알약 이미지 객체탐지 프로젝트

스마트폰 사진에서 알약 종류 (Class) + 위치 (Bounding Box) 동시 탐지

3 팀

목차

※ 본 자료는 " 7. Streamlit 을 활용한 서비스 구현" 이전까지만 포함합니다 .

프로젝트 개요 / 목표

저희 프로젝트는 스마트폰으로 촬영된 알약 이미지에서 알약의 종류 (Class) 와 위치 (Bounding Box) 를 동시에 탐지하는 객체탐지 모델을 개발하는 것입니다 .
데이터는 Kaggle 에서 제공되며 , 원천은 AI Hub 형식이고 , 실무적으로는 "사진 한 장을 찍으면 약명 / 전문의약품 여부 / 효능 , 효과 / 연령 / 복용법 / 병용금기성분과 주의사항"를 바로 알려주는 흐름을 목표로 했습니다 .

목표

1) Kaggle 데이터 (AI Hub 원천) 구조를 파악하고 학습 가능한 형태로 정리하기
2) 학습 파이프라인 구축 : 전처리 → 데이터셋 → 학습 → 추론 → 제출 파일 생성까지 end-to-end 로 만들기
3) 비약 서비스 정보 매핑 : 단순 분류가 아니라 , 약 이름 / 제조사 / 성분 같은 서비스 정보를 class id 와 연결하기

1. 데이터 셋 준비

1-1. 모듈 및 라이브러리 로드 / GPU 확인

환경 세팅입니다. 한글 폰트 설치, Colab 기준으로 GPU를 먼저 잡고, Torch/vision, OpenCV, numpy 같은 기본 라이브러리를 로드했습니다.

```
### **2.1 라이브러리 로드 & 장치(GPU) 설정**
"""

import os
import random
import json
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

# 한글 폰트 설정
!pip install -q koreanize-matplotlib

import koreanize_matplotlib
import matplotlib.pyplot as plt

# 테스트
plt.figure(figsize=(4, 2))
plt.plot([1, 2, 3])
plt.title("한글 폰트 정상 작동 확인")
plt.show()

# 시드 고정
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    print(f"Seed set to {seed}")

set_seed(42)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Current Device: {device}")

# 한글 폰트(나눔글꼴) 설치
!sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf
```

1-2. 데이터 다운로드 (Kaggle)

데이터 다운로드는 Kaggle competition 파일을 받는 방식입니다.
- kaggle.json 업로드 → ~/.kaggle 이동 및 권한 설정
- kaggle competitions download -c ai06-level1-project
- zip unzip 해서 ./data 폴더에 풀기

```
1
2     # 1. 기존 설정 초기화
3     !rm -f kaggle.json
4     !rm -rf ~/.kaggle
5
6     print("'kaggle.json'을 선택해주세요!")
7     uploaded = files.upload() # <--- 데이터가 없을 때만 실행됨
8
9     # 2. 파일 이동 및 권한 설정
10    if 'kaggle.json' in uploaded.keys():
11        print("✅ 파일 업로드 성공! 인증을 진행합니다.")
12        !mkdir -p ~/.kaggle
13        !mv kaggle.json ~/.kaggle/
14        !chmod 600 ~/.kaggle/kaggle.json
15
16        # 3. 다운로드 및 압축 해제
17        print("⬇️ 데이터 다운로드 시작...")
18        !kaggle competitions download -c ai06-level1-project
19
20        print("📦 압축 해제 중...")
21        !unzip -q ai06-level1-project.zip -d ./data
22        print("🎉 데이터 준비 완료! (./data 폴더)")
23
24    else:
25        print("❌ 파일 업로드가 취소되었습니다.")
26
27 """### **2.3 구글 드라이브 마운트 및 백업**"""
28
29 from google.colab import drive
```

1-3. 드라이브 마운트 및 백업 (재현성 / 안전장치)

Colab 은 런타임이 꺼지면 데이터가 날아갈 수 있어서, 다운로드 zip 을 Google Drive 로 백업했습니다.
이 단계는
"프로젝트를 기간 내 완성"
해야 해서, 중간에 런타임
리셋으로 망하는 리스크를 줄이는
목적이었습니다.

```python
import os

# 1. 구글 드라이브 연결 (로그인 창이 뜨면 승인해주세요)
print("□ 구글 드라이브를 연결합니다...")
drive.mount('/content/drive')
MY_BACKUP_FOLDER = "/content/drive/MyDrive/My_AI_Project/team_project"

# 2. 백업 실행
zip_file = './ai06-level1-project.zip'

if os.path.exists(MY_BACKUP_FOLDER):
    if os.path.exists(zip_file):
        print(f"□ 드라이브로 복사 시작... (약 1~2분 소요)")
        print(f"Source: {zip_file}")
        print(f"Target: {MY_BACKUP_FOLDER}")

        # 파일 복사
        shutil.copy(zip_file, MY_BACKUP_FOLDER)

        print("□ 백업 완료! 이제 데이터가 드라이브에 안전하게 저장되었습니다.")
    else:
```

1-4. (핵심) 분산된 JSON 병합 → train.json 생성

원본 라벨이 폴더 / 파일 단위로 수백 개 JSON 으로 쪼개져 있어서 , 그대로는 학습이 불가능했습니다 .
그래서 모든 json 을 재귀로 탐색해서 ,
images / annotations / categories 를 하나의 COCO JSON 으로 병합했습니다 .

로직
- glob(..., recursive=True) 로 모든 json 경로 수집
- seen_image_ids, seen_category_ids 로 중복 방지
- annotations 는 그대로 extend (이미지 id 기준으로 연결됨)
- 결과를 ./data/train.json 으로 저장

```python
import os
import json
import glob
from tqdm import tqdm

# 1. 경로 설정 (Config 없이 직접 지정)
DATA_ROOT = './data'  # 임시 정의
ANNOTATION_ROOT = os.path.join(DATA_ROOT, 'train_annotations')
OUTPUT_JSON = os.path.join(DATA_ROOT, 'train.json')

print(f"🔍 검색 경로: {ANNOTATION_ROOT}")

# 2. 흩어진 모든 .json 파일 찾기 (재귀 탐색)
json_files = glob.glob(os.path.join(ANNOTATION_ROOT, '**', '*.json'), recursive=True)
print(f"📂 발견된 JSON 파일 개수: {len(json_files)}개")

# 3. 데이터 병합 (Merging)
merged_data = {
    "images": [],
    "annotations": [],
    "categories": []
}

# 중복 방지용 Set
seen_category_ids = set()
seen_image_ids = set()

print("🚀 병합 시작...")

for json_path in tqdm(json_files):
    with open(json_path, 'r', encoding='utf-8') as f:
        try:
            data = json.load(f)

            # (1) Images 병합
            for img in data.get('images', []):
                if img['id'] not in seen_image_ids:
                    merged_data['images'].append(img)
                    seen_image_ids.add(img['id'])

            # (2) Annotations 병합
            if 'annotations' in data:
                merged_data['annotations'].extend(data['annotations'])

            # (3) Categories 병합
            for cat in data.get('categories', []):
                if cat['id'] not in seen_category_ids:
                    merged_data['categories'].append(cat)
                    seen_category_ids.add(cat['id'])

        except json.JSONDecodeError:
            print(f"⚠️ 에러: 깨진 파일 건너뜀 - {json_path}")

# 4. 결과 저장
with open(OUTPUT_JSON, 'w', encoding='utf-8') as f:
    json.dump(merged_data, f, ensure_ascii=False)

print("-" * 30)
print(f"✅ 병합 완료!")
print(f"📁 저장 위치: {OUTPUT_JSON}")
print(f"- 총 이미지: {len(merged_data['images'])}")
print(f"- 총 박스(Annotations): {len(merged_data['annotations'])}")
print(f"- 총 알약 종류(Categories): {len(merged_data['categories'])}")
```

glob 재귀 탐색 + 중복 방지 (seen_* set) + images/annotations/categories 병합

2. 경로 및 하이퍼파라미터 설정

2-1. 기본 경로 설정 (Config)

그 다음은 실행 환경에 따라 파일이 다를 수 있어서 , Config 를 만들어 데이터셋 파일 자동 선택 로직을 넣었습니다 .

- train_final_augmented.json (증강 포함 고성능 버전)
- train.json (병합만 된 원본 버전)

이미지 사이즈 , 배치 , lr, epoch 같은
기본 하이퍼파라미터를 Config 로 묶어서
이후 모델들 (Faster R-CNN/Detectron2/YOLO) 에도
일관되게 적용하려고 했습니다 .

```python
class Config:
    # 1. 기본 경로 설정
    DATA_ROOT = './data'
    TRAIN_IMG_DIR = os.path.join(DATA_ROOT, 'train_images')
    TEST_IMG_DIR = os.path.join(DATA_ROOT, 'test_images')

    # 2. 데이터셋 경로 자동 감지 (Integrity Logic)
    # 우선순위 1: 증강된 최종 파일 (우리가 복구한 것)
    # 우선순위 2: 원본 병합 파일 (처음 시작하는 사람용)

    AUGMENTED_PATH = os.path.join(DATA_ROOT, 'train_final_augmented.json')
    ORIGINAL_PATH = os.path.join(DATA_ROOT, 'train.json')

    if os.path.exists(AUGMENTED_PATH):
        TRAIN_JSON = AUGMENTED_PATH
        STATUS = "Augmented (High Performance)"
    elif os.path.exists(ORIGINAL_PATH):
        TRAIN_JSON = ORIGINAL_PATH
        STATUS = "Original (Baseline)"
    else:
        # 둘 다 없으면 에러 방지를 위해 None 처리 (2.4를 안 돌렸을 경우)
        TRAIN_JSON = None
        STATUS = "Missing"

    # 3. 하이퍼파라미터
    IMG_SIZE = 224
    BATCH_SIZE = 16
    LEARNING_RATE = 1e-4
    NUM_EPOCHS = 20
    NUM_WORKERS = 2

# == [검증 및 안내] ==
print("-" * 40)
if Config.STATUS == "Missing":
    print("⚠ 학습용 JSON 파일이 없습니다!")
    print("   ▶ [처음이라면?] 2.4 단계를 먼저 실행해서 train.json을 만드세요.")
    print("   ▶ [복구했다면?] 파일 경로가 ./data 안에 제대로 들어왔는지 확인하세요.")
else:
    print(f"✅ Config 설정 완료!")
    print(f"📂 사용 데이터셋: {Config.STATUS}")
    print(f"📄 파일 경로: {Config.TRAIN_JSON}")

    if Config.STATUS == "Original (Baseline)":
        print("💡 Tip: 성능을 높이려면 [3.5 데이터 증강] 단계를 실행하거나 백업파일을 복구하세요.")
```

3. 데이터 이해와 상비약 서비스 매핑

3-1. 데이터 로드 및 ID-Name 매핑

categories 에는 단순 클래스 이름뿐 아니라 제조사 (dl_company), 성분 (dl_material) 도 들어있었습니다 .
그래서 pill_info_map 을 만들어서
class_id → (이름 / 회사 / 성분) 으로 매핑했습니다 .
이게 서비스 목표 (" 탐지 결과를 약 정보로 연결") 의 기반이 됩니다 .

전체 56 개 클래스 중에서 , " 서비스 MVP" 로 먼저 제공할 Top 30 상비약 후보를 뽑았습니다 .
annotations 를 기반으로 클래스별 샘플 수를 계산하고 , count 내림차순 정렬 후 상위 30 개를 CSV 로 저장했습니다 .

결과물 :
- top30_pills_service.csv (팀 공유용)

```
# 1. JSON 파일 로드
print(f"📂 데이터 로드 중... ({Config.TRAIN_JSON})")
with open(Config.TRAIN_JSON, 'r', encoding='utf-8') as f:
    data = json.load(f)

# 2. ID <-> 약 정보 매핑 (Dictionary 생성)
pill_info_map = {}

for cat in data['categories']:
    pill_info_map[cat['id']] = {
        'name': cat['name'],              # 약 이름
        'company': cat.get('dl_company', ''), # 제조사
        'material': cat.get('dl_material', '') # 성분 (서비스 핵심 정보!)
    }

print(f"✅ 매핑 완료! 총 {len(pill_info_map)}개의 알약 정보를 확보했습니다.")
# 테스트로 첫 번째 알약 정보 출력해보기
first_id = list(pill_info_map.keys())[0]
print(f"📝 예시 (ID {first_id}): {pill_info_map[first_id]}")

"""### **3.2 클래스 분포 확인**
- 가장 데이터가 많은 약 30개를 추출
"""

# 1. 알약 등장 횟수 세기 (Annotations 카운트)
# 이미지 속에 어떤 알약(category_id)이 몇 번 나왔는지 셉니다.
pill_counts = Counter([ann['category_id'] for ann in data['annotations']])

# 2. 분석용 데이터프레임(표) 생성
df = pd.DataFrame.from_dict(pill_counts, orient='index', columns=['count'])
df.index.name = 'class_id'
df.reset_index(inplace=True)

# 3. 아까 만든 'pill_info_map'을 이용해 이름과 성분 채워 넣기
df['pill_name'] = df['class_id'].map(lambda x: pill_info_map.get(x, {}).get('name', 'Unknown'))
df['company'] = df['class_id'].map(lambda x: pill_info_map.get(x, {}).get('company', 'Unknown'))
df['material'] = df['class_id'].map(lambda x: pill_info_map.get(x, {}).get('material', 'Unknown'))

# 4. 개수 많은 순서로 정렬 (내림차순)
df = df.sort_values(by='count', ascending=False).reset_index(drop=True)

# 5. [핵심] 서비스 타겟 Top 30 추출 및 저장
top30_df = df.head(30)
save_csv_path = './top30_pills_service.csv'

# 엑셀에서 한글 깨짐 방지를 위해 'utf-8-sig' 인코딩으로 저장
top30_df.to_csv(save_csv_path, index=False, encoding='utf-8-sig')

print("-" * 50)
print(f"💾 [서비스용 파일 저장 완료] {save_csv_path}")
print("👉 코랩 왼쪽 파일 폴더에서 이 파일을 다운로드해서 팀원들과 공유하세요!")
print("-" * 50)
print("📊 [데이터 수량 Top 10 알약]")
print(top30_df[['class_id', 'pill_name', 'count', 'company']].head(10).to_string(index=False))
```

4. 데이터 불균형 해결

4. 데이터 불균형 해결 : Offline 증강

클래스 분포를 보니 56 개 중 대부분이 50 장 미만이었고, 심한 건 3~5 장 수준이라 학습이 무너질 가능성이 컸습니다.

증강 설계 포인트 :
- bbox 가 있는 데이터라서 이미지 변환과 bbox 변환이 같이 가야 함
- Albumentations bbox_params(format='coco') 로 해결
- SafeRotate 경고 이슈가 있어서 value 인자를 빼서 버전 호환 처리
- 생성된 증강 샘플은 id 충돌을 막기 위해 이미지 / 어노테이션 id 를 200000 번대부터 새로 부여

4. 데이터 불균형 해결 : Offline 증강

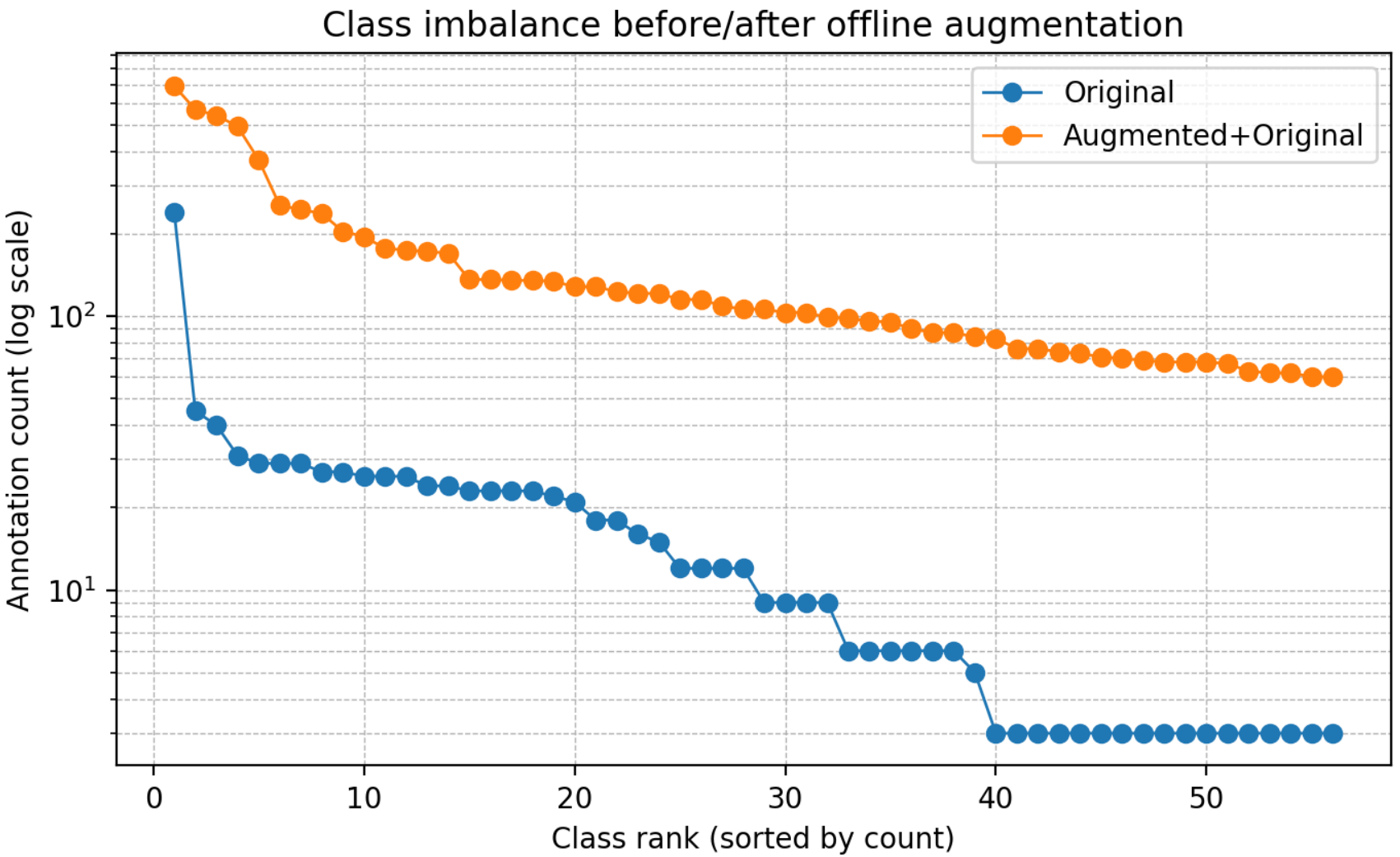
실사용 환경에서는 알약을 "약 봉지 / 비닐 봉투"에 담아 촬영하는 경우가 있을 수 있음봉지 촬영 상황에서도 탐지가 가능하도록
증강 / 전처리에 신경을 썼습니다 .
단순히 데이터 수를 늘리는 목적을 넘어
봉지 속 알약의 현실적인 촬영 조건
회전 , 반전 , 조명 , 난반사 , 흐림 , 노이즈 등을 학습 데이터에 반영했습니다 .

- SafeRotate(limit=20)
- HorizontalFlip
- VerticalFlip
- RandomBrightnessContrast (brightness=0.2, contrast=0.2)
- GaussianBlur(blur_limit=(3,5))
- GaussNoise(var_limit=(10,50))
- CLAHE(clip_limit=4.0)
- Rotate(limit=180)

```python
import albumentations as A
from albumentations.pytorch import ToTensorV2

# 1. 학습용 (Train): 약 봉지 시뮬레이션 완결판
train_transform = A.Compose([
    A.Resize(Config.IMG_SIZE, Config.IMG_SIZE),

    # (1) 회전: 봉지 안에서 굴러다님
    A.Rotate(limit=180, p=0.7),

    # (2) 반전: 앞뒤 구분 모호함
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),

    # (3) [New] 비닐 질감 시뮬레이션 
    # Blur: 비닐 때문에 뿌옇게 보이는 현상
    A.GaussianBlur(blur_limit=(3, 7), p=0.3),
    # CLAHE: 비닐 난반사로 인한 국소적 대비 변화
    A.CLAHE(clip_limit=4.0, tile_grid_size=(8, 8), p=0.3),

    # (4) 조명 변화 (기존 유지)
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),

    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2()
], bbox_params=A.BboxParams(format='coco', label_fields=['category_ids']))
```

[시각화] 불균형 개선 : Rank plot (Before vs After)

원본 min=3 → 전체 (min)=60 (증강 포함).



[코드 캡처] Offline 증강 파이프라인 (SafeRotate 경고 대응)

SafeRotate value 인자 제거 (버전 호환) + bbox 변환 동시 처리

```
# --- [4] 증강 파이프라인 정의 (Warning Fixed) ---
# SafeRotate에서 경고를 유발하던 value 인자 제거 (버전 호환성)
pipeline = A.Compose([
    A.SafeRotate(limit=20, p=0.7, border_mode=cv2.BORDER_CONSTANT),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    A.GaussianBlur(blur_limit=(3, 5), p=0.3),
    # GaussNoise는 버전에 따라 경고가 뜰 수 있으나 기능상 문제 없으므로 유지
    A.GaussNoise(var_limit=(10.0, 50.0), p=0.3),
], bbox_params=A.BboxParams(format='coco', label_fields=['category_ids'], min_visibility=0.3))
```

5. 데이터셋 / 로더 구축 학습 파이프라인

5-1. Smart Select (증강 / 전처리 / 원본 자동 선택)

학습할 때는 train_final_augmented.json 이 있으면 그걸 우선 사용하고 , 없으면 train_final.json, 없으면 train.json 으로 자동 선택하게 했습니다 .
이건 팀원이 각자 환경에서 실행해도
흐름이 유지되게 하는 장치입니다 .

```python
# 1. 경로 설정
if os.path.exists('./data/train_final_augmented.json'):
    TARGET_JSON = './data/train_final_augmented.json'
    print(f"□ 증강된 데이터셋 사용: {TARGET_JSON}")
else:
    TARGET_JSON = './data/train_final.json'
    print(f"□ 원본 데이터셋 사용: {TARGET_JSON}")

IMG_DIR = './data/train_images'

# 2. PillDataset 정의
class PillDataset(Dataset):
    def __init__(self, json_path, root_dir):
        self.root_dir = root_dir
        with open(json_path, 'r') as f:
            data = json.load(f)

        self.images = {img['id']: img for img in data['images']}
        self.annotations = {}
        for ann in data['annotations']:
            img_id = ann['image_id']
            if img_id not in self.annotations: self.annotations[img_id] = []
            self.annotations[img_id].append(ann)

        self.image_ids = list(self.images.keys())

        # 라벨 매핑
        categories = sorted(data['categories'], key=lambda x: x['id'])
        self.cat_id_to_label = {cat['id']: i+1 for i, cat in enumerate(categories)}
        self.label_to_cat_id = {v: k for k, v in self.cat_id_to_label.items()}

        self.num_classes = len(categories) + 1
        print(f"   -> 데이터셋 준비 완료: 클래스 {self.num_classes}개")
```

5-2. PillDataset 구현 (bbox/label 매핑, 예외 처리)

PillDataset 에서는 COCO bbox(x,y,w,h) 를 xyxy 로 바꾸고, 화면 밖으로 나가는 박스는 clipping 했습니다.
그리고 실제로는 라벨을 모델 학습용으로 1 부터 다시 매핑했습니다.
또 실전에서 자주 생기는 문제로:
- 이미지 파일이 실제로 없거나 로드 실패하면 다음 샘플로 넘어가고,
- 박스가 0 개면 학습이 깨지니까 빈 박스 데이터는 자동 스킵했습니다.

```python
def __getitem__(self, idx):
    img_id = self.image_ids[idx]
    info = self.images[img_id]
    path = os.path.join(self.root_dir, info['file_name'])

    if not os.path.exists(path): path = os.path.join('./data', info['file_name'])
    image = cv2.imread(path)

    # 이미지 로드 실패 시 -> 다음 이미지 시도 (재귀 호출)
    if image is None:
        return self.__getitem__((idx + 1) % len(self))

    height, width = image.shape[:2]

    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = image.transpose((2, 0, 1))
    image = np.ascontiguousarray(image)
    image = torch.from_numpy(image).float() / 255.0

    boxes, labels = [], []
    for ann in self.annotations.get(img_id, []):
        x, y, w, h = ann['bbox']

        # Clipping
        x1 = max(0, x)
        y1 = max(0, y)
        x2 = min(width, x + w)
        y2 = min(height, y + h)

        # 유효한 박스만 담기
        if x2 > x1 and y2 > y1:
            boxes.append([x1, y1, x2, y2])
            new_label = self.cat_id_to_label.get(ann['category_id'], 0)
            labels.append(new_label)

    # ⭐ [핵심 수정] 만약 유효한 박스가 하나도 없다면? -> 다음 이미지 가져와!
    if len(boxes) == 0:
        # print(f"⚠️ [Skip] 박스 없는 이미지 건너뜀: {info['file_name']}")
        return self.__getitem__((idx + 1) % len(self))

    target = {}
    target["boxes"] = torch.as_tensor(boxes, dtype=torch.float32)
    target["labels"] = torch.as_tensor(labels, dtype=torch.int64)
    target["image_id"] = torch.tensor([img_id])
    return image, target
```

6. 모델 학습 타임 라인

6. 모델 학습 타임라인

1 차 시도 : Faster R-CNN 베이스라인 (기준점 확보)

처음엔 결과로 기준점을 얻기 위해 torchvision 기반 Faster R-CNN 으로 베이스라인을 만들어 제출했습니다 .
파이프라인이 정상 작동하고 제출이 되는지 확인하는 것이 목표였고 , 결과적으로 약 0.7712 점수로 기준선을 확보했습니다 .

6. 모델 학습 타임라인

2 차 시도 : Detectron2 로 성능 끌어올리기 (막힘 → 방향 전환)
그 다음 목표는 0.80 이상을 노리는 것이었고 , Detectron2 로 넘어갔습니다 .
COCO 기반 설정이 잘 되어 있어서 “제대로만 붙으면 더 올라가겠다”는 기대가 있었는데 ,
여기서 문제가 생겼습니다 .

6. 모델 학습 타임라인

그 다음 목표는 0.80 이상을 노리는 것이었고, Detectron2 로 넘어갔습니다.
COCO 기반 설정이 잘 되어 있어서 점수가 높아질거라 예상했지만,
실제로는 과다 예측(박스가 너무 많이 나오는 현상)이 발생했고, 제출 점수가 0.68491 로 오히려 떨어졌습니다.
threshold/NMS 튜닝, 라벨 역변환 정합성 검증에 시간이 많이 들어가서, 기간 내 완성을 위해 깊게 파지 않고 전환하는 방법을 택했습니다.

6. 모델 학습 타임라인

3 차 시도 : YOLOv8 로 실용성과 점수 동시에 확보

- 학습 / 추론 속도가 빠르고
- 실시간 탐지에 더 현실적이며
- 튜닝 사이클이 짧아서 기간 내 성능 확보에 유리했습니다 .

여기서 필요한 작업은 COCO → YOLO 포맷 변환이었고 ,
yolo_data_config.yaml 에 0~55 클래스 이름을 등록해서 학습을 진행했습니다 .

결과 :

- YOLO 기본 모델 점수 0.82073 까지 상승했습니다 .

6. 모델 학습 타임라인

3 차 -1. TTA 시도
이후 " augment=True" 를 사용한 TTA 도 해봤는데 , 결과는 0.82061 로 살짝 하락했습니다 .
즉 "해봤지만 이득이 거의 없거나 오히려 떨어지는 케이스"였고 , 이건 최종안에서 단독
채택은 하지 않았습니다 .

6. 모델 학습 타임라인

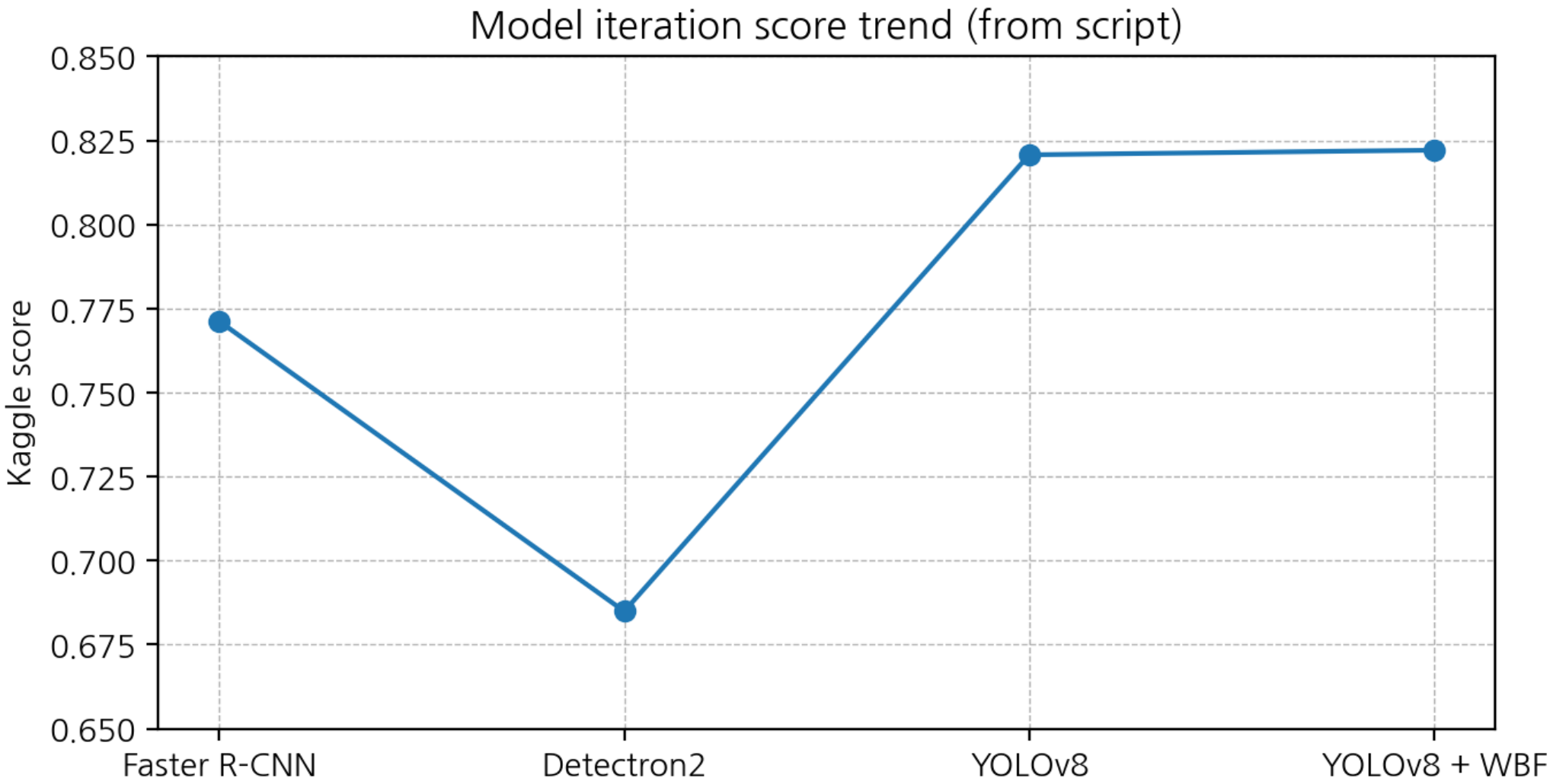
3 차 -2. WBF 앙상블 + 라벨 보정
마지막으로 점수를 올리기 위해 선택한 것이 WBF(Weighted Boxes Fusion) 앙상블입니다.
핵심은 두 가지 추론 결과를 결합하는 것입니다.
- 같은 모델이지만 augment=False 와 augment=True 두 예측을 만들고,
- 박스를 정규화한 뒤 WBF 로 합쳤습니다.

그리고 여기서 중요한 포인트는 YOLO 라벨은 0~55 인데 Kaggle 제출은 1~56 을 요구합니다.
그래서 최종 저장할 때 category_id = int(label) + 1 보정이 반드시 필요했습니다.

최종적으로 WBF 앙상블 점수 0.82211 로 상승했습니다.

[요약] 실험 타임라인 & 점수 변화

Faster R-CNN(0.7712) → Detectron2(0.68491) → YOLOv8(0.82073) → WBF 최종 (0.82211)



6. 모델 학습 타임라인

정리하면 , 저희는
데이터 병합 → 서비스 매핑 (Top30) → 불균형 증강 (+ 봉지 촬영 상황 반영) → 데이터 셋 / 로더 → 모델 실험 (베이스라인→막힘→피벗→최종 앙상블)
의 타임라인으로 프로젝트를 완성했습니다 .

- 증강 포함 라벨 : train_final_augmented.json
- 최종 모델 가중치 : yolov8_best.pt
- 최종 제출 : submission_FINAL_1BASED.csv 또는 WBF 제출 csv

YOLO + WBF 로 전략을 전환한 것이 최종 성능을 만들었고 최종 산출물로 채택하여 다음 단계인 Streamlit 서비스 구현으로 넘어가기로 했습니다 .

[코드 캡처] COCO → YOLO 포맷 변환 (폴더 / 매핑)

train/val split + labels(.txt) 생성

```python
def convert_coco_to_yolo():
    print("="*60)
    print("🚀 COCO(JSON) 데이터를 YOLO(TXT) 포맷으로 변환합니다...")

    # 1. 경로 설정 (입력)
    JSON_PATH = './data/train_final_augmented.json'
    IMG_DIR = './data/train_images' # 원본 이미지 경로

    # 2. 경로 설정 (출력 - YOLO 구조)
    BASE_DIR = os.path.abspath('./datasets/pill_data')

    # 폴더 초기화 (꼬임 방지)
    if os.path.exists(BASE_DIR):
        shutil.rmtree(BASE_DIR)

    for split in ['train', 'val']:
        os.makedirs(os.path.join(BASE_DIR, 'images', split), exist_ok=True)
        os.makedirs(os.path.join(BASE_DIR, 'labels', split), exist_ok=True)

    # 3. JSON 로드
    with open(JSON_PATH, 'r') as f:
        data = json.load(f)

    # 카테고리 매핑 (Detectron2 때와 동일하게 정렬)
    categories = sorted(data['categories'], key=lambda x: x['id'])
    # 원본ID -> 0~55 인덱스
    id_map = {cat['id']: i for i, cat in enumerate(categories)}
    class_names = {i: cat['name'] for i, cat in enumerate(categories)}

    # 이미지 리스트 정리
    images = data['images']
    annotations = data['annotations']

    # 이미지별 어노테이션 그룹화
    img_to_anns = {}
    for ann in annotations:
        img_id = ann['image_id']
        if img_id not in img_to_anns: img_to_anns[img_id] = []
        img_to_anns[img_id].append(ann)

    # 4. 데이터 분할 (Train / Val) - 재현성 고정
    train_imgs, val_imgs = train_test_split(images, test_size=0.2, random_state=42, shuffle=True)

    print(f"📂 데이터 분할: Train {len(train_imgs)}장 / Val {len(val_imgs)}장")
```

[코드 캡처] YOLO bbox 정규화 + data.yaml 생성

x_center/y_center/w/h 정규화 + names 등록

```
            # (2) 라벨 파일(.txt) 생성
            label_file = os.path.splitext(file_name)[0] + '.txt'
            dst_label = os.path.join(BASE_DIR, 'labels', split, label_file)

            with open(dst_label, 'w') as f_out:
                anns = img_to_anns.get(img_id, [])
                for ann in anns:
                    if ann['category_id'] not in id_map: continue

                    cat_idx = id_map[ann['category_id']]
                    bbox = ann['bbox'] # x, y, w, h

                    # YOLO 포맷 변환 (Normalized Center-X, Center-Y, W, H)
                    x_center = (bbox[0] + bbox[2]/2) / w
                    y_center = (bbox[1] + bbox[3]/2) / h
                    width_norm = bbox[2] / w
                    height_norm = bbox[3] / h

                    # 범위 클리핑 (0~1 사이로 안전장치)
                    x_center = max(0, min(1, x_center))
                    y_center = max(0, min(1, y_center))
                    width_norm = max(0, min(1, width_norm))
                    height_norm = max(0, min(1, height_norm))

                    f_out.write(f"{cat_idx} {x_center:.6f} {y_center:.6f} {width_norm:.6f} {height_norm:.6f}\n")

    process_data(train_imgs, 'train')
    process_data(val_imgs, 'val')

    # 6. data.yaml 생성 (가장 중요!)
    yaml_content = {
        'path': BASE_DIR,
        'train': 'images/train',
        'val': 'images/val',
        'test': 'images/val', # 테스트는 임시로 val 사용
        'names': class_names
    }

    yaml_path = os.path.join(BASE_DIR, 'data.yaml')
    with open(yaml_path, 'w') as f:
        yaml.dump(yaml_content, f, allow_unicode=True)
```

[코드 캡처] yolo_data_config.yaml: names(0~55)

클래스 이름을 YOLO 학습 / 추론에 연결

```
names:
  0: 보령부스파정 5mg
  1: 뮤테란캡슐 100mg
  2: 일양하이트린정 2mg
  3: 기넥신에프정(은행엽엑스)(수출용)
  4: 무코스타정(레바미피드)(비매품)
  5: 알드린정
  6: 뉴로메드정(옥시라세탐)
  7: 에어탈정(아세클로페낙)
  8: 리렉스펜정 300mg/PTP
  9: 아빌리파이정 10mg
  10: 다보타민큐정 10mg/병
  11: 써스펜8시간이알서방정 650mg
  12: 에빅사정(메만틴염산염)(비매품)
  13: 리피토정 20mg
  14: 크레스토정 20mg
  15: 가바토파정 100mg
  16: 동아가바펜틴정 800mg
  17: 오마코연질캡슐(오메가-3-산에틸에스테르90)
  18: 리리카캡슐 150mg
  19: 종근당글리아티린연질캡슐(콜린알포세레이트)
  20: 콜리네이트연질캡슐 400mg
  21: 트루비타정 60mg/병
  22: 스토가정 10mg
  23: 노바스크정 5mg
```

```
  40: 트라젠타정(리나글립틴)
  41: 비모보정 500/20mg
  42: 레일라정
  43: 리바로정 4mg
  44: 트라젠타듀오정 2.5/850mg
  45: 아질렉트정(라사길린메실산염)
  46: 자누메트엑스알서방정 100/1000mg
  47: 글리아타민연질캡슐
  48: 신바로정
  49: 에스원엠프정 20mg
  50: 글리틴정(콜린알포세레이트)
  51: 제미메트서방정 50/1000mg
  52: 아토젯정 10/40mg
  53: 로수젯정10/5밀리그램
  54: 로수바미브정 10/20mg
  55: 카발린캡슐 25mg
path: /content/datasets/pill_data
test: images/val
train: images/train
val: images/val
```

[코드 캡처] YOLOv8 학습 실행

imgsz/batch/patience 등으로 빠른 튜닝 사이클 확보

```
model = YOLO('yolov8x.pt')

# 2. 학습 시작
# data: 아까 만든 yaml 파일 경로
# imgsz: 640 (Colab T4에서 X모델 돌리려면 640이 안전)
# epochs: 25 (충분한 학습량)
# batch: 4 (메모리 터짐 방지)
results = model.train(
    data='./datasets/pill_data/data.yaml',
    epochs=25,
    imgsz=640,
    batch=4,
    name='yolov8_pill_train',
    project='./yolo_output',
    exist_ok=True,      # 덮어쓰기 허용
    patience=5,         # 5번 동안 성능 안 오르면 조기 종료 (시간 절약)
    seed=42,            # 재현성 고정
    verbose=True
)

print("\n" + "="*60)
print("□ YOLO 학습 완료!")
print(f"□ 결과 저장 경로: ./yolo_output/yolov8_pill_train")
print("="*60)
```

[코드 캡처] WBF: augment False/True 앙상블

두 예측을 정규화 후 weighted_boxes_fusion 으로 결합

```
        # image_id 추출
        file_name = os.path.basename(img_path)
        numbers = re.findall(r'\d+', file_name)
        if not numbers:
            continue
        img_id = int(''.join(numbers))

        # 두 가지 예측
        pred1 = model.predict(img_path, conf=0.01, imgsz=640, augment=False, verbose=False)[0]
        pred2 = model.predict(img_path, conf=0.01, imgsz=640, augment=True, verbose=False)[0]

        # 정규화
        boxes_list = []
        scores_list = []
        labels_list = []

        for pred in [pred1, pred2]:
            if len(pred.boxes) == 0:
                boxes_list.append([])
                scores_list.append([])
                labels_list.append([])
                continue

            boxes = pred.boxes.xyxy.cpu().numpy().copy()
            boxes[:, [0, 2]] /= w_img
            boxes[:, [1, 3]] /= h_img
            boxes = np.clip(boxes, 0, 1)

            boxes_list.append(boxes.tolist())
            scores_list.append(pred.boxes.conf.cpu().numpy().tolist())
            labels_list.append(pred.boxes.cls.cpu().numpy().tolist())

        # WBF
        if any(len(b) > 0 for b in boxes_list):
            try:
                boxes, scores, labels = weighted_boxes_fusion(
                    boxes_list, scores_list, labels_list,
                    weights=[2, 1], iou_thr=0.55, skip_box_thr=0.01
                )
            except:
                continue

            # 역정규화 및 저장
            for box, score, label in zip(boxes, scores, labels):
```

[코드 캡처] 최종 제출 라벨 보정 (+1)

YOLO(0~55) → Kaggle 제출 (1~56) 로 category_id = int(label) + 1

```
            y1 = max(0, min(y1, h_img))
            x2 = max(0, min(x2, w_img))
            y2 = max(0, min(y2, h_img))

            w = x2 - x1
            h = y2 - y1

            if w < 5 or h < 5:
                continue

            # □□□ 핵심: YOLO 0~55 › 캐글 1~56 □□□
            category_id = int(label) + 1

            results.append({
                'annotation_id': annotation_id,
                'image_id': img_id,
                'category_id': category_id,
                'bbox_x': float(x1),
                'bbox_y': float(y1),
                'bbox_w': float(w),
                'bbox_h': float(h),
```

여기까지 (Streamlit 파트는 별도 발표)