

Deep learning project  
**자율주행 시각 엔진(Vision Engine) 구현**

4조 임현수 정민주



# INDEX

01

## 프로젝트 개요

- 주제 및 목적
- 기획 배경과 목표
- 알고리즘 및 tool 소개

02

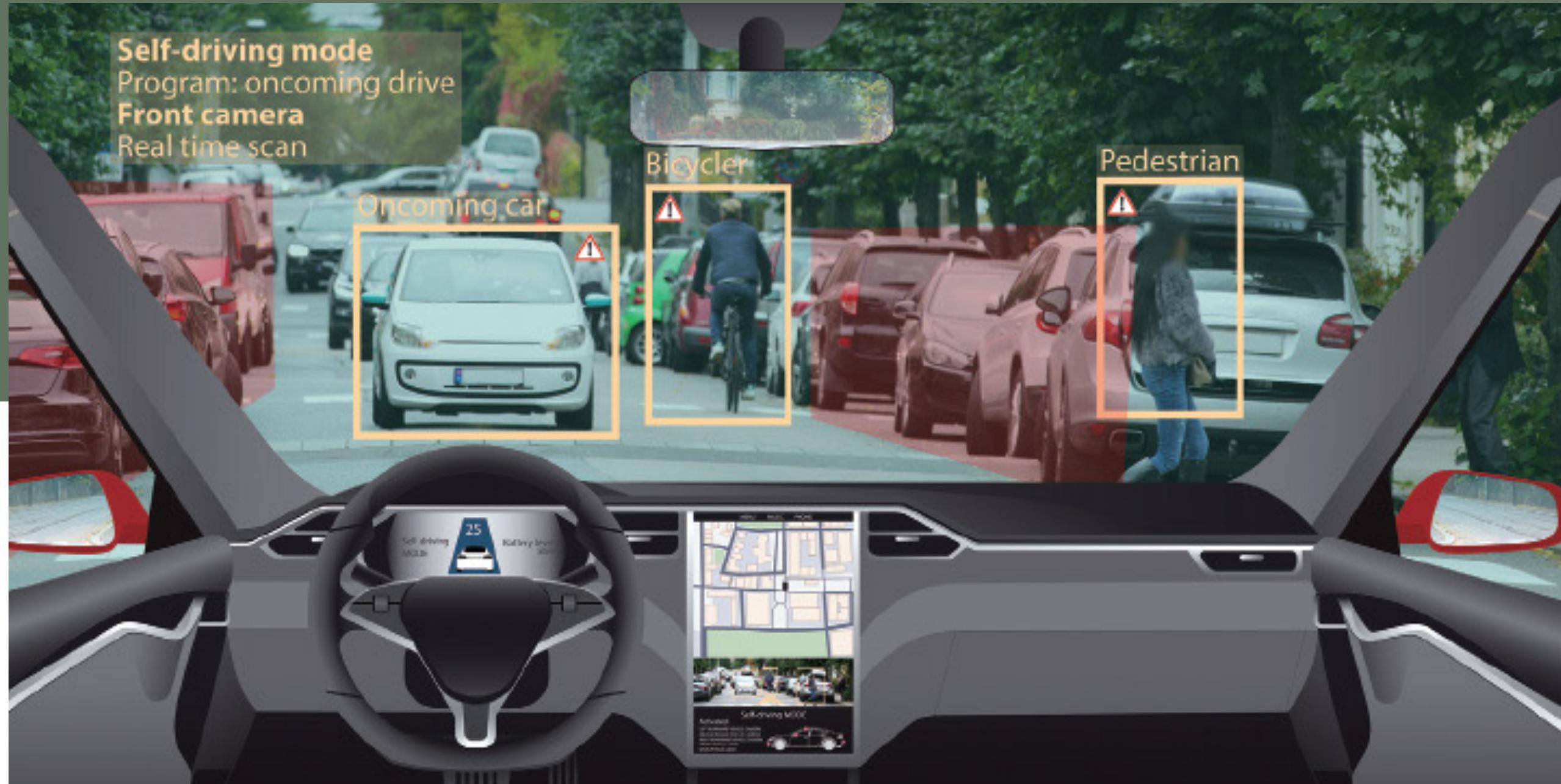
## 프로세스

- 프로세스 소개
- 데이터 수집과 거리 추정
- 1 ~ 4차 시도

03

## 프로젝트 결과

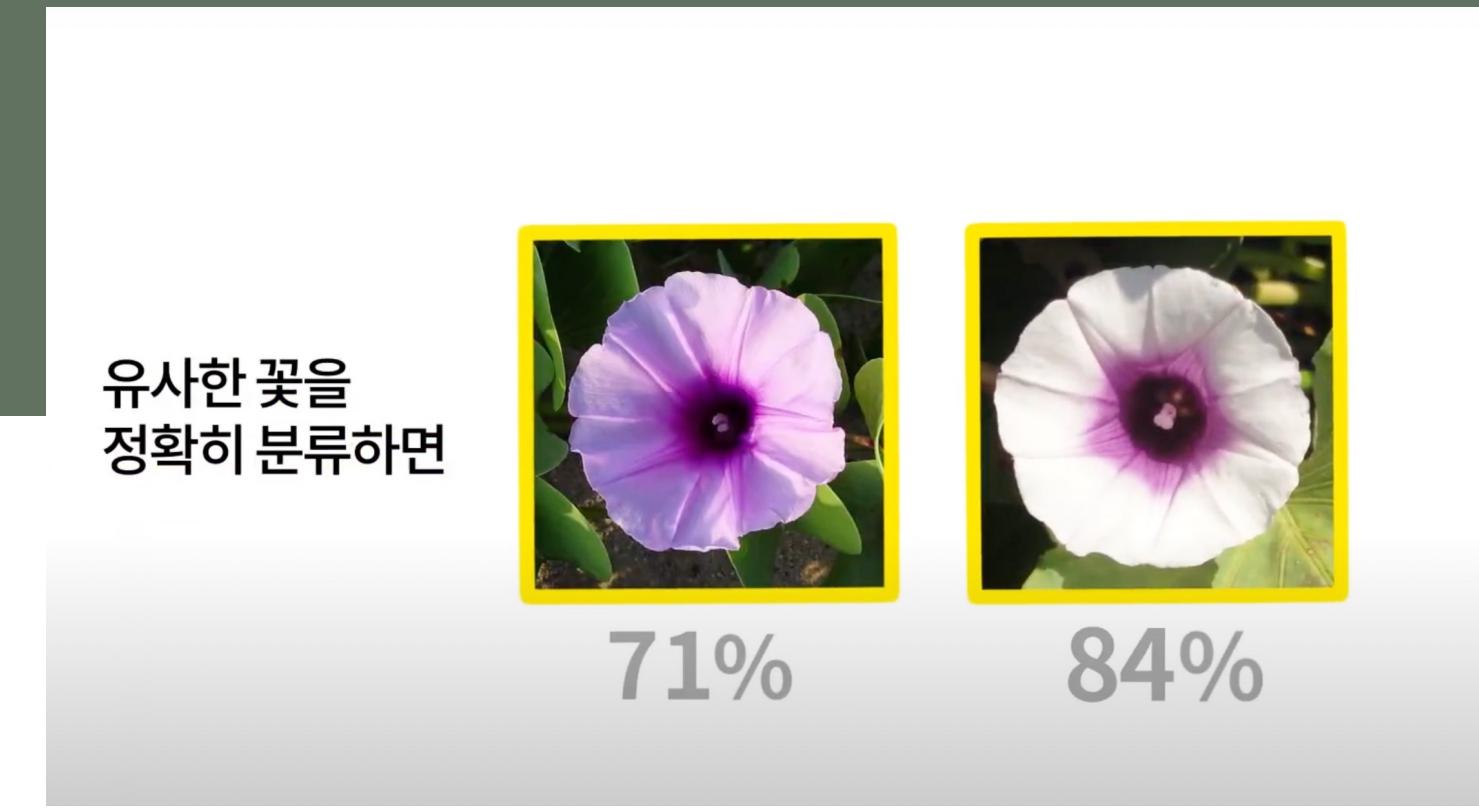
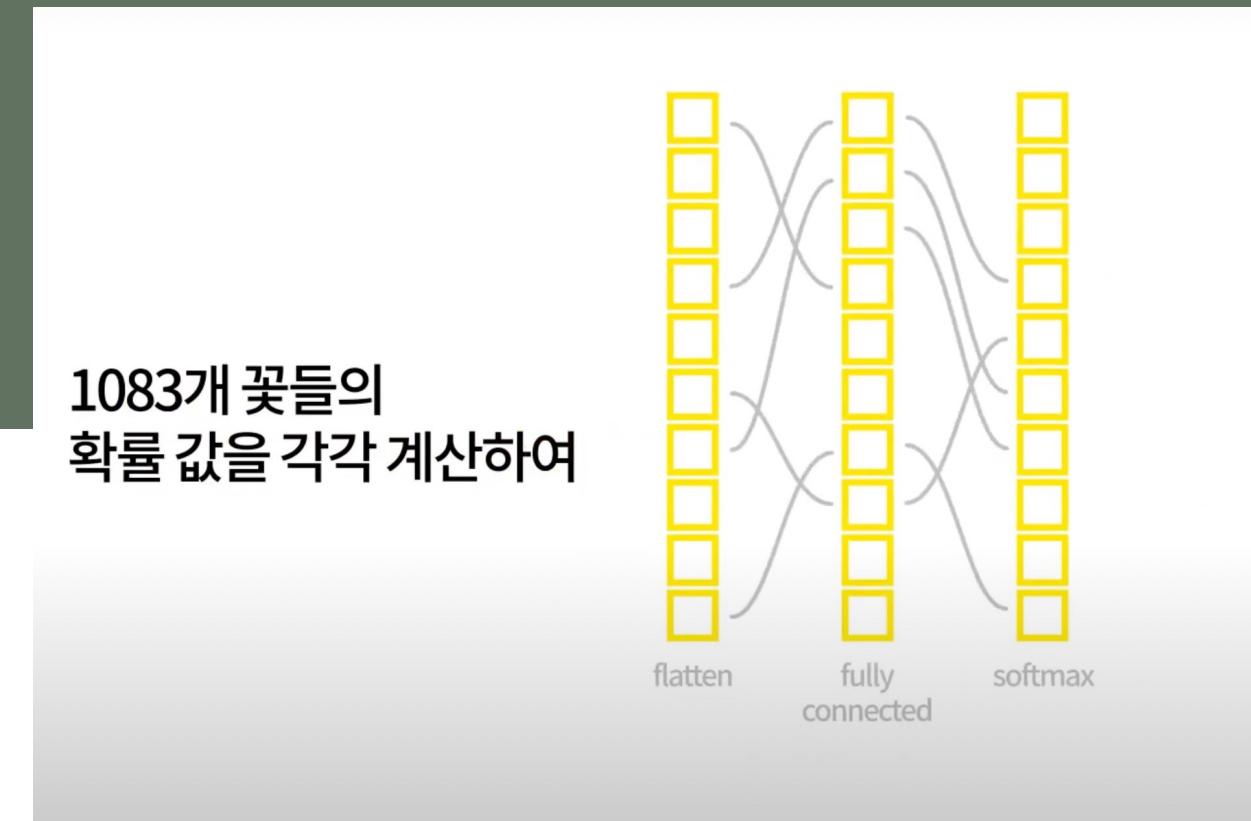
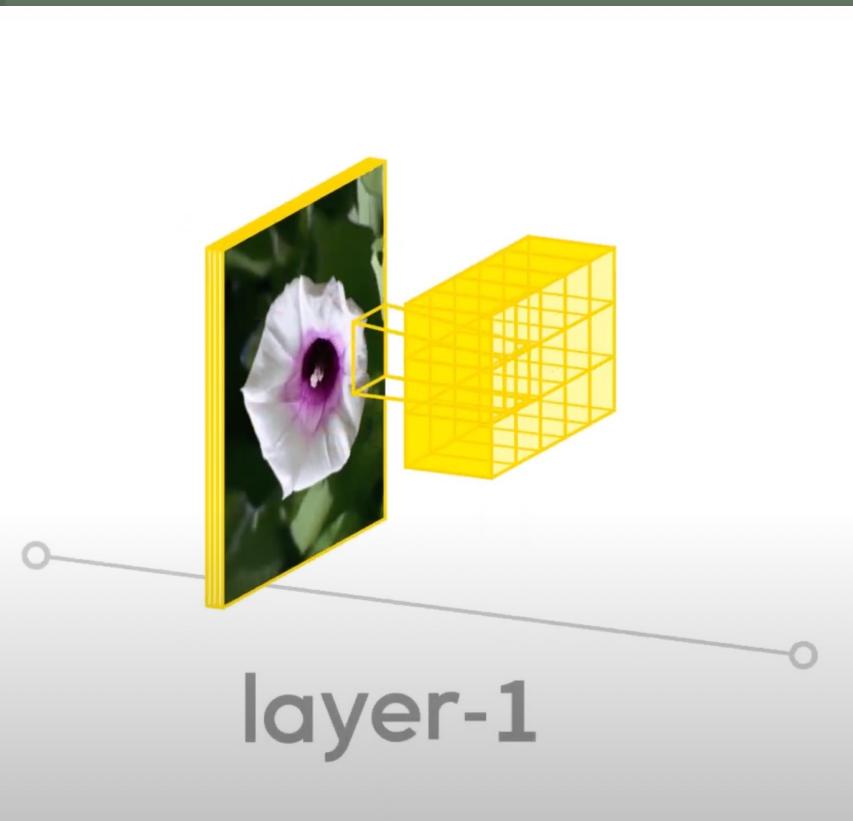
- 결과 영상
- 추후 보완



**프로젝트 주제**

---

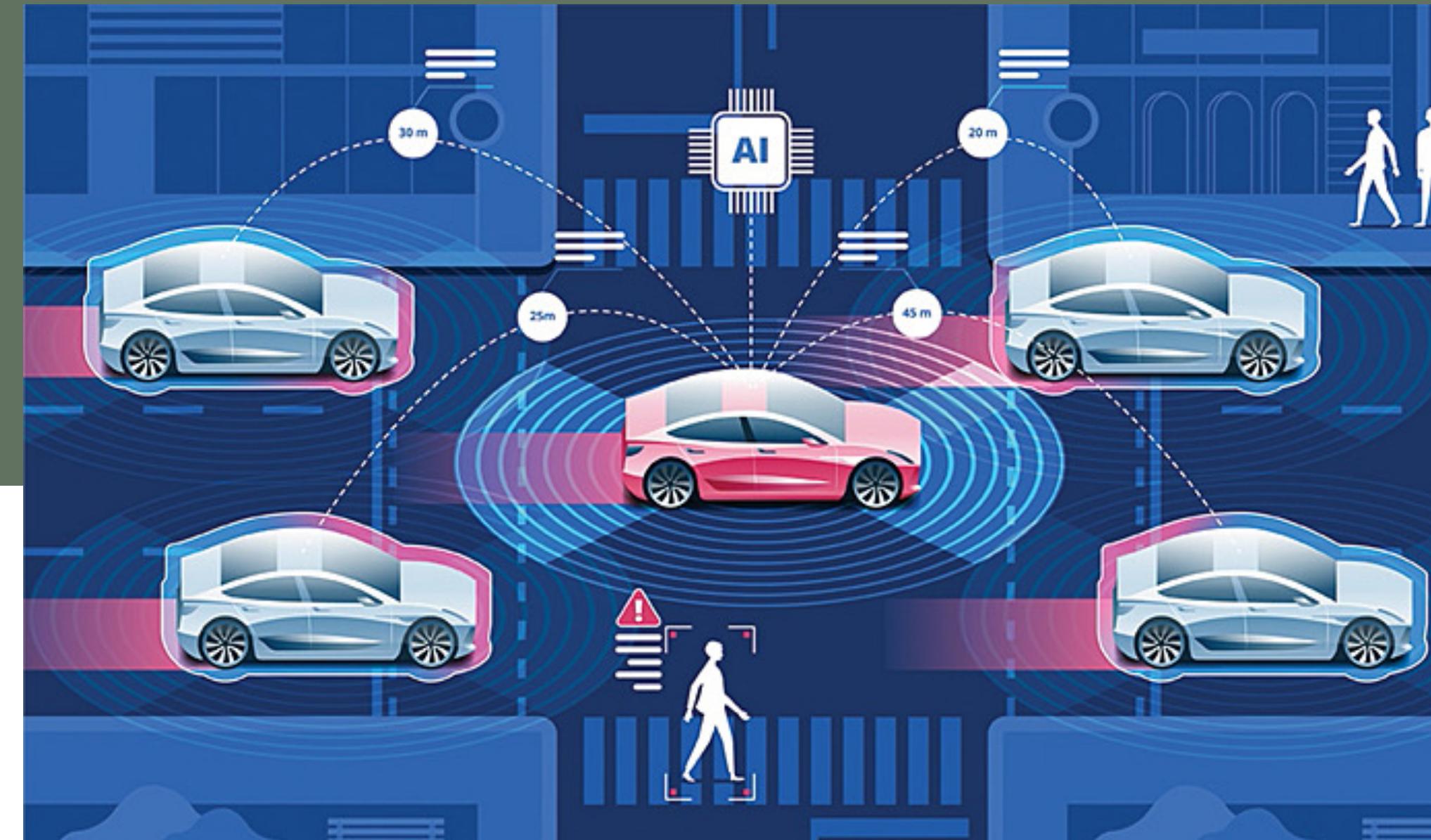
## 자율주행 시각 엔진(Vision Engine)\* 구현



### 프로젝트 주제

시각엔진 : 이미지에 담긴 다양한 정보를 인식 및 분석하여 이미지를 이해하는 엔진

얼굴 인식, 유사 상품 추천 등에 사용 가능 (출처: 카카오엔터프라이즈 홈페이지)



**프로젝트 목적**

---

운전 시 사람이 시각 정보를 수집하는 과정을 모방,  
자율주행 시각엔진을 간단한 수준에서 구현해보자 함

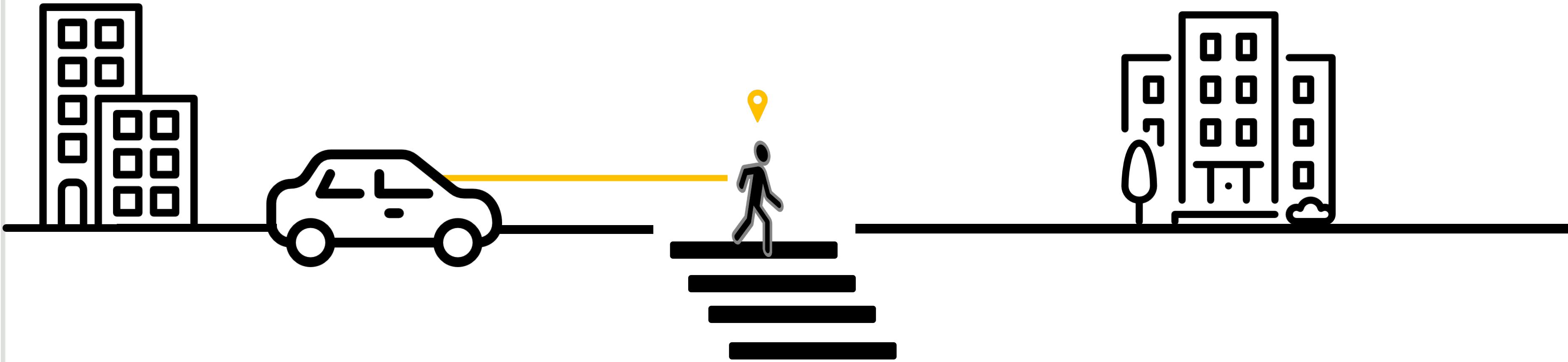
## ☑ 프로젝트 기획 배경

[거리 인지]

step 1. ‘가까이’에 갑자기 물체가 나타남

step 2. 우선 급정거

→ step 0. 거리를 인지하였음



## ☑ 프로젝트 기획 배경

[객체 탐지]

step 1. 내 시야 '멀리'에 무언가 있다

step 2. 위협이 되지 않는다면 무시,

피해야한다면 그 대상과의 거리에 따라  
적절하게 대응

→ step 0. 발견한 것의 종류를 인지함



## ☑ 프로젝트 기획 배경 [트랙킹]

· '가까워지기' 위해 쫓아가는 경우,  
눈에 보이는 다양한 대상들 중에서 하나의 고유한 대상을 인지하고 추적함



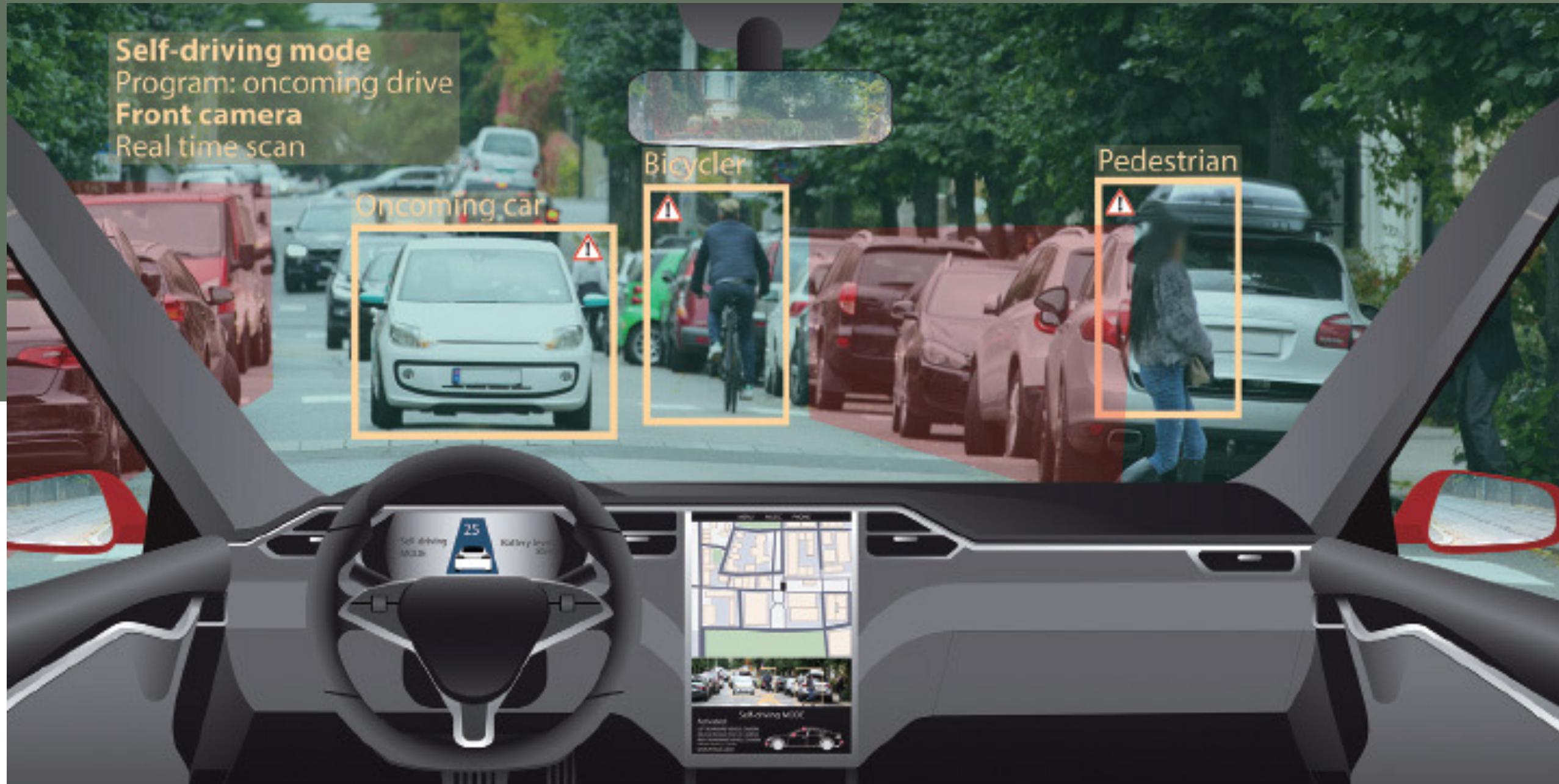
## ☑ 프로젝트 기획 배경

[거리 인지]

[객체 탐지] → ‘거리’에 대한 인지가 기반이 되며, ‘거리’란 반대편의 ’객체 탐지’를 전제로 함

[트래킹] 이 두가지가 충족될 때 트래킹 등 다양한 어플리케이션으로 확장 가능함



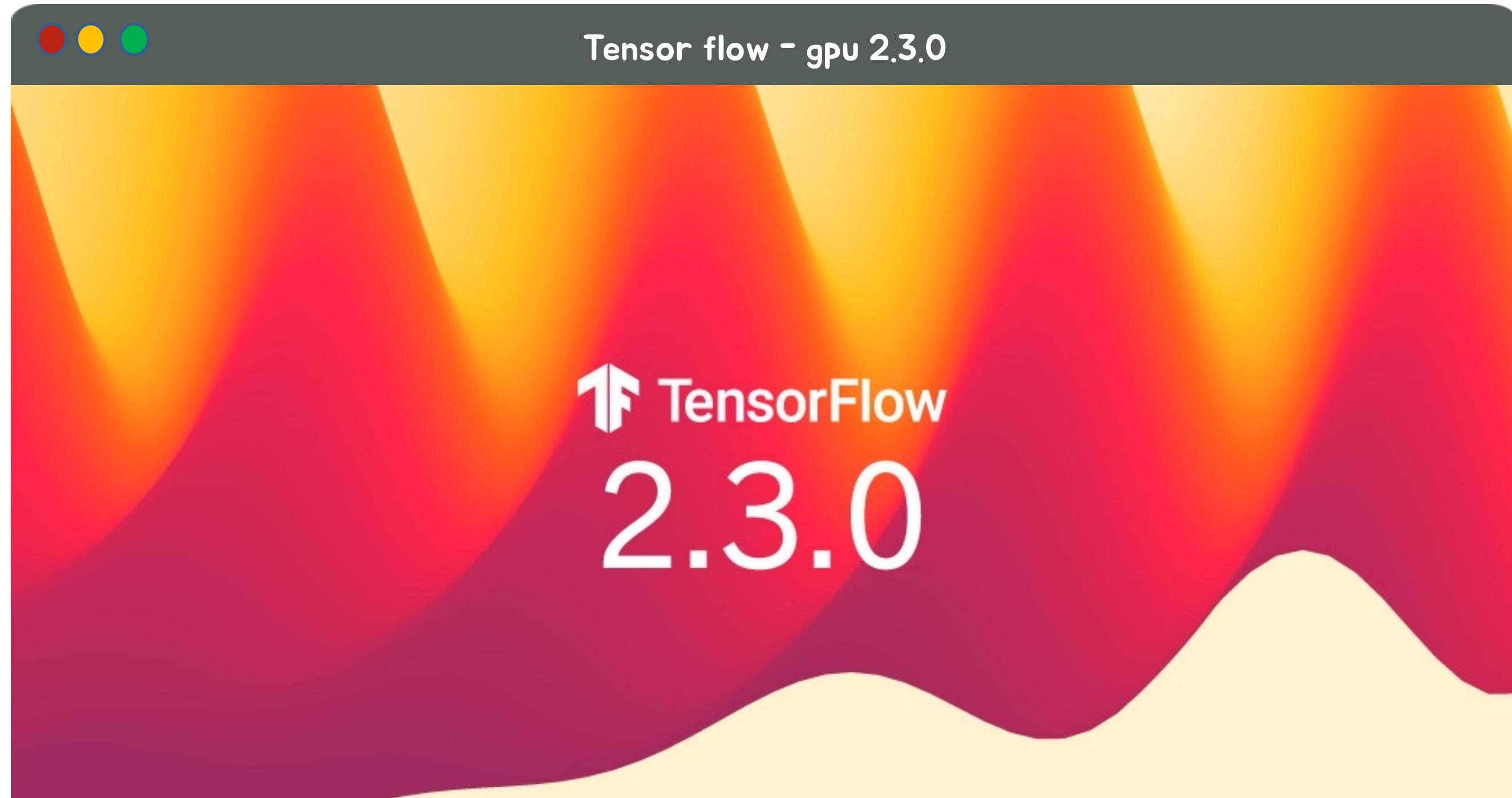


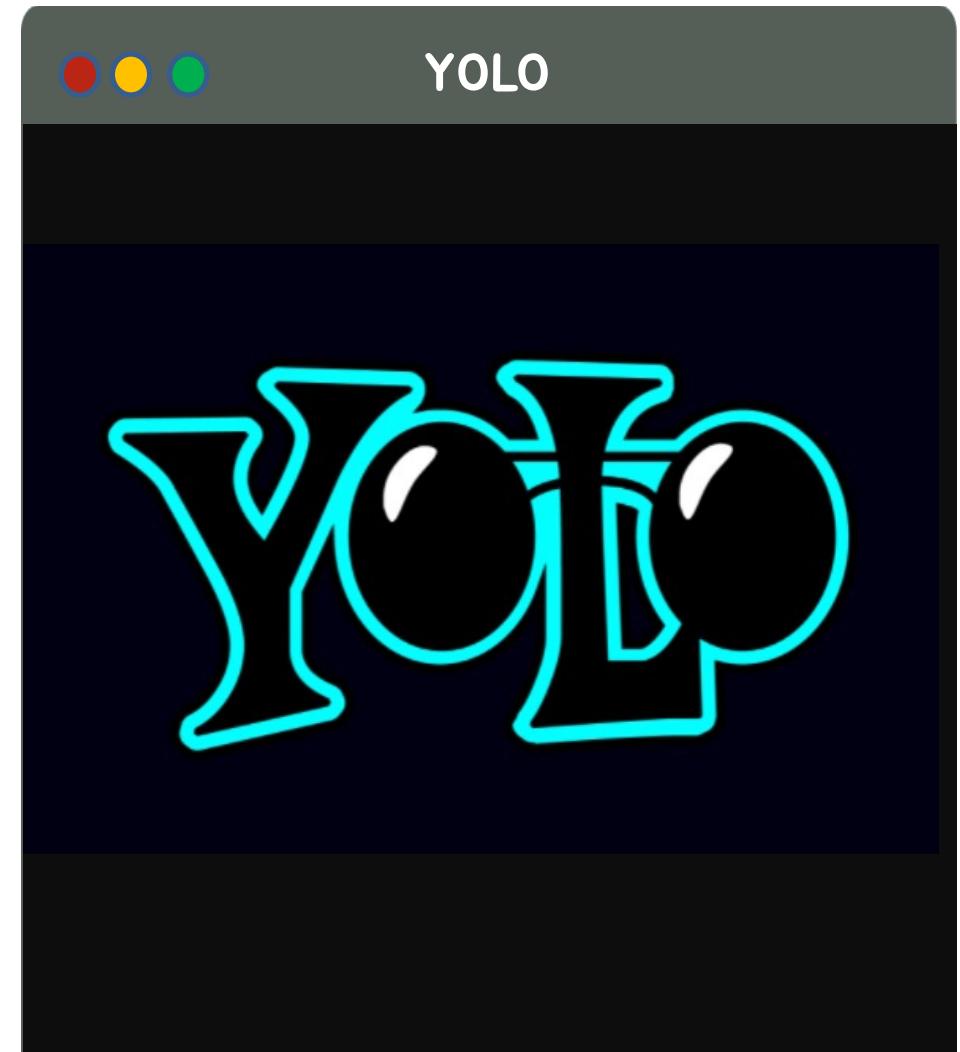
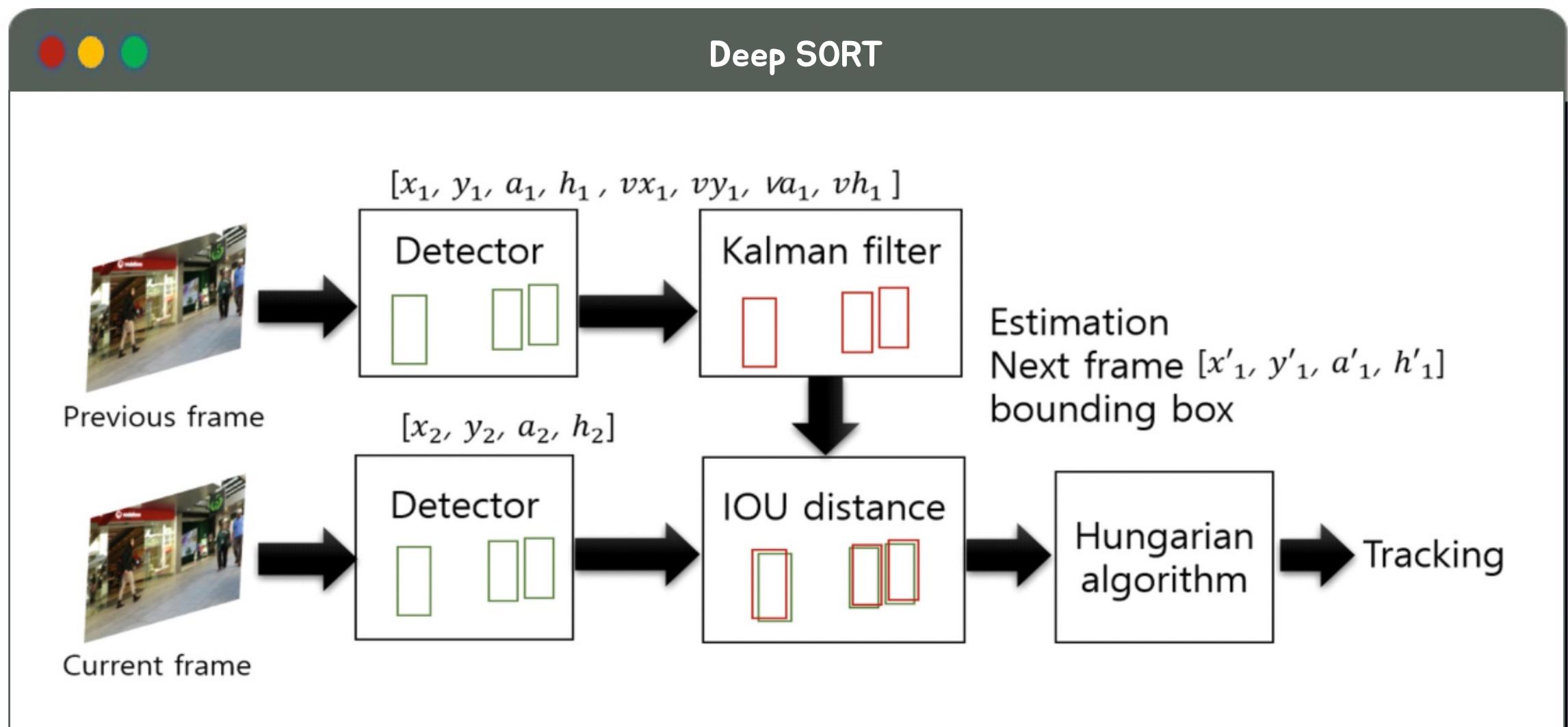
**프로젝트 목표**

우수한 객체 탐지(Object Detection) 성능과 거리 추정 정확도 확보  
위 내용을 기반으로 한 다양한 적용 (Applications)



## 프레임워크

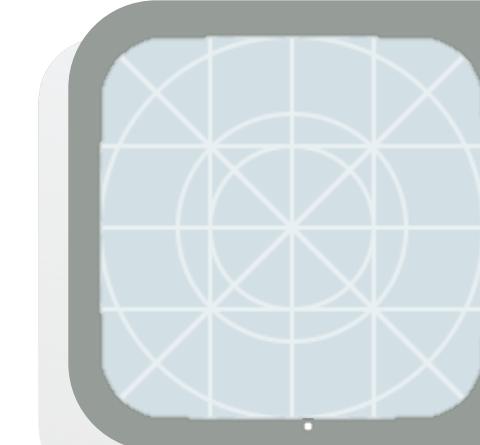


 알고리즘



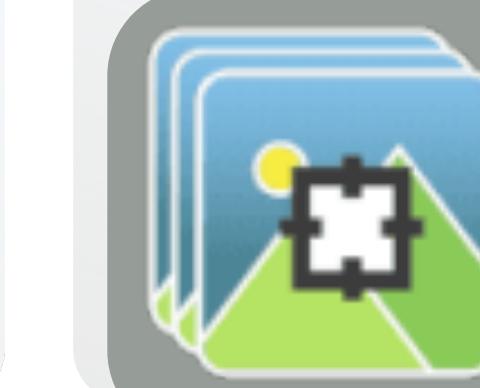
Tool

Google Colab

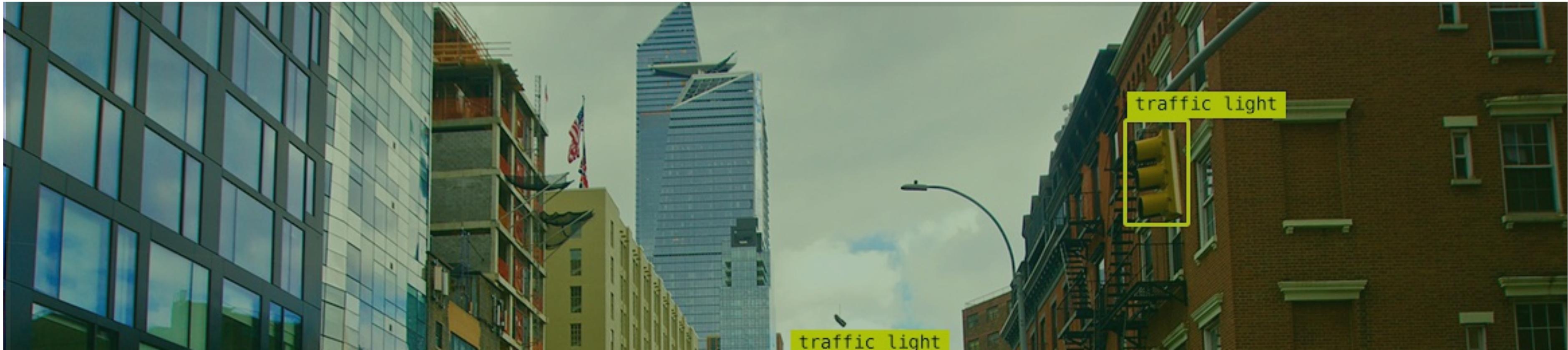


Yololabel

Iriun



LabelImg



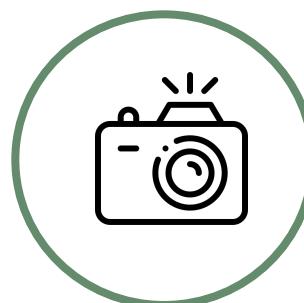
# 02

## 프로세스

- 프로세스 소개
- 데이터 수집과 거리 추정
- 1 ~ 4차 시도

 **프로세스 소개**

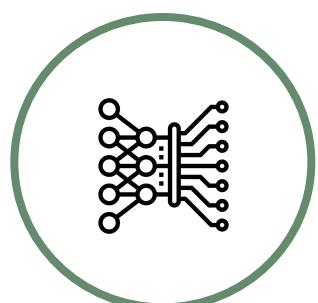
Step 1  
데이터 수집



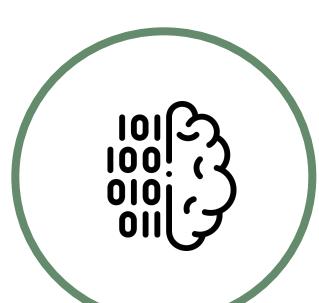
Step 2  
라벨링



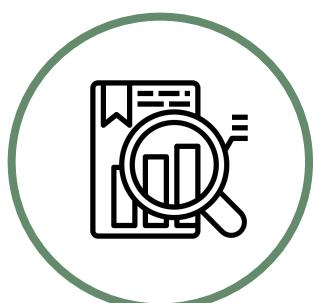
Step 3  
모델 훈련



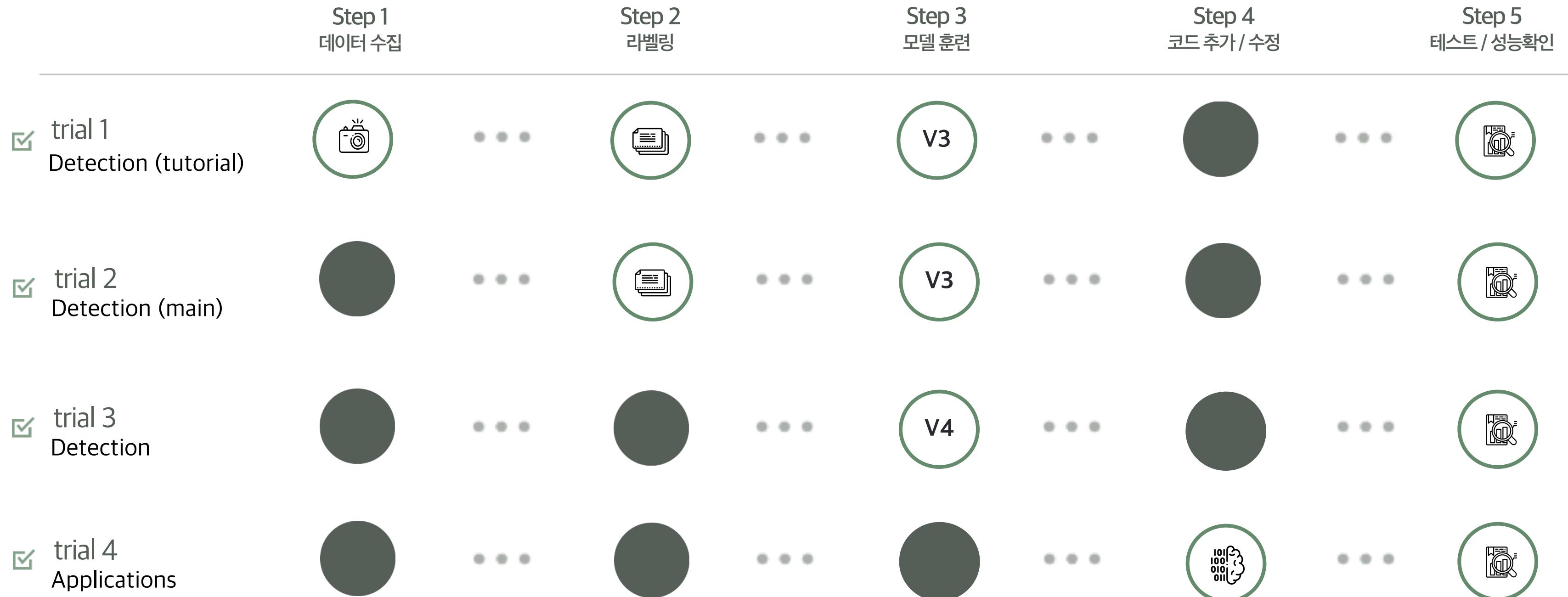
Step 4  
코드 추가 / 수정

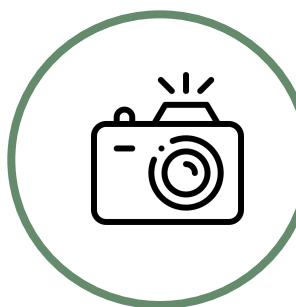


Step 5  
테스트 / 성능 확인



## ✓ 프로세스 소개

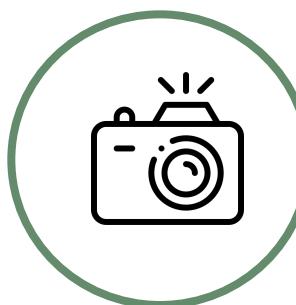


 데이터 수집Step 1  
데이터 수집

## [문제] 지상에서 거리 추정 시 발생하는 문제

- 지상에서 : 사람의 얼굴이 분간 가능하기 때문에 초상권 등의 이슈 발생 가능
- 거리를 추정하기 : 사진 속 객체와 촬영 근원지(카메라) 간 거리 추정이 기존 모듈이나 라이브러리 등의 형식으로 존재하지 않는다.

→ ‘공중에서’ ‘직접 실측하고 맞춤화된 거리 추정 함수를 도출’ 해야함

 데이터 수집Step 1  
데이터 수집

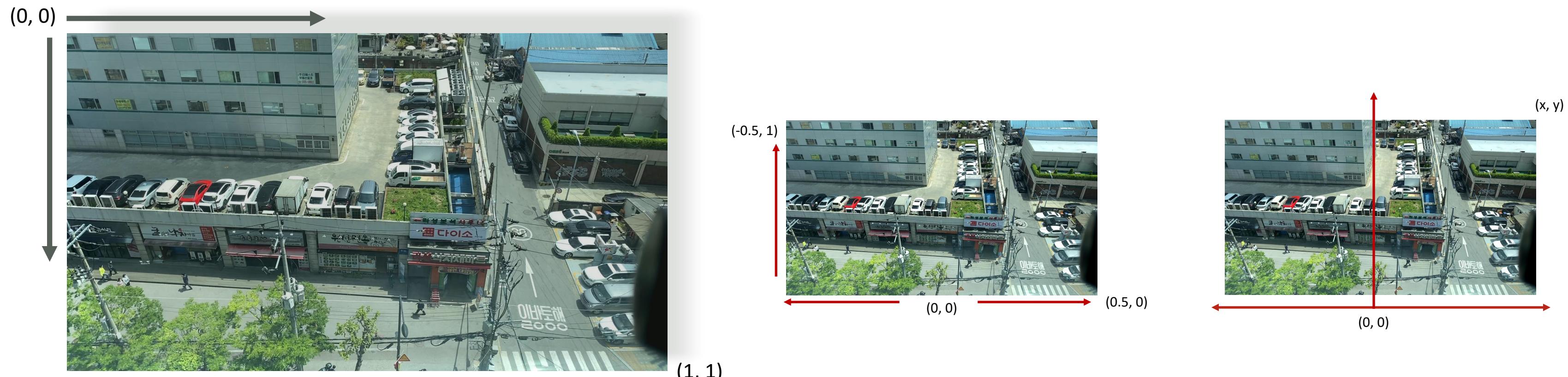
## [해결] 공중에서 촬영한 이미지/ 영상 데이터 수집

- 지상 8층 높이의 건물에서 삼각대로 카메라 고정 후 촬영
- Iriun을 사용하여 휴대폰 카메라를 웹캠으로 활용
- 유동인구가 많은 점심시간대인 11 ~ 14시 동안 2천 장의 이미지 수집
- Appendix에 이미지 자동저장 코드 첨부

## 거리 추정 사진 내 위치 좌표로 거리를 출력하는 함수식 도출 & 적용

### 3차원 좌표계 변환을 통한 Distance Estimation

- 사진 내 픽셀 위치값(0~1 사이의 값)을 좌표 변환 후, 변환된 픽셀 값을 실제 거리로 환산



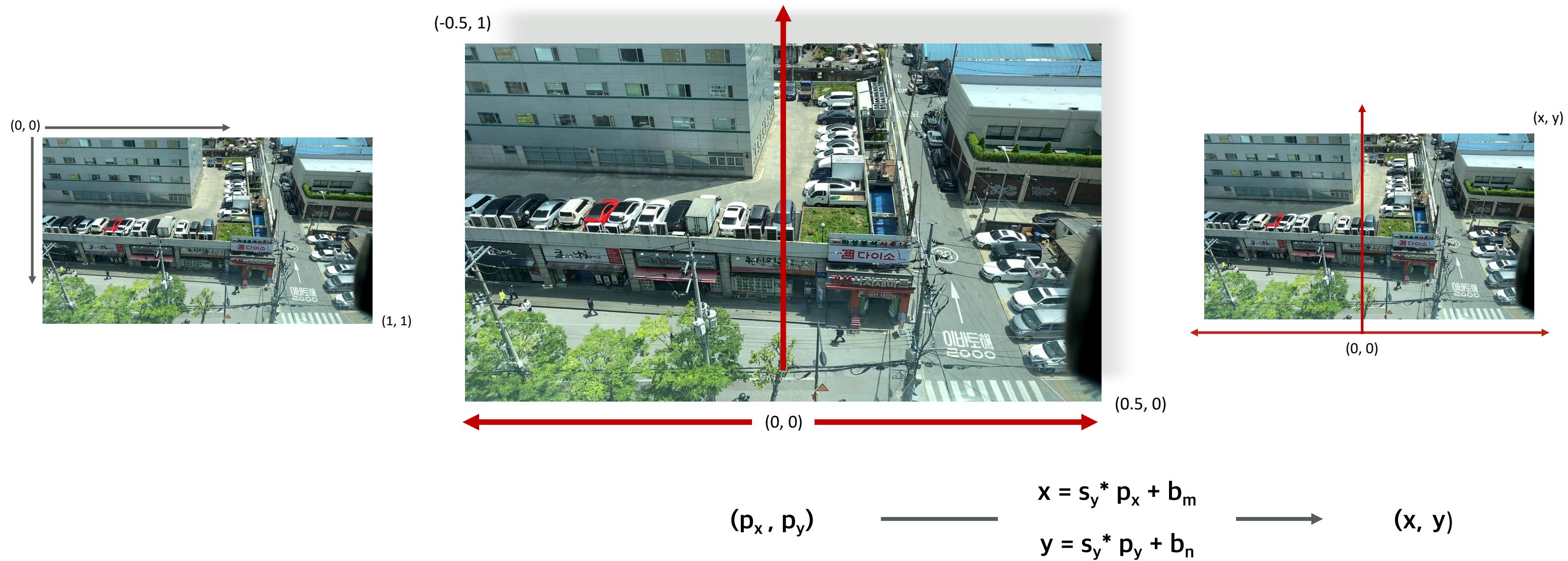
$$\begin{array}{c}
 (x_0, y_0) \\
 \xrightarrow{\quad} \\
 x_0 \rightarrow x_0 - 0.5 (p_x) \\
 y_0 \rightarrow 1 - y_0 (p_y)
 \end{array}
 \longrightarrow
 (p_x, p_y)$$



## 거리 추정 사진 내 위치 좌표로 거리를 출력하는 함수식 도출 & 적용

### 3차원 좌표계 변환을 통한 Distance Estimation

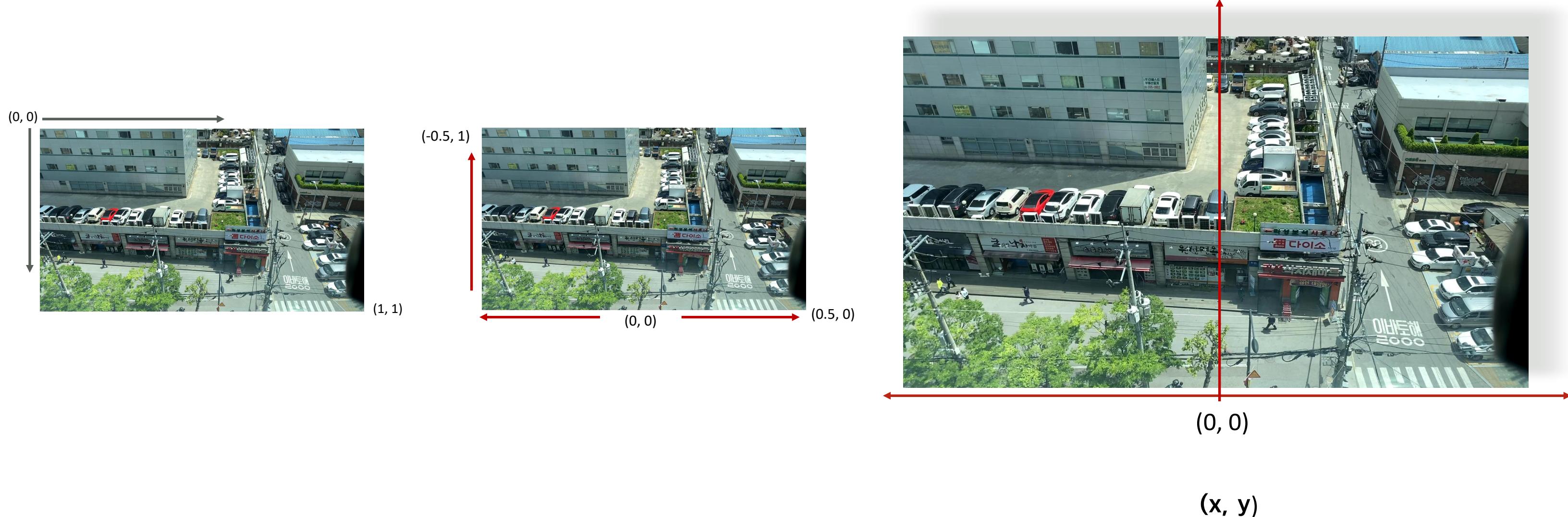
- 사진 내 픽셀 위치값(0~1 사이의 값)을 좌표 변환 후, 변환된 픽셀 값을 실제 거리로 환산



## 거리 추정 사진 내 위치 좌표로 거리를 출력하는 함수식 도출 & 적용

### 3차원 좌표계 변환을 통한 Distance Estimation

- 사진 내 픽셀 위치값(0~1 사이의 값)을 좌표 변환 후, 변환된 픽셀 값을 실제 거리로 환산



## 거리 추정 사진 내 위치 좌표로 거리를 출력하는 함수식 도출 & 적용

### 3차원 좌표계 변환을 통한 Distance Estimation

- 변환식 ( $p_x \rightarrow x, p_y \rightarrow y$ ) 및 계수 도출
- 원근에 따른 변화를 반영하기 위해 scale factor ( $s_y$ ) 적용함



#### [가정]

임의의 한 픽셀에서

x방향으로의 단위 픽셀에 대한 실제 거리값과

y방향으로의 단위 픽셀에 대한 실제 거리값은 동일하다

$$s_y = a_y * p_y + b_y$$

$s_y$  : y에 대한 scale factor

$$x = s_y * p_x + b_m$$

$p_x, p_y$  : 변환된 x축, y축의 픽셀값

$$y = s_y * p_y + b_n$$

(0~1 사이의 값을 가짐)

$x, y$  : 환산된 실제 거리 (meter 단위)

$b_m, b_n$  : bias

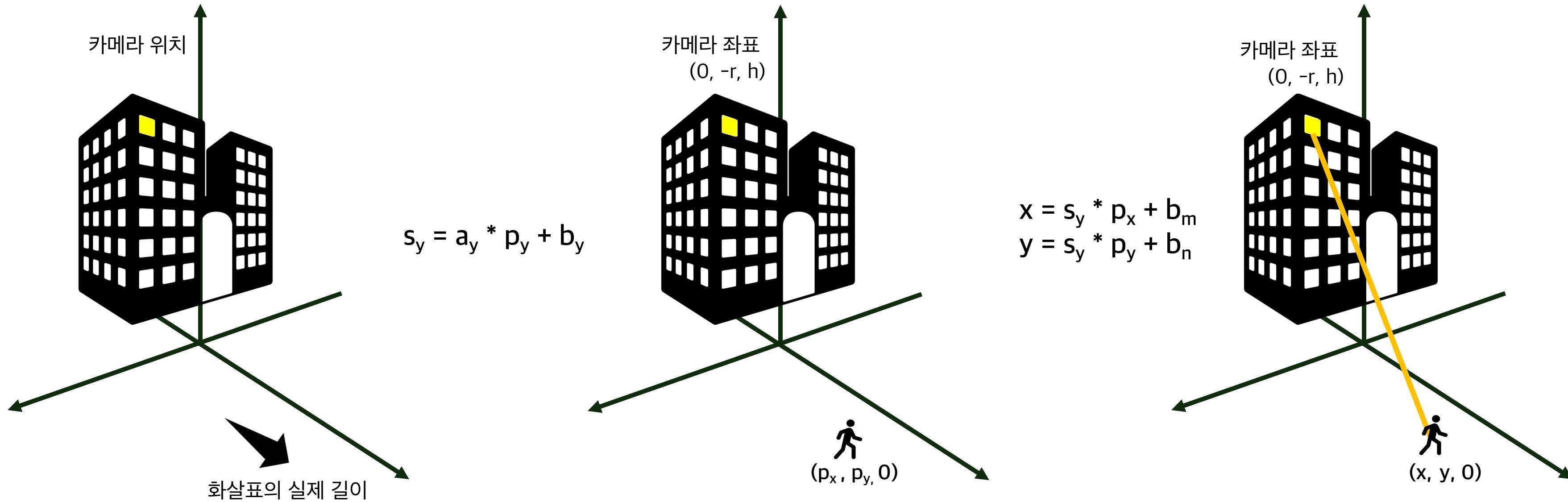
## 거리 추정 사진 내 위치 좌표로 거리를 출력하는 함수식 도출 & 적용

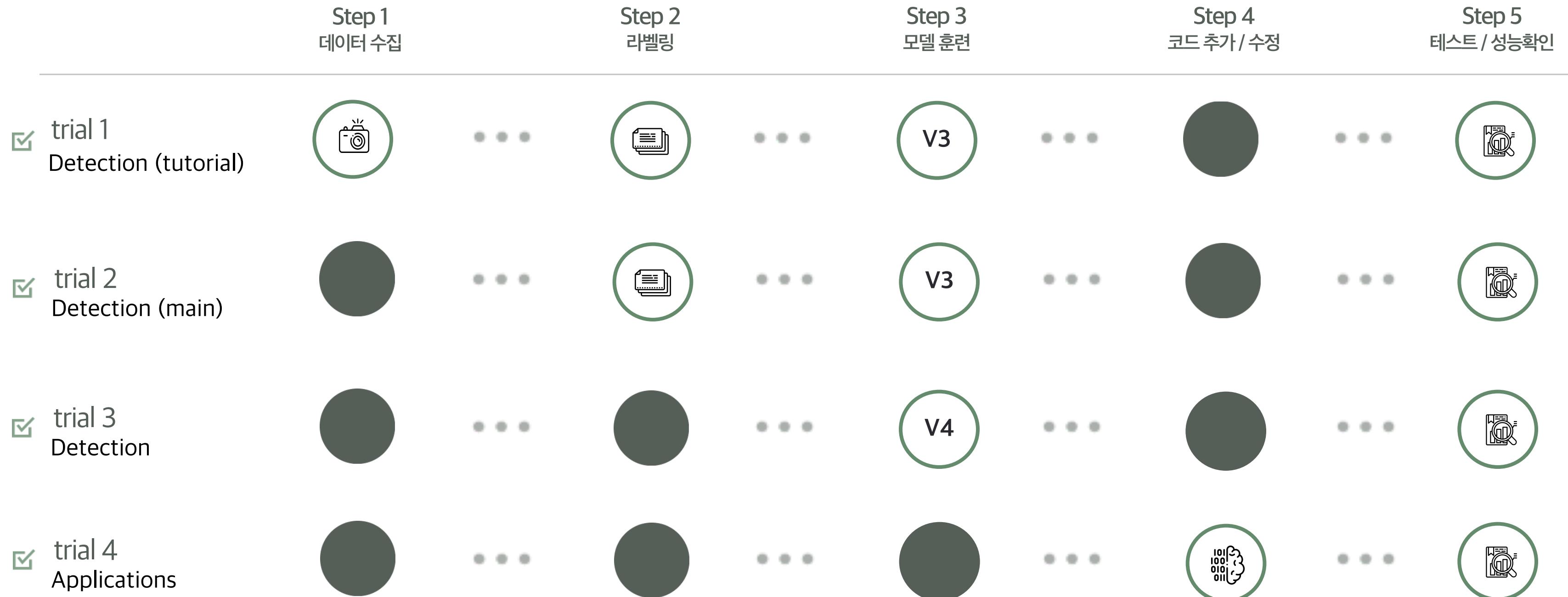
### 3차원 좌표계 변환을 통한 Distance Estimation

- 변환식 ( $p_x \rightarrow x, p_y \rightarrow y$ ) 및 계수 도출
- 원근에 따른 변화를 반영하기 위해 scale factor ( $s_y$ ) 적용함

$h$  : 카메라의 실제 높이

$r$  : 좌표변환된 이미지의 원점과 카메라 간 실제 거리

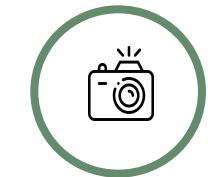


 프로세스

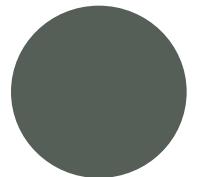
## Trial 1. YOLO v3 for Object Detection

Step 1  
데이터 수집

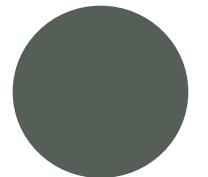
Detection (tutorial)



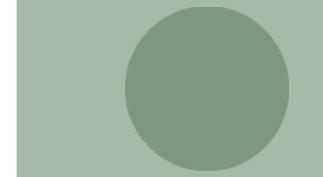
Step 2  
라벨링



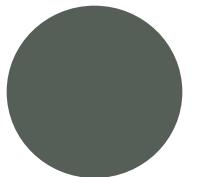
Step 3  
모델 훈련



Step 4  
코드 추가/수정



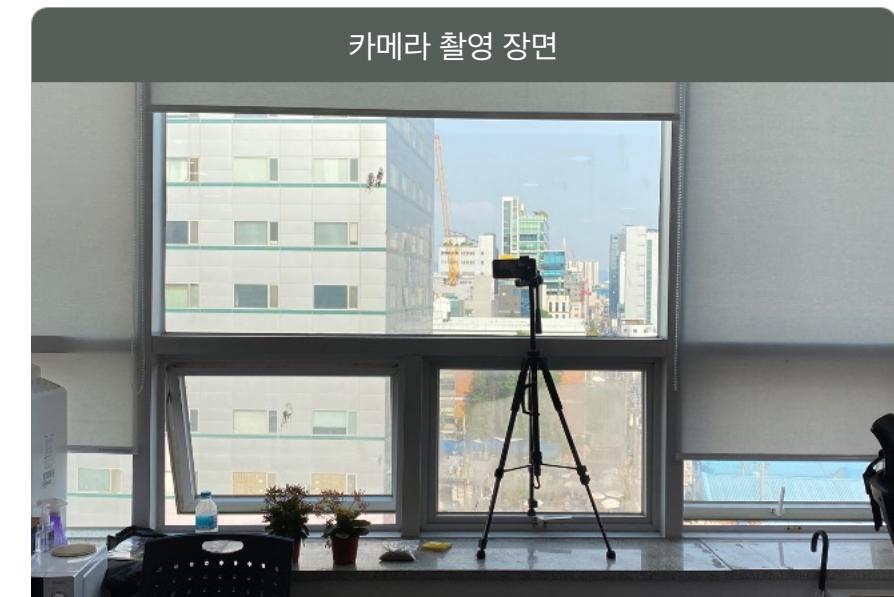
Step 5  
테스트/성능확인



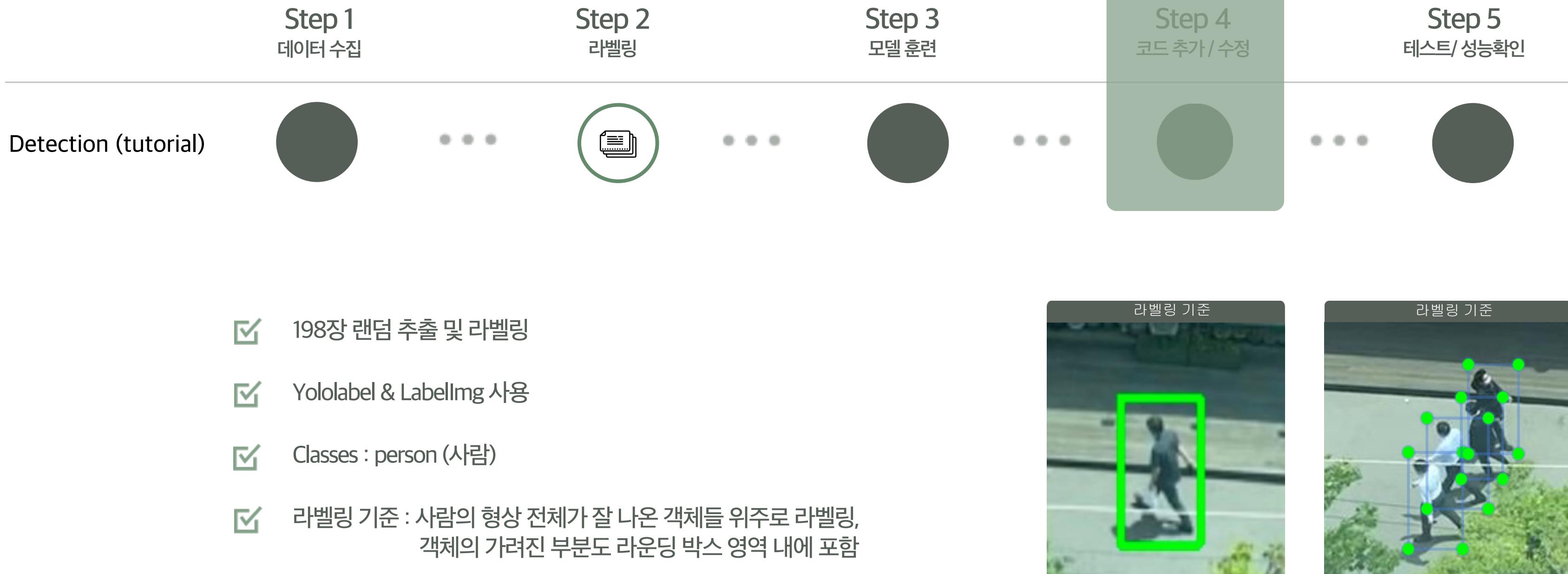
삼각대 & iriun 앱 사용

카메라 위치 고정

유동인구가 많은 점심시간대인 11~14시 동안  
4초 간격으로 총 2,000장 촬영



## Trial 1. YOLO v3 for Object Detection



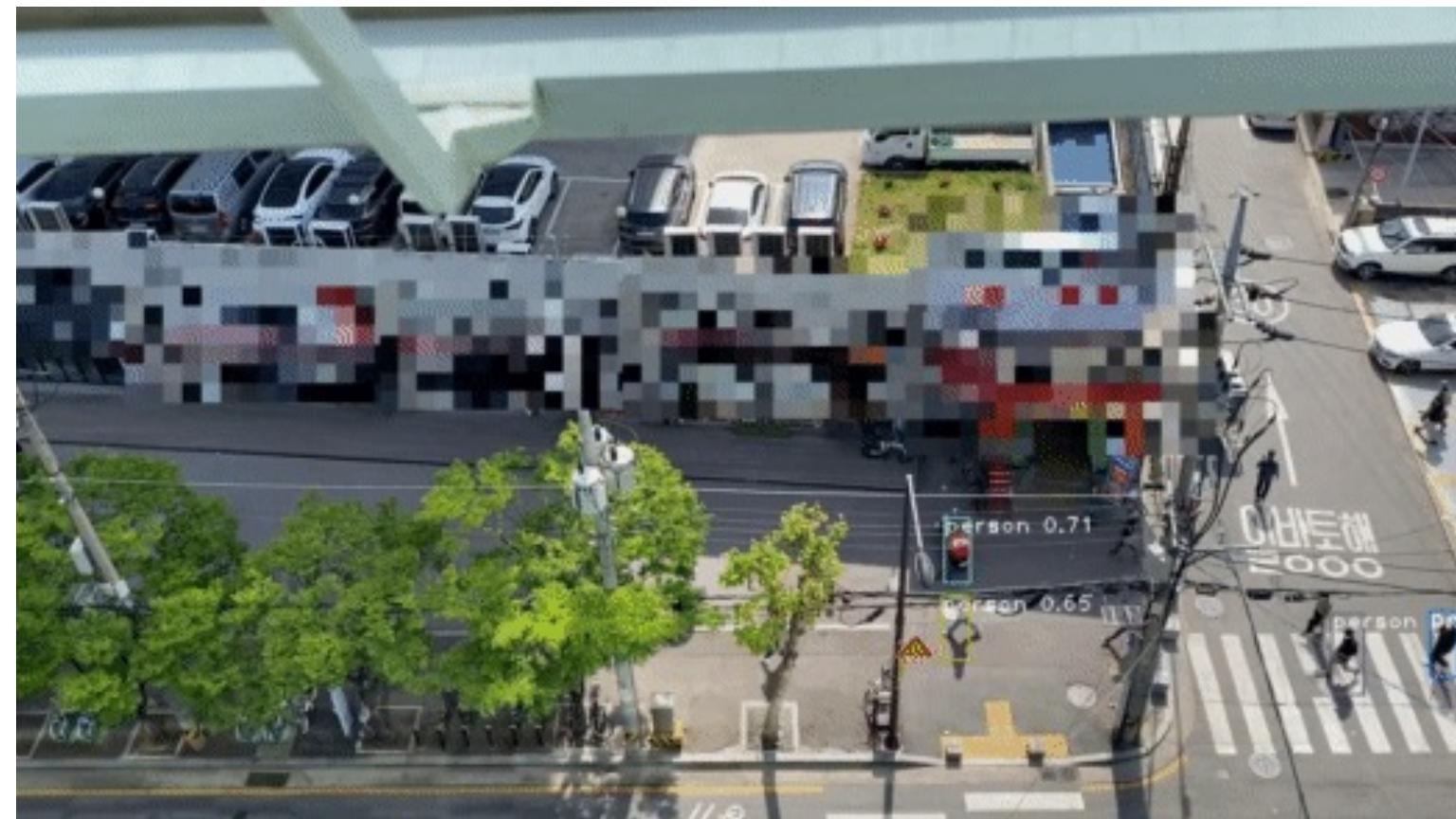
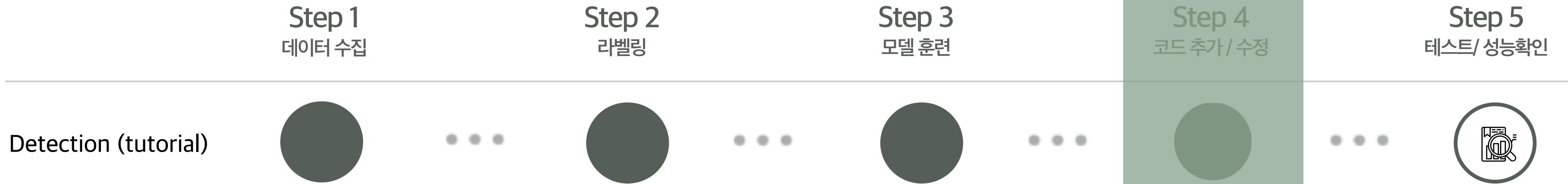
## Trial 1. YOLO v3 for Object Detection



- cfg 설정
- 1,000 epoch 돌리고 중간 중지하였음
- [convolutional] yolo layer 직전의 layers

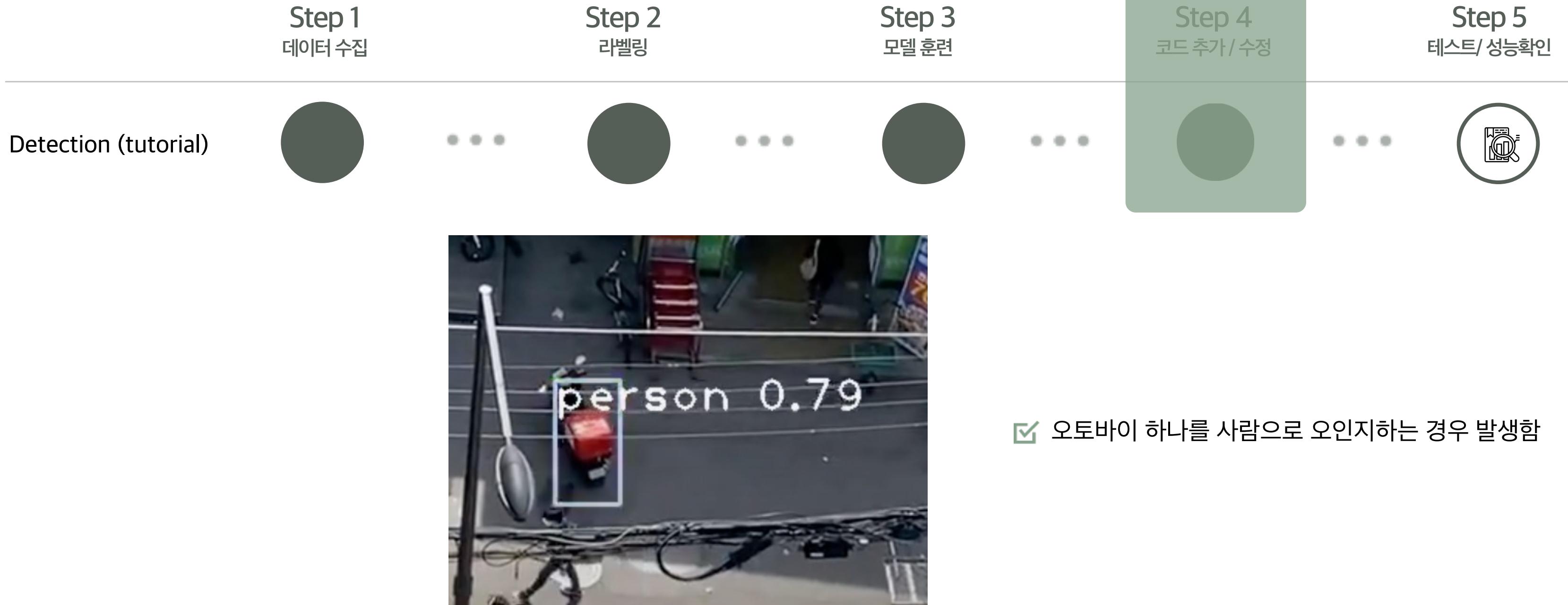
layer	parameter	default	customizing
net	batch	1	64
	Subdivisions	1	16
	max_batches	500200	2000
yolo	classes	80	1
convolutional	filters	255	18

## Trial 1. YOLO v3 for Object Detection

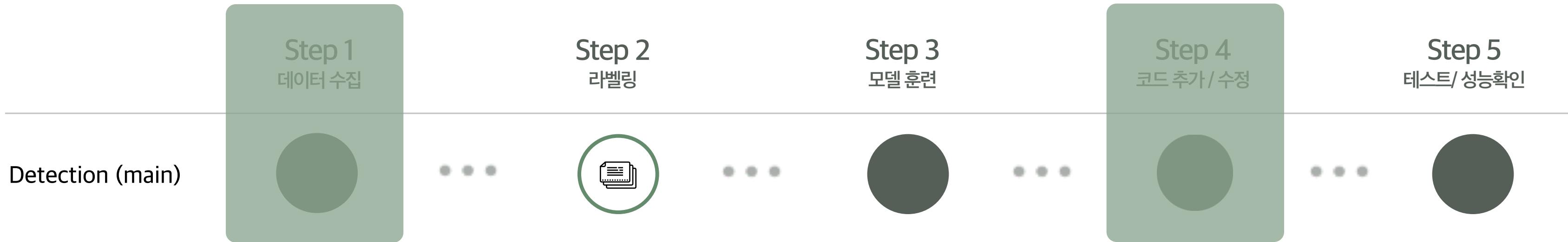


훈련을 도중 중지하였음에도 양호한 detection 성능을 보임

## Trial 1. YOLO v3 for Object Detection



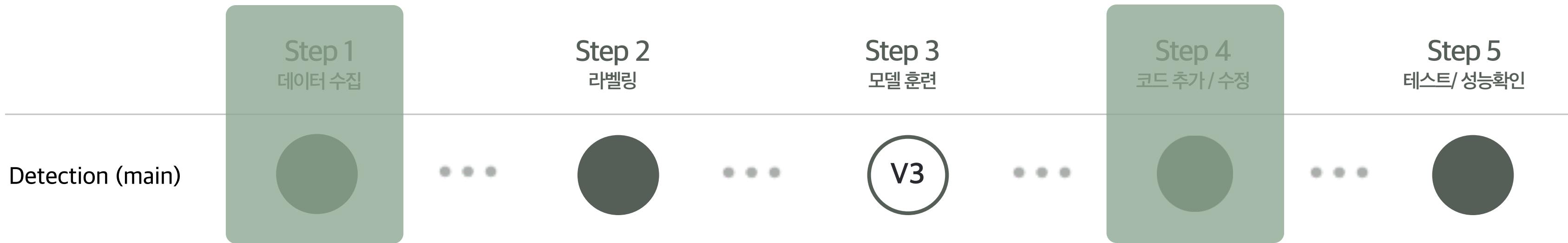
## Trial 2. YOLO v3 for Object Detection



- 500장으로 확장하여 랜덤 추출 및 라벨링
- Classes : pedestrian (보행자) , bike (이륜차) , person on bike (보행자와 이륜차 위에 앉은 사람)로 확대
- 라벨링 기준 : 상체 혹은 하체만 나와도 보행자로 라벨링  
가려진 부분은 '그 부분만' or '전체'를 혼합하여 진행



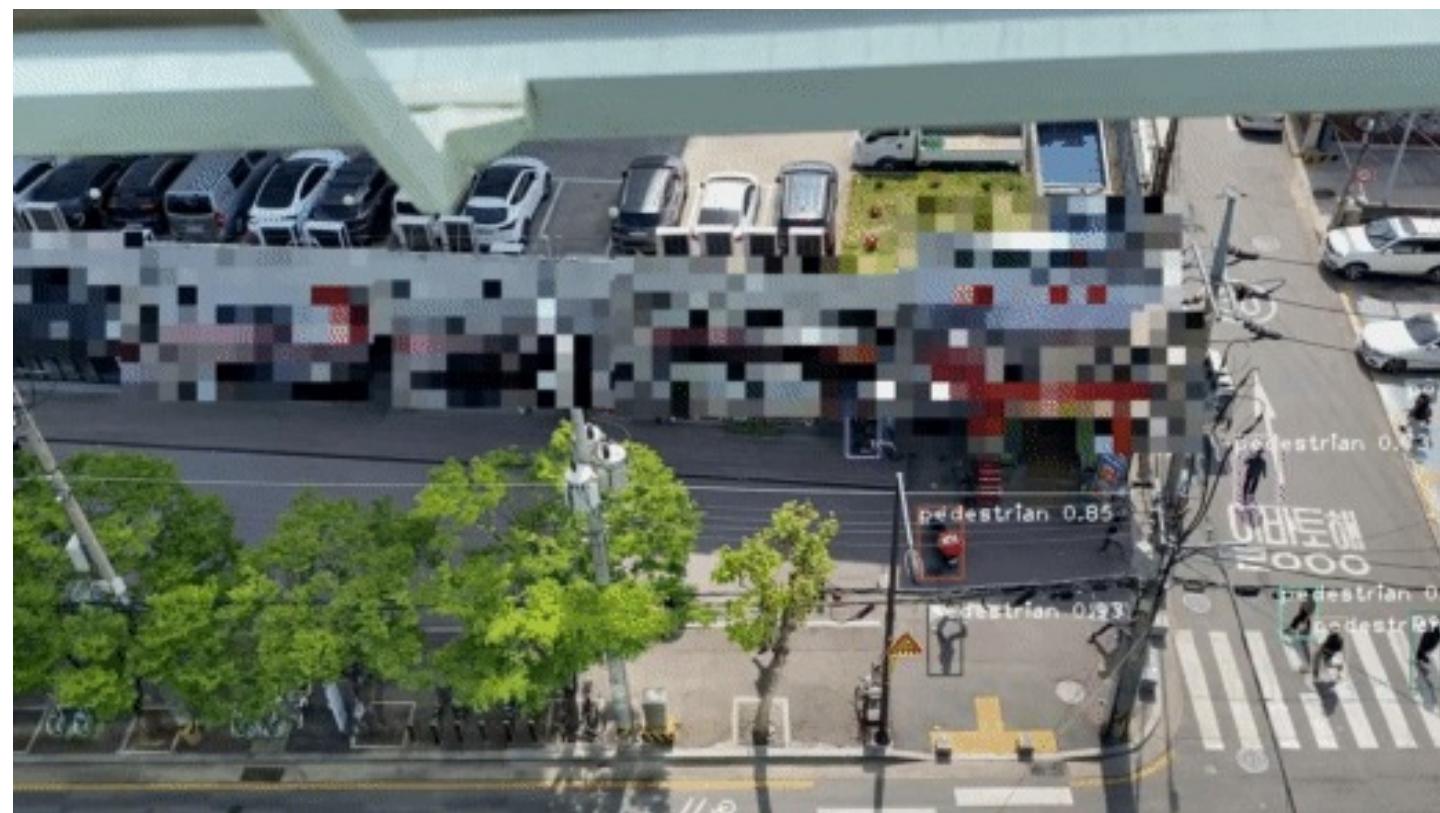
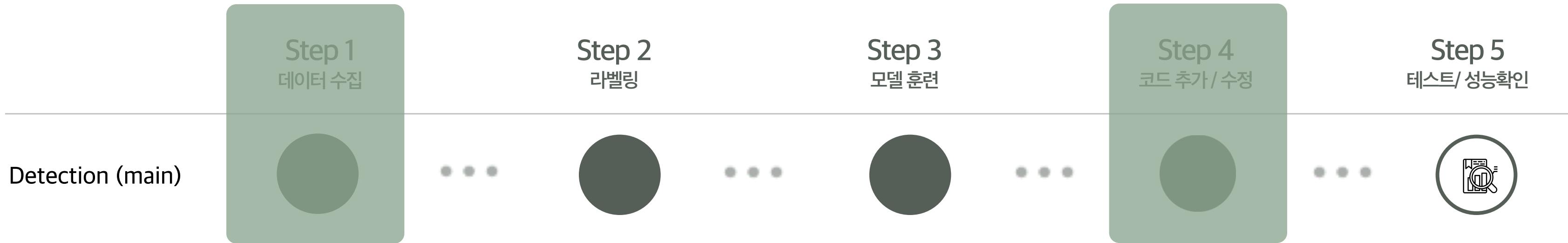
## Trial 2. YOLO v3 for Object Detection



- cfg 수정
- 6,000 epoch (약 12시간 소요)
- [convolutional] yolo layer 직전의 layers

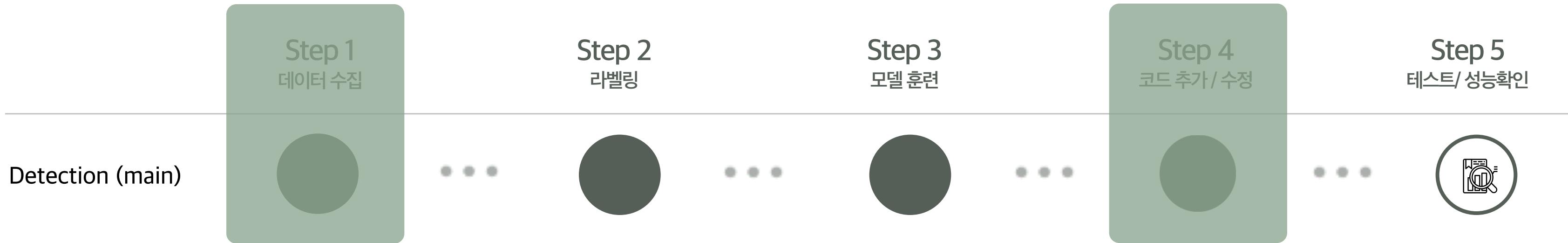
layer	parameter	default	customizing
net	batch	1	64
	Subdivisions	1	16
	max_batches	500200	<b>6000</b>
yolo	classes	80	3
convolutional	filters	255	24

## Trial 2. YOLO v3 for Object Detection



- Trial 1 보다 사람을 더 잘 잡아내고 confidence도 높아짐
- 여전히 오토바이 하나를 보행자로 잘못 인지함
- 요구르트 카트까지 보행자로 오인하는 경우 발생함

## Trial 2. YOLO v3 for Object Detection



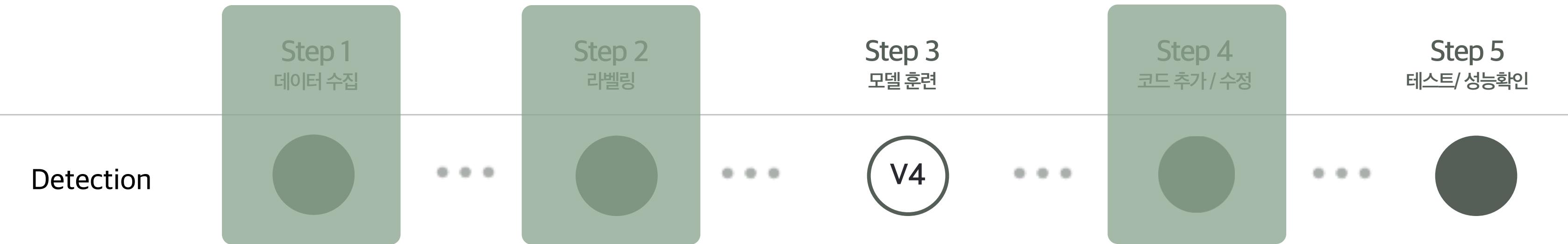
- Trial 1 보다 사람을 더 잘 잡아내고 confidence도 높아짐
- 여전히 오토바이 하나를 보행자로 잘못 인지함
- 요구르트 카트까지 보행자로 오인하는 경우 발생함

[보행자로 잘못 인지된 요구르트 카트 (좌), 오토바이 (우)]

## Trial 3. YOLO v4를 활용한 이유

- YOLO v3를 활용한 Tracking의 레퍼런스 부족
- 찾아낸 소수 레퍼런스들 간 Tensorflow 버전이 혼재  
→ YOLO v3 ~ v5 중 비교적 tracking관련 레퍼런스가 풍부했던 v4를 활용하기로 결정함

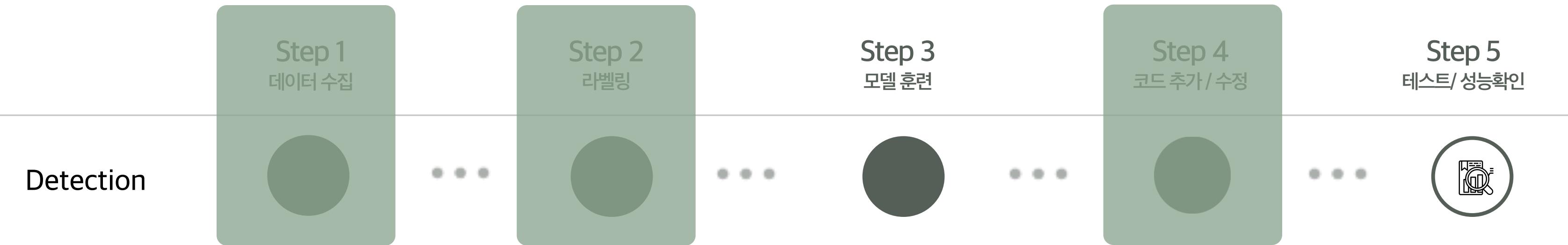
## Trial 3. YOLO v4 for Object Detection



- cfg 수정
- 6,000 epoch (약 30시간 소요)
- [convolutional] yolo layer 직전의 layers

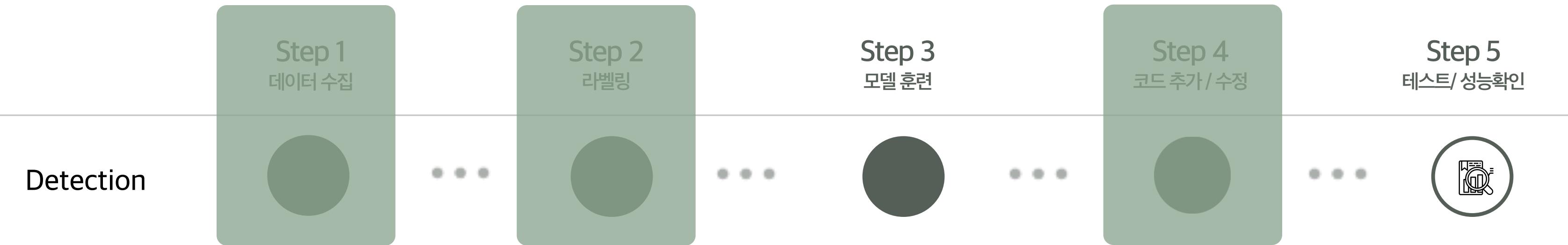
layer	parameter	default	customizing
net	batch	1	64
	Subdivisions	8	64
	max_batches	500200	6000
	steps	400000, 450000	4800, 5400
yolo	classes	80	3
convolutional	filters	255	24

## Trial 3. YOLO v4 for Object Detection



- Trial 2 (YOLO v3)와 유사한 성능을 보임
- 총 2개 영상에 대해 진행
  - test 영상의 모자이크 처리 시 불편함 존재
  - 앵글을 고정하여 새로운 영상 촬영

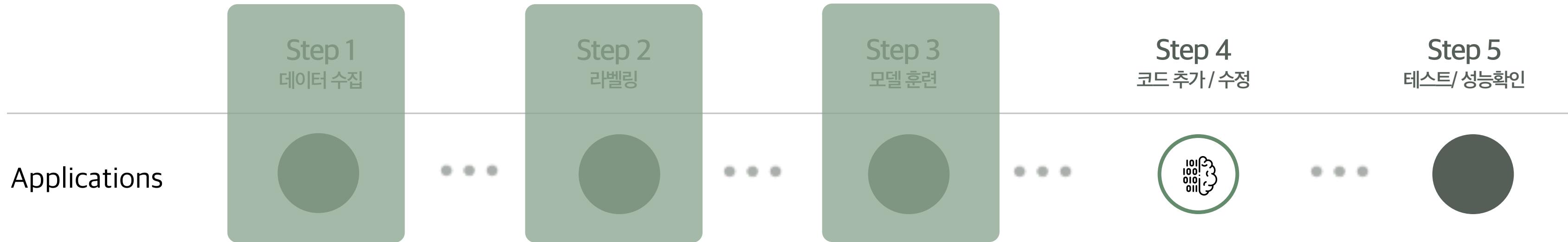
## Trial 3. YOLO v4 for Object Detection



[보행자로 잘못 인지된 오토바이]

- Trial 2 (YOLO v3)와 유사한 성능을 보임
- 총 2개 영상에 대해 진행
  - test 영상의 모자이크 처리 시 불편함 존재
  - 앵글을 고정하여 새로운 영상 촬영

## Trial 4. YOLO v4 for Applications

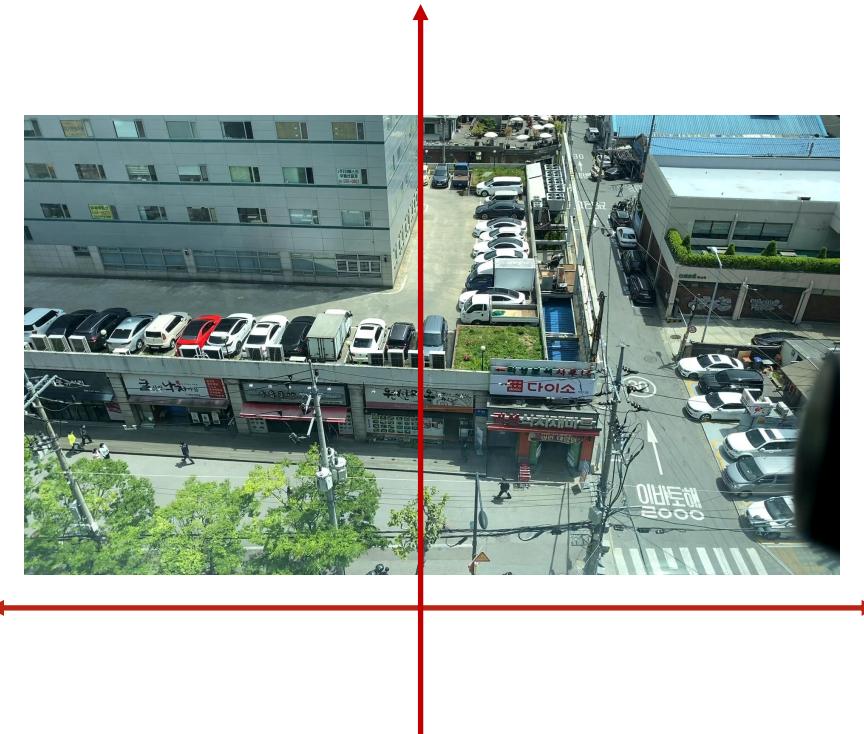


### Deep SORT를 활용한 Object Tracking & Counting

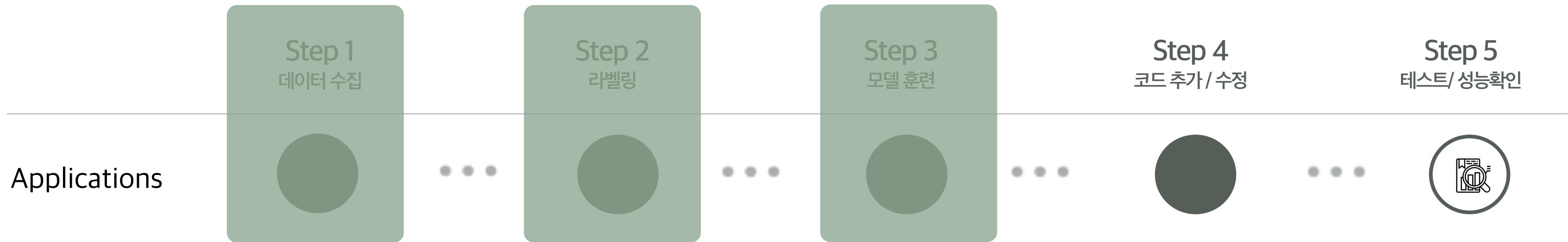
- 레퍼런스를 참고해 tracking 코드를 실행
- 특정 구역에 대한 누적/실시간 Counting 코드 추가

### 3차원 좌표계 변환을 통한 Distance Estimation

- 동일한 3차원 좌표계에 카메라와 object들을 반영
- Euclidean distance를 활용함



## Trial 4. YOLO v4 for Applications



### GOOD

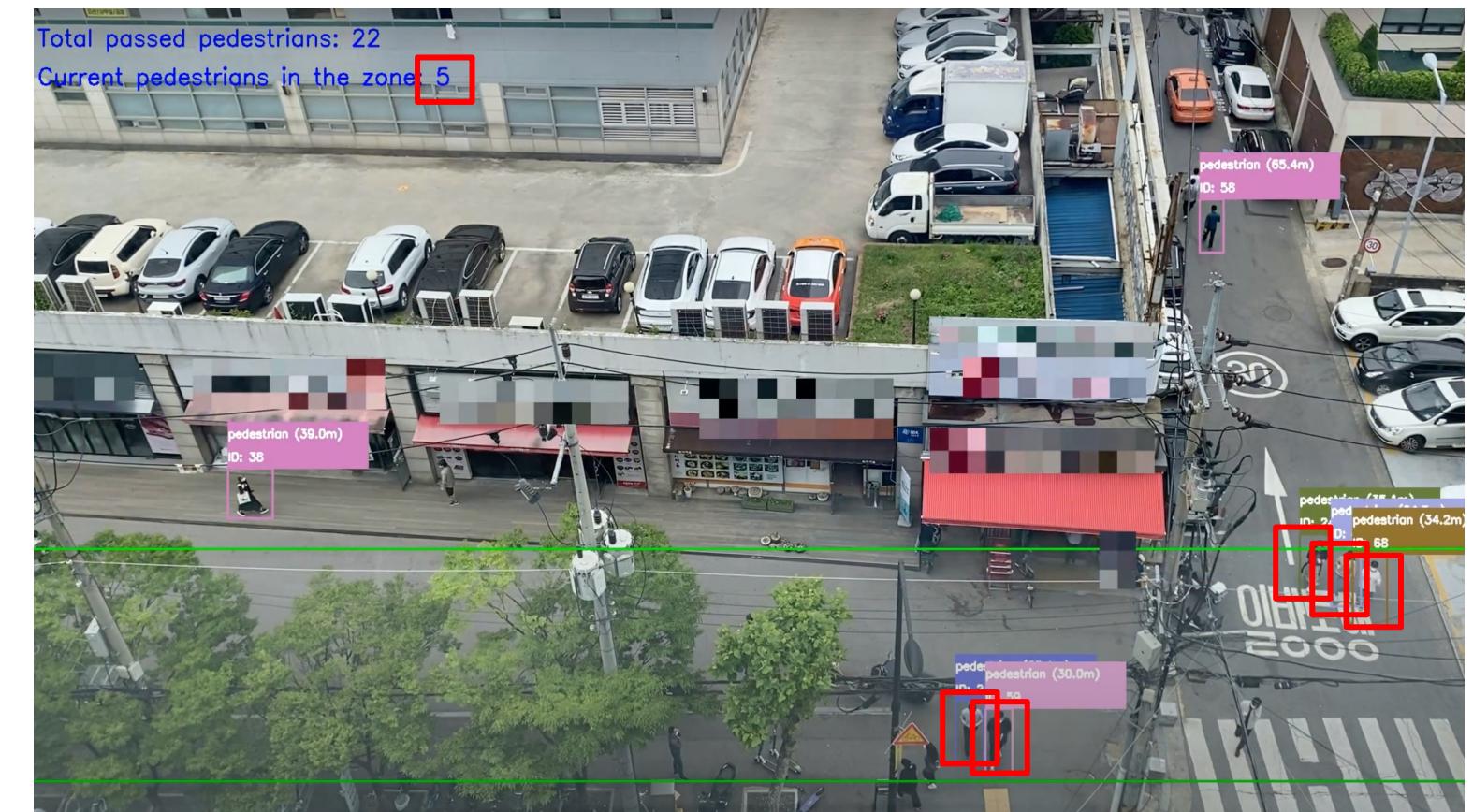
트래킹과 카운팅 모두 양호

추정한 거리도 실제값과 1% 내외의 오차율로 우수한 정확도를 보임  
(이미지 전체 세로축(y축) 길이 기준)

### BAD

트래킹의 경우, 보행자가 진행 방향을 심하게 바꾸었을 때

(ex. 측면->정면) 별개의 객체로 인식하는 경우 발생



[Tracking & 지정한 영역 내 객체 Counting (빨간 박스)]

## ✓ Trial 4. YOLO v4 for Applications

### 3차원 좌표계 변환을 통한 Distance Estimation

- flags 설정

```
31 # define flags
32 flags.DEFINE_string('framework', 'tf', '(tf, tflite, trt')
33 flags.DEFINE_string('weights', './checkpoints/yolov4-416',
34 | | | | |   'path to weights file')
35 flags.DEFINE_integer('size', 416, 'resize images to')
36 flags.DEFINE_boolean('tiny', False, 'yolo or yolo-tiny')
37 flags.DEFINE_string('model', 'yolov4', 'yolov3 or yolov4')
38 flags.DEFINE_string('video', './data/video/test.mp4', 'path to input video or set to 0 for webcam')
39 flags.DEFINE_string('output', None, 'path to output video')
40 flags.DEFINE_string('output_format', 'XVID', 'codec used in Videowriter when saving video to file')
41 flags.DEFINE_float('iou', 0.45, 'iou threshold')
42 flags.DEFINE_float('score', 0.50, 'score threshold')
43 flags.DEFINE_boolean('dont_show', False, 'dont show video output')
44 flags.DEFINE_boolean('info', False, 'show detailed info of tracked objects')
45
46 ##### Uncomment the 'count' flag which was already made, as we will replace it to the one we newly coded
47 # flags.DEFINE_boolean('count', False, 'count objects being tracked on screen')
48
49 ##### Add two new flags
50 flags.DEFINE_boolean('count', False, 'count objects which have passed specified area')
51 flags.DEFINE_boolean('distance', False, 'estimate distances between the camera and objects')
```

## ✓ Trial 4. YOLO v4 for Applications

### 3차원 좌표계 변환을 통한 Distance Estimation

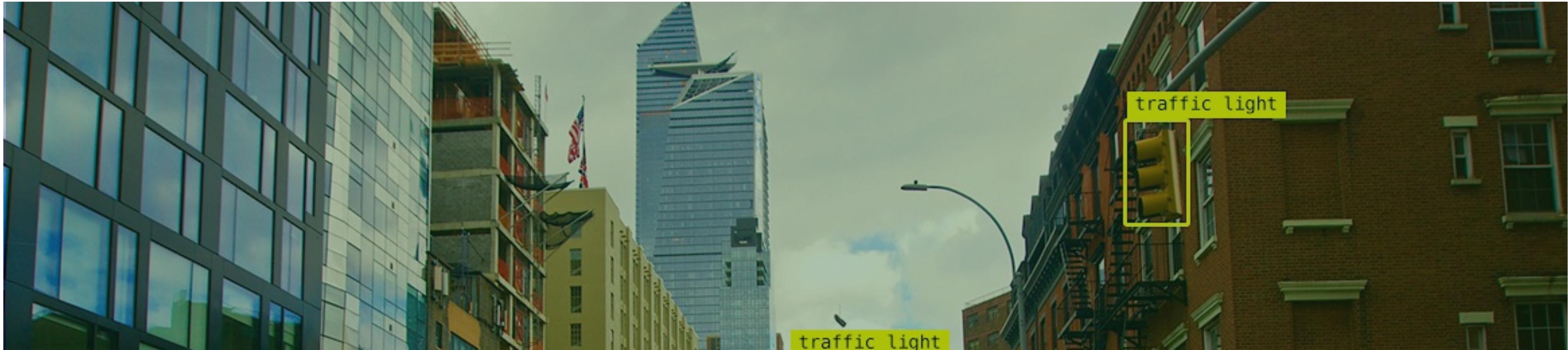
- flags 설정

```
49 ##### Add two new flags
50 flags.DEFINE_boolean('count', False, 'count objects which have passed specified area')
51 flags.DEFINE_boolean('distance', False, 'estimate distances between the camera and objects')
```

```
4. Apply tracking & counting & distance estimation after flags update (only for 2nd test video)
[-] ▶ M↓
# Counting 0, Distance X
!python object_tracker_count_distance.py --video ./data/video/test_2nd.MOV --output ./outputs/
results_yes_count_no_distance.mp4 --output_format DIVX --model yolov4 --count --dont_show --info

[-] ▶ M↓
# Counting X, Distance 0
!python object_tracker_count_distance.py --video ./data/video/test_2nd.MOV --output ./outputs/
results_no_count_yes_distance.mp4 --output_format DIVX --model yolov4 --distance --dont_show --info

[-] ▶ M↓
# Counting 0, Distance 0
!python object_tracker_count_distance.py --video ./data/video/test_2nd.MOV --output ./outputs/
results_yes_count_yes_distance.mp4 --output_format DIVX --model yolov4 --distance --count --dont_show --info
```



# 03

## 프로젝트 결과

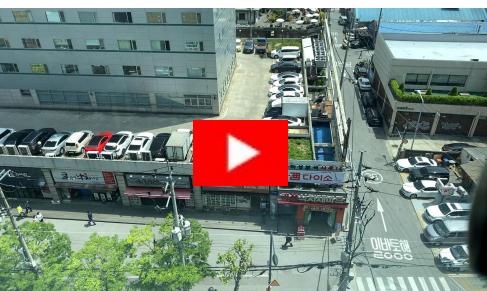
결과 영상

추후 보완

## 결과 영상

### Detection

Trial 1. YOLO v3 (demo)



### Detection

Trial 2. YOLO v3 (main)



### Detection

Trial3. YOLO v4



### Detection & Applications

Trial 4. Deep SORT



 **추후 보완****정량적인 성과 측정 지표 반영**

- 추후 성능 개선 여부를 논의하기 위해서는 정량 지표가 필요
- 라벨링된 테스트 데이터를 분지, mAP 등 정량적인 지표를 기준으로 성과 측정 필요

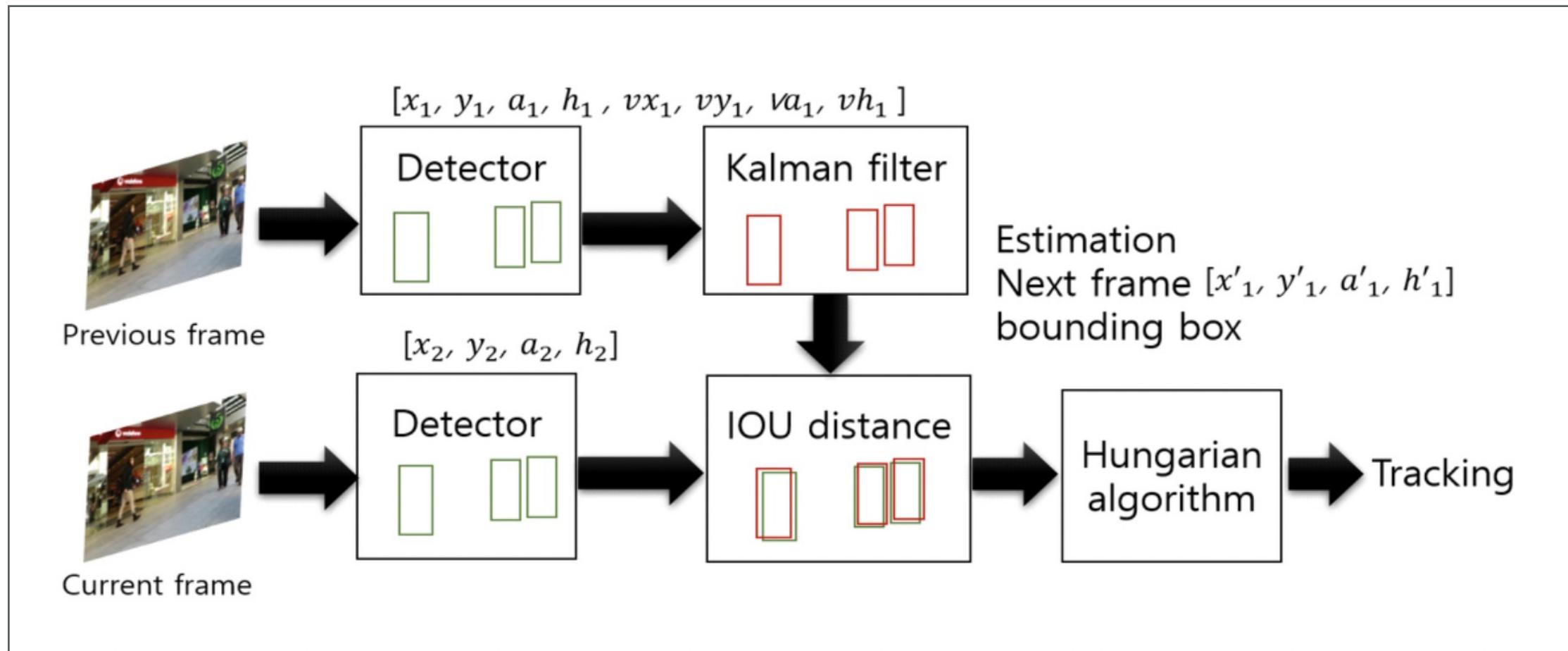
## ☒ Appendix

```
1 일정 주기마다 캡쳐할 수 있는 코드
2 import sys
3 import cv2
4 import time
5 import datetime
6
7
8 cap = cv2.VideoCapture(0)
9
10 if not cap.isOpened():
11     print("camera open failed!")
12     sys.exit()
13
14 fps = cap.get(cv2.CAP_PROP_FPS)
15 delay = round(1000/fps)
```

```
17 while True:
18     ret, frame = cap.read()
19     if not ret:
20         break
21
22     cv2.imshow('frame', frame)
23
24     now = datetime.datetime.now().strftime('%y%m%d_%H%M%S')
25     cv2.imwrite(f'210506/pic_{now}.jpg', frame)
26     time.sleep(4)
27
28     if cv2.waitKey(delay) == 27:
29         break
30
31
32
33 cap.release()
34 cv2.destroyAllWindows()
```

## ✓ Appendix

### SORT (Simple Online and Realtime Tracking)



### Deep SORT

SORT

+

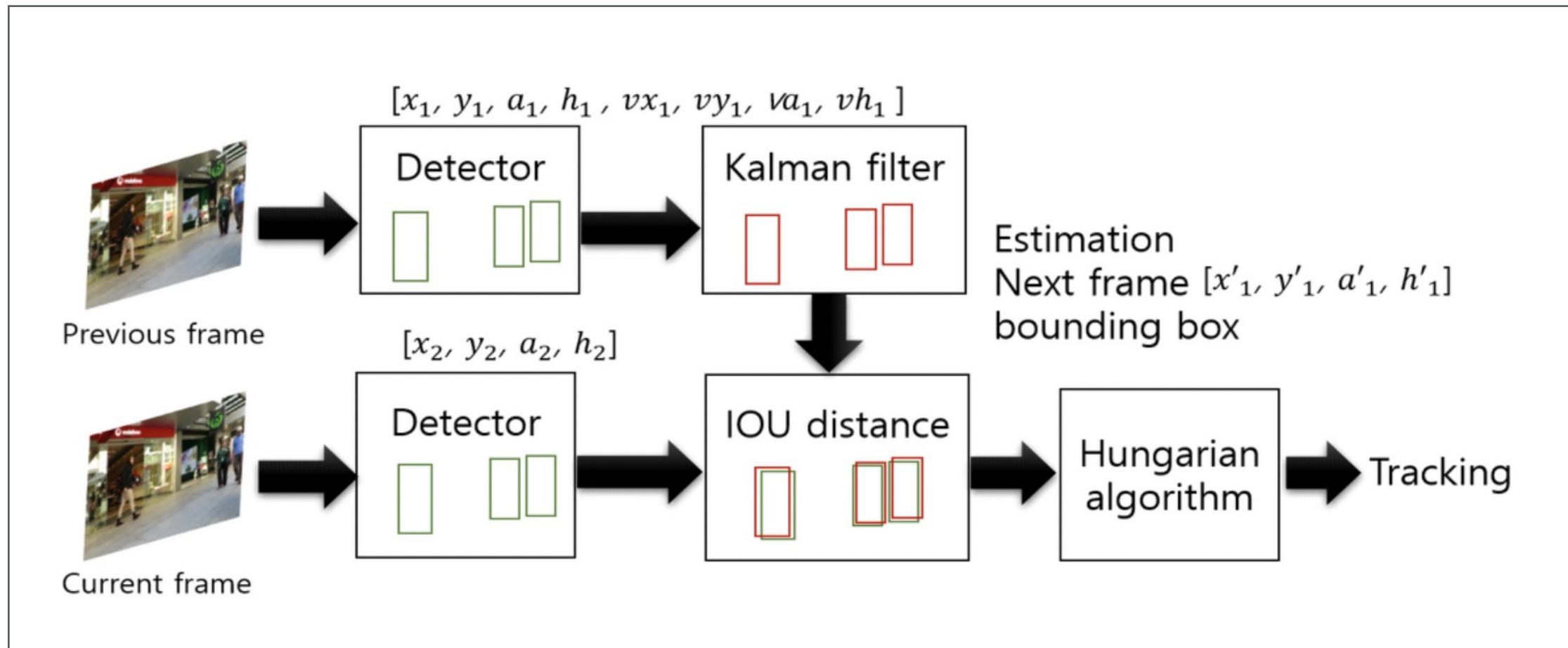
Deep Learning

#### 1) 위치 추정: Kalman Filter

- 이전 프레임에서 탐지된 객체의 bounding box를 이용하여 객체의 속도를 추정,  
현재 프레임에서 객체의 위치를 예측

## ✓ Appendix

### SORT (Simple Online and Realtime Tracking)



### Deep SORT

SORT

+

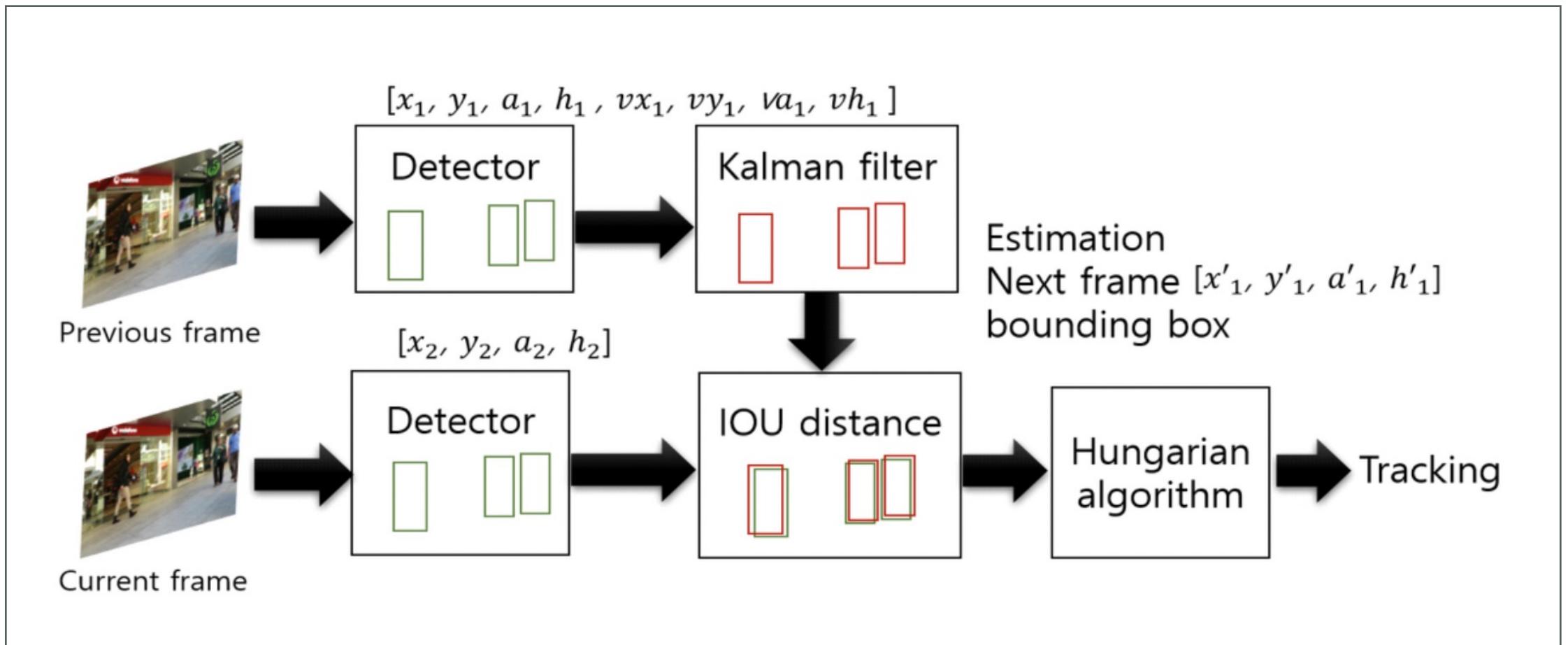
Deep Learning

#### 2) 매칭: Hungarian algorithm

- Kalman Filter를 통해 얻은 예측값과 실제값의 IOU distance를 이용하여 객체 정보를 업데이트
- 일종의 최적 matching algorithm
- 최소 IOU를 설정하여 그보다 낮은 경우 매칭 거부 가능

## ✓ Appendix

### SORT (Simple Online and Realtime Tracking)



### Deep SORT

SORT

+

Deep Learning

### 3) 한계

- 객체의 겹침이나 장애물로 인한 가림 문제에 취약

## Appendix

### **SORT (Simple Online and Realtime Tracking)**

#### 3) 한계

- 객체의 겹침이나 장애물로 인한 가림 문제에 취약



### **Deep SORT**

**SORT**

**+**

**Deep Learning**

#### 1) 딥러닝 기반 객체 추적 시스템 혼용

- 객체의 feature appearance vector를 반영하여 SORT의 한계를 보완, 성능 개선

#### 2) 사용 알고리즘

- IOU distance algorithm을 사용하지 않음  
→ 대신 Mahalanobis distance algorithm을 적용
- CNN 기반 네트워크의 이미지 특징 정보 추출  
→ Deep cosine metric algorithm을 적용

## ✓ Appendix

# YOLO v3 & v4의 cfg 파일 디폴트값 차이

	Trial 1 (yolo v3)	Trial 2 (yolo v3)	Trial 3 (yolo v4)
[net]			
batch	64	64	64
subdivisions	16	16	64
width	416	416	608
height	416	416	608
channels	3	3	3
momentum	0.9	0.9	0.949
decay	0.0005	0.0005	0.0005
angle	0	0	0
saturation	1.5	1.5	1.5
exposure	1.5	1.5	1.5
hue	0.1	0.1	0.1
learning_rate	0.001	0.001	0.0013
burn_in	1000	1000	1000
max_batches	2000	6000	6000
policy	steps	steps	steps
steps	400000, 450000	400000, 450000	4800, 5400
scales	0.1, 0.1	0.1, 0.1	0.1, 0.1
[convolutional] (yolo 직전)			
filters	18	24	24
[yolo]			
classes	1	3	3
num	9	9	9
jitter	0.3	0.3	0.3
ignore_thresh	0.7	0.7	0.7
truth_thresh	1	1	1

## 1) Subdivisions

- (v3) 16으로 세팅해도 에러가 발생하지 않음  
(v4) 16으로 세팅 시 에러가 발생하며, 64로 세팅해야 코드가 정상 작동됨

## 2) Width & Height

- (v3) 디폴트값으로 416이 세팅되어 있음  
(v4) 디폴트값이 608이며, v3 대비 고해상도 값임

## 3) Momentum

- (v3) 디폴트값으로 0.9가 세팅되어 있음  
(v4) 디폴트값이 0.949이며, v3 대비 높은 값임

## 4) learning\_rate

- (v3) 디폴트값으로 0.001가 세팅되어 있음  
(v4) 디폴트값이 0.0013이며, v3 대비 높은 값임

## Appendix

### 기타 YOLO v3 & v4의 cfg 파일 차이

---

#### 1) 'mosaic' 변수 유무 여부

- (v3) 없음  
(v4) 1이라는 값이 할당되어 있음 (mosaic = 1)

#### 2) 활성함수 종류

- (v3) 레이어 전반에 걸쳐 'leaky'를 사용  
(v4) 전반에서는 'mish', 후반에서는 'leaky'를 사용

#### 3) 레이어의 수

- (v3) 107개  
(v4) 162개

#### 4) SPP 유무 여부

- (v3) 없음  
(v4) 있으며, 'maxpool'이라는 레이어를 가지고 있음

#### 5) 'yolo' 레이어의 mask값 순서

- (v3) 8부터 시작해서 0까지 1씩 감소  
(v4) 0부터 시작해서 8까지 1씩 증가

#### 6) 'yolo' 레이어의 anchor값 차이 및 파라미터 종류

- (v4) v3에는 없는 파라미터들이 있다
  - scale\_x\_y = 1.2
  - iou\_thresh=0.213
  - cls\_normalizer=1.0
  - iou\_normalizer=0.07
  - iou\_loss=ciou
  - nms\_kind=greedy\_nms
  - beta\_nms=0.6
  - max\_delta=5

E . O . D

