNet 및 VFNet + Cascade 간 앙상블

- 의도
  - Varifocal Loss를 사용하는 one-stage 계열(VFNet)이
    IoU-aware classification으로 AP를 끌어올릴 수 있는지 확인
- 결과
  - VFNet 단독: 0.3751
  - VFNet + Cascade 간 간단 앙상블: 0.4401 (Cascade 단독보다 낮음)
- 원인 추정
  - data level re-balancing 없이 config 기본값만 사용
  - scale jitter, epoch, warmup 등을 깊게 튜닝하지 못해
    VFNet의 잠재력을 충분히 살리지 못함
- 교훈
  - 서로 다른 모델을 무조건 섞는다고 성능이 올라가지 않는다
  - 약한 모델이 강한 모델의 prediction을 오히려 “희석”시키는 케이스를 경험

## Deformable DETR (ResNet-50)

- 의도
  - Deformable Attention 기반 transformer detector가
    multi-scale·cluttered scene에서 어떤 성능을 내는지 확인
- 결과
  - 0.252로 two-stage 계열보다 많이 낮음
- 원인 추정
  - 기본 config 수준의 epoch / lr schedule만 사용
  - Deformable DETR 특유의 긴 학습 스케줄과 튜닝이 부족
- 교훈
  - transformer detector는 “구조 좋은 논문”만 믿고 짧게 돌리는 것보다
    처음부터 충분한 스케줄·튜닝 계획을 세워야 의미 있는 결과를 얻을 수 있다

### TOOD / Sparse R-CNN / RTMDet (실패 실험)

- TOOD (0.2686)
  - anchor-free + task-aligned head 구조
  - one-stage 계열에서 필수적인 scale jitter, longer schedule, warmup 조정이 부족
- Sparse R-CNN (0.2873)
  - learnable proposal 기반 query detector
  - 데이터량 대비 자유도가 커서 underfitting / 튜닝 난이도↑
- RTMDet (0.0468)
  - 사실상 학습 실패 수준
  - COCO config를 거의 그대로 사용했고,
    클래스 수·augmentation·lr 스케줄 모두 데이터 특성에 맞게 조정하지 못함

교훈

→ 새 구조를  단순히 가져오면

오히려 baseline보다 성능이 떨어질 수 있다는 것을 경험했다.

# 3. DETA 4-fold 실험과 실패 원인

## 3-1. 실험 목표

- 목표
  - Swin-L 기반 DETA를 4-fold로 학습해

    DDQ-DETR 계열과 비슷한 레벨의 성능을 확보하고,

    나중에 팀원 DDQ-DETR와도 앙상블 가능한 후보 모델로 쓰기

## 3-2. 진행 과정

- `make_4fold_coco.py` 로 train을 4개의 COCO json으로 분할
- `train_deta_4fold_v3.py` 로 fold별 학습
  - 각 fold에서 `best_coco_bbox_mAP_epoch_*.pth` 체크포인트 생성
- `infer_deta_4fold.py` / `infer_deta_4fold_ensemble_csv.py` 로

  fold별 추론 및 부분 앙상블 csv 생성 시도

이 과정에서 크게 두 종류의 문제가 발생했다.

## 3-3. 기술적 이슈와 버그

1. num_classes / label mapping 꼬임

- COCO 기반 DETA config는 기본 80 classes
- 대회는 10 classes
- 처음에는 `cfg_options`를 통해
  - `model.bbox_head.num_classes = 10`
  - `model.dn_query_generator.num_classes = 10`

    로 패치하려 했지만,
  - DetInferencer 버전에서 `cfg_options`를 지원하지 않음
- 해결을 위해
  - base config를 로드 후, num_classes를 직접 수정하고

    `temp .py` 파일로 dump하는 `_make_temp_cfg_for_ckpt` 함수 작성
  - 이 임시 config를 넣어 DetInferencer를 생성하는 방식으로 우회

그럼에도 결과 로그에서

`unique labels: [0, 1, 6, 7]`처럼 일부 클래스에만 예측이 몰리는 현상이 나왔고,

train.json의 category id 순서와 pretrained weight의 class index가
완전히 일치하지 않는 가능성이 남아 있었다.

1. score calibration / ensemble 파이프라인 문제

- fold별 예측에서는 class 7에 대해 0.7~0.8대 confidence가 나왔지만,
  fold 간 WBF/NMS를 수행하면서 score가 0.1 이하로 떨어지는 현상 발생
- 원인
  - WBF에서 score를 단순 평균으로 계산
  - 여러 fold가 비슷한 위치를 "애매한 score"로 찍으면
    최종 score가 과도하게 희석되는 구조
  - score threshold, IoU threshold, label mapping이 한꺼번에 얽혀 있어
    디버깅 난이도가 높았다.

**▼ 추가 설명 및 정리**

# DETA 4-Fold 실험 정리

## 1. 학습 단계 요약

- DETA 4-fold 학습 자체는 겉보기에는 정상적으로 진행됨.
- 각 fold별로 `best_coco_bbox_mAP_epoch_*.pth` 형태의 체크포인트가 정상 저장됨.
- 학습 로그 상에서 loss는 감소하고, mAP도 일정 수준까지는 상승하는 양상을 보임.

→ 학습만 놓고 보면 "완전히 망한 학습"이라고 보기는 어려운 상태였음.

---

## 2. 추론(inference) 및 앙상블 파이프라인 이슈

### 2.1 DetInferencer + cfg_options 문제

- DetInferencer 사용 시 `cfg_options` 인자로
  `model.bbox_head.num_classes=10`, `dn_query_generator.num_classes=10`을 전달하려 했으나,
  사용 중이던 버전이 해당 인자를 지원하지 않음.

- 결과적으로:
  - 에러가 발생하거나,
  - 설정이 무시되어 실제 추론 config에는 반영되지 않는 문제가 있었음.

## 2.2 config 직접 수정 및 temp config 생성

- 위 문제를 우회하기 위해 다음과 같은 절차를 사용함:
  1. base config 로드
  2. config 객체 내 `num_classes` 관련 설정을 10으로 직접 수정
  3. 수정된 config를 임시 `.py` 파일로 dump
  4. DetInferencer의 `config` 인자로 이 temp `.py`를 전달
- 이 과정을 수행하는 헬퍼가 `_make_temp_cfg_for_ckpt` 역할을 담당함.

## 2.3 추론 결과에서의 label 분포 이상

- 수정된 config로 추론 후 로그를 확인했을 때,

  예측 label 분포가 비정상적으로 나타남.
- 예시:
  - `unique labels: [0, 1, 6, 7]` 등, 일부 class id에만 예측이 집중됨.
- 대회 기준 class 개수는 10개인데, 실제 예측은 0, 1, 6, 7 등의 일부 클래스에만 거의 편중된 상태였음.
- 이 현상은:
  - 실제 데이터 분포 문제일 수도 있으나,
  - pretrained weight와 새로운 category 순서/메타 정보가 어긋난 label mapping 문제 가능성이 큼.

## 2.4 앙상블(WBF) 이후 score 급감과 박스 소실

- 각 fold 모델 개별 추론에서는 일부 박스가 score 0.7~0.8 수준으로 비교적 높은 confidence로 예측됨.
- 그러나 fold 간 WBF(Weighted Box Fusion) 앙상블을 수행하는 과정에서:
  - score를 단순 평균하는 구현을 사용하면서,
  - 일부 fold에서 낮은 score(0.1~0.3) 또는 위치가 어긋난 박스가 섞여 들어감.

- 그 결과:
  - 하나의 객체에 대해 “강한 예측 + 약한 예측 + 위치 어긋난 예측”이 한 클러스터로 합쳐지면서

    최종 score가 0.1대 수준으로 떨어지는 케이스가 빈번하게 발생.
- 이후 파이프라인에서:
  - score threshold 적용,
  - NMS,
  - label mapping 문제까지 중첩되면서

    상당수 박스가 제거된 상태로 최종 csv가 생성됨.

---

# 3. 리더보드 결과 및 해석

- 최종 리더보드 점수:
  - DETA: 0.0031
  - DETA_nms: 0.0025
- 위 점수는:
  - 학습이 전혀 되지 않았다고 보기보다는,
  - 추론 및 앙상블 파이프라인에서 박스가 거의 모두 소실된 결과에 가깝다고 해석하는 것이 타당함.
- 정리하면:
  - 학습 자체도 최적은 아니었을 수 있으나,
  - “0.00대에 가까운 점수”는 거의 전적으로 inference + ensemble 파이프라인 설계 및 구현 문제로 보는 것이 합리적인 수준임.

---

# 4. 주요 용어 정리

## 4.1 num_classes

- 의미: 모델이 구분해야 하는 클래스 개수 설정 값.
- 예시:
  - COCO: 80개 클래스 → `num_classes = 80`

    - 본 대회: 쓰레기 10개 클래스 → `num_classes = 10`
- 중요성:
    - 마지막 classification 레이어 출력 차원이 `num_classes`에 의해 결정됨

      (예: cls logit shape = `[num_queries, num_classes]`).
    - 이 값이 실제 데이터의 클래스 개수와 다르면:
        - 잘못된 class index로 예측하거나,
        - loss 계산 시 엉뚱한 차원을 학습하는 문제가 발생함.

## 4.2 label mapping

- 의미: "레이블 번호 → 실제 클래스 의미"를 연결하는 규칙.
    - 예: `0 → 일반쓰레기`, `1 → 플라스틱`, `2 → 종이` ...
- pretrained 모델(COCO)은:
    - `0 → person`, `1 → bicycle`, `2 → car` ... 와 같은 mapping을 기본으로 가짐.
- 문제점:
    - 단순히 `num_classes`만 10으로 바꾸고,
    - category id / metainfo(클래스 이름, 순서)를 제대로 맞추지 않으면,

      "5번 클래스" 등 특정 index가 모델 입장에서는 전혀 다른 의미로 해석될 수 있음.
- DETA 실험에서:
    - `unique labels: [0,1,6,7]`만 주로 나오는 현상은

      label mapping 및 metainfo 정합성 문제 가능성을 강하게 시사함.

## 4.3 cfg_options

- MMDetection에서 config 일부를 런타임에 override하기 위한 옵션.
- 일반적인 사용 예:
    - `DetInferencer(config='xxx.py', cfg_options={'model.bbox_head.num_classes': 10})`
- 역할:
    - 원본 config 파일을 직접 수정하지 않고도, 특정 key 값만 덮어쓰기 위해 사용.
- 이번 실험에서는:

- 사용한 DetInferencer 버전이 `cfg_options`를 지원하지 않아
  인자를 줘도 무시되거나 에러가 발생하는 문제가 있었음.

## 4.4 DetInferencer

- MMDetection에서 제공하는 high-level 추론 도구.
- 입력: config, checkpoint, 이미지 경로 등.
- 내부에서:
  - 데이터 로딩
  - 전처리
  - 모델 forward
  - 후처리(NMS 등)
  - 시각화/저장
    을 한 번에 수행.
- 장점:
  - 별도의 추론 스크립트를 작성하지 않아도 쉽게 결과를 확인 가능.
- 단점(본 케이스 기준):
  - 내부 로직이 추상화되어 있어 `num_classes`, metainfo, pipeline을 세밀하게 제어하기 어려움.
  - 버전별 지원 인자, 출력 타입(DetDataSample 등)이 달라 디버깅 난이도가 높음.

## 4.5 base config / temp .py dump

- base config:
  - DETA 학습 시 사용한 원본 설정 파일 (예: `deta_swinl_1024.py` 등).
- temp `.py` dump:
  - 파이썬 코드에서 base config 객체를 수정한 뒤,
  - 수정된 config를 `.py` 파일로 저장한 임시 설정 파일.
- 사용 이유:
  - DetInferencer가 `cfg_options` override를 지원하지 않기 때문에,

"이미 수정된 config"를 새로운 .py 파일로 만들어 직접 넘기는 우회 전략으로 사용.

## 4.6 score, calibration, score threshold

- score (confidence):
  - "이 박스가 특정 클래스일 것이라는 모델의 확신 정도"를 0.0~1.0 사이 값으로 표현한 것.
- calibration:
  - score 값이 실제 정답 확률과 얼마나 잘 대응하는지의 정도.
  - 예: score 0.8인 박스가 실제로 약 80% 정도 맞는지 여부.
- score threshold:
  - 일정 값 이하의 박스를 제거하는 컷라인.
  - 예: `score_thr=0.03`일 때, score 0.03 미만 박스는 모두 제거.

## 4.7 IoU, NMS, WBF

- IoU (Intersection over Union):
  - 두 박스의 겹치는 정도를 0~1 사이 값으로 표현.
  - 0: 전혀 안 겹침, 1: 완전히 동일.
- NMS (Non-Maximum Suppression):
  - 동일 객체를 여러 박스가 덮고 있을 때,
  - 가장 score가 높은 박스를 남기고,

    IoU가 threshold 이상인 나머지 박스를 제거하는 알고리즘.
- WBF (Weighted Box Fusion):
  - 여러 박스를 버리지 않고,

    좌표는 score 기반 가중 평균, score는 정의한 방식(예: 단순 평균)으로 합쳐 하나의 박스로 만드는 방법.
  - 예: A(score 0.9), B(score 0.8)이 충분히 겹치는 경우

    좌표는 두 박스 좌표의 가중 평균, score는 (0.9+0.8)/2 = 0.85 등의 방식.

## 4.8 fold ensemble (k-fold ensemble)

- k-fold cross validation에서 각 fold별로 학습된 모델들을

추론 시 다시 합쳐 사용하는 방식.

- 본 실험:
  - 4-fold 구조 사용.
  - 동일 이미지에 대하여 fold1~fold4 모델이 각각 detection 수행.
  - 이후 이 detection 결과들을 WBF/NMS로 합치는 방식으로 앙상블 시도.

---

# 5. 당시 문제 구조 요약

## 5.1 label / num_classes 관련 흐름

1. 학습 단계

- config 상 `num_classes=10`으로 맞추고,

  DETA를 4-fold로 학습했다고 가정.

1. 추론 초기 버전

- DetInferencer에 "원본 config(80 클래스 기준)" + checkpoint를 그대로 전달.
- `cfg_options`로 `num_classes=10`을 덮어쓰려 했지만,

  버전 이슈로 실제로는 80 클래스 그대로 추론되었을 가능성이 있음.

1. temp config 버전

- 위 문제를 해결하기 위해:
  - config 객체를 직접 수정,
  - `.py`로 dump 후 DetInferencer에 전달.
- 이 과정에서:
  - metainfo(category names/ids),
  - pretrained head weight alignment

    중 하나라도 어긋나면,

    예측이 일부 클래스에만 집중되는 현상이 발생할 수 있음.

1. 결과 증상

- 로그 상 `unique labels: [0, 1, 6, 7]`처럼

  10개 클래스 중 일부에만 예측이 몰리는 현상 관측.

- 이는 label mapping, category 순서, num_classes 정합성 문제 가능성을 강하게 시사.

## 5.2 score 0.7 → 0.1로 떨어진 과정

1. fold별 예측 가정 예시

- 어떤 객체(예: 병)에 대해:
  - fold1: class=7, score=0.80, 박스 A
  - fold2: class=7, score=0.75, 박스 B
  - fold3: class=7, score=0.78, 박스 C
  - fold4: detection 없음

1. 이상적인 WBF 동작

- A/B/C의 IoU가 충분히 크면 같은 cluster로 묶이고,
  - 좌표: A/B/C 가중 평균
  - score: (0.80+0.75+0.78)/3 ≈ 0.776 수준 유지

1. 실제 문제 상황

- 일부 fold에서 동일 객체에 대해 score 0.1~0.3의 약한 박스가 섞여 있었고,
- 어떤 fold는 bounding box 위치가 어긋나 IoU 기준에서 다른 cluster로 분리되기도 함.
- 구현 상:
  - 여러 클래스를 잘못 섞어서 cluster를 구성했을 가능성,
  - score 계산 로직의 설정 문제 가능성도 존재.

1. 최종 영향

- "강한 예측 + 약한 예측 + 위치 어긋난 예측"이 섞인 cluster에서

  최종 score가 0.1~0.2대로 내려감.
- 이후:
  - score threshold,
  - NMS,
  - label mapping 문제 등이 연속적으로 적용되면서

    많은 박스가 제거된 상태로 submission이 생성됨.

- 이로 인해 리더보드 상 점수가 0.00대에 가까운 수준으로 기록됨.

## 3-4. 리더보드 결과와 평가

- DETA: 0.0031
- DETA_nms: 0.0025

실질적으로 “모델 자체의 학습 성능”을 확인하기도 전에

inference 파이프라인과 label mapping bug 때문에 거의 모든 bbox가

잘못된 class로 기록되거나, threshold에서 사라져버린 실패 실험이다.

교훈

- 새로운 transformer detector를 도입할 때는
  1. train loop가 돌아가는 것
  2. 단일 이미지에서 추론 결과를 사람이 직접 eye-check하는 것
  3. 그 결과로 submission csv 한 줄을 정확히 만들어 보는 것

     이 세 단계가 모두 검증된 후에야 대량 inference·앙상블로 넘어가야 한다.
- fold ensemble을 모델 내부에서 직접 구현하려 하면

  복잡도가 급격히 올라가고 bug risk가 커진다는 것을 체감했다.

---

# 4. CSV 기반 앙상블 전략

## 4-1. 설계 목표

DETA 쪽에서 model-level ensemble이 꼬이면서,

전략을 완전히 바꿔서

- “각 모델이 만든 submission csv만 신뢰한다”
- “csv의 PredictionString을 파싱해서 박스 레벨에서 앙상블한다”

라는 방향으로 전환했다.

이를 위해 `ensemble_from_submissions.py`를 직접 구현했다.

## 4-2. ensemble_from_submissions.py 핵심 구조

1. PredictionString 파싱

- `"cls score x y w h ..."` →

  `[(cls, score, x1, y1, x2, y2), ...]` 리스트로 변환
- 좌표 변환
  - x2 = x + w, y2 = y + h
  - 최종 저장 시 다시 w/h로 되돌림

1. 이미지별, 클래스별로 박스 모으기

- csv 여러 개에 대해
  - 같은 image_id, 같은 class의 박스들을 모아서

    모델별 boxes / scores 리스트 생성
- 모델별 weight 옵션
  - score에 `weight`를 곱해 모델 신뢰도 반영

1. WBF 또는 NMS

- WBF 모드 (`-use-wbf`)
  - IoU threshold 이상인 박스들을 하나의 cluster로 묶고
  - cluster 내 bbox 좌표는 score-weighted average
  - 최종 score는 처음엔 평균, 이후 실험에서 최대값으로 변경
- NMS 모드
  - 모든 박스를 한 리스트로 모은 뒤

    class-wise greedy NMS 수행
  - 위치는 가장 score 높은 박스가 대표가 됨

1. score_thr, iou_thr

- iou_thr: 0.5
- score_thr: 0.03
- score_thr는 precision/recall trade-off에 큰 영향을 주기 때문에

  여러 번 값 바꿔가며 실험했다.

# 4-3. 앙상블 구성과 점수 변화

## 초기 앙상블 (ensemble, vfnet_cascade_ensemble 등)

- 구성
  - 내가 직접 학습한 모델들(Cascade, VFNet, Deformable DETR 등)을 단순히 PredictionString 기준으로 합치거나 NMS를 적용
- 결과
  - ensemble: 0.3918
  - vfnet_cascade_ensemble: 0.4401
- 평가
  - 단일 Cascade(0.4642), rcnn_sin(0.4760)보다 낮았기 때문에 “내 모델들만 섞어서 얻을 수 있는 이득은 제한적”이라는 결론

## ensemble_2까지 (DDQ 2종 + Cascade 조합, WBF 중심)

- 사용 csv (3개)
  - ddq_swinl_10_full 또는 ddq_swinl_1024_18e_full (약 0.6185)
  - ddq_swinl_1024_12e (0.5986)
  - cascade_rcnn_1024_12e (0.5320)
- 전략
  - WBF 사용
  - fused score = cluster 내 score 평균
  - score_thr = 0.03, iou_thr = 0.5
- 결과 (대표)
  - ensemble_2: 0.4978
- 원인
  - DDQ 두 모델은 매우 강력한데, 상대적으로 약한 Cascade를 같이 섞으면서 noisy box가 증가
  - score를 평균으로 내는 WBF 방식 때문에 원래 0.7~0.9대 박스들이 0.2~0.3대로 희석
  - 결과적으로 잘 잡힌 박스도 점수가 낮아져 precision이 떨어짐

## ensemble_4, ensemble_5 (DDQ 2종 + CODetr 조합, 최종 전략)

- 사용 csv (3개)
  - ddq_swinl_10_full / 1024_18e_full (0.6185)
  - ddq_swinl_1024_12e (0.5986)
  - codetr_swinl_1024_7e (0.6910) ← Cascade 대신 교체
- 변경점 1: 모델 조합
  - Cascade 대신 CODetr Swin-L을 포함
  - 세 모델 모두 Swin-L backbone + transformer 계열로
    구조적으로 비슷하지만 학습 세팅이 달라 상호보완 기대
- 변경점 2: WBF 점수 계산 수정
  - 평균 → 최대값
    - `fs = sum(scores)/len(scores)` → `fs = max(scores)`
- ensemble_4
  - WBF + 수정된 max score 방식
  - 결과: 0.5035
  - 이전보다 소폭 개선이 있었지만 single CODetr 0.6910에는 한참 못 미침
- ensemble_5 (최종 제출)
  - WBF 사용하지 않고 NMS-only로 전환
  - weights
    - ddq_swinl_10_full(또는 1024_18e_full): 0.85
    - ddq_swinl_1024_12e: 0.8
    - codetr_swinl_1024_7e: 1.0
  - score_thr = 0.03, iou_thr = 0.5
  - 결과: 0.6138

해석

- WBF를 제거하고 NMS만 사용했을 때 오히려 점수가 크게 올랐다.
  - 세 모델 모두 bbox 위치 자체는 꽤 정확하다고 보고,
    단일 모델의 “가장 자신 있는 박스”를 살려주는 방향이
    이 데이터에서는 더 유리했던 것 같다.

- CODetr 단일(0.6910)에는 못 미쳤지만,

  DDQ 계열 두 모델의 장점도 일부 가져오면서

  어느 정도 보수적인 선택을 한 셈이다.

---

# 5. 기술적 인사이트와 실패 요약

## 5-1. 모델 선택에 대한 인사이트

- 제한된 시간에서
  - 처음부터 DETR/DETA 같은 transformer 계열에 올인하는 것보다는

    Faster/Cascade R-CNN으로 robust한 baseline을 빠르게 확보하는 것이 효율적이다.
- 그 위에
  - 데이터 특성(작은 객체, dense scene, class imbalance 등)을 고려해

    VFNet, TOOD, Sparse R-CNN 같은 대안을 "한 개씩" 추가하는 전략이 현실적이다.

## 5-2. mmdet / mmengine 환경 이슈

- 같은 모델이라도 버전마다
  - DetInferencer 인자 지원 여부
  - 출력 포맷(DetDataSample vs dict)
  - config key 경로가 다르다.
- 이번 프로젝트에서
  - 임시 config 파일을 dump해서 num_classes를 직접 패치하는 방식이

    `cfg_options`보다 안정적인 해결책이라는 것을 경험했다.

## 5-3. 실패 사례 요약

1. RTMDet, TOOD, Sparse R-CNN

   → config·스케줄을 데이터에 맞게 조정하지 않은 "빠른 시도"의 한계
2. DETA / DETA_nms

→ num_classes, label mapping, inference 파이프라인 복합 버그로 실질적 실패

3. 초기 CSV 앙상블 (ensemble_2, ensemble_4 이전)

   → WBF score 평균 때문에 강한 박스도 score가 눌려서 precision 하락

교훈

- detection에서 앙상블은

  "어떤 모델을 섞느냐"만큼 "어떻게 score를 계산하느냐"가 중요하다.

- 특히 WBF 사용 시 score 스케일이 변하는 걸 항상 체크해야 한다.

---

# 6. 개인 회고

## 6-1. 경험

1. 다양한 구조를 실제로 돌려보며 감각을 쌓음
   - Cascade R-CNN, VFNet, Deformable DETR, TOOD, Sparse R-CNN, RTMDet 등

     논문에서만 보던 구조들을 직접 대회 데이터에 적용해 보고,

     각 구조가 어떤 상황에서 강/약점을 가지는지 경험적으로 이해했다.
2. DETA 실패를 통해 inference 파이프라인을 깊게 이해
   - 단순히 "점수 망했다"에서 끝내지 않고

     label 분포, unique labels, num_classes, metainfo, score 분포를 하나씩 확인했다.
   - 이 과정에서 mmdet config 구조와 DetInferencer 동작을 꽤 깊게 파고들 수 있었고,

     다음에 비슷한 문제를 겪을 때 빠르게 해결할 수 있겠다라는 생각이 들었다.
3. submission-level 앙상블 유틸 구현
   - 프레임워크와 무관하게 csv만 있으면

     NMS/WBF 앙상블을 시도할 수 있는 `ensemble_from_submissions.py` 를 만들었다.
   - 최종 결과인 ensemble_5(0.6138)는 이 파이프라인 위에서

     ddq_swinl 계열 2종과 codetr_swinl 모델을 조합해서 얻어낸 결과다.
   - 이 코드는 이후 다른 detection 대회에서도 재사용 가능한 도구가 된다.

## 6-2. 아쉬웠던 점과 한계

1. 모델 포트폴리오 설계가 사전에 정리되지 않음
    - “이 모델도 재밌어 보이니까 한 번” 식으로 실험이 늘어나다 보니

      RTMDet, Sparse R-CNN 등 시간 대비 효율이 낮은 시도가 꽤 많았다.
    - anchor 계열, transformer 계열, one-stage 계열을

      역할 나눠서 계획적으로 배치했으면 더 탄탄한 조합을 만들 수 있었을 것 같다.
2. validation / 에러 분석 부족
    - public LB 위주로 판단하다 보니

      per-class mAP, 사이즈별 AP, 어려운 이미지 집합에 대한 분석은 거의 못 했다.
    - 어떤 클래스에서 ensemble이 이득이고 손해인지까지는 도달하지 못한 게 아쉽다.
3. ensemble 전략의 보수적인 선택과 탐색 부족
    - CODetr 단일 0.6910이 워낙 강해서,

      이걸 기반으로 더 과감한 조합을 여러 개 실험해 볼 여지가 있었지만

      시간 부족과 private risk 부담 때문에 탐색 폭이 제한적이었다.

## 6-3. 다음 프로젝트에서의 개선 계획

1. 초반 설계 문서화
    - “anchor 기반 2개 + transformer 기반 2개 + one-stage 1개”처럼

      기본 포트폴리오를 먼저 정리하고,

      각 모델의 가설(예: 작은 객체 강화, class imbalance 개선)을 명시한 뒤 실험 시작하기.
2. DETR/DETA 계열 사용 시 체크리스트화
    - class id mapping, pretrained dataset, num_classes, metainfo를

      도식으로 먼저 그려 놓고,
    - 단일 이미지 inference → submission 한 줄 검증 → 전체 추론 순으로

      단계별 검증 절차를 템플릿으로 만들어 둘 계획이다.
3. 에러 분석·앙상블 자동화
    - per-class AP, size-wise AP, score calibration 리포트를

      자동으로 뽑는 스크립트를 따로 만들고,

- `ensemble_from_submissions.py`를 확장해서
  - 모델별 temperature scaling
  - class별 weight
  - 다양한 조합 탐색을 스크립트 한 번으로 돌릴 수 있게 만드는 것이 목표다.

---