

4팀 최종 결과보고서

팀장 이승철

팀원 손성경

최지혁

정예진

멘토 김상현

Contents

01. 주제

02. 협업 도구

03. EDA & 문제점 확인

04. 데이터 전처리

05. SSD/Faster-RCNN/YOLOv8/YOLOv10
초기 성능 측정

06. 클래스 불균형 해소

07. 클래스 불균형 해소 후 YOLO 모델 성능 측정

08. 학습 및 평가 파이프라인 구성 및 자동화

01. 주제

경구약제 이미지 인식 기능을 이용한 유저 헬스케어 데이터 수집 서비스 구축



- 데이터 탐색, 전처리 및 가공하는 과정을 통해 경구약제 이미지 데이터를 종합적으로 이해
- 문제를 해결하기 위해 객체 인식 모델을 실험하고 성능을 개선하기 위한 다양한 딥러닝 이론(이미지 전처리, 이미지 증강 등)을 직접 적용
- PyTorch를 사용해 모델을 설계, 훈련 및 검증 진행
- 캐글에 결과물을 제출하고 테스트 데이터에 대한 모델 성능을 확인
- 시각화 도구(Matplotlib, opencv 등)를 활용해 데이터 시각화 및 의미 전달

02. 협업 도구

1. GitHub

- 사용 목적

- 프로젝트 소스코드 버전 관리
- 팀원 간 코드 변경 사항 추적 및 공유
- 실시간 코드 리뷰 및 오류 수정

- 사용 방식

- 팀장이 GitHub Repository를 생성 후 나머지 팀원들을 Collaborator로 초대
- 각자 로컬 환경에서 브랜치(branch)를 만들어 작업 후 Pull Request(PR)로 병합

- 협업 효과

- 코드 충돌 최소화 및 수정 이력 투명화
- 모델 및 데이터 처리 파이프라인을 모든 팀원이 동일하게 유지
- 버전별 변경사항을 손쉽게 추적 가능

2. Google Drive

- 사용 목적

- 대용량 데이터셋 및 결과 파일 공유
- 모델 출력 결과, 로그 파일 등 문서화된 결과 보관

- 사용 방식

- 팀원 한 명이 구글 드라이브 폴더를 생성 후 팀원 전체에게 공유 권한 부여
- 학습용 데이터, 테스트 이미지 및 결과 파일 등을 업로드
- 팀원들은 구글 드라이브를 통해 동일한 파일 구조를 유지하면서 접근

- 협업 효과

- 로컬 저장소 용량 문제 해결
- 파일 버전 관리 및 팀 전체의 데이터 일관성 유지
- 발표 자료 및 실험 결과를 실시간으로 공유 가능

03. EDA & 문제점 확인

폴더 구조 확인

```
Train_annotations/
├── K-001900-016548-018110-029345/      # ► 이미지 이름 폴더
│   ├── 016548/                          # ► 알약 ID 폴더
│   │   ├── bbox_0001.json                # ► bbox별 JSON 파일 (각기 다른 바운딩 박스)
│   │   ├── bbox_0002.json
│   │   └── bbox_0003.json
│   └── 018110/
│       ├── bbox_0001.json
│       └── bbox_0002.json
└── K-002000-014555-017780-021111/
    ├── 014555/
    │   ├── bbox_0001.json
    │   └── bbox_0002.json
    └── 017780/
        ├── bbox_0001.json
        └── bbox_0002.json
...
... (다른 이미지 폴더들)
```

- 주어진 annotations 폴더는 이미지마다 있는 객체 하나마다 annotation 정보(바운딩 박스, 클래스) 가 담긴 json 파일이 하나씩 있었음
- json 파일에 annotation 정보 외에 모델 학습에 필요 없는 정보가 많이 포함되어 있었음

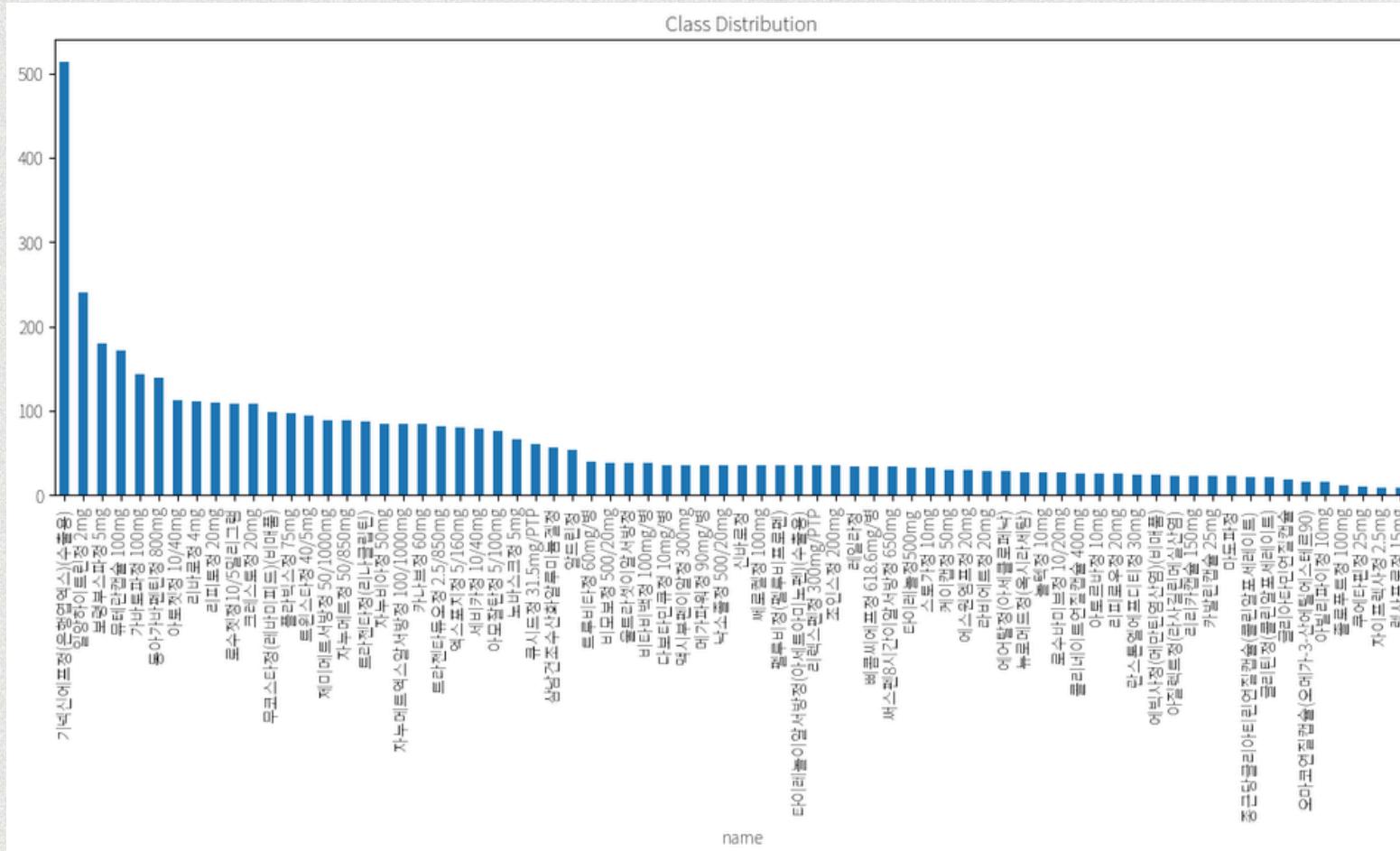
이미지 크기 분포 확인

이미지 크기 분포			
width	height	count	
0	976	1280	1489

- 데이터 일관성 확인
 - 모든 이미지의 해상도가 동일한지 확인(동일함)
- 모델 효율성 확보
 - 해상도가 크면 학습 속도나 메모리 영향을 주기 때문에 적정 크기 정하기 위해 분석

03. EDA & 문제점 확인

클래스 분포 확인



총 알약 클래스 수 확인

클래스별 Annotation 개수

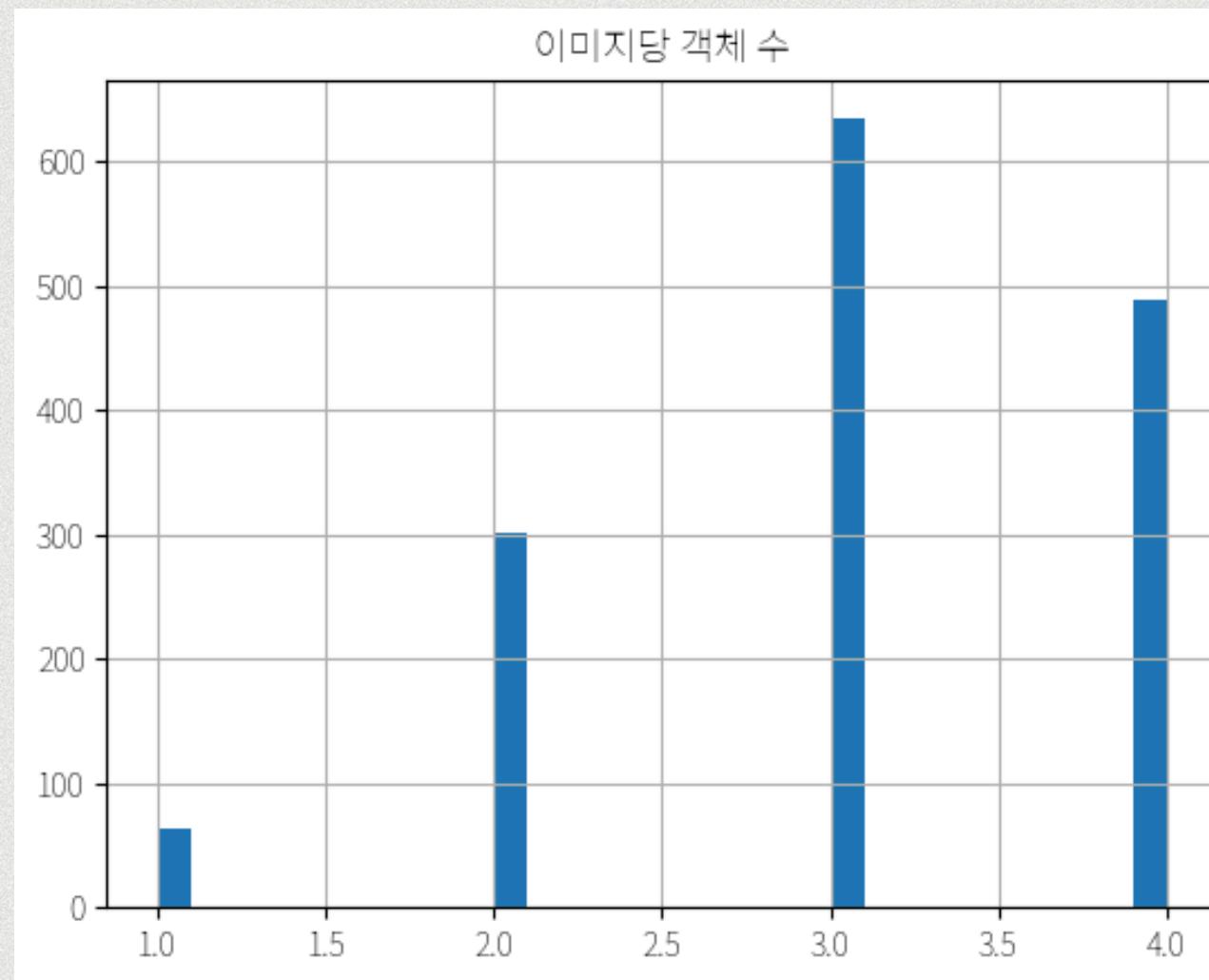
name	count
기넥신에프정(은행엽엑스)(수출용)	514
일양하이트린정 2mg	240
보령부스파정 5mg	180
뮤테란캡슐 100mg	172
가바토파정 100mg	143
...	
졸로푸트정 100mg	11
쿠에타핀정 25mg	10
자이프렉사정 2.5mg	9
렉사프로정 15mg	9
브린텔릭스정 20mg	7

Name: count, Length: 73, dtype: int64

- 데이터 불균형이 심해, 이를 해결하기 위해 특정 데이터를 증강하거나, 비워진 라벨을 채워 최대한 데이터 불균형을 해소해야함.
- 알약의 클래스 종류는 73가지

03. EDA & 문제점 확인

이미지당 객체 수 시각화



annotation 정보 시각화



- annotation 파일 정보를 기반해서 이미지당 객체 수를 파악
- 실제 이미지는 알약이 3개 or 4개인데 1개, 2개인 경우도 있음

- 주어진 annotation 정보를 시각화해보니 빈 부분 & 에러가 있는 것을 발견함

04. 데이터 전처리

```
IMG_DIR    = rf"{ROOT}\train_images"
ANN_ROOT   = rf"{ROOT}\train_annotations" # <- 여기를 루트로 재귀 탐색
OUT_JSON   = rf"{ROOT}\train_annotations_merged.json"

> def load_json(p): ...
> def ensure_image_size(file_name, img_dir): ...

# 머지 컨테이너
merged = {"images": [], "annotations": [], "categories": []}

# 카테고리는 "이름" 기준으로 통합(이름이 없으면 id를 키로)
cat_name_to_newid = OrderedDict()
next_cat_id = 1

# image / ann id는 전역 카운터
next_img_id = 1
next_ann_id = 1

# (file_name basename, w, h) -> new image_id
seen_images = {}

# 재귀적으로 모든 .json 수집
json_list = glob.glob(os.path.join(ANN_ROOT, "**", "*.json"), recursive=True)
json_list.sort()
print(f"[INFO] Found json files: {len(json_list)}")
if len(json_list) == 0:
    print("[ERROR] No JSON found under ANN_ROOT. 경로를 확인해주세요.")
    sys.exit(1)

# 본격 머지
> for idx, jp in enumerate(json_list, 1): ...

# 저장
> with open(OUT_JSON, "w", encoding="utf-8") as f: ...
print(f"[OK] Saved merged COCO: {OUT_JSON}")
print(f"  #images={len(merged['images'])}  #anns={len(merged['annotations'])}  #cats={len(merged['categories'])}")


```

알약 객체마다 있던 annotation 파일을 하나의 annotation 파일로 병합

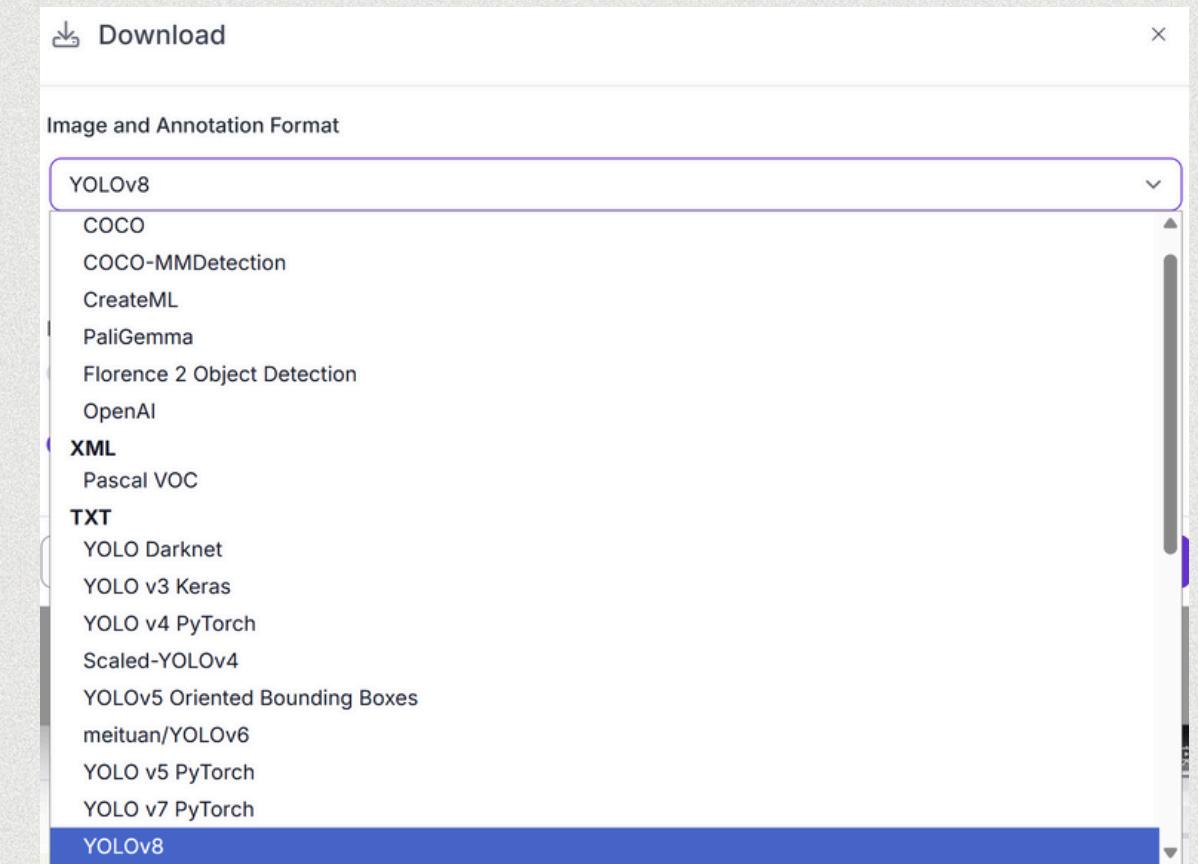
- 원본 annotation 데이터는 COCO 포맷의 json 파일이 객체마다 하나씩 존재
- 모델 학습과 데이터셋 편집 도구인 roboflow에서 사용하려면 하나의 json 파일로 병합 필요

→ 기존 객체마다 하나씩 있었던 annotation 파일을 COCO 포맷을 유지하며 하나의 annotation 파일로 병합해주는 코드 구현

04. 데이터 전처리

비어있는 라벨 채우기 & 잘못 배치된 라벨 고쳐 넣기

웹 기반 데이터셋 편집 도구인 Roboflow를 사용하여 기존에 가지고 있었던 데이터셋을 Roboflow에 업로드하고 이미지마다 바운딩 박스와 라벨이 없는 객체에 annotation 정보를 추가



COCO JSON 포맷의 데이터셋 - YOLO 포맷의 데이터셋 변환

COCO 포맷의 annotation을 사용하는 SSD, RCNN 모델과는 달리

YOLO는 독립적인 포맷을 사용

→ roboflow에서 모델 별 annotation 포맷 전환을 지원해 쉽게 변환할 수 있었음

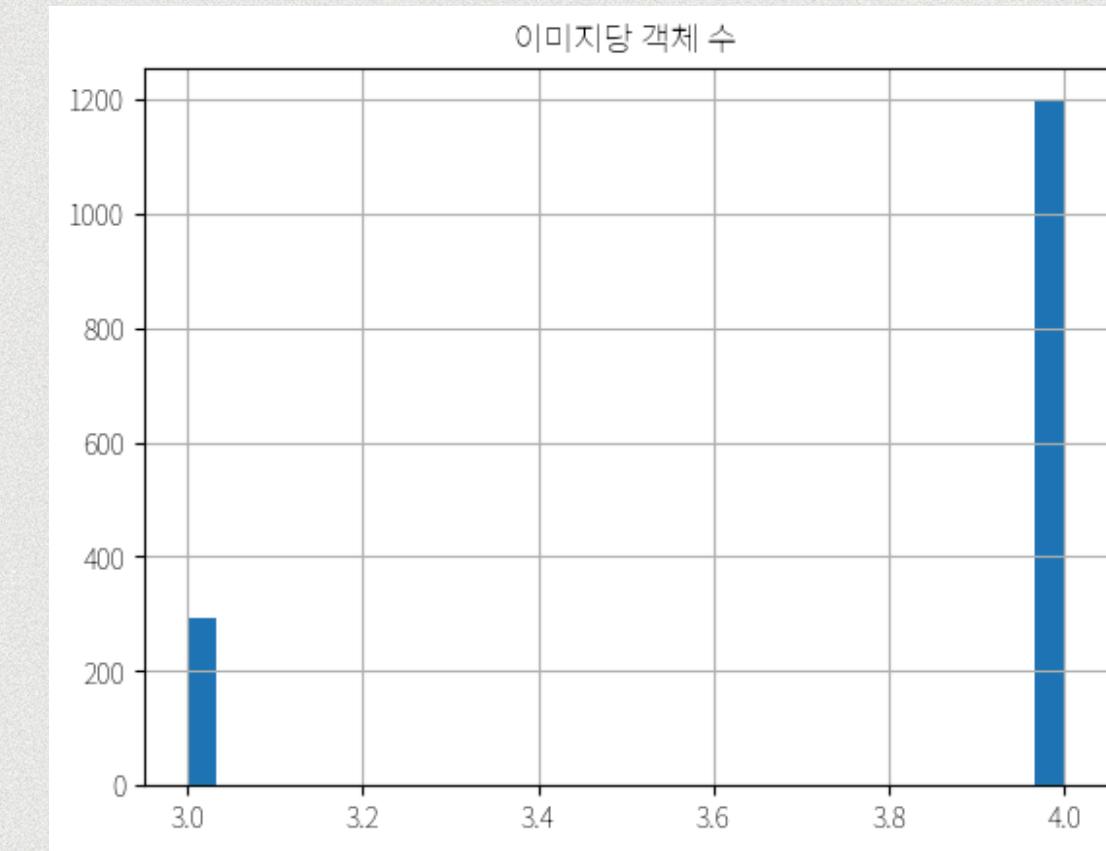
04. 데이터 전처리

데이터 전처리 후 클래스별 annotation 수 확인

클래스별 Annotation 개수	
name	
Ginexin-F Tab-	622
Hytrin Tab- 2mg Ilyang	294
Muteran Cap- 100mg	225
Buspar Tab- 5mg Boryung	222
Gabatopa Tab- 100mg	174
...	
Zoloft Tab- 100mg	12
Brintellix Tab- 20mg	12
Quetapin Tab- 25mg	12
Lexapro Tab- 15mg	12
Nexium Tab- 40mg	6
Name: count, Length: 74, dtype: int64	

- 모델의 일반화 성능 확보를 위해 분포 조정함
- 일부 클래스의 데이터가 과도하게 적거나 많을 경우, 모델이 다수 클래스만 학습되는 경우가 있음
- 데이터 불균형이 완전히 해소되진 않았지만, 극단적인 편향이 일부 완화

데이터 전처리 후 이미지당 객체 수 분포 확인



- 대부분의 이미지가 4개 객체를 포함하고 있음
- 객체 수 분포가 균일해 학습 시 배치 간 난이도 차이가 줄어듦

05. 사용한 모델

Faster RCNN

- Two-Stage detector
- Region Proposal Network (RPN)로 bbox 설정
- CNN이 bbox를 분류하고, 위치를 정교하게 보정
- 복잡한 장면, 중첩된 객체에서도 성능 우수

SSD

- Single Shot detector
- 후보영역을 제안하지 않고 CNN 한 번으로 탐지 & 분류
- 훨씬 빠름 (실시간 가능)
- 단순한 구조, 다양한 해상도 대응 가능

YOLOv8

- One-Stage detector
- 헤드에서 Classification과 Regression을 분리
- NMS 대신 Task-aligned Assigners로 개선된 훈련
- 학습과 배포가 쉬움 (ultralytics 라이브러리)

YOLOv10

- Non-Max Suppression(NMS) 제거 → “End-to-End” 구조
- 속도는 유지하면서 정확도 상승
- COCO 벤치마크 기준, YOLOv8보다 +1~2% mAP 향상
- 파라미터 수 감소 + 추론 속도 향상

05. Faster RCNN

카테고리 맵핑 문제와 해결과정

문제상황

- 학습 어노테이션과 카테고리 정보가 불일치함
- 모델 출력과 실제 카테고리가 서로 다른 값으로 매핑됨
- 시각화나 submission 단계에서 클래스 이름/아이디 불일치, KeyError 등 오류 발생

해결방법

- 카테고리 정보 어노테이션 기준으로 {label_id → category_id} dictionary 생성
- 테스트 시 model output을 원래 category_id로 변환
- 시각화 시 올바른 카테고리명 표시 가능

05. Faster RCNN Error 해결

클래스 수 불일치로 인한 모델 오류

문제

- 모델의 출력층(`cls_score`, `bbox_pred`)이 프로젝트의 데이터셋 클래스수와 맞지 않아 size mismatch 발생

해결

출력층 수정을 통한 클래스 수 및 바운딩 박스 수 맞춤

```
model = fasterrcnn_resnet50_fpn(  
    weights=wt,  
    trainable_backbone_layers=trainable_backbone_layers  
)  
model.roi_heads.box_predictor.cls_score = nn.Linear(in_features=1024, out_features=num_classes)  
model.roi_heads.box_predictor.bbox_pred = nn.Linear(in_features=1024, out_features=num_classes*4)  
return model
```

프로젝트 데이터셋에 맞춰 클래스 수와 바운딩 박스 회귀 출력을 재정의

- `cls_score`: 분류기 → `nn.Linear(1024, num_classes)`
- `bbox_pred`: 박스 회귀기 → `nn.Linear(1024, num_classes * 4)`

05. Faster RCNN Error 해결 후 결과

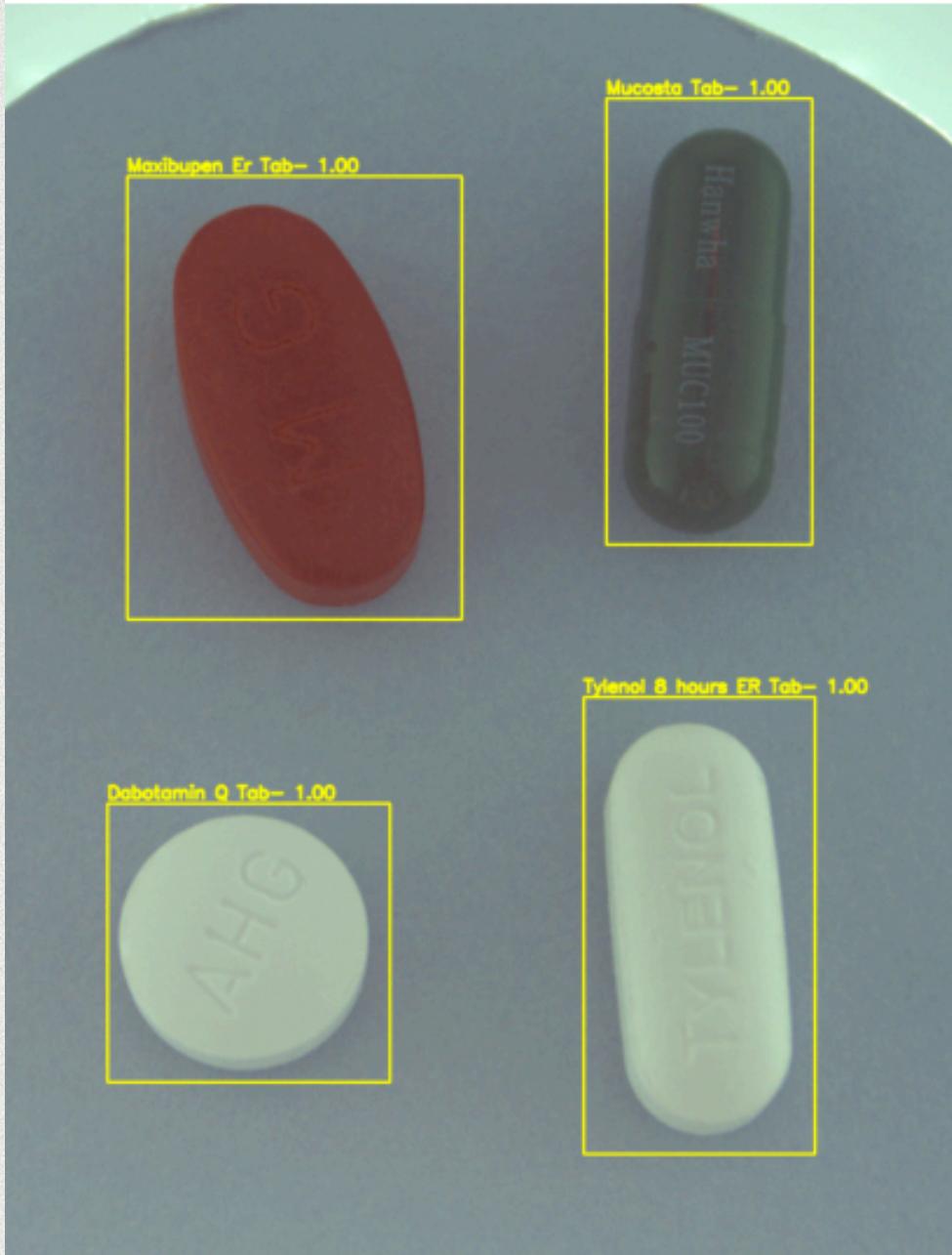


학습모델설정

- epoch: 80
- lr: 0.001
- batch size: 8
- 최적화 알고리즘: SGD
- 학습률 스케줄러: LinearLR

시각화 결과를 확인해봤을 때 바운딩 박스 예측과 클래스 분류가 잘 된것을 확인할 수 있었음

05. SSD



학습 모델 : ssd300_vgg16

학습 파라미터 : Epoch : 10

Size : 300 x 300

Optimizer : SGD

Learning Rate : 2e-4

Scheduler : StepLR(step_size = 5, gamma = 0.2)

학습 결과

- 학습 속도는 총 30분
- train mAP는 0.75 기준 0.986, recall은 0.910로 좋은 학습 결과를 지닌다.

결과

submission (1).csv

0.92715

Complete · Terry_Choi · 3h ago

05. SSD 모델 구동 중 생긴 문제점

```
total_classes = num_classes + 1 # 배경 포함

# head에 있는 num_classes 교체
try:
    model.replace_head(total_classes)
except AttributeError:
    try:
        # 직접 교체
        model.head.classification_head.num_classes = total_classes
    except Exception:
        # 구버전 완전 수동 교체
        num_anchors = model.anchor_generator.num_anchors_per_location()
        in_channels = [512, 1024, 512, 256, 256, 256] # SSD300 기본 구조 유지
        model.head.classification_head = SSDClassificationHead(in_channels, num_anchors, total_classes)
```

라벨링 : 배경을 포함한 라벨을 넣어야 하지만, 라벨이 밀리거나, 학습 내용과 라벨이 불일치하게 나오는 등의 문제가 상당히 많이 나왔다.

과학습 : 학습 데이터에서 중복이 많아 mAP@0.75의 변화는 없었지만, train_loss 값이 크게 떨어지는 경우가 많았다.

(10 epochs 기준 {mAP : 0.968 → 0.986} / {train_loss : 1.895 → 0.146})

Precision의 부재 : PR graph를 확인하기 위해 precision 값을 구하려 했지만, 구하는 과정에서 에러가 빈번히 발생해 precision 값을 제대로 구할 수 없었다.

05. YOLOv10

모델 및 학습 설정

- YOLOv10n 모델(소형 모델) 사용
- 비어있는 annotation을 모두 채운 데이터셋을 학습
- 배치 사이즈 16
- 학습률 0.001
- 최적화 알고리즘 - Adam 채택
- 100에폭 학습 - 10에폭마다 학습 가중치 저장
- 검증 손실이 가장 낮았을 때 가중치를 테스트에 사용

검증 데이터 모델 예측 성능 요약표

- mAP@50: 0.978
- mAP@50–95: 0.936
- Precision: 0.914
- Recall: 0.936

캐글 점수 측정



[submission_yolov10_noaug.csv](#)

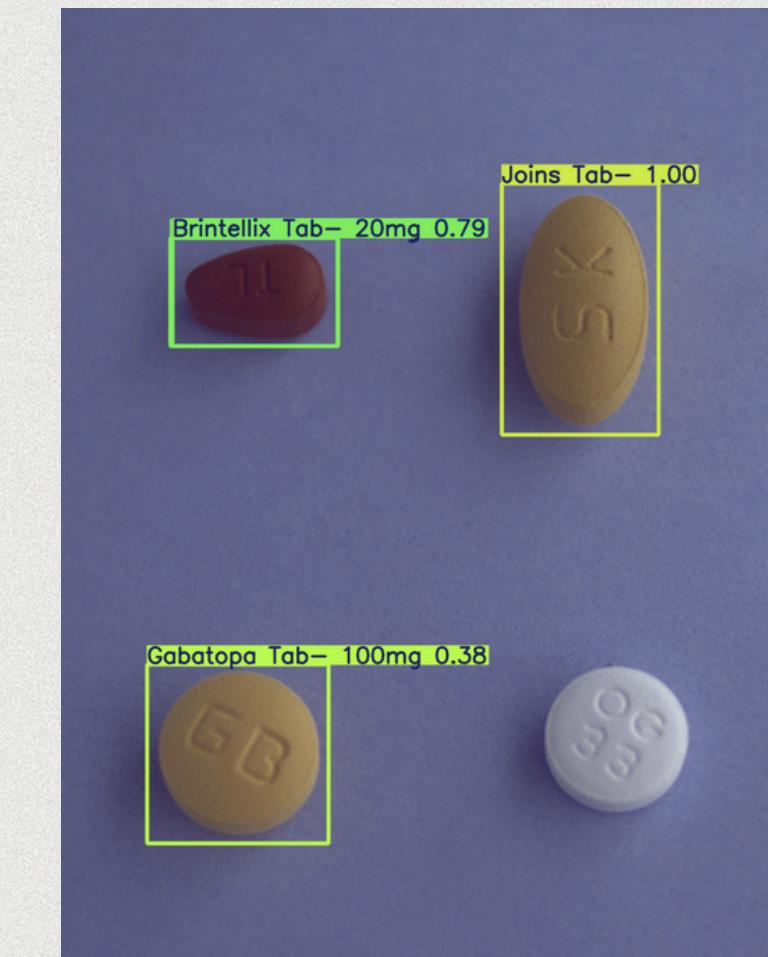
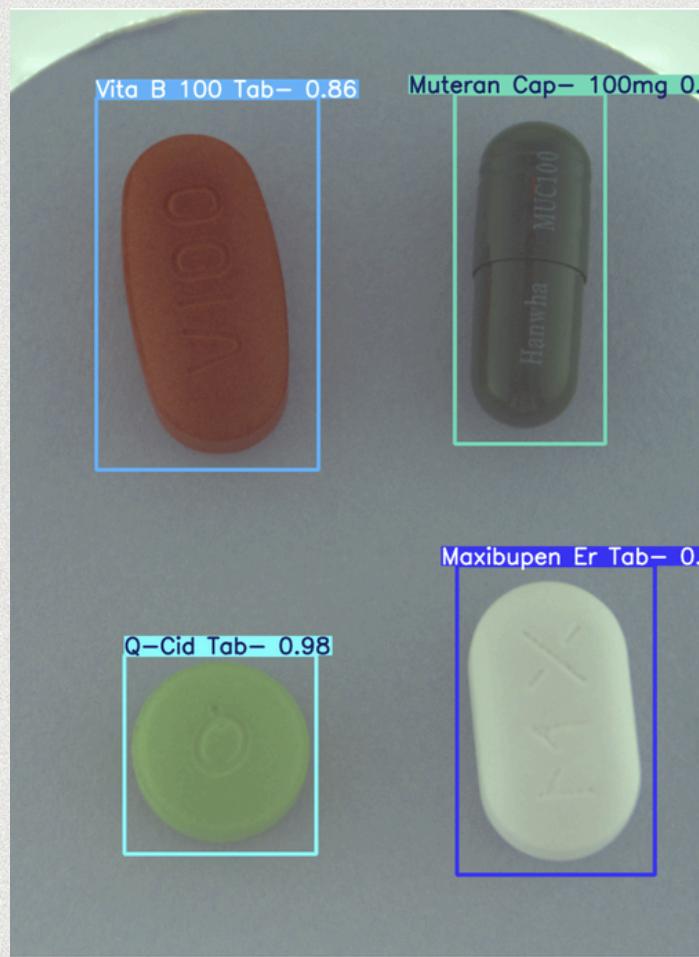
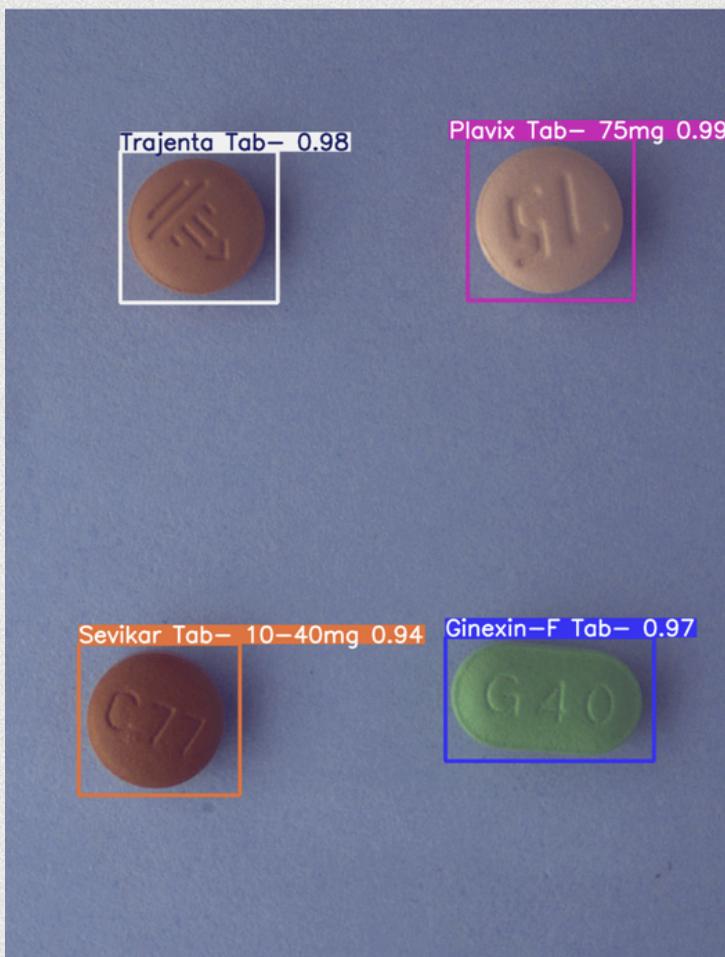
Complete · Yi Seung Cheol · 5h ago · YOLOv10 모델 사용(데이터 증강x, 하이퍼파라미터 튜닝x, 클래스 불균형 해소x)

0.96914

05. YOLOv10

모델 예측 결과 시각화 자료

모델 예측시 신뢰도 임계값 0.1로 설정 ➡ 신뢰도 임계값 0.1 이하의 예측은 모두 제거



일반적인 예측이 잘 맞으나
일부 알약에 대한 예측이
누락됨을 확인함

05. YOLOv8

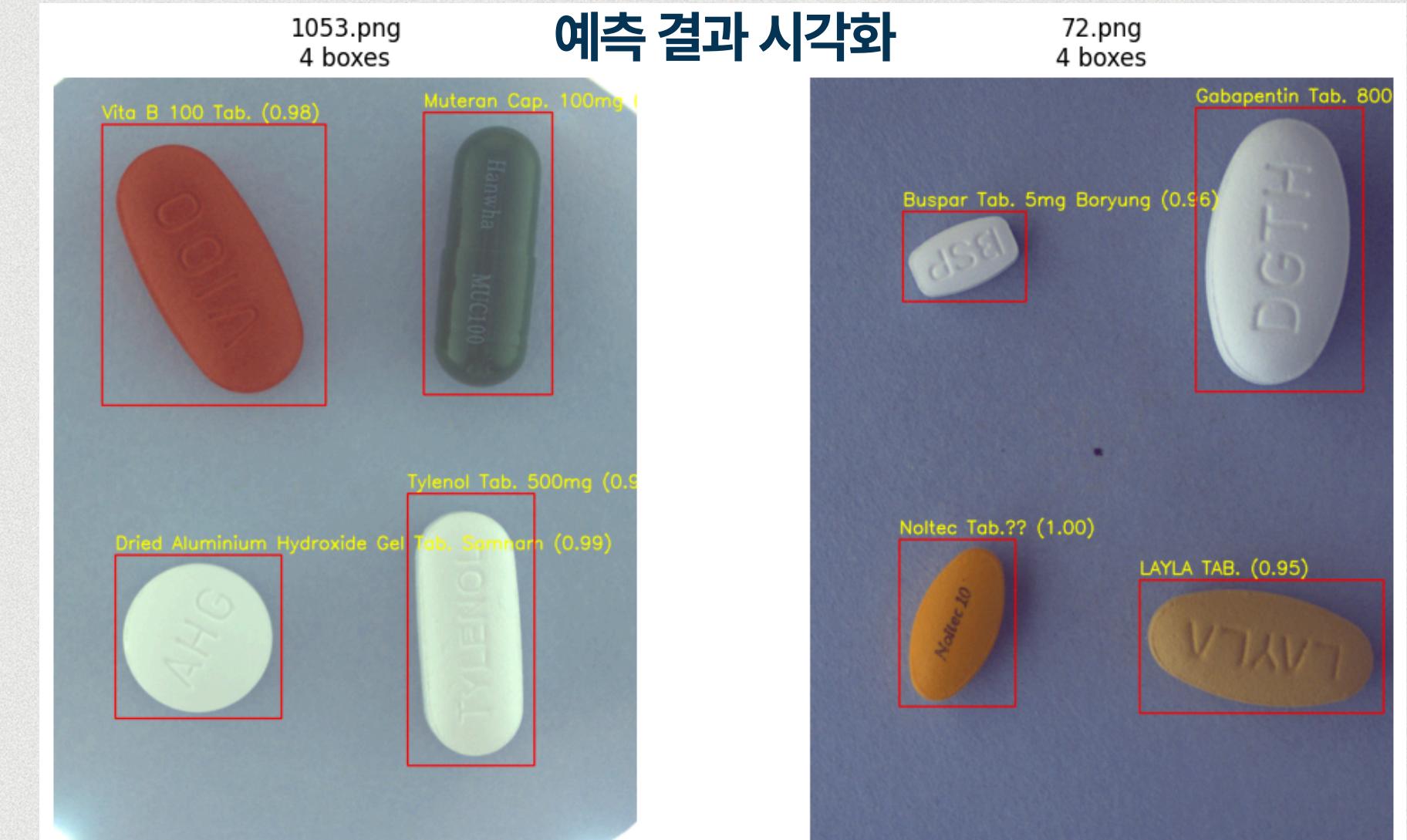
1차 학습 파라미터 설정

- yolov8n.pt 가장 가벼운 Nano 모델로 1차 학습
- Epochs 100
- Batch Size 8
- Image Size 640 x 640
- Optimizer SGD (기본값)

성능 요약표 (Val)

- mAP@50 : 0.9950
- mAP@50–95 : 0.9537
- Precision : 0.9891
- Recall : 0.9954

캐글점수 0.96804



- rare class에 대한 학습 성능을 높이기 위해 클래스 불균형 해소 방안 고민

06. 클래스 불균형 해소

초기 계획

- 학습 프로세스 설계

- 1차: 전체 데이터로 기본 feature 학습
- 2차 학습: 50개 미만의 rare class만 별도로 k-fold 방식으로 fine-tuning
→ 1차 모델 점수와 2차 k-fold 평균 점수를 합산·비교하여 성능 개선 여부 검증

- 데이터셋 설계

- 기존 데이터셋을 train / val / test 8:1:1 비율로 나눠서 자체 테스트 데이터셋 확보 시도
- rare class의 val 데이터는 0으로 설정 → test 세트를 충분히 확보하여 1차 평가에 집중
 - 이유: val 세트를 따로 분리할 경우 검증 데이터가 너무 적어 통계적 의미가 떨어짐
 - 따라서 rare class는 train/test에만 포함해 학습시키고, 이후 별도 fine-tuning 단계에서 k-fold 검증으로 보완
- rare class는 crop, 합성 및 데이터 증강을 통해 부족한 개수 보완
→ fine tune 학습과 8:1:1로 분할한 데이터셋은 최종적으로 미적용

06. 클래스 불균형 해소

최종 실행 방안

앞서 Roboflow로 라벨 누락을 보완한 데이터셋에서 50개 미만 있는 rare class 알약을 파악하고, 해당하는 알약이 포함된 이미지의 객체들을 잘라서 합성해 새로운 이미지 데이터와 그에 맞는 annotation 정보를 만들기로 함

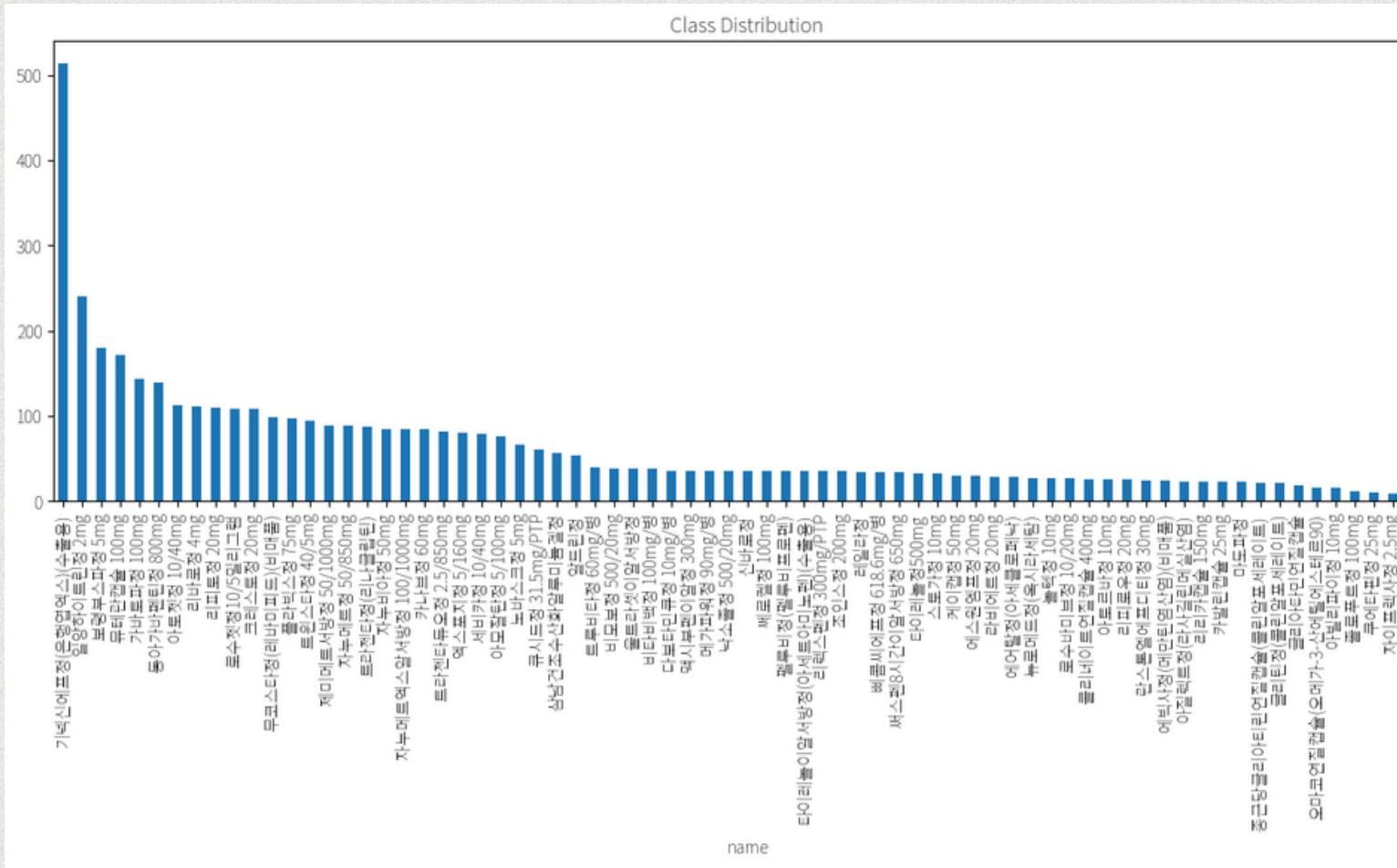
알약 crop 후 이미지 생성

- 알약들의 바운딩 박스 정보를 기반으로 이미지 크롭
- 크롭한 객체 이미지의 배경을 U-2-Net 모델을 사용해 제거
- 알약의 종류 당 100번씩 들어가도록 이미지 생성 후 COCO, YOLO 포맷의 annotation 파일 생성

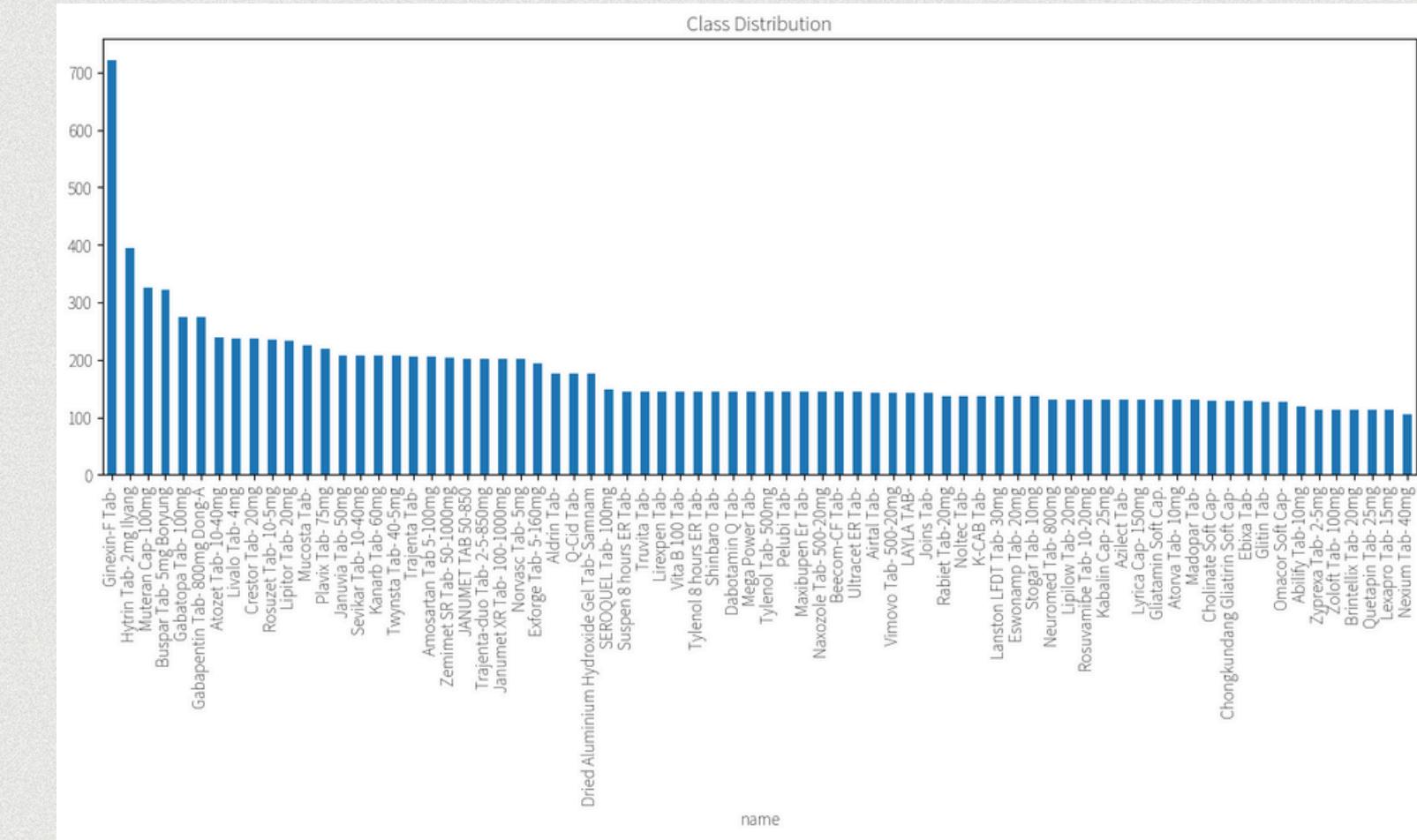


06. 클래스 불균형 해소

초기 데이터셋 Class 분포



데이터 라벨링 및 크롭+합성 후 Class 분포



07. 클래스 불균형 해소 후 YOLO 모델 성능 측정

모델 학습 설정

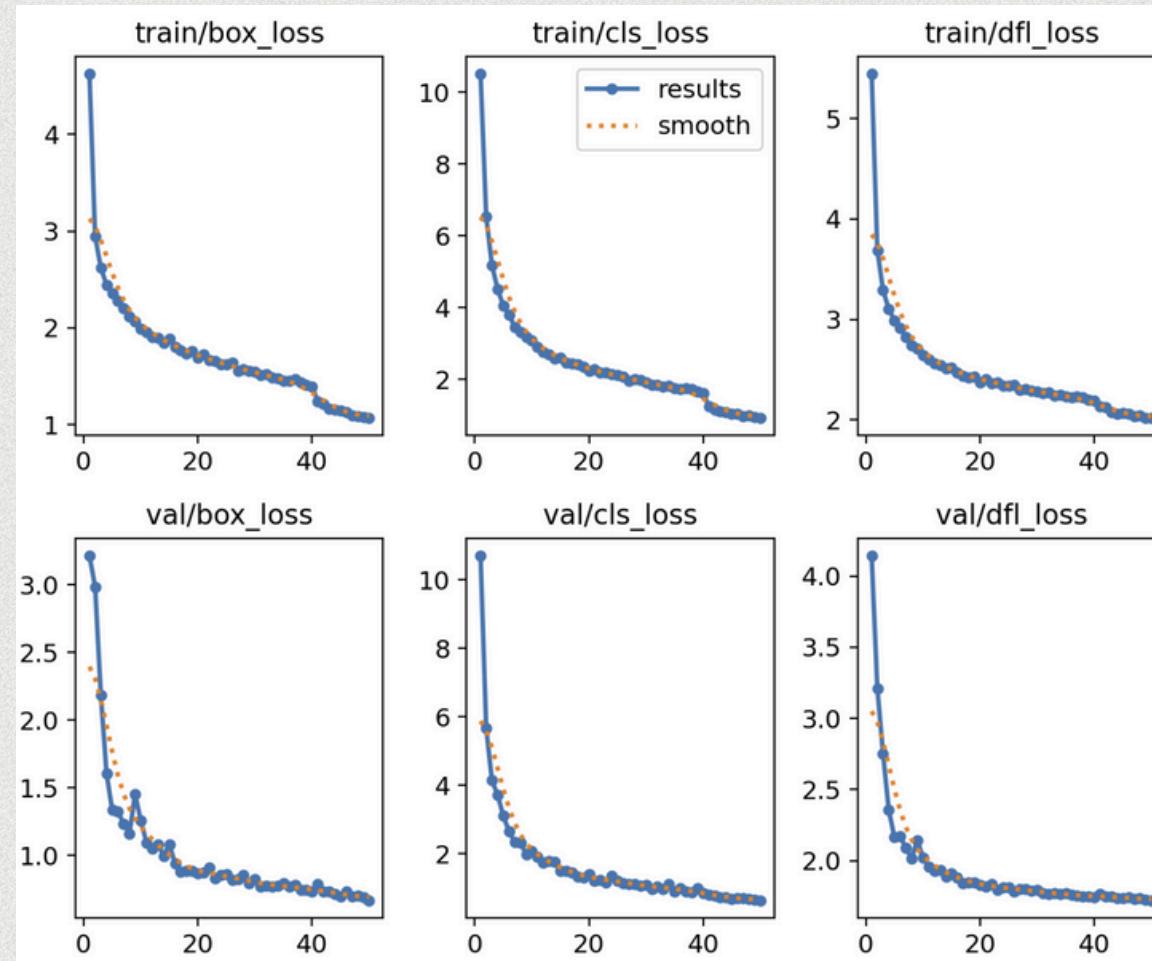
- 기존 학습 방식에서 100이었던 에폭 수를 50으로 줄임
- 나머지 하이퍼파라미터, 검증 세팅은 동일하게 설정

학습 곡선 분석

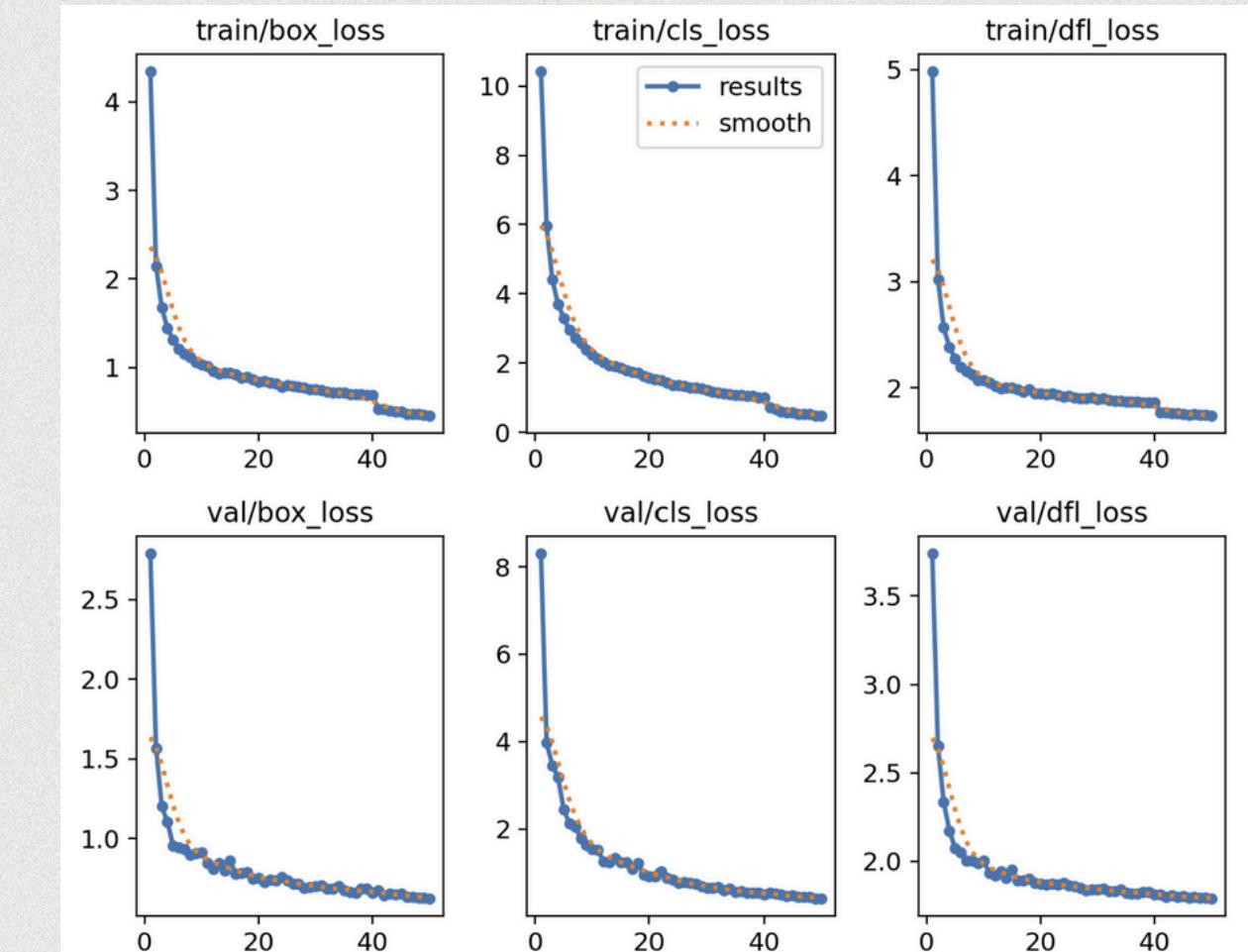
- 박스 예측 오차 곡선, 클래스 예측 오차 곡선의 y축을 봤을 때 오차 감소 폭이 커지는 것을 관찰할 수 있었음
- 데이터 증강 전에는 오차가 갑자기 큰 폭으로 튀는 경우가 있었지만 증강 후 그런 경우가 거의 없어짐

데이터 증강 전/후 학습 곡선 비교

증강 전



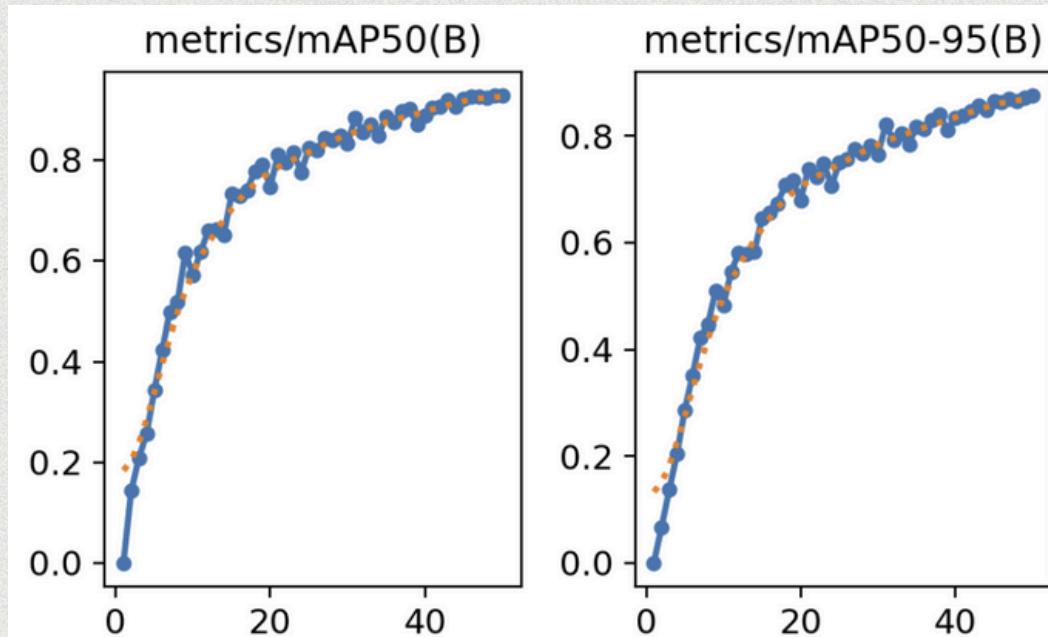
증강 후



07. 클래스 불균형 해소 후 YOLO 모델 성능 측정

데이터 증강 전/후 mAP 지표 비교

증강 전



증강 후

mAP 지표 분석

- mAP는 객체 인식 모델의 성능을 측정하는 지표
- 데이터 증강 전/후 매 에폭마다 측정된 mAP 지표 곡선을 비교해 보았을 때, 데이터 증강 후 mAP 값이 더 빠르게 1에 가까이 수렴한 것을 볼 수 있음
- 캐글 점수도 100에폭 학습했을 때의 점수(약 0.969)와 데이터 증강 후 50에폭 학습했을 때의 점수(약 0.961)가 큰 차이가 없음
- 더 적은 학습으로 같은 성능을 낼 수 있음을 의미

데이터 증강 후 데이터 학습 YOLOv10 모델 캐글 점수 측정



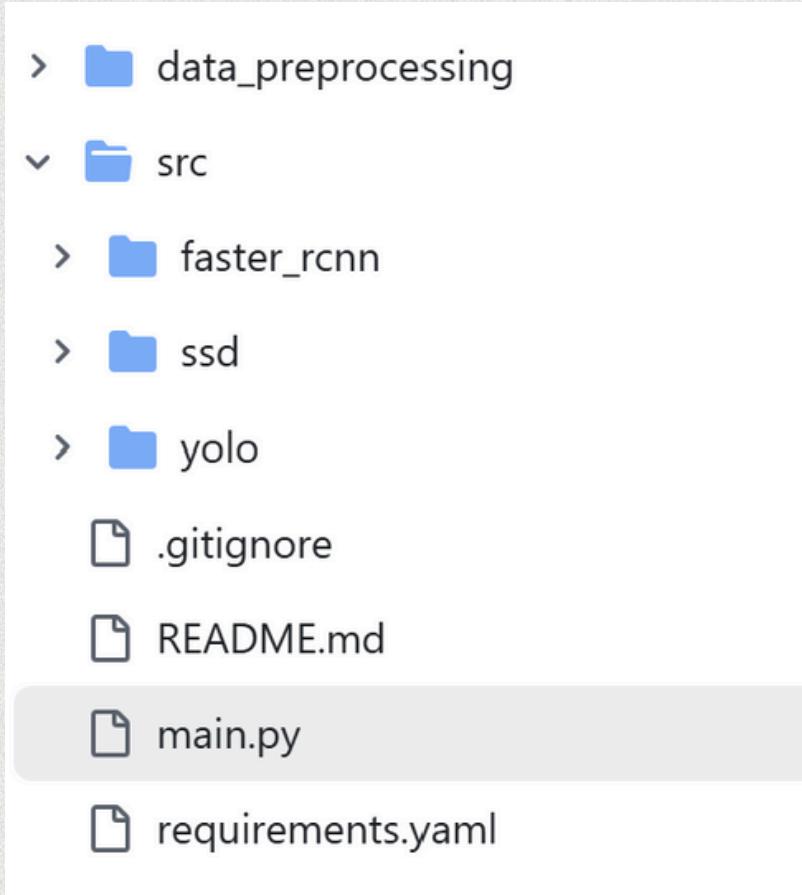
submission_yolov10_afteraug3.csv

Complete · Yi Seung Cheol · 1m ago

0.96065

08. 학습 및 평가 파이프라인 구성 및 자동화

main.py에서 모델을 사용할 수 있게 제작



- **data_preprocess**

- 데이터 EDA 및 전처리 코드를 넣어둔 폴더

- **src**

- main.py에서 학습할 수 있는 모델을 넣어둔 폴더.
- main.py 실행 시 -model 옵션에 SSD, RCNN, YOLOv8, YOLOv10중 하나를 입력해 선택한 모델로 학습할 수 있도록 코드 구성
- 각 모델에 맞는 데이터 로드 - 학습 - 평가 - 캐글 제출용 csv 파일 생성까지의 과정 자동화

출처

팀 Git : https://github.com/charls1114/Codeit_Pill_Detection_Project

팀 공유 데이터 폴더 : <https://drive.google.com/drive/>

협업 일지 : <https://www.notion.com/ko>

SSD : <https://github.com/amdegroot/ssd.pytorch>

Faster R-CNN : https://docs.pytorch.org/vision/main/models/faster_rcnn.html

YOLO : <https://github.com/ultralytics/ultralytics>

U-2-Net : <https://github.com/xuebingin/U-2-Net>

Roboflow : <https://roboflow.com/>

우리 친구 : <https://chatgpt.com/>

Q & A

감사합니다