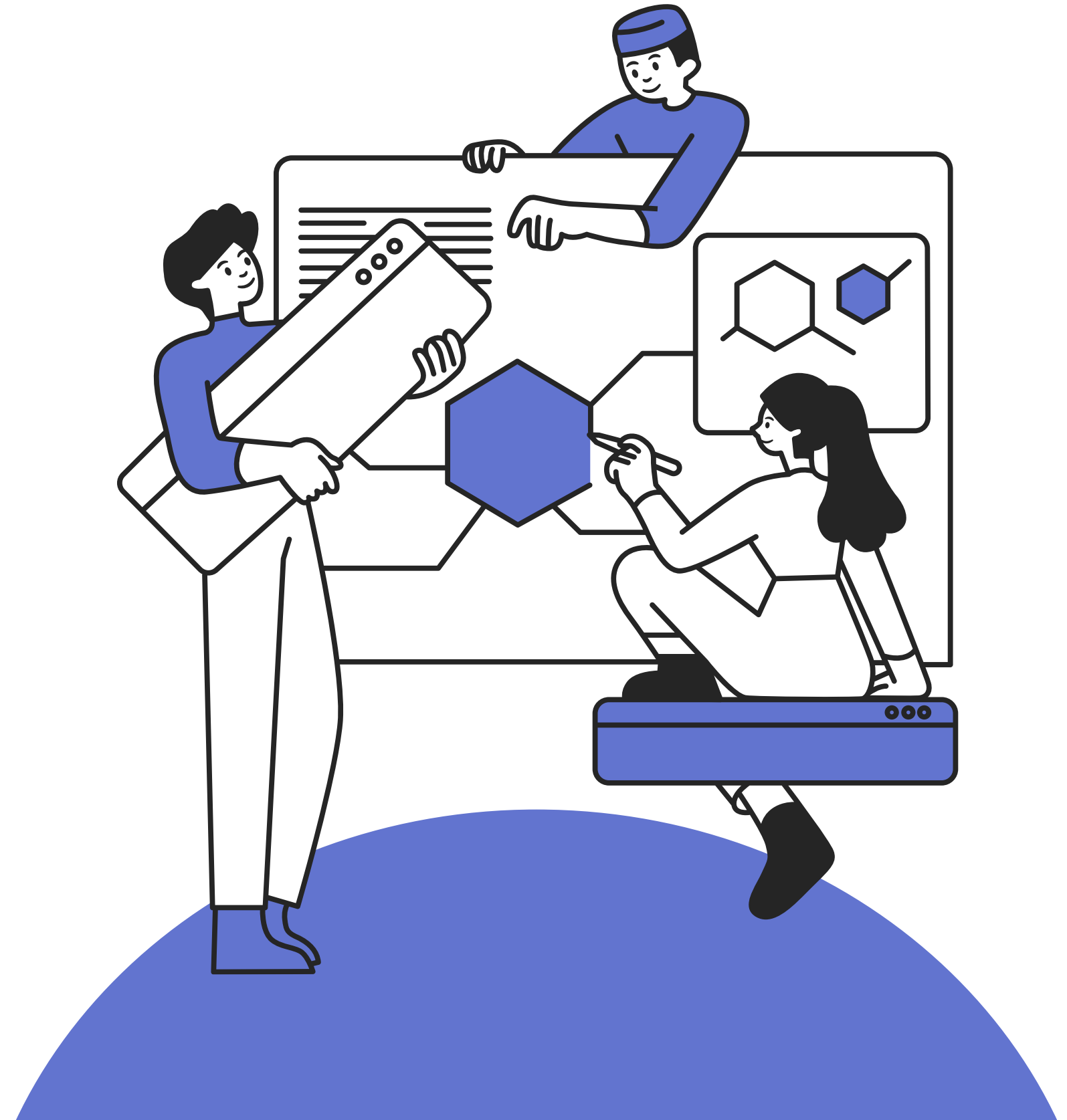


4 Way

교차로 차량·보행자 모니터링을 통한 혼잡도 및 안전도 분석 시스템

이보라 권 봄 임다연 강은비



CONTENTS

- 1 팀 구성
- 2 프로젝트 개요
- 3 목표 및 기대효과
- 4 주요 기능
- 5 프로젝트 진행 과정
- 6 모델 데이터 학습 과정
- 7 프로젝트 구현
- 8 추후 프로젝트 진행 방향 및 목표
- 9 자체 평가 의견
- 10 Q&A

01

팀 구성



이보라
팀장



권 봄
서기



임다연
팀원



강은비
팀원

02

프로젝트 개요

프로젝트 주제

교차로 차량·보행자 모니터링 및 이상 알림 시스템

교차로 내 차량과 보행자의 움직임을 실시간으로 추적하고,
이상 상황을 감지하여 교차로의 혼잡도와 안전도를 분석하는
시스템입니다.

교통사고를 예방하고, 교차로의 효율적인 관리 및 안전한 환
경 조성을 목표로 하며, 실시간 알림을 통해 교차로 관리자와
관련 당국이 신속하게 대응할 수 있도록 돕습니다.



02

프로젝트 개요

기술 스택

DATA 공공데이터포털
.GO.KR

데이터 수집용 CCTV

@roboflow

데이터 라벨링

데이터셋 분할

전처리, 증강



모델 학습



파이썬 3.9

영상 처리, 모델 추론, 웹 UI 구성



모델 학습

출력 화면



Streamlit

서버 구축

03

목표 및 기대 효과

혼잡도 완화



교차로의 교통 흐름 개선

사고 예방



보행자와 운전자의 안전 ↑↑

실시간 데이터 제공



교차로의 교통 상황 효율적 관리

04

주요 기능



차량 및 보행자
모니터링



안전거리 분석



혼잡도 분석



알림 시스템

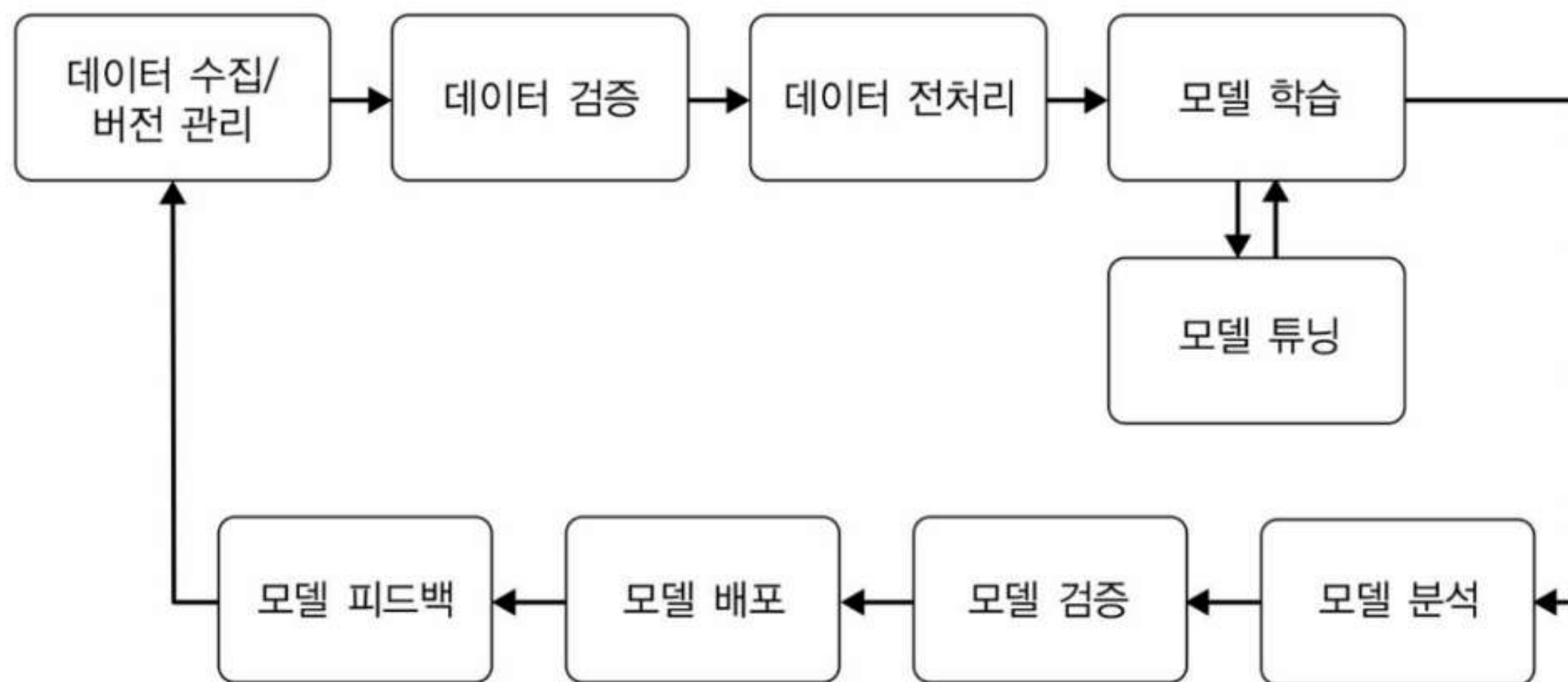
05

프로젝트 진행 과정



06

모델 데이터 학습 과정



06

모델 데이터 학습 과정

5. 데이터 수집

충청남도 천안시_교통정보 CCTV_20220922

- 위 링크에서 HTTP 링크를 입수하여 5-3 코드 돌리기

- 세집매 삼거리
- 서부역 입구 삼거리
- 역말 오거리
- 천안로삼거리
- 삼명대 입구 삼거리
- 방죽안오거리
- 천안역
- 남부오거리
- 교보삼거리
- 청삼교차로
- 신방삼거리
- 아마트 사거리
- 쌍용삼거리
- 봉서삼거리
- 구상골 사거리
- 인쇄창삼거리
- 부석초교삼거리
- 극동삼거리
- 백석삼거리

운동장 사거리

쌍용도서관 삼거리

동일하이빌 삼거리

고속철 입구

개목삼거리

우마란A 삼거리

세무서

천수 국민은행

생태터널삼거리

원천육교

도록원점 삼거리

직삼삼거리

저리-교차로

성성교차로

도장리

예술의전당

대흥교차로

단대입구 사거리

구정삼거리

오룡지하차도

와촌초교

불당산도사 입구 삼거리

푸르지오 106동 사거리

우마란 102동 사거리

시청삼거리

후반201동 사거리

불당교

불당현대삼거리

고속화도로입구삼거리

새말삼거리

```
1 import cv2
2 import os
3 import time
4 from datetime import datetime
5 import schedule # => pip install schedule
6
7 # 1. 이미지 수집 간격 설정
8 interval_time = 10
9
10 # 2. 저장 디렉토리 설정
11 SAVE_DIR = "capture_images"
12 os.makedirs(SAVE_DIR, exist_ok=True)
13
14 # 3. 데이터 수집 함수
15 def capture_image():
16     # 3-1. 카메라 불러오기
17     cap = cv2.VideoCapture("http://210.99.70.120:1935/live/cctv007.stream/playlist.m3u8")
18     # 3-2. 카메라 확인
19     if not cap.isOpened():
20         raise RuntimeError("카메라 확인해주세요~")
21     print("카메라 연결 완료")
22
23     # 3-3. 카메라 프레임 읽기
24     success, frame = cap.read()
25     if success:
26         timestamp = datetime.now().strftime("%Y%m%d_%H_%M_%S")
27         file_path = os.path.join(SAVE_DIR, f"get_{timestamp}.jpg")
28
29         # 3-4. 수집 이미지 저장
30         cv2.imwrite(file_path, frame)
31         print(f"사진이 저장됐습니다. {file_path}")
32         print(f"{interval_time}초 간격으로 이미지를 수집합니다.")
33
34     else:
35         print("프레임을 못 읽었습니다.")
36
37     # 3-5. 카메라 해제
38     cap.release()
39     cv2.destroyAllWindows()
40
41 #schedule 실행 설정
42 schedule.every(interval_time).seconds.do(capture_image)
43
44 while True:
45     schedule.run_pending()
46     time.sleep(1)
47
```

모델 데이터 학습 과정

약 6000장 수집

더불어






06

모델 데이터 학습 과정

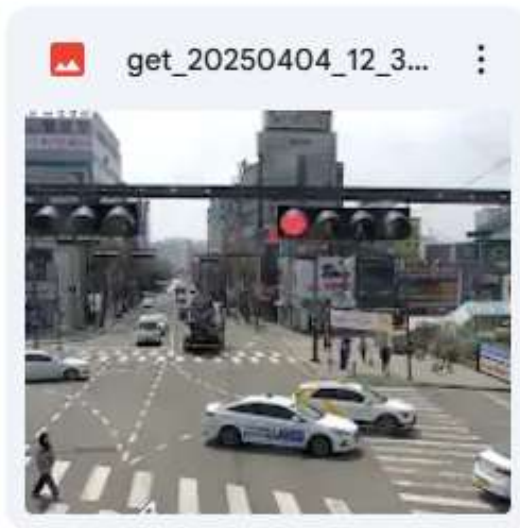
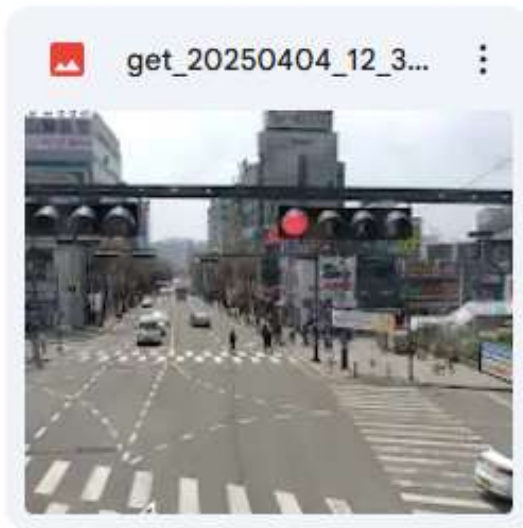
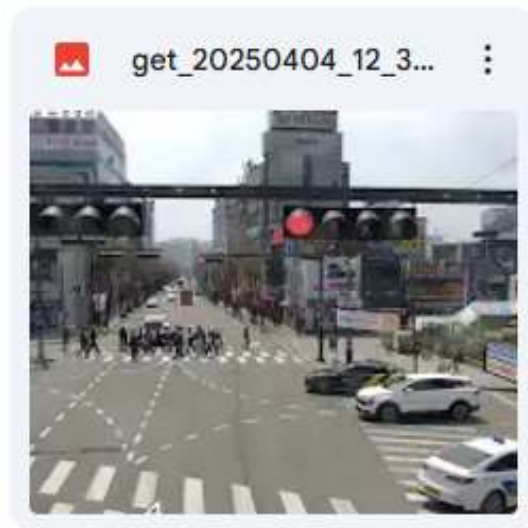
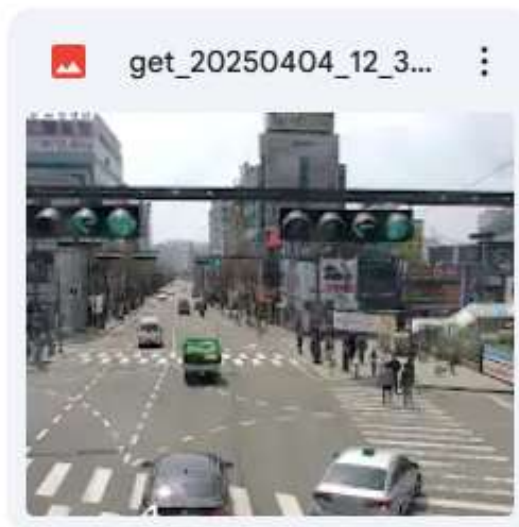
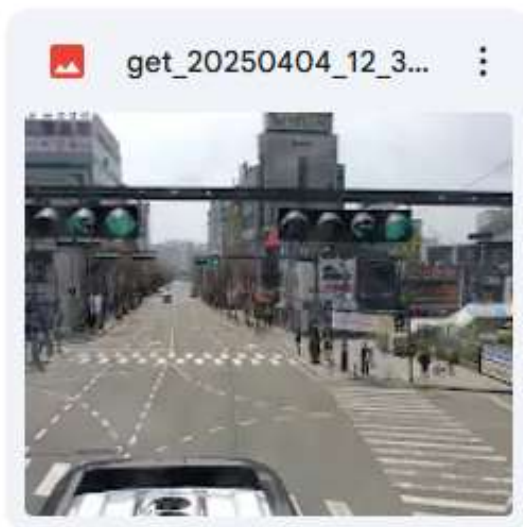
공유 문서함 > AI_4WAY > capture_images > capture_images_CCTV0... ▾

☰ ✓ 96 ⓘ

유형 ▾ 사람 ▾ 수정 날짜 ▾ 출처 ▾

파일

↑ 이름 ▾ ⋮



06

모델 데이터 학습 과정 - roboflow



KB_Folder: capture_images - Auto Label [🔗](#)

Return For Edits

Add Approved to Dataset

✕

Progress

1000 Images

● 627 Approved

● 21 Rejected

○ 652 Annotated

○ 0 Unannotated

Instructions

[🔗 Edit](#)

Automatic labeling job

Assignment

[🔗 Reassign](#)

AUTOLABEL
Labeler

Timeline

?

\$Job created via API and assigned it to AUTOLABEL.


2025. 4. 7. 오전 10:06:23

To Do 652

Approved 327


Rejected 21

Sort By Newest ▾




TRAIN

get_202504&d_&H25.j...




TRAIN

get_20250404_12_28...




TRAIN

get_20250404_12_30...




TRAIN

get_20250404_12_33...




TRAIN

get_20250404_12_34...




TRAIN

get_20250404_12_36...




TRAIN

get_20250404_12_36...




TRAIN

get_20250404_12_36...




TRAIN

get_20250404_12_371...




TRAIN

get_20250404_12_37...




TRAIN

get_20250404_12_41...




TRAIN

get_20250404_12_43...




TRAIN

get_20250404_12_46...




TRAIN

get_20250404_12_44...




TRAIN

get_20250404_12_45...




TRAIN

get_20250404_12_46...




TRAIN

get_20250404_12_48...




TRAIN

get_20250404_12_52...




TRAIN

get_20250404_12_49...




TRAIN

get_20250404_12_50...




TRAIN

get_20250404_12_51...




TRAIN

get_20250404_12_50...



TRAIN

get_20250404_12_54...



TRAIN

get_20250404_12_54...

06

모델 데이터 학습 과정

데이터셋
버전1 = 1727장
버전2 = 2914장
버전3 = 5221장

Dataset

7 Jobs

[See all 5221 images](#)

Folder: capture_images: Job 2

Labeler: 이보라

487 Images

EB_Folder: capture_images - Auto Label

Labeler: ⚡ Automatic Labeling

861 Images

EB_Folder: capture_images - Auto Label

Labeler: ⚡ Automatic Labeling

866 Images

EB_Folder: capture_images - Auto Label

Labeler: ⚡ Automatic Labeling

713 Images

Dataset

[How to Search](#)

[+ New Dataset Version](#) [Train Model](#)

Search images

Filter by filename Split Classes Sort By Newest Search by Image

☐ 0 images selected

The image grid displays 36 images arranged in a 4x9 layout. Each image is a dark, low-light scene, likely from a video game, showing various objects and characters. The images are labeled with bounding boxes and text. The labels are in Korean and include names like 'get_20250404_12_3...' and 'get_20250404_12_4...'. Some images have a blue circle with a white 'x' icon, while others have a yellow circle with a white 'x' icon. The grid is titled '0 images selected' and has a search bar at the top.

06

모델 데이터 학습 과정

Versions

2025-04-16 5:44pm

v1 12458

This version doesn't have a model.

Train an optimized, state of the art model with Roboflow or upload a custom trained model to use features like Label Assist and Model Evaluation and deployment options like our auto-scaling API and edge device support.


Custom Train

Uses Roboflow CreditsView usage ↗

How to Upload Custom Weights

12458 Total Images

View All Images →



Dataset Split

TRAIN SET

10892 Images

87%

VALID SET

1044 Images

8%

TEST SET

522 Images

4%

06

모델 데이터 학습 과정 - 버전1-1(데이터셋1, 10회)



The image shows a VS Code editor window with a Python script named `version_test.py` and its terminal output. The script imports `YOLO` from `ultralytics`, loads a model, and trains it for 10 epochs using a specific data path. The terminal output shows the training process, including the automatic selection of an optimizer and the start of training for 100 epochs.

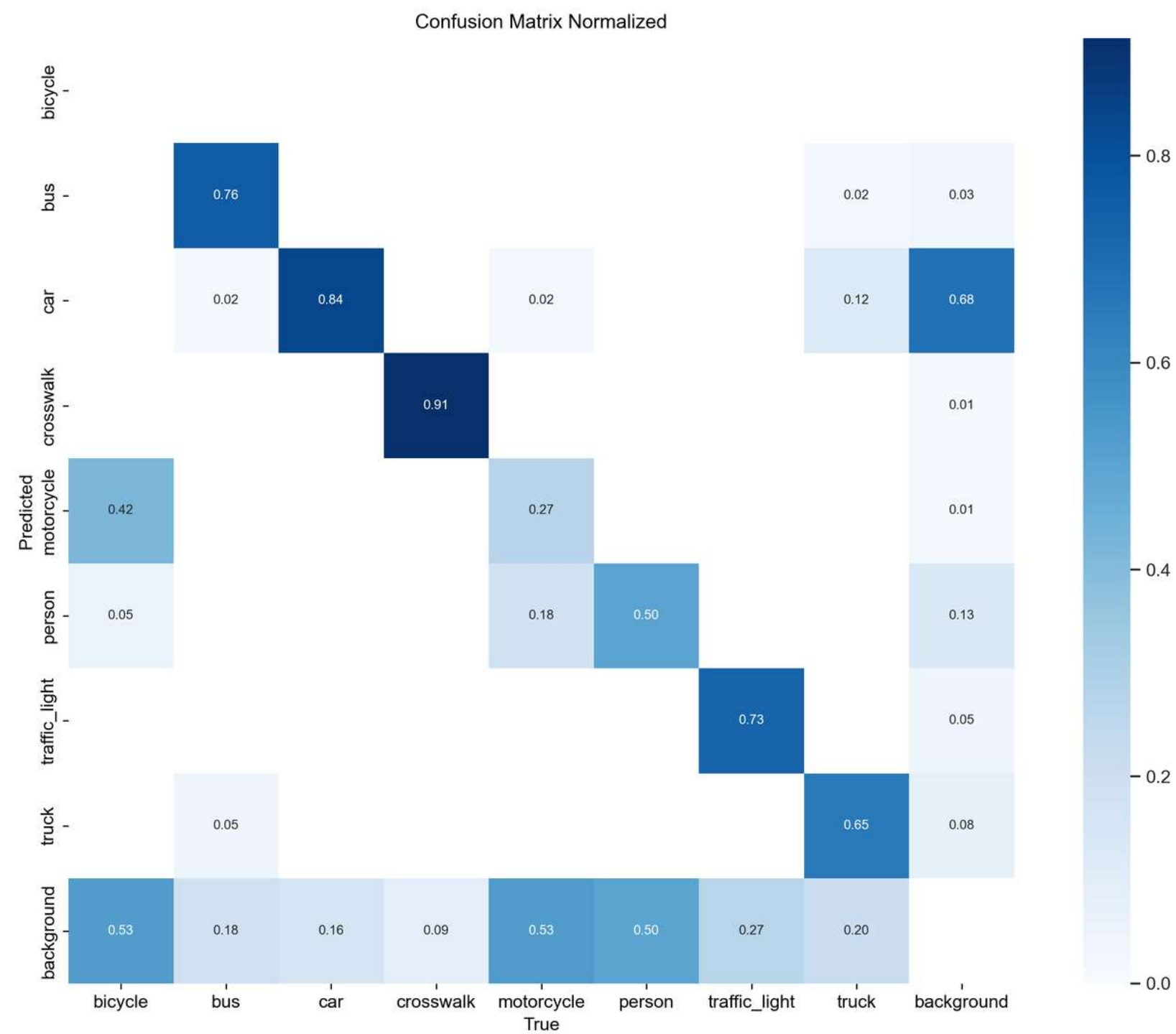
```
version_test.py • v1_class.py requirements.txt
wtcd > team_project > version_test.py > ...
1  from ultralytics import YOLO
2
3
4  # 1. YOLO 모델 로드
5  model = YOLO("yolo11n.pt")
6
7  # 2. 모델 훈련
8  model.train(
9      epochs=10,
10     data="C:/Users/Administrator/Desktop/ai/wtcd/team_project/4way_v1i_yolov11/data.yaml"
11 )
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS Python Debug Console

Plotting labels to runs\detect\train4\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.000833, momentum=0.9) with parameter groups 81 weight(decay=0.0), 88 weight(decay=0.0005), 87 bias(decay=0.0)
TensorBoard: model graph visualization added 
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\train4
Starting training for 100 epochs...

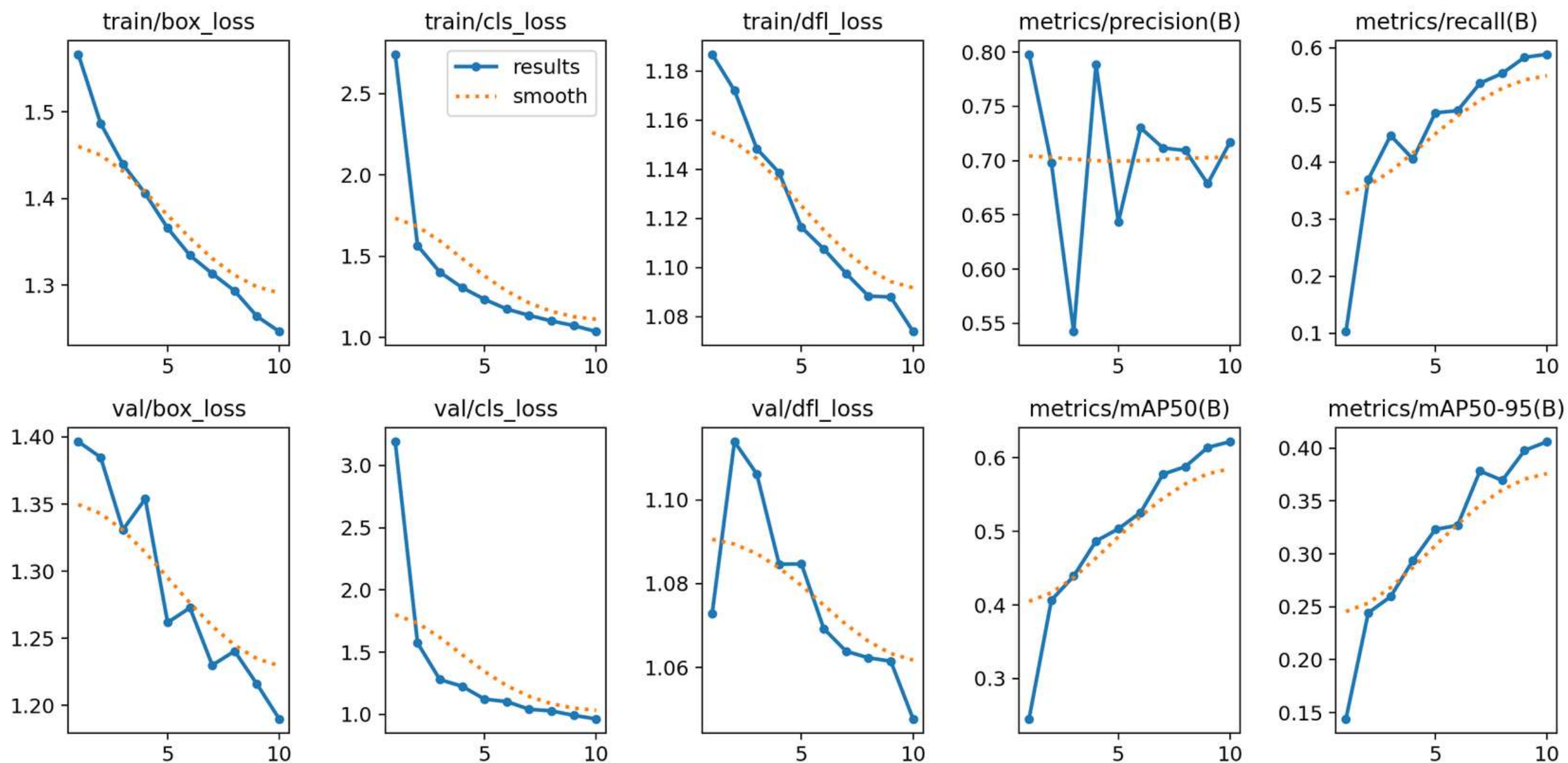
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/100	0G	1.599	2.943	1.181	234	640: 100% ██████████ 76/76 [07:35<00:00, 5.99s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 11/11 [00:38<00:00, 3.49s/it]
	all	345	4248	0.912	0.105	0.215 0.133

모델 데이터 학습 과정 - 버전1-1(데이터셋1, 10회) 컴퓨전



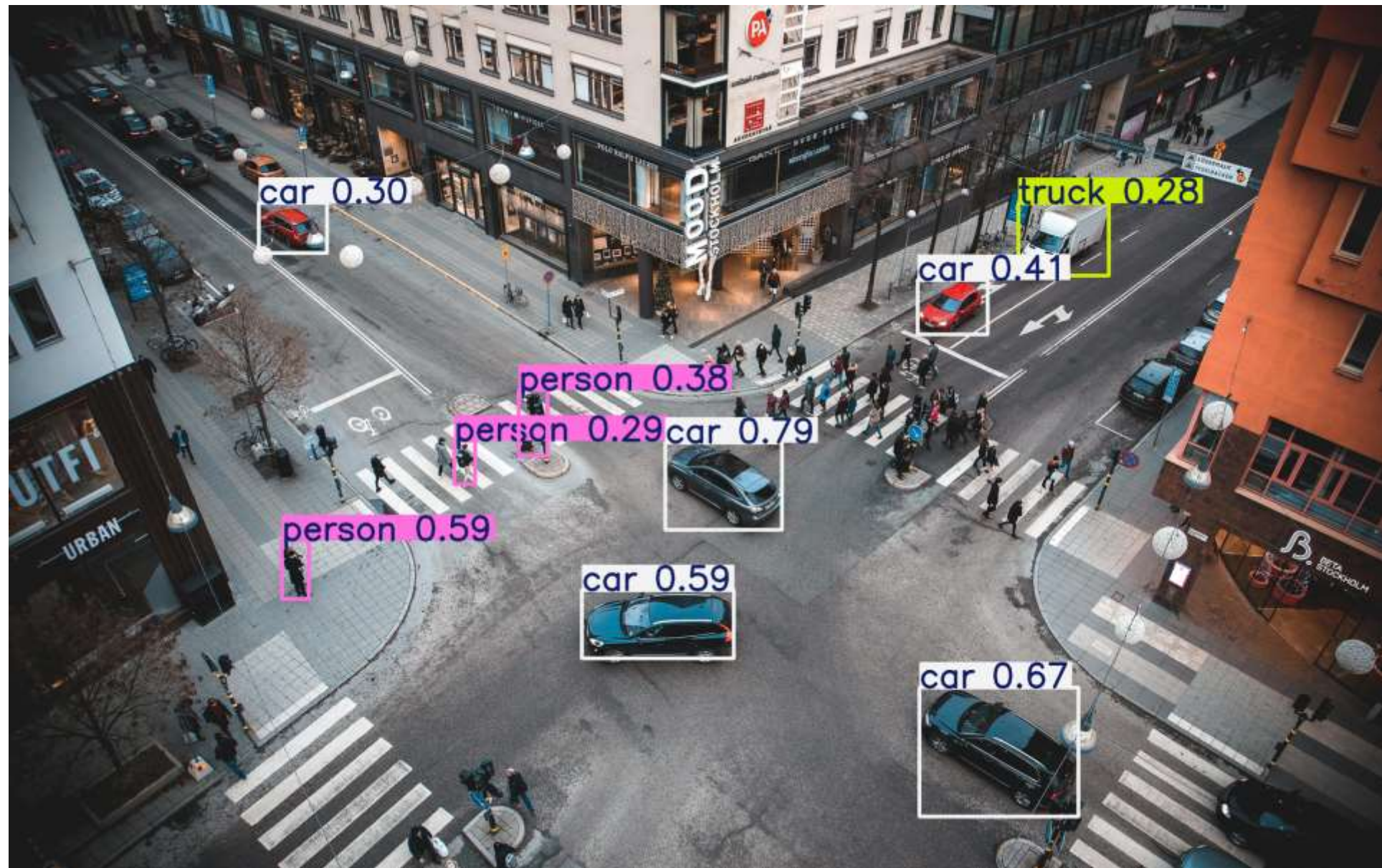
06

모델 데이터 학습 과정 - 버전1-1(데이터셋1, 10회)결과



06

모델 데이터 학습 과정 - 버전1-1(데이터셋1, 10회) 결과



06

모델 데이터 학습 과정 - 버전1 - 2(데이터셋1, 100회)



The image shows a VS Code editor window with a Python script named `version_test.py` and its terminal output. The script imports `YOLO` from `ultralytics`, loads a YOLO11n model, and trains it for 100 epochs using a specific data path. The terminal output shows the training process, including the automatic selection of the AdamW optimizer and the start of training for 100 epochs. A table at the bottom displays the training progress for the first epoch.

```
version_test.py • v1_class.py requirements.txt
wtcd > team_project > version_test.py > ...
1 from ultralytics import YOLO
2
3
4 # 1. YOLO 모델 로드
5 model = YOLO("yolo11n.pt")
6
7 # 2. 모델 훈련
8 model.train(
9     epochs=100,
10     data="C:/Users/Administrator/Desktop/ai/wtcd/team_project/4way_v1i_yolov11/data.yaml"
11 )
```

Plotting labels to runs\detect\train4\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.000833, momentum=0.9) with parameter groups 81 weight(decay=0.0), 88 weight(decay=0.0005), 87 bias(decay=0.0)
TensorBoard: model graph visualization added 
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\train4
Starting training for 100 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/100	0G	1.599	2.943	1.181	234	640: 100% ██████████ 76/76 [07:35<00:00, 5.99s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 11/11 [00:38<00:00, 3.49s/it]
	all	345	4248	0.912	0.105	0.215 0.133

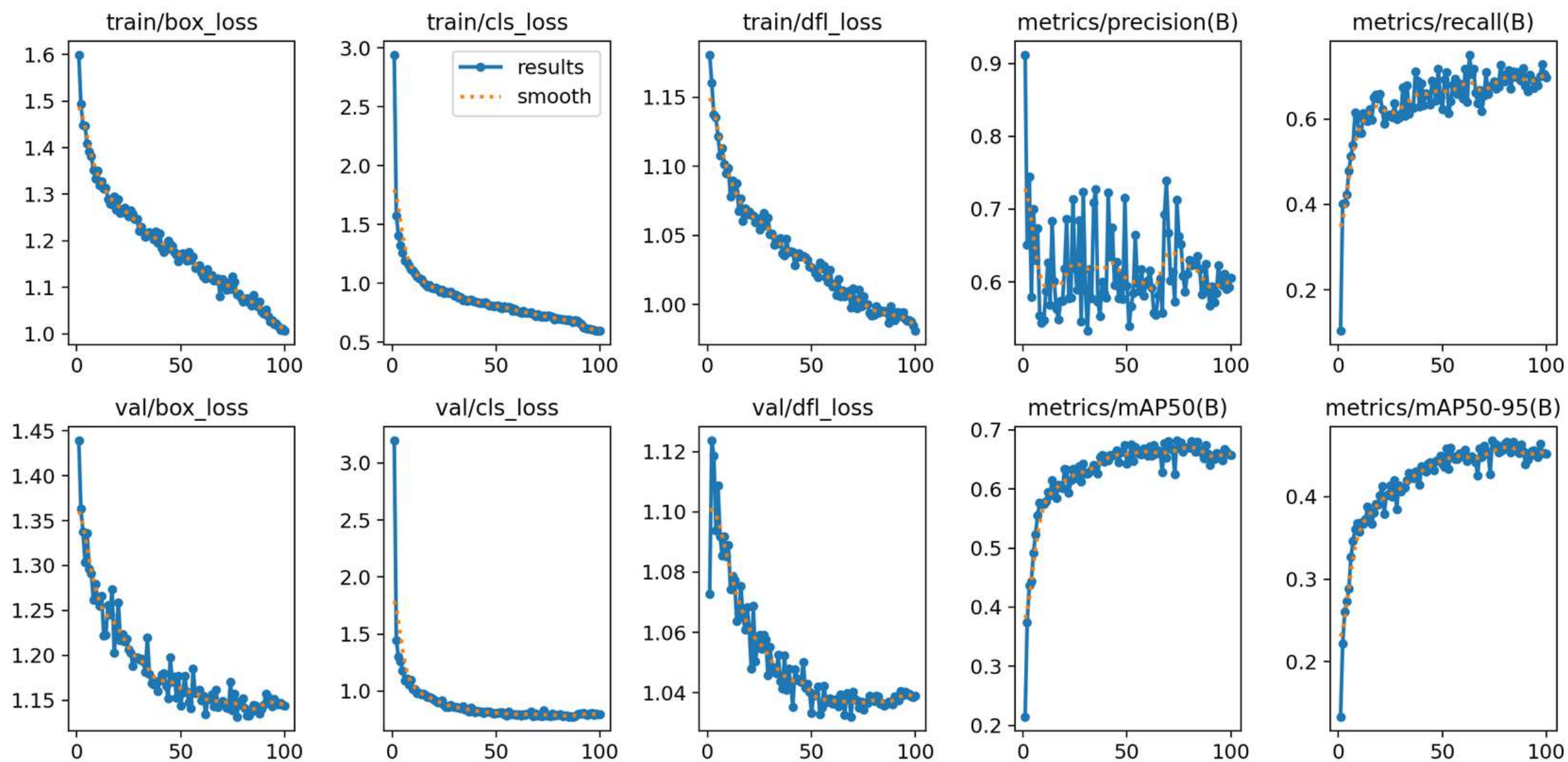
06

모델 데이터 학습 과정 - 버전1-2(데이터셋1, 100회) 컴퓨전



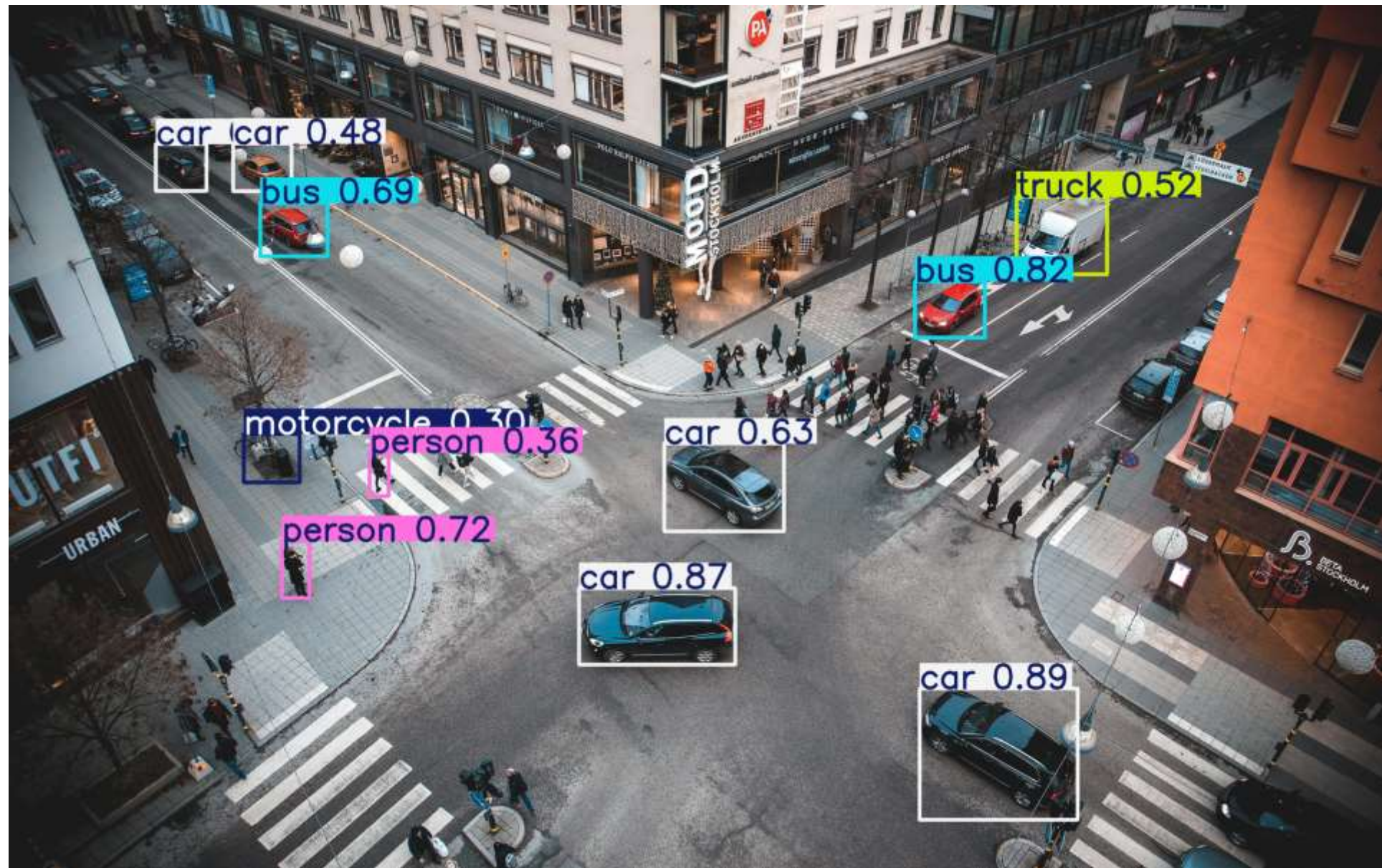
06

모델 데이터 학습 과정 - 버전1-2(데이터셋1, 100회) 결과



06

모델 데이터 학습 과정 - 버전1-2(데이터셋1, 100회) 결과



06

모델 데이터 학습 과정 - 버전2(데이터셋2, 300회)

```
# 모델 훈련
from ultralytics import YOLO

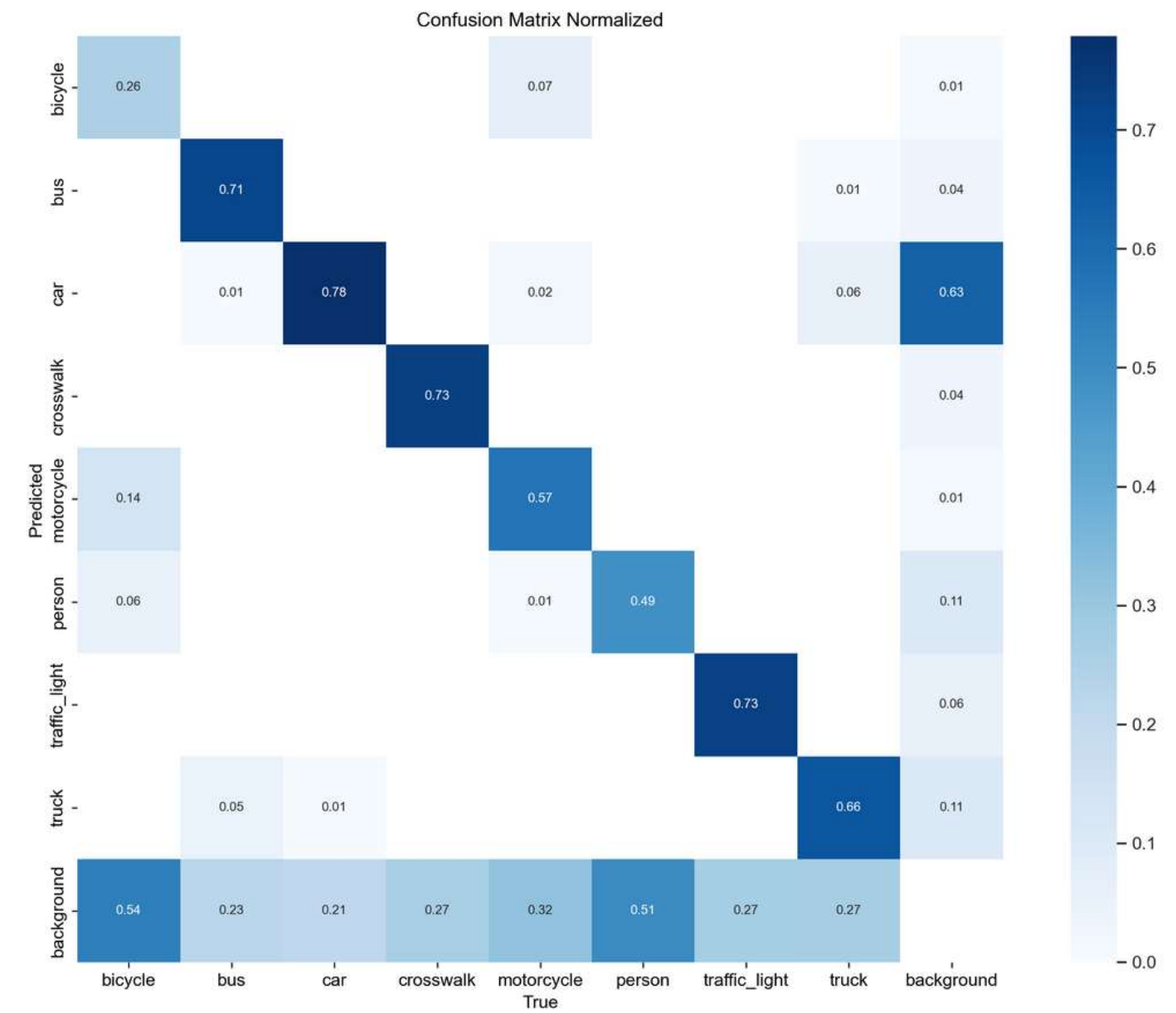
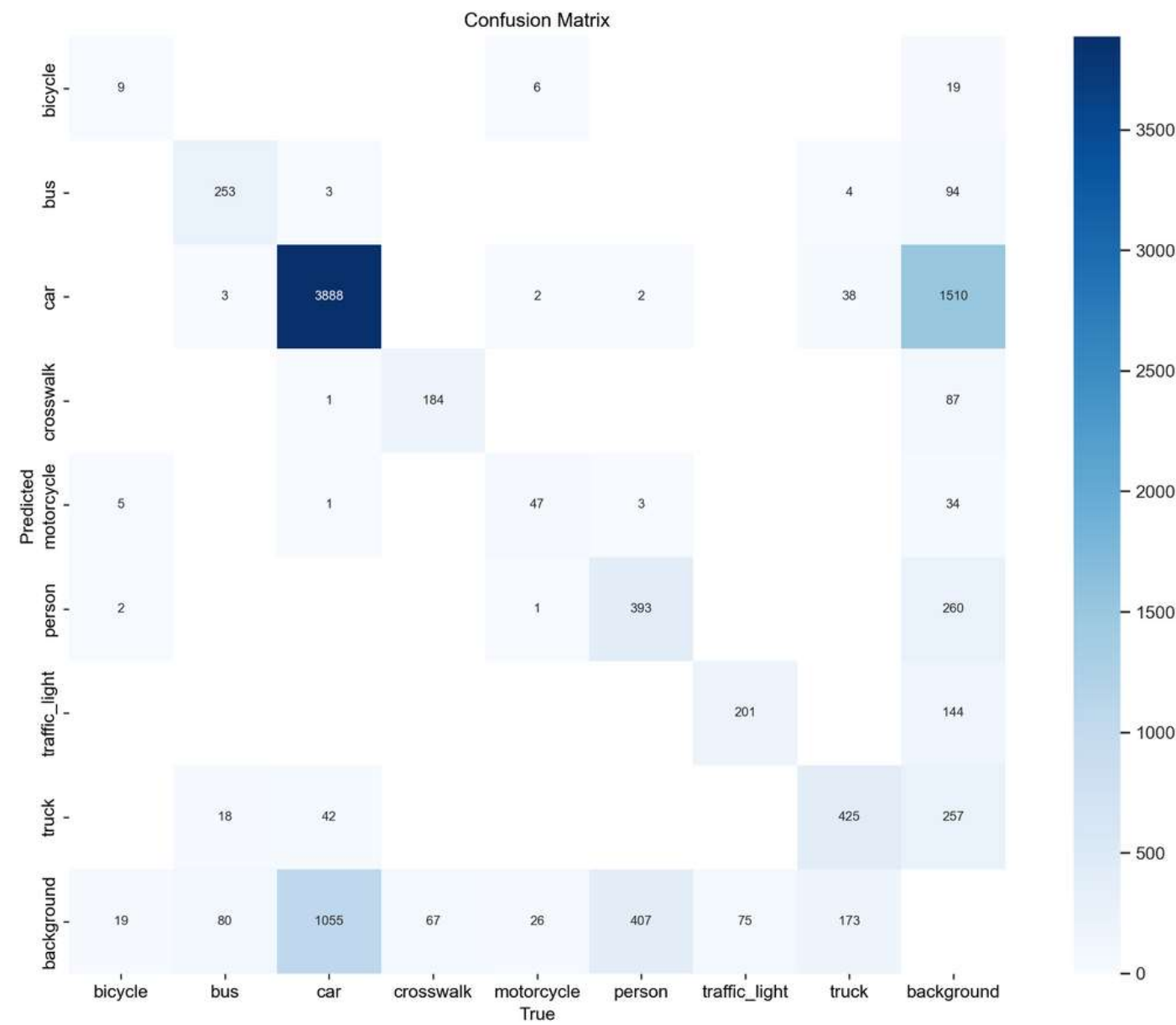
# 1. YOLO 모델 로드
model = YOLO("/content/yolo11n.pt")

# 2. 모델 훈련
model.train(
    data="/content/data.yaml",
    epochs=300,
    project="/content/drive/MyDrive/yolo_project",
    name="yolo_train"
)
```

```
Ultralytics 8.3.109 Python-3.11.12 torch-2.6.0+cu124 CPU (Intel Xeon 2.20GHz)
engine/trainer: task=detect, mode=train, model=/content/yolo11n.pt, data=/content/data.yaml, epochs=300, time=None, patience=100,
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100%|██████████| 755k/755k [00:00<00:00, 14.7MB/s]
Overriding model.yaml nc=80 with nc=8
```

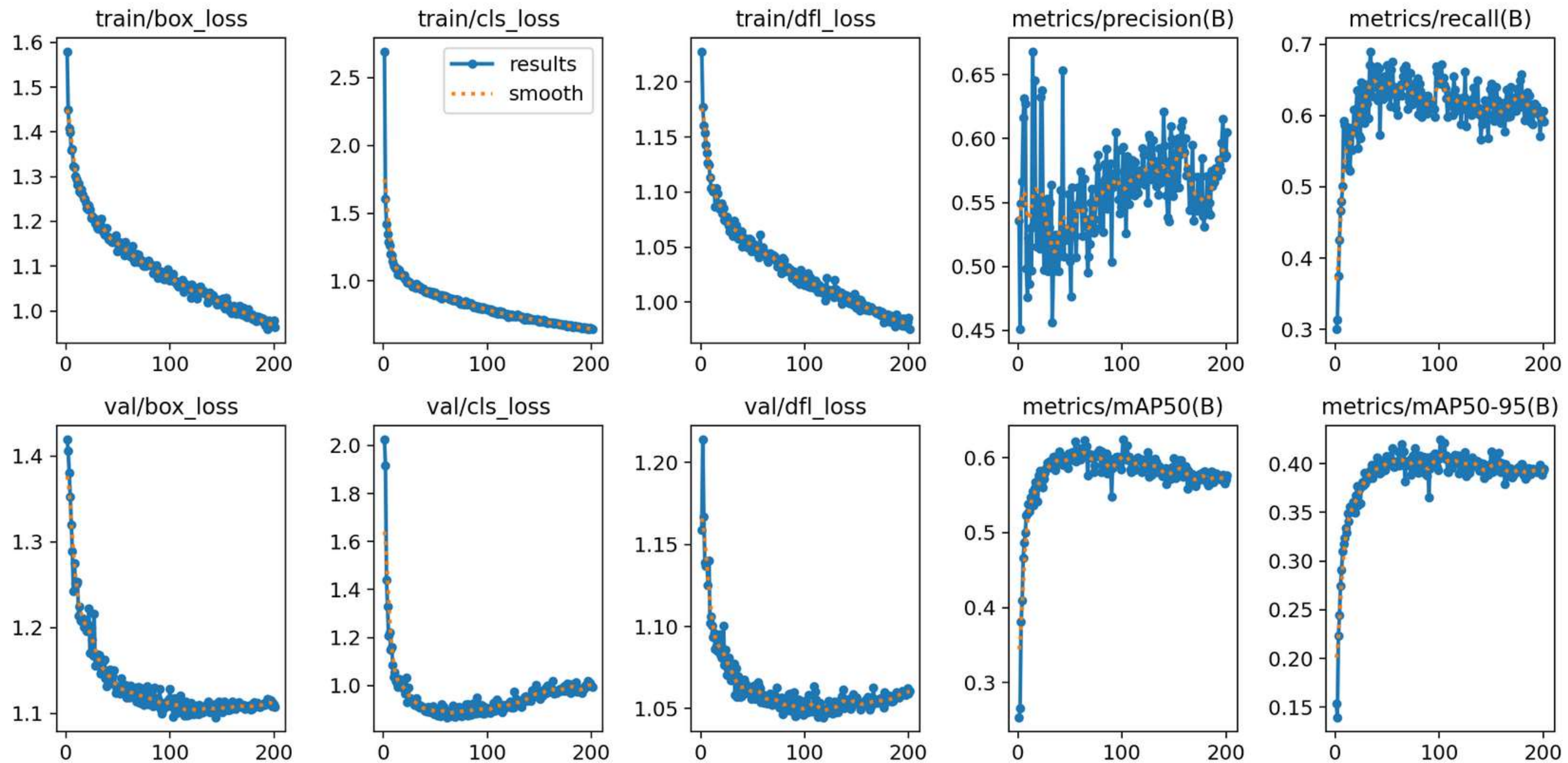
06

모델 데이터 학습 과정 - 버전2(데이터셋2, 300회) 컴퓨전



06

모델 데이터 학습 과정 - 버전2(데이터셋2, 300회) 결과



06

모델 데이터 학습 과정 - 버전2(데이터셋2, 300회) 결과



06

모델 데이터 학습 과정 - 버전3(데이터셋3, 150회)

```
[ ] # 모델 훈련
from ultralytics import YOLO

# 1. YOLO 모델 로드
model = YOLO("/content/yolo11n.pt")

# 2. 모델 훈련
model.train(
    data="/content/data.yaml",
    epochs=150,
    project="/content/drive/MyDrive/yolo_project",
    name="yolo_train"
)
```

🔄 Ultralytics 8.3.109 🚀 Python-3.11.12 torch-2.6.0+cu118
engine/trainer: task=detect, mode=train, model=/content/yolo11n.pt
Downloading <https://ultralytics.com/assets/Arial.ttf>
100%|██████████| 755k/755k [00:00<00:00, 14.7MB/s]
Overriding model.yaml nc=80 with nc=8

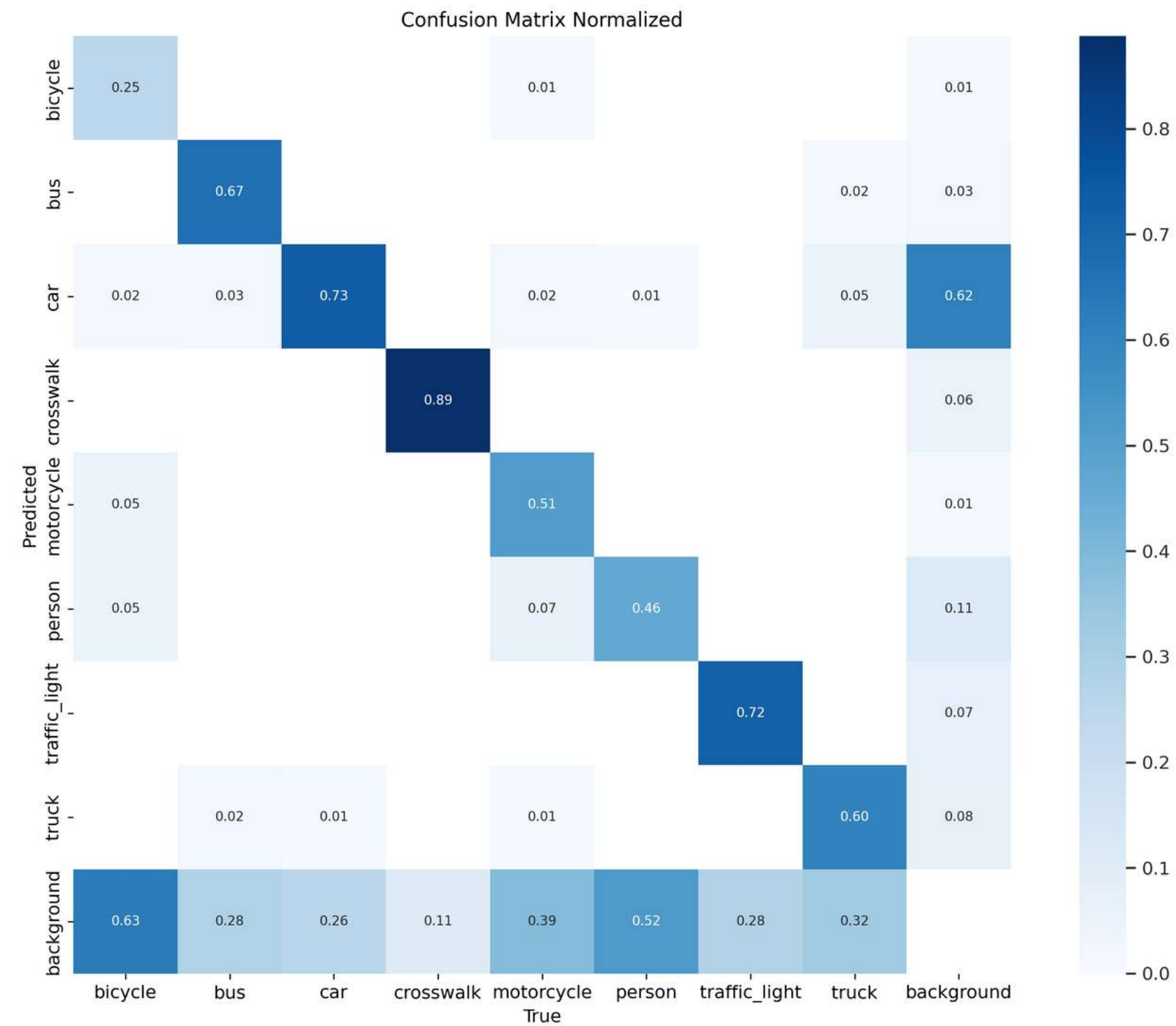
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
147/150	4.41G	0.8012	0.4744	0.9303	194	640: 100% ██████████ 500/500 [02:19<00:00, 3.59it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 24/24 [00:05<00:00, 0.555 0.382]
	all	762	10141	0.565	0.602	
148/150	4.42G	0.802	0.4725	0.9302	267	640: 100% ██████████ 500/500 [02:21<00:00, 3.53it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 24/24 [00:06<00:00, 0.555 0.382]
	all	762	10141	0.565	0.6	
149/150	4.43G	0.7975	0.4701	0.9308	174	640: 100% ██████████ 500/500 [02:20<00:00, 3.56it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 24/24 [00:05<00:00, 0.555 0.382]
	all	762	10141	0.565	0.6	
150/150	4.44G	0.7953	0.4692	0.929	227	640: 100% ██████████ 500/500 [02:23<00:00, 3.50it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 24/24 [00:06<00:00, 0.555 0.382]
	all	762	10141	0.565	0.6	

52 epochs completed in 2.223 hours.
Optimizer stripped from /content/drive/MyDrive/yolo_project/yolo_train5/weights/last.pt, 5.5MB
ultralytics.utils.metrics.DetMetrics object with attributes:

ap_class_index: array([0, 1, 2, 3, 4, 5, 6, 7])
box: ultralytics.utils.metrics.Metric object
confusion_matrix: <ultralytics.utils.metrics.ConfusionMatrix object at 0x7c56499e6f10>
curves: ['Precision-Recall(B)', 'F1-Confidence(B)', 'Precision-Confidence(B)', 'Recall-Confidence(B)']
curves_results: [[array([0.001001, 0.002002, 0.003003, 0.004004, 0.005005, 0.006006, 0.007007,

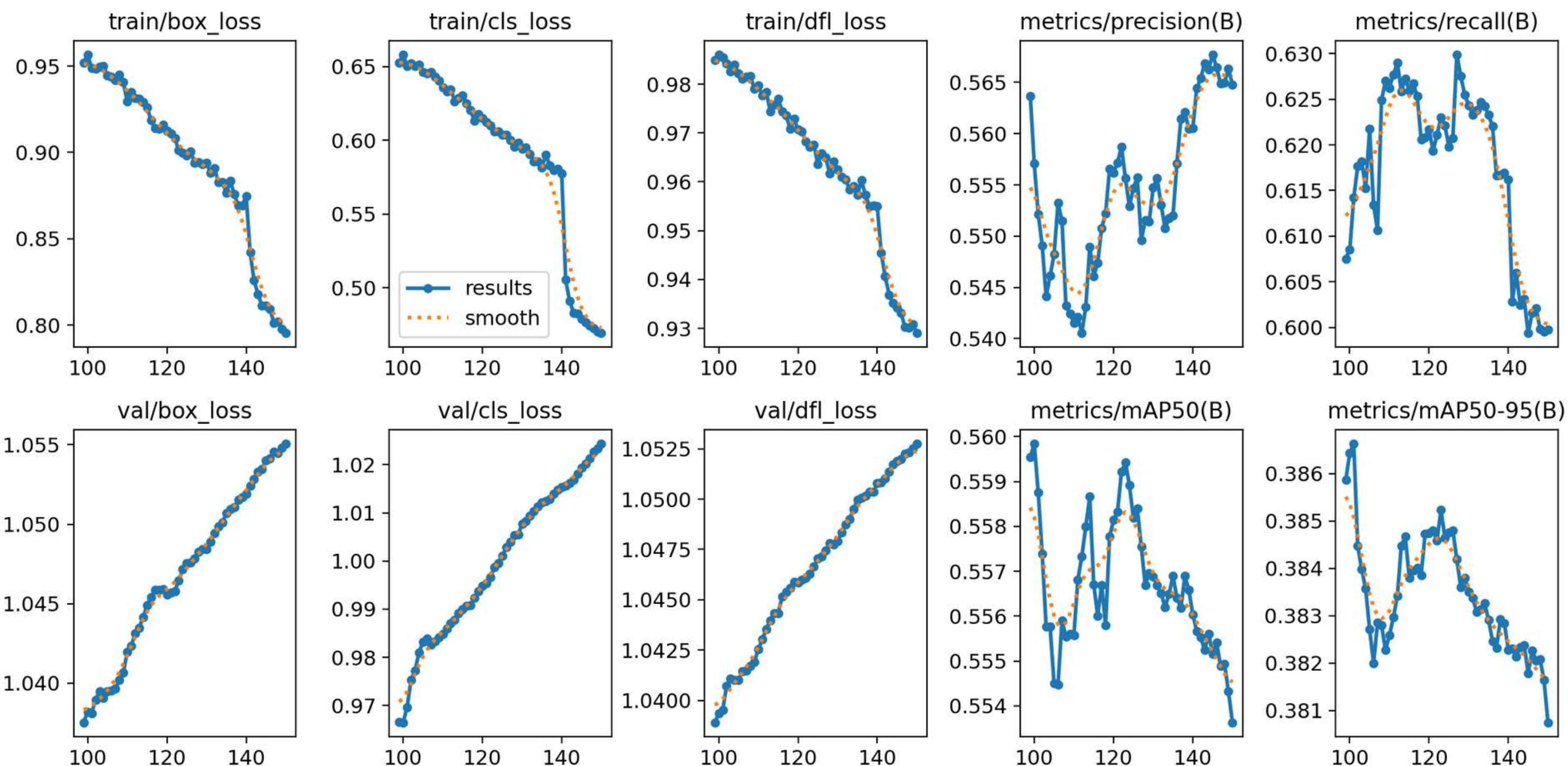
06

모델 데이터 학습 과정 - 버전3(데이터셋3, 150회) 컴퓨전



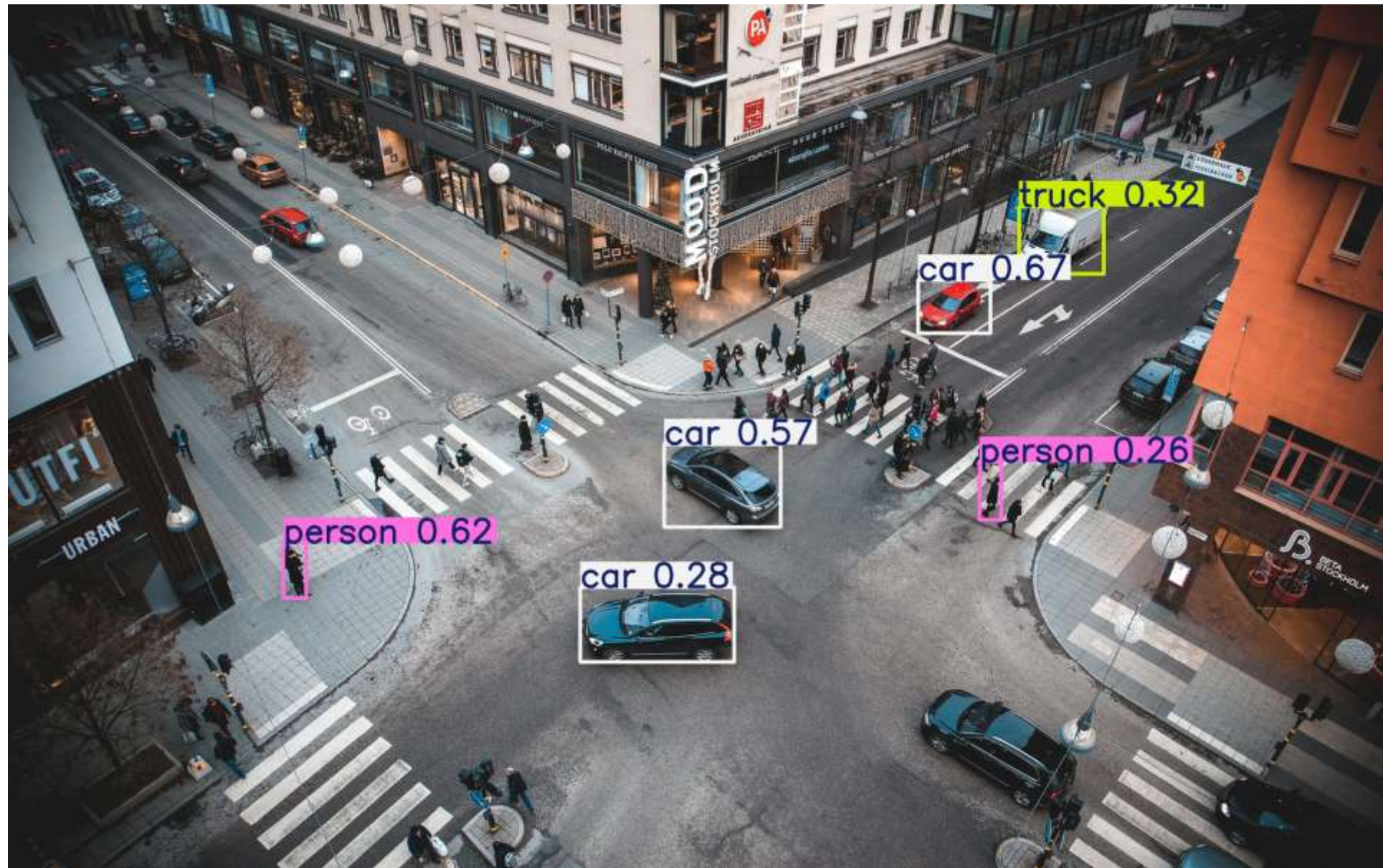
06

모델 데이터 학습 과정 - 버전3(데이터셋3, 150회) 결과



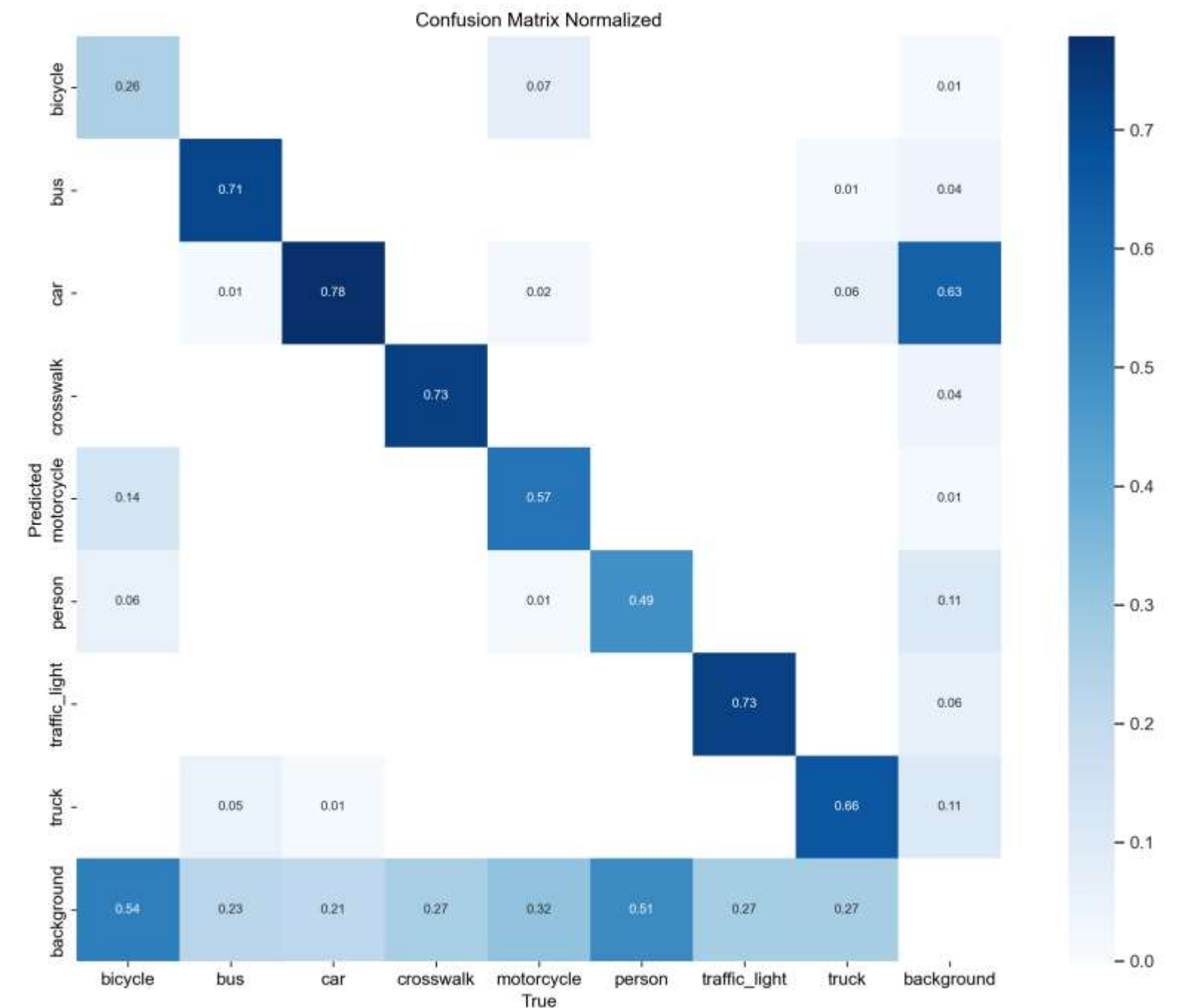
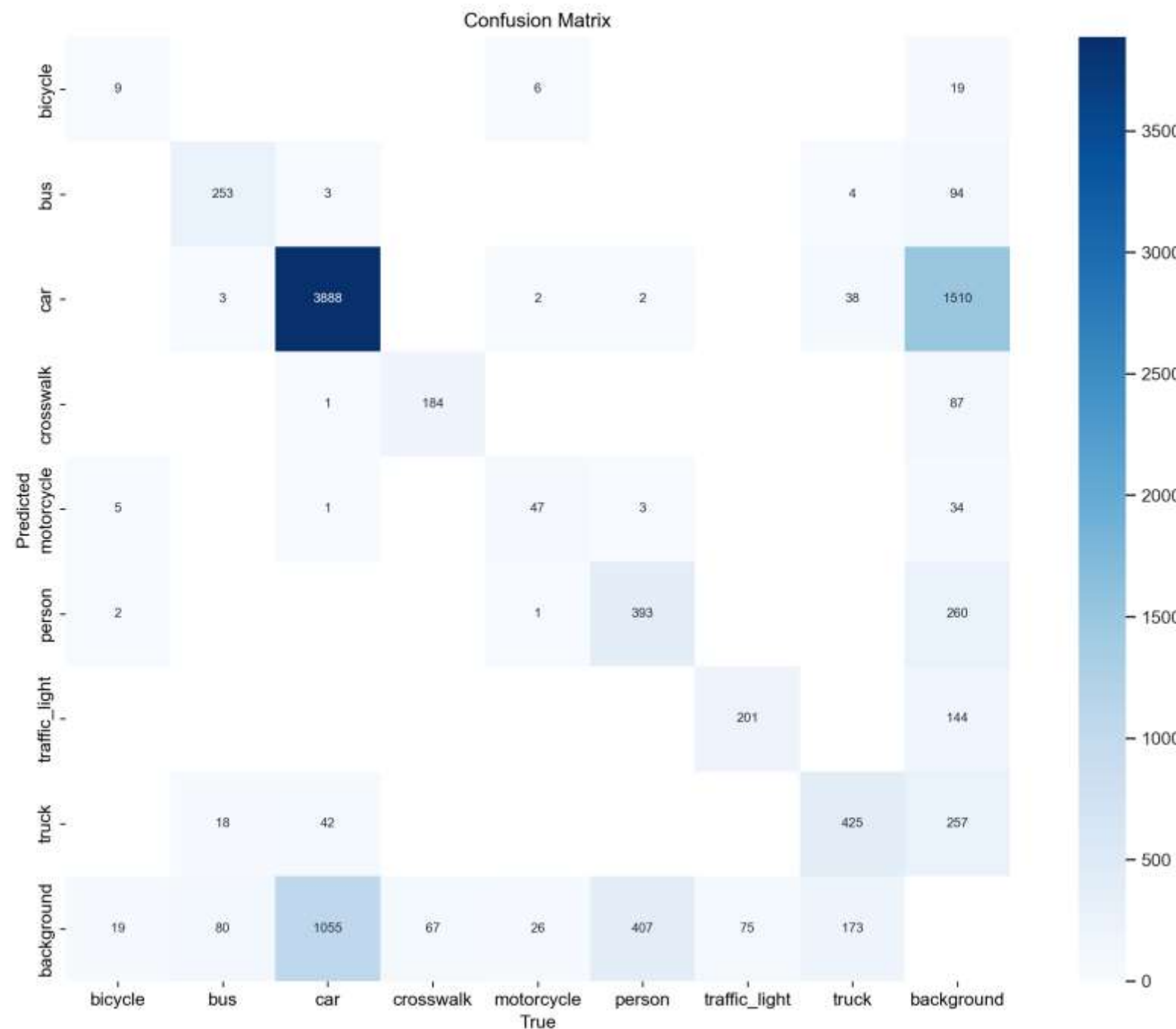
06

모델 데이터 학습 과정 - 버전3(데이터셋3, 150회) 결과



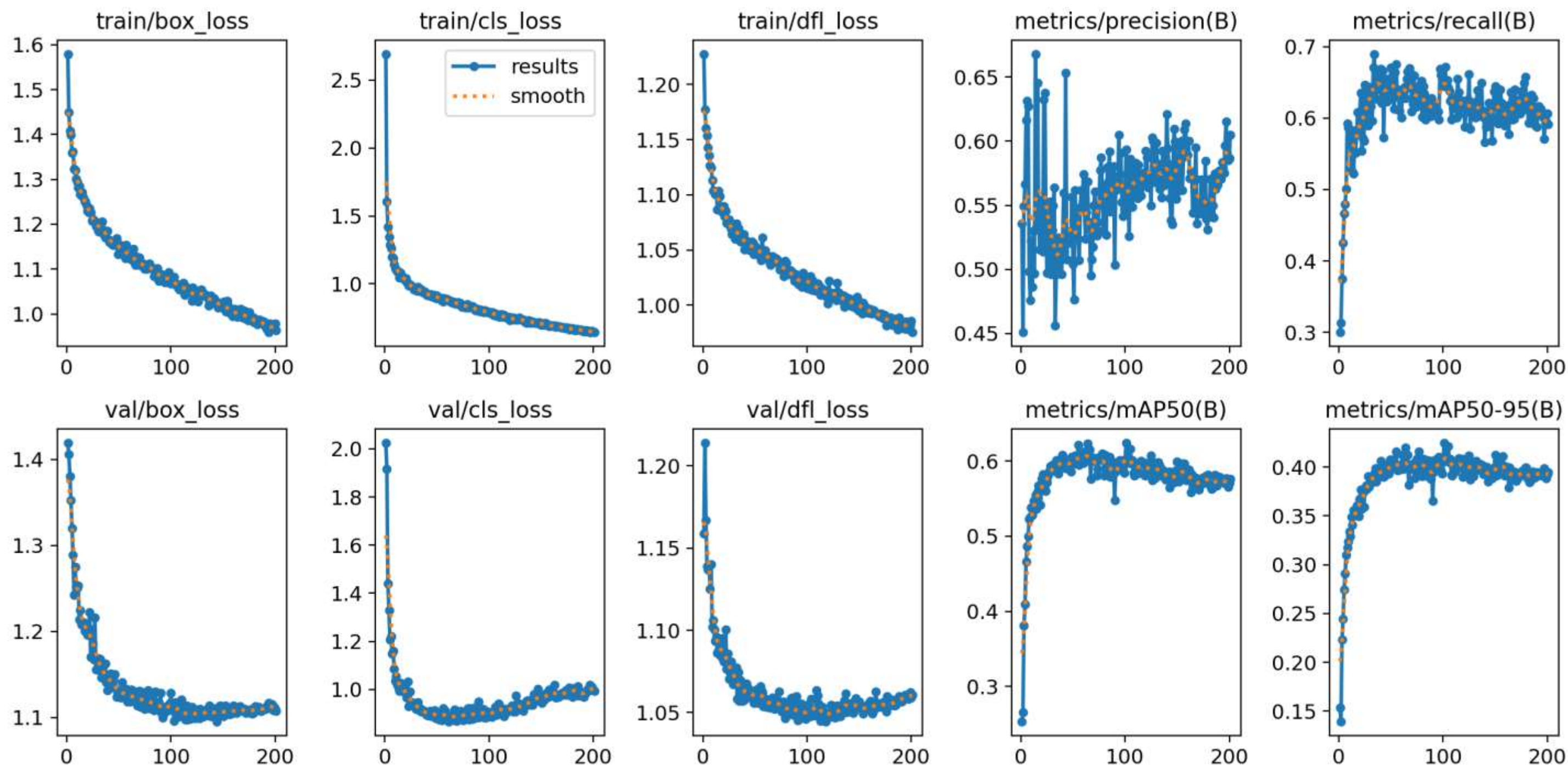
06

모델 데이터 학습 과정 - 버전4(데이터셋3, 22/300) 컴퓨전



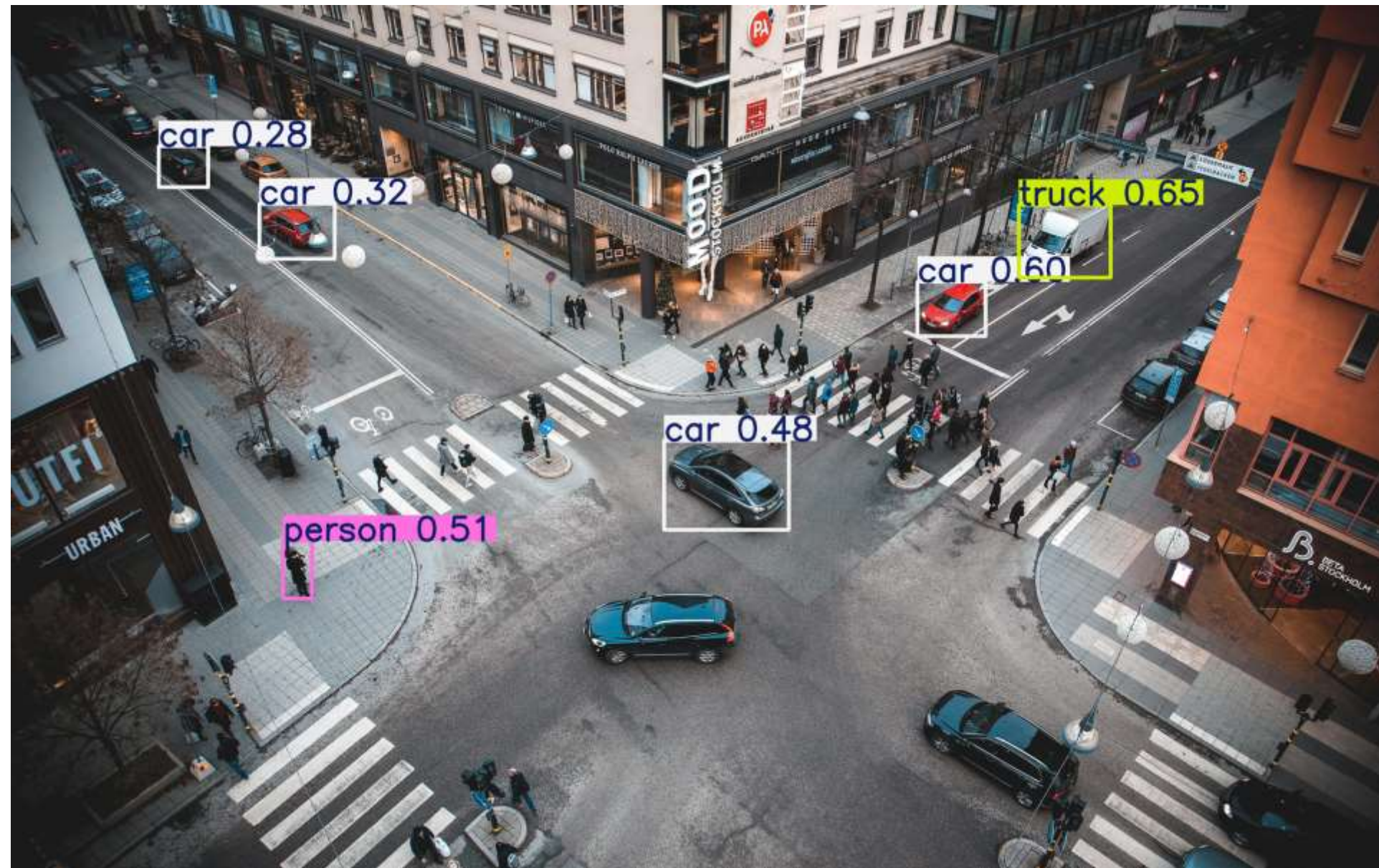
06

모델 데이터 학습 과정 - 버전4(데이터셋3, 22/300) 결과



06

모델 데이터 학습 과정 - 버전4(데이터셋3, 22/300) 결과



07

프로젝트 구현

구현 코드

```
1 from ultralytics import YOLO
2 import cv2
3 import streamlit as st
4 import pandas as pd
5 import plotly.express as px
6 import datetime
7 import time
8 import uuid
9 import os
10 # 환경 변수 설정으로 dll 충돌 문제 해결
11 os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
12
13 # 모델 로드
14 model = YOLO("C:/Users/Administrator/Desktop/4way/4way-2/4way-main/te
15
16 # 클래스 그룹 정의
17 person_id = 5
18 vehicle_ids = {0, 1, 2, 4, 7}
19
20 # 혼잡도 상태 분류 함수
21 def get_status(count):
22     if count >= 15:
23         return "매우 혼잡"
24     elif count >= 10:
25         return "혼잡"
26     elif count >= 5:
27         return "보통"
28     else:
29         return "원활"
30
31 # Streamlit 레이아웃 설정
32 st.set_page_config(layout="wide")
33 st.title("🚦 4Way 교차로 분석 시스템 🚦")
34
35 # 영상 및 결과 표시 영역
36 video_area = st.empty()
37 alert_area = st.empty()
38 info_area = st.empty()
39 info1_col, info2_col, info3_col, info4_col = st.columns([1, 1, 1, 1])
40 info1 = info1_col.empty()
41 info2 = info2_col.empty()
42 info3 = info3_col.empty()
43 info4 = info4_col.empty()
44 col1, col2 = st.columns([1, 1])
45 chart_person_area = col1.empty()
46 chart_vehicle_area = col2.empty()
```

```
47
48 # 비디오 경로 설정
49 cap = cv2.VideoCapture("http://210.99.70.120:1935/live/cctv007.stream/playlist.m3u8")
50
51 # 시간 관리 변수 추가
52 last_history_update = time.time()
53 last_graph_update = time.time()
54 last_alert_time = 0
55 alert_timeout = 3
56 update_interval = 5
57
58 previous_alerts = set()
59
60 # 히스토리 저장용 리스트 추가
61 history = []
62
63 # 비디오 프레임 처리
64 while cap.isOpened():
65     suc, frame = cap.read()
66
67     st.warning("프레임을 가져올 수 없습니다.")
68     break
69
70 now = time.time()
71 now_str = datetime.datetime.now().strftime("%H:%M:%S")
72 frame = cv2.resize(frame, (640, 480))
73
74 results = model.track(frame, persist=True, tracker="bytetrack.yaml")
75 boxes = results[0].boxes
76 cls_list = boxes.cls
77 ids_list = boxes.id
78
79 # 바운딩 박스 및 객체 추적 리스트
80 tracked_objects = []
81 for box in boxes:
82     cls_id = int(box.cls[0])
83     obj_id = -1
84     if box.id is not None and len(box.id) > 0:
85         obj_id = int(box.id[0])
86     bbox = list(map(int, box.xyxy[0]))
87     tracked_objects.append((obj_id, cls_id, bbox))
88
89 # 경고 메시지 저장 리스트
90 alerts = []
91
92 # 위험 객체 ID 저장용 집합
93 danger_ids = set()
94
95 # 안전도 분석
96 # 거리 계산 함수
97 def get_center(box):
98     x1, y1, x2, y2 = box
99     return ((x1 + x2) // 2, (y1 + y2) // 2)
100
101 def euclidean_distance(p1, p2):
102     return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2) ** 0.5
103
104 # 차량 객체 필터링
105 vehicle_objects = [(obj_id, bbox) for obj_id, cls_id, bbox in tracked_objects if c
106
107 # 차량 간 거리 측정
108 for i in range(len(vehicle_objects)):
```

```
109
110 # 차량 간 거리 측정
111 for i in range(len(vehicle_objects)):
112     for j in range(i + 1, len(vehicle_objects)):
113         id1, box1 = vehicle_objects[i]
114         id2, box2 = vehicle_objects[j]
115         dist = euclidean_distance(get_center(box1), get_center(box2))
116
117         if dist < 50:
118             msg = f"🚗 [{now_str}] 차량 거리 위험({int(dist)}px): 차량 {id1} + 차량 {id2}"
119             if msg not in previous_alerts:
120                 alerts.append(msg)
121                 previous_alerts.add(msg)
122                 danger_ids.update([id1, id2]) # 🚗 위험 차량 ID 저장
123
124 # 보행자 객체 필터링
125 person_objects = [(obj_id, bbox) for obj_id, cls_id, bbox in tracked_objects if cls_id == person_id]
126
127 # 보행자와 차량 간 거리 측정
128 for p_id, p_box in person_objects:
129     for v_id, v_box in vehicle_objects:
130         dist = euclidean_distance(get_center(p_box), get_center(v_box))
131
132         if dist < 60:
133             msg = f"🚶 [{now_str}] 보행자와 차량 거리 위험({int(dist)}px): 보행자 {p_id} + 차량 {v_id}"
134             if msg not in previous_alerts:
135                 alerts.append(msg)
136                 previous_alerts.add(msg)
137                 danger_ids.update([p_id, v_id]) # 🚶 위험 차량 ID 저장
138
139 # 위험 객체의 바운딩 박스 색상 표시
140 for box in boxes:
141     cls_id = int(box.cls[0])
142     obj_id = -1
143     if box.id is not None and len(box.id) > 0:
144         obj_id = int(box.id[0])
145     cls_name = model.names[cls_id]
146     conf = float(box.conf[0])
147     x1, y1, x2, y2 = map(int, box.xyxy[0])
148     label = f"ID:{obj_id} {cls_name}"
149
150 # 박스 색상 체계:
151 # - 위험: 빨간색
152 # - 비위험 보행자: 노란색
153 # - 비위험 차량: 초록색
154 if obj_id in danger_ids:
155     box_color = (0, 0, 255) # 🚗 빨간색
156     text_color = (0, 0, 255)
157 else:
158     # 🚶 비위험 보행자 → 노란색
159     if cls_id == person_id:
160         box_color = (0, 255, 255) # 🚶 노란색
161         text_color = (0, 255, 255)
162     # 🚗 비위험 차량 → 초록색
163     elif cls_id in vehicle_ids:
164         box_color = (0, 255, 0) # 🚗 초록색
165         text_color = (0, 255, 0)
166     else:
167         # 기타 클래스 (예외 처리)
168         box_color = (255, 0, 0)
169         text_color = (255, 255, 255)
170
171 cv2.rectangle(frame, (x1, y1), (x2, y2), box_color, 2)
172 cv2.putText(frame, label, (x1, y1 - 10),
173             cv2.FONT_HERSHEY_SIMPLEX, 0.6, text_color, 2)
```


07

프로젝트 구현

구현 코드

```
174 cv2.FONT_HERSHEY_SIMPLEX, 0.6, text_color, 2)
175
176 # 프레임 RGB 변환 후 표시
177 frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
178 video_area.image(frame, channels="RGB", width=640) #use_column_w
179
180 # 인파 / 차량 각각 카운팅
181 person_count = sum(int(cls) == person_id for cls in cls_list)
182 vehicle_count = sum(int(cls) in vehicle_ids for cls in cls_list)
183
184 # 상태 평가
185 person_status = get_status(person_count)
186 vehicle_status = get_status(vehicle_count)
187
188 # 최신 알람 시간 갱신
189 if alerts:
190     last_alert_time = now
191
192 # 경고 메시지 표시 또는 제거
193 if alerts:
194     alert_text = "### ⚠️ **실시간 안전 경고**\n"
195     alert_text += "\n".join([f"- {msg}" for msg in alerts])
196     alert_area.markdown(alert_text)
197 elif now - last_alert_time < alert_timeout:
198     # 이전 알람 이후 경과 시간이 timeout 이내면 유지
199     pass
200 else:
201     alert_area.empty()
202     previous_alerts.clear() # timeout 이후 이전 알람 기록도 초기화
203
204 # 정보 출력
205 info_area.markdown("### 🔍 실시간 교차로 혼잡도 정보")
206 info1.metric("👤 보행자 수", person_count)
207 info2.metric("🚦 보행자 혼잡도", person_status)
208 info3.metric("🚗 차량 수", vehicle_count)
```

```
209 info4.metric("🚦 차량 혼잡도", vehicle_status)
210
211 # 5초마다 꺾은선 그래프 갱신
212 if now - last_history_update >= update_interval:
213
214     # 타임스탬프 생성
215     timestamp = datetime.datetime.now()
216
217     # 현재 시점 정보 저장
218     history.append({
219         "시간": timestamp,
220         "구분": "사람",
221         "수량": person_count,
222         "상태": person_status
223     })
224     history.append({
225         "시간": timestamp,
226         "구분": "차량",
227         "수량": vehicle_count,
228         "상태": vehicle_status
229     })
230
231 # 히스토리 개수 제한 / 사람/차량 각각 100개씩
232 history = history[-200:]
233
234 # 히스토리 데이터프레임 변환 및 시간순 정렬
235 history_df = pd.DataFrame(history)
236 history_df = history_df.sort_values(by=["시간", "구분"])
237
238 # 주이 시각화 (꺾은선 그래프)
239 if len(history_df) >= 4:
240     # 사람 / 차량 데이터 분리
241     df_person = history_df[history_df["구분"] == "사람"].sort_values("시간").tail(10)
242     df_vehicle = history_df[history_df["구분"] == "차량"].sort_values("시간").tail(10)
243
244     # 상태별 색상 매핑
245     color_map = {
246         "유휴": "green",
247         "보통": "yellow",
248         "혼잡": "orange",
249         "매우 혼잡": "red"
250     }
251
252     # 상태 연속 구간별로 자르기 위한 함수 (마커 추가용)
253     def split_by_status(df):
254         segments = []
255         if df.empty:
256             return segments
257         current_status = df.iloc[0]["상태"]
258         segment = [df.iloc[0]]
259         for i in range(1, len(df)):
260             row = df.iloc[i]
261             if row["상태"] == current_status:
262                 segment.append(row)
263             else:
264                 segments.append(pd.DataFrame(segment))
265                 segment = [row]
266                 current_status = row["상태"]
267         segments.append(pd.DataFrame(segment))
268         return segments
269
270     # 보행자 상태별 꺾은선 생성
271     fig_person = px.line(df_person, x="시간", y="수량", title="👤 보행자 혼잡도 주이")
272     fig_person.update_traces(mode='lines', line=dict(color="blue"))
273
274     # 차량별 마커 추가 (상태가 바뀐 지점마다 마커)
275     added_status = set()
276
277     for seg in split_by_status(df_person):
```

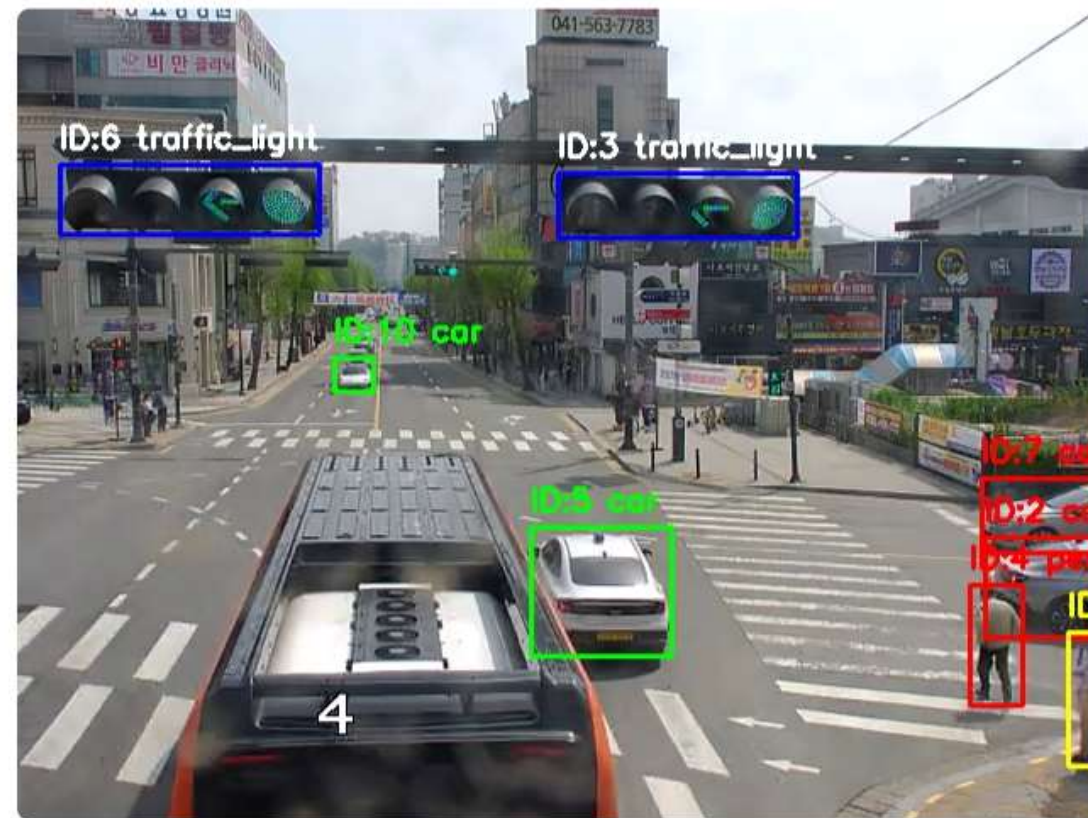
```
277
278
279     status = seg.iloc[0]["상태"]
280     show_legend = status not in added_status
281     fig_person.add_scatter(
282         x=seg["시간"], y=seg["수량"],
283         mode="markers",
284         marker=dict(color=color_map.get(status, "gray"), size=8, symbol="circle"),
285         name=status if show_legend else None,
286         showlegend=show_legend
287     )
288     added_status.add(status)
289
290 # x축을 5초 간격으로 설정
291 fig_person.update_layout(
292     xaxis=dict(
293         tickformat="%H:%M:%S",
294         tickangle=45,
295         tickmode="linear",
296         dtick=5 * 1000 # 5초 간격으로 설정 (밀리초 단위)
297     )
298 )
299
300 # 차량 상태별 꺾은선 생성
301 fig_vehicle = px.line(df_vehicle, x="시간", y="수량", title="🚗 차량 혼잡도 주이")
302 fig_vehicle.update_traces(mode='lines', line=dict(color="blue")) # 선은 하나의 색으로 고정
303
304 # 상태별로 마커 추가 (상태가 바뀐 지점마다 마커)
305 added_status = set()
306
307 for seg in split_by_status(df_vehicle):
308     status = seg.iloc[0]["상태"]
309     show_legend = status not in added_status
310     fig_vehicle.add_scatter(
311         x=seg["시간"], y=seg["수량"],
312         mode="markers",
313
314     # 상태별로 마커 추가 (상태가 바뀐 지점마다 마커)
315     added_status = set()
316
317     for seg in split_by_status(df_vehicle):
318         status = seg.iloc[0]["상태"]
319         show_legend = status not in added_status
320         fig_vehicle.add_scatter(
321             x=seg["시간"], y=seg["수량"],
322             mode="markers",
323             marker=dict(color=color_map.get(status, "gray"), size=8, symbol="circle"),
324             name=status if show_legend else None,
325             showlegend=show_legend
326         )
327         added_status.add(status)
328
329 # x축을 5초 간격으로 설정
330 fig_vehicle.update_layout(
331     xaxis=dict(
332         tickformat="%H:%M:%S",
333         tickangle=45,
334         tickmode="linear",
335         dtick=5 * 1000 # 5초 간격으로 설정 (밀리초 단위)
336     )
337 )
338
339 # 시각화 업데이트
340 chart_person_area.plotly_chart(fig_person, use_container_width=True, key=f"line_chart_person_{uuid.uuid4()}")
341 chart_vehicle_area.plotly_chart(fig_vehicle, use_container_width=True, key=f"line_chart_vehicle_{uuid.uuid4()}")
342
343 last_history_update = now
344
345 release()
346 destroyAllWindows()
```


07

프로젝트 구현

구현 화면

🚦 4Way 교차로 분석 시스템 🚦



⚠️ 실시간 안전 경고

- 🚗 [14:35:55] 차량 거리 위험(42px): 차량 2 ↔ 차량 7
- 🚗 [14:35:55] 보행자와 차량 거리 위험(43px): 보행자 4 ↔ 차량 2

🔍 실시간 교차로 혼잡도 정보

🚶 보행자 수

2

🚶 보행자 혼잡도

원활

🚗 차량 수

4

🚗 차량 혼잡도

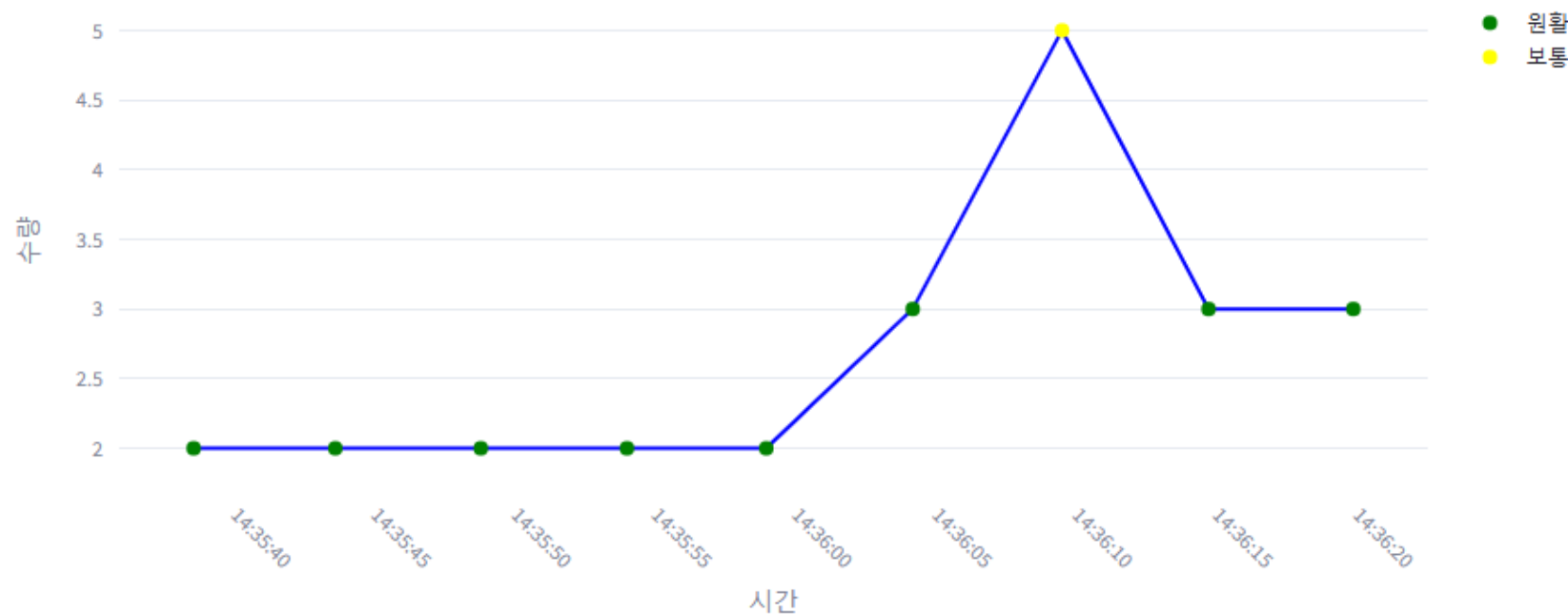
원활

07

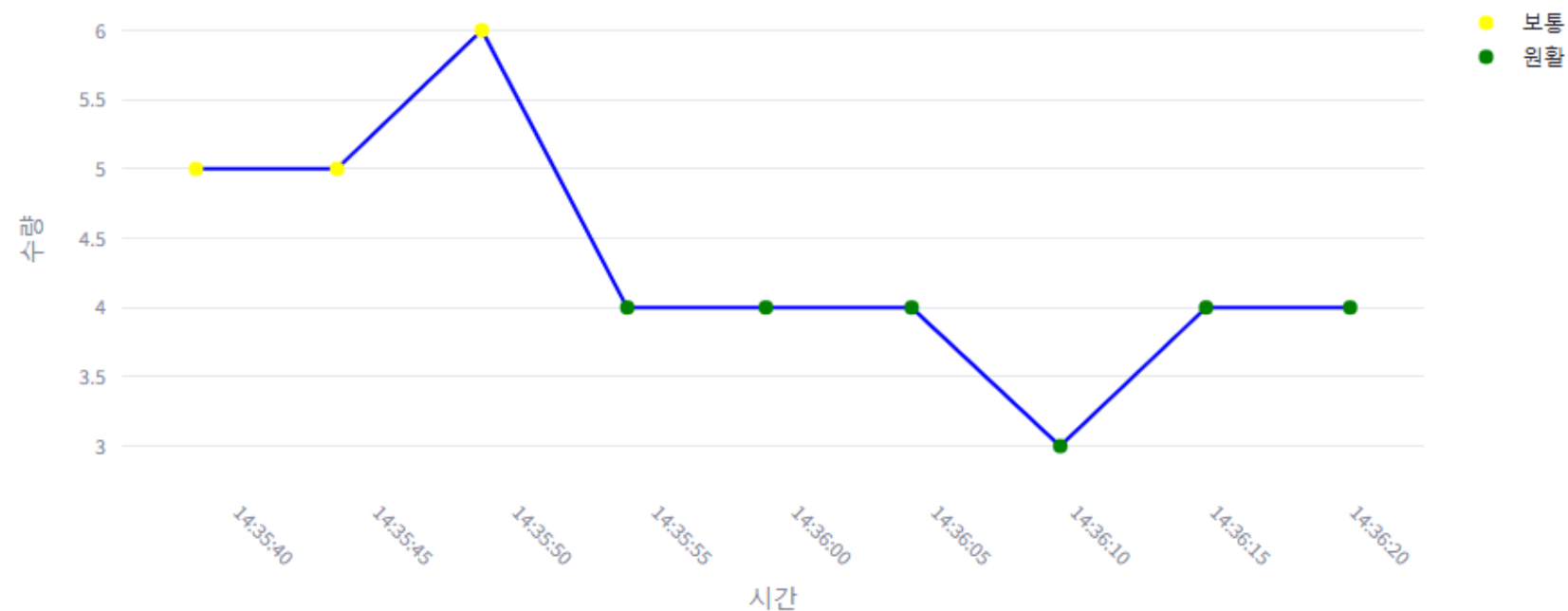
프로젝트 구현

구현 화면

보행자 혼잡도 추이



차량 혼잡도 추이



08

자체 평가 의견

1) 결과물에 대한 완성도 평가

팀원	기능	디자인
이보라	7점	8점
권 봄	8점	8점
임다연	8점	8점
강은비	7점	8점

08

자체 평가 의견

2) 프로젝트 자체 평가와 느낀 점

팀원	내용
이보라	객체 탐지에서 데이터와 라벨링의 중요성을 체감했고, 많은 시간을 투자했으며 비전 모델 개발 과정과 팀원 간 협업의 중요성도 함께 배울 수 있었음
권 봄	팀원 모두 라벨링에 참여하며 프로젝트 전반을 이해할 수 있어 좋았음. 시간이 많이 소요되어 일정이 촉박해져 다음엔 시간 배분에 신경 써야겠다는 느낌
임다연	계획했던 기능들을 구현하게 되어서 만족함, 모델 학습하는 과정에 있어서 예상보 다 많은 시간이 소요되어 최적화가 중요하다고 느꼈음
강은비	라벨링 정확도가 모델 성능에 중요함을 깨달음. 데이터 수집·라벨링에 시간 많이 들었고, 다음엔 시간 배분 더 효율적으로 할 필요 있음.

09

추후 프로젝트 보완할 점

01

객체 탐지
정확도 향상

모델 재학습

데이터 정제

오탐 개선

02

다양한 환경
데이터 보강

야간 영상 수집

날씨 조건 보강

테스트 강화

비·안개 대응

03

추가 기능 확장

무단횡단 탐지

신호위반 감지

상황 인식 추가

09

QnA



THANK YOU !
