

# 고정카메라영상을 통한 차량탐지

AI 14기 류제욱

# 목차

01

프로젝트 목적

02

프로젝트 배경

03

프로젝트 구현 내용

04

프로젝트 수행 결과

05

프로젝트 고찰

# 1. 프로젝트 목적

01 딥러닝을 통한 객체 탐지에 대한 이해

02 실시간 영상처리를 통한 자동차 객체 탐지

03 자동차 객체 탐지 기술 응용

# 2. 프로젝트 배경

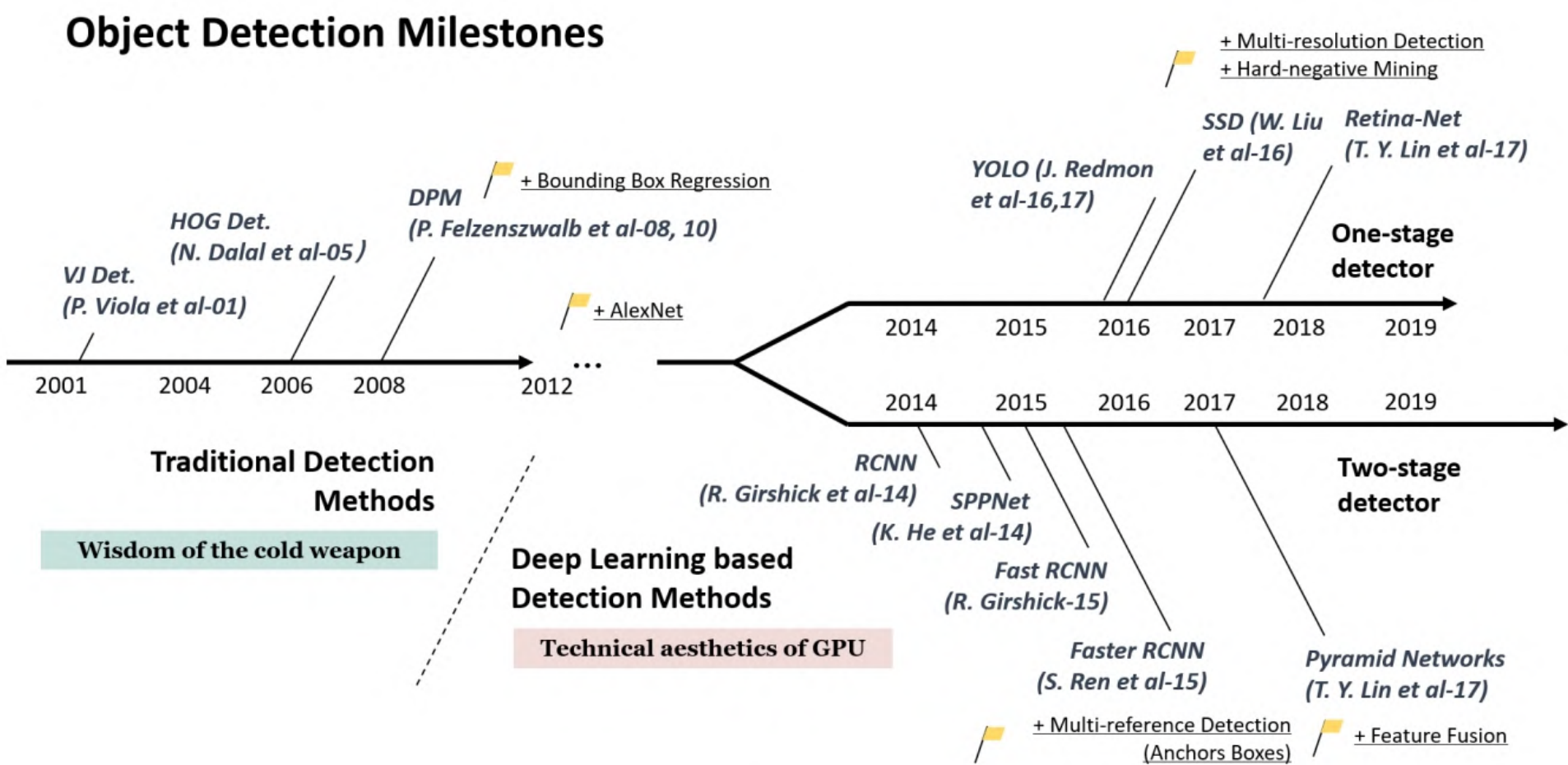
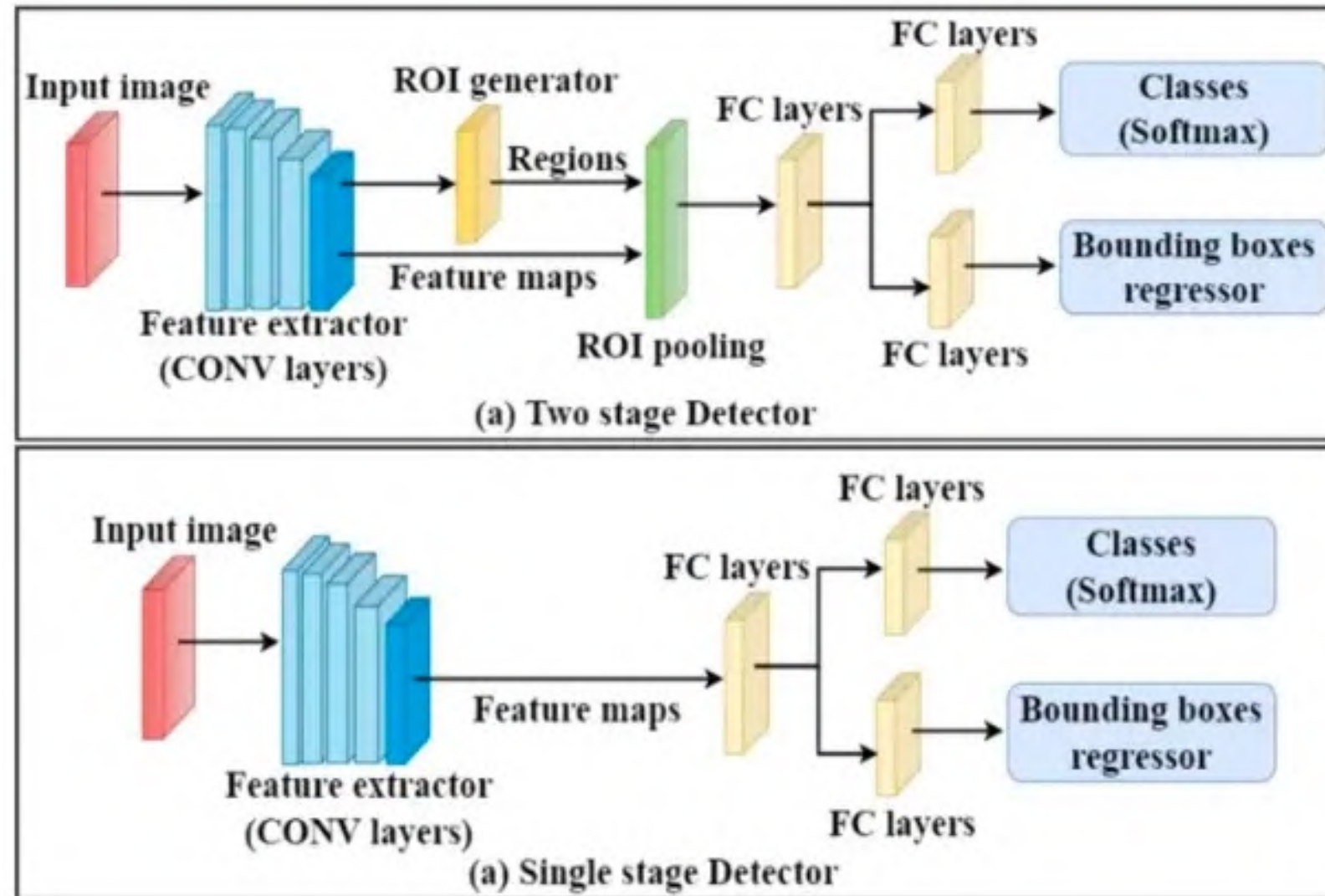


Fig. 2. A road map of object detection. Milestone detectors in this figure: VJ Det. [10, 11], HOG Det. [12], DPM [13–15], RCNN [16], SPPNet [17], Fast RCNN [18], Faster RCNN [19], YOLO [20], SSD [21], Pyramid Networks [22], Retina-Net [23].

- 객체 탐지 모델이 실질적으로 시작된 것은 Viola-Jones(2001) 모델
  - haar 필터를 통해 중요 특징들을 추출하는 모델
  - Sliding window를 사용해 모든 픽셀을 확인함으로 많은 시간 소요
- 2014년부터 R-CNN을 시작으로 딥러닝을 통한 객체 탐지 모델들이 개발되었음
- 2016년에 YOLO가 등장하며 제일 좋은 성능 보임과 동시에 처리 시간을 대량 단축
- 딥러닝 객체 탐지 대표적인 모델들로 R-CNN과 YOLO가 있음

## 2. 프로젝트 배경

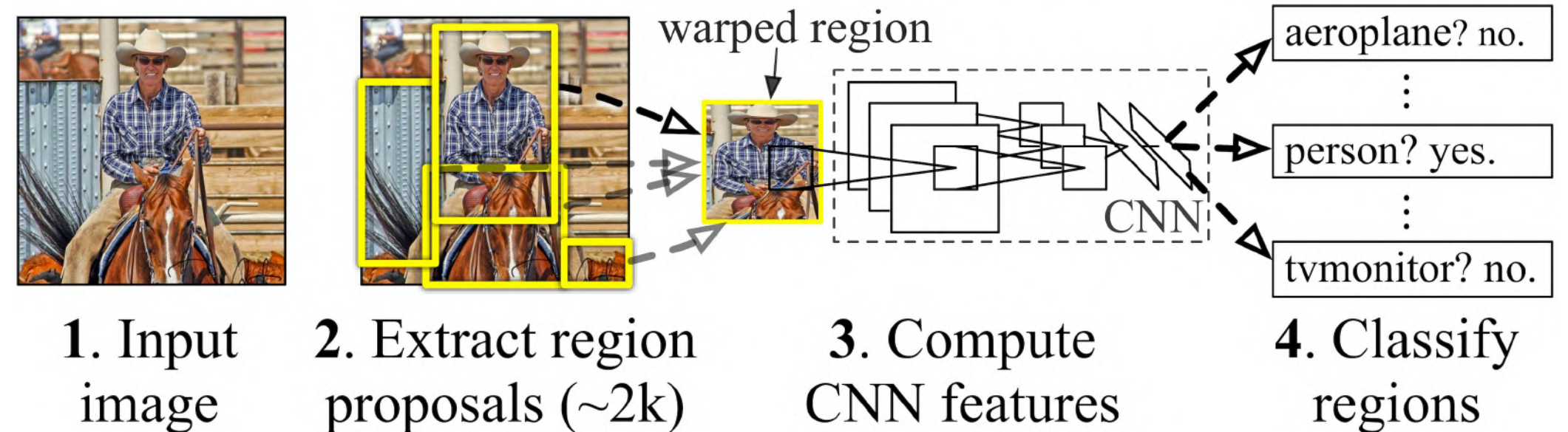


- 딥러닝 객체 탐지 모델은 크게 두가지 종류로 나뉨:
  - 단일단계 방식(i.e. YOLO, SSD)
  - 이단계 방식(i.e. R-CNN)
- 두 종류 모두 다 CNN 모델을 바탕으로 개발
- 제일 큰 차이점은 모델들의 처리 단계로 인한 탐지/분류 처리 시간
  - 이단계 방식은 후보영역을 선정한 후 해당 영역들에 대한 예측이 순차적으로 진행되었지만 단일단계 방식은 영역 제안을 함과 동시에 영역 내에 물체에 대한 예측을 진행

## 2. 프로젝트 배경

- R-CNN은 Selective Search를 통해 후보영역(RoI)들을 먼저 찾고 제안된 모든 영역에 대해 예측이 순차적으로 진행
  - warping으로 인해 이미지가 변형되고 모든 영역에 대해 예측을 진행하기 때문에 시간이 비교적 오래 걸린다
- 처리 시간이 길어 실시간 데이터 처리 사용에 적합하지 않았음
- 하지만 Fast R-CNN, Faster R-CNN등 모델을 개선하여 처리 시간을 단축시킴
  - Faster R-CNN에선 영역을 제안하는 단계에서 RPN을 통해 영역에 객체 포함 유무에 대해 이진 분류를 진행하여 후보영역을 추림

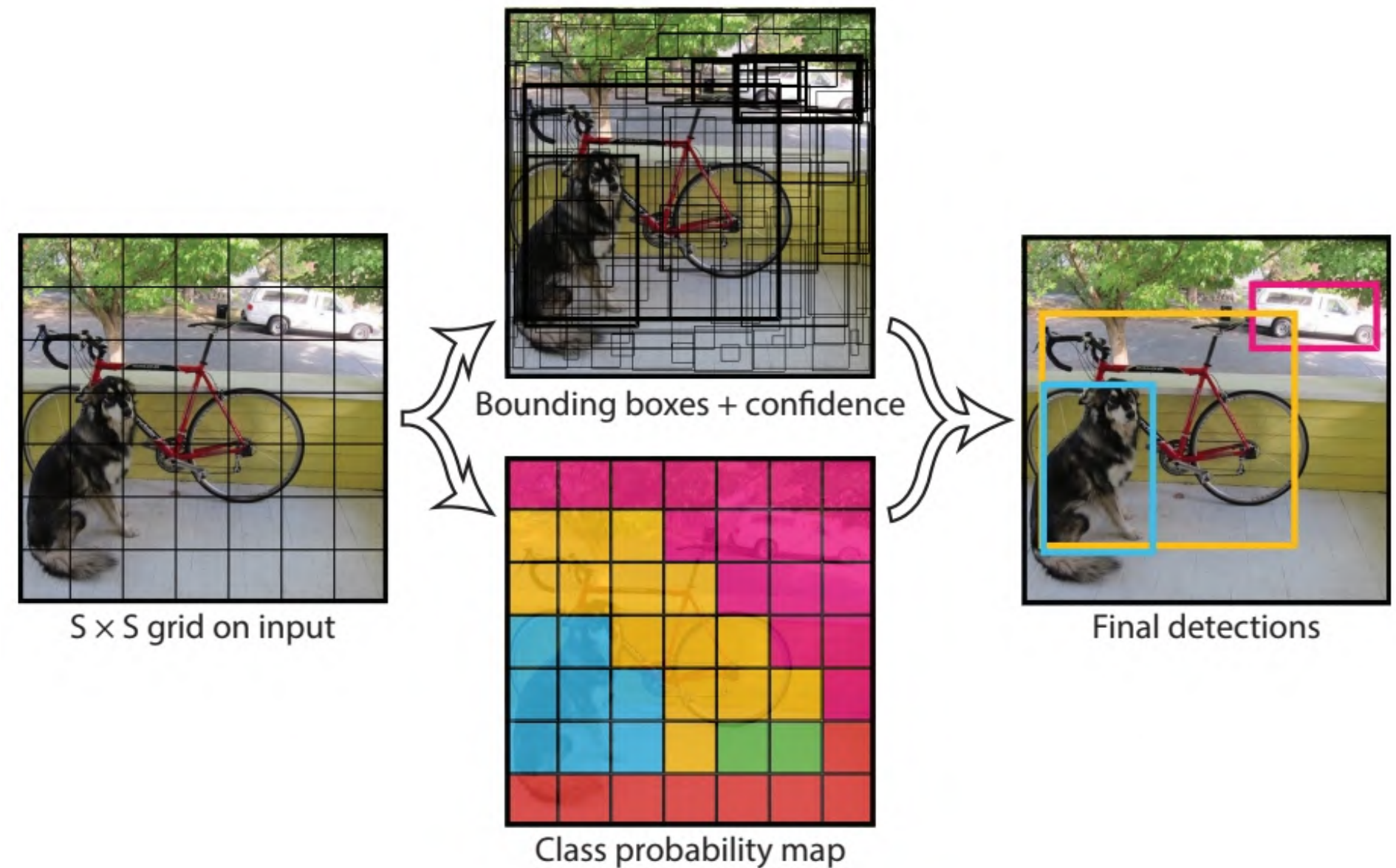
### R-CNN: *Regions with CNN features*





















## 2. 프로젝트 배경

- 대표적인 단일단계 모델로, input 데이터를 동일한 사이즈의 grid로 나눠 그리드 중심, bounding box와 신뢰도를 기반으로 객체를 예측
- 모든 과정을 한번에 같이 진행하므로 데이터 처리 시간이 짧음
  - 이단계 방식 모델보다 단일단계 방식 모델이 더 실시간 데이터 처리 사용에 적합함
- 하지만 정해진 사이즈의 grid로 나눠서 예측을 진행하기에 너무 작은 물체들에 대한 예측은 성능이 떨어짐
- 현재 v7까지 개발



## 2. 프로젝트 배경

Rank	Model	box ↑ AP	FPS (V100, b=1)	FPS	Paper	Code	Result	Year	Tags
1	YOLOv7-E6E (36 fps)	56.8	36	36	YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors			2022	YOLO
2	YOLOv7-D6 (44 fps)	56.6	44	44	YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors			2022	YOLO
3	YOLOv7-E6 (56 fps)	56	56	56	YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors			2022	YOLO
4	ConvNeXt-XL (C-M-RCNN)	55.2	8.6						
5	ConvNeXt-XL++ (9 fps)	55.2	8.6		A ConvNet for the 2020s			2022	
6	EfficientDet-D7x	55.1	6.5						
7	YOLOv7-W6 (84 fps)	54.9	84	84	YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors			2022	YOLO
8	ConvNeXt-L++	54.8	10		A ConvNet for the 2020s			2022	
9	PP-YOLOE-plus X	54.7	45.0		PP-YOLOE: An evolved version of YOLO			2022	
10	ConvNeXt-B++	54	11.5		A ConvNet for the 2020s			2022	

- 현재 제일 좋은 성능을 보이는 모델을 States-of-the-Art(SOTA)라고 칭함
- 실시간 영상 처리 부문에서 YOLOv7이 제일 높은 성능을 보임

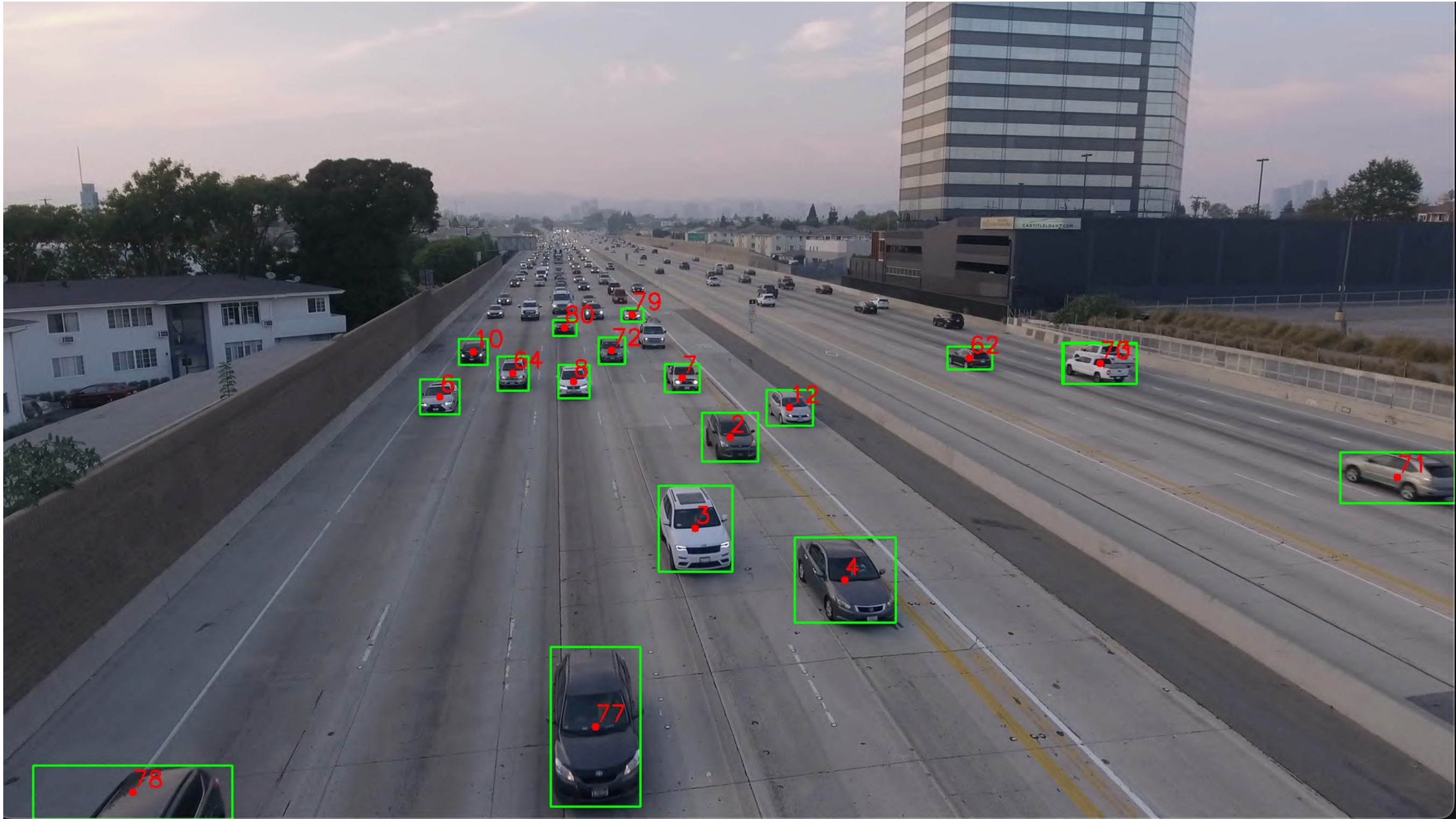
<https://paperswithcode.com/sota/real-time-object-detection-on-coco>



### 3. 프로젝트 구현 내용

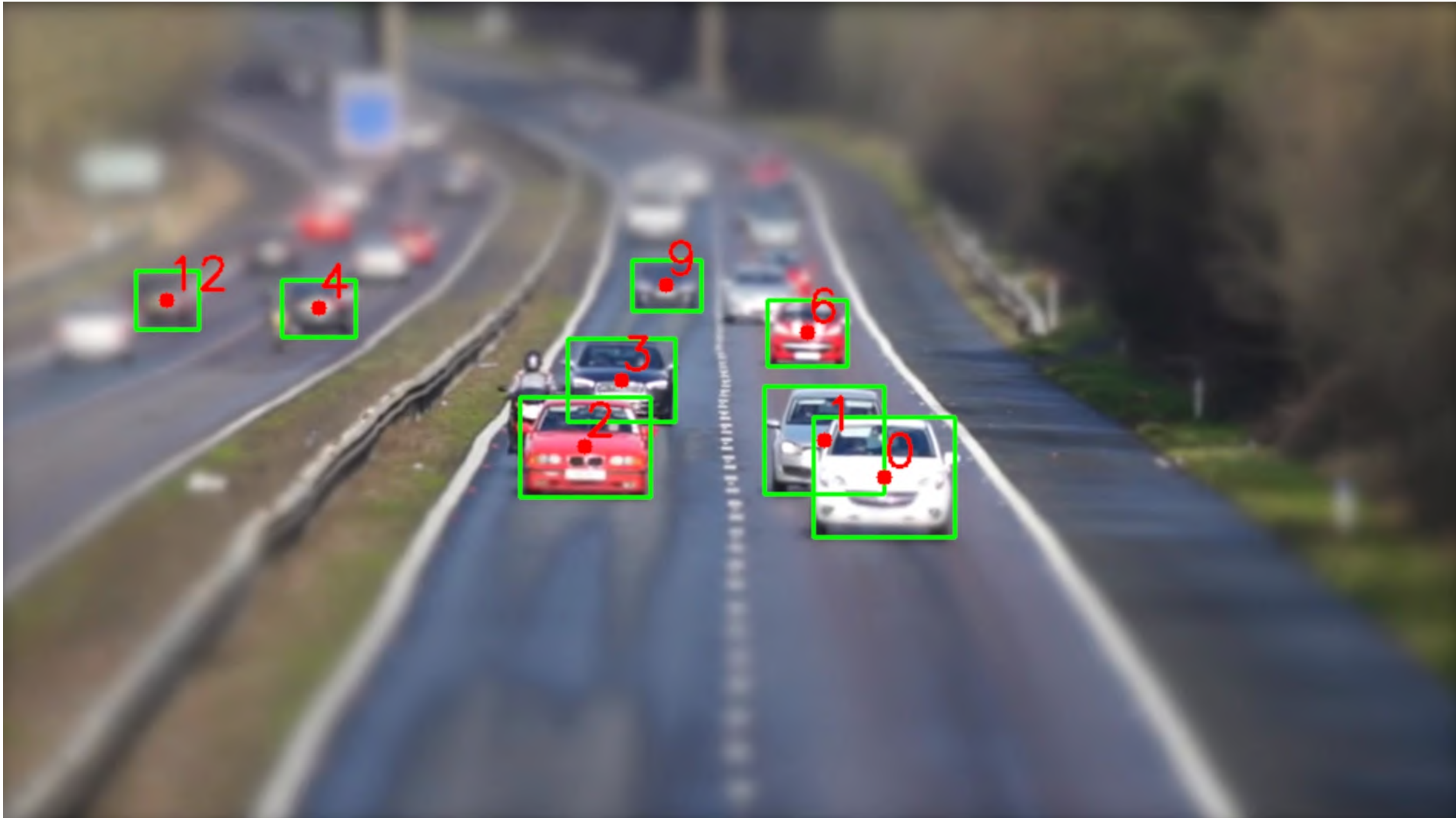
- 실시간 영상 처리/분석
- 자동차에게 고유 번호를 지정하여 객체 추적
- YOLOv4를 사용

## 4. 프로젝트 수행 결과



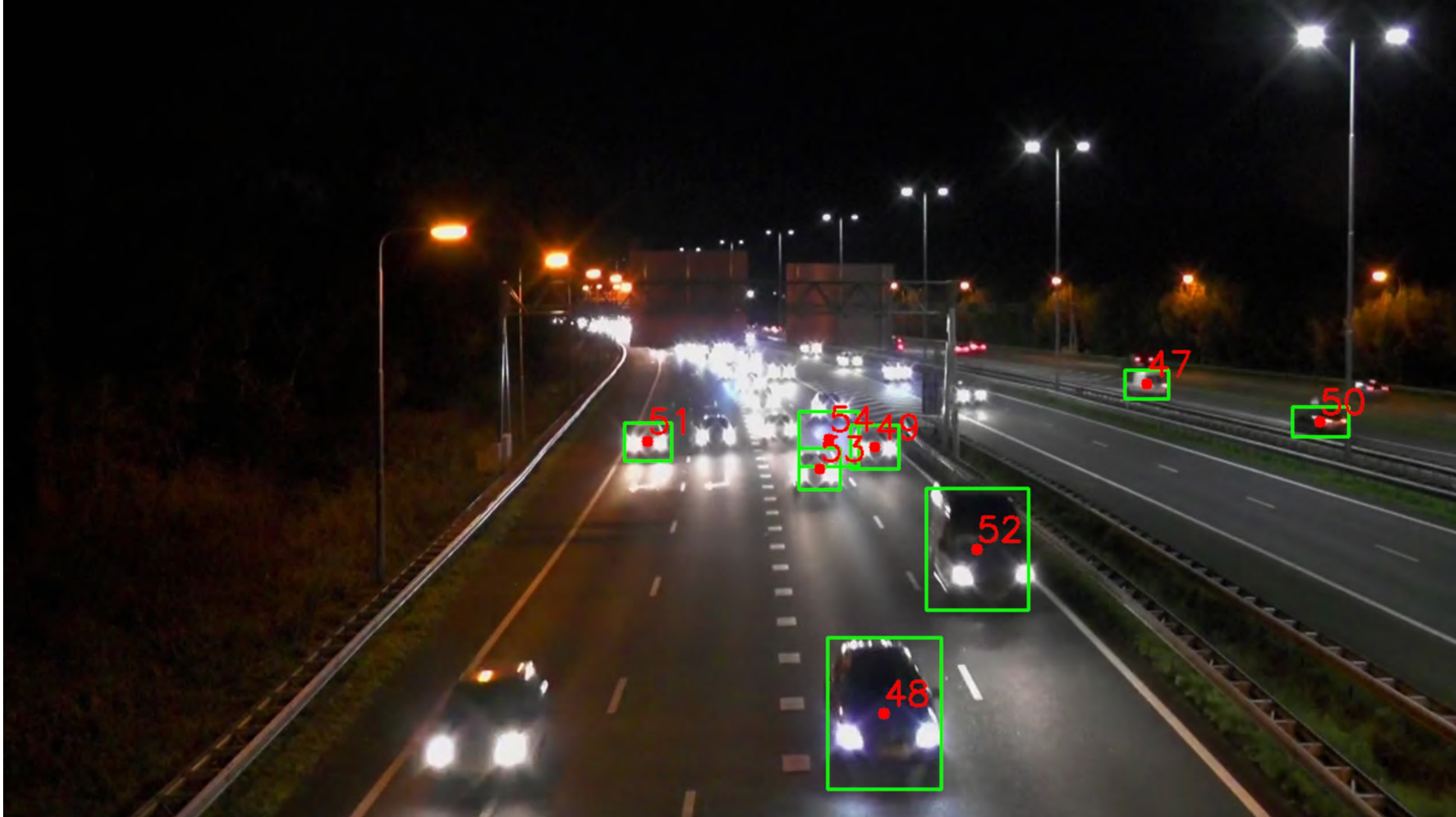


## 4. 프로젝트 수행 결과



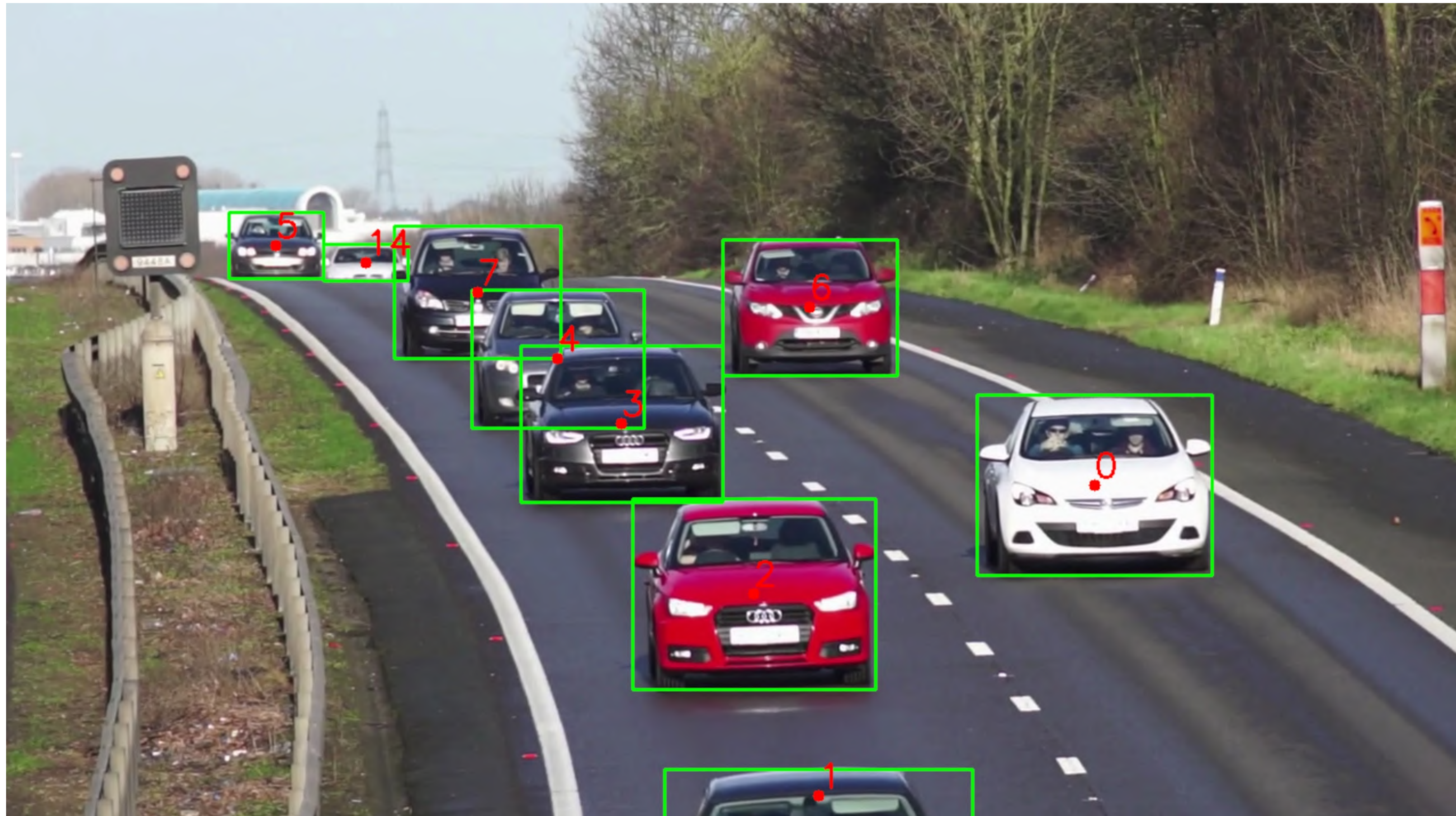


## 4. 프로젝트 수행 결과





## 4. 프로젝트 수행 결과





## 4. 프로젝트 수행 결과

```
class ObjectDetection:
    def __init__(self, weights_path="dnn_model/yolov4.weights", cfg_path="dnn_model/yolov4.cfg"):
        print("Loading Object Detection")
        print("Running opencv dnn with YOLOv4")
        self.nmsThreshold = 0.4
        self.confThreshold = 0.5
        self.image_size = 608

        # Load Network
        net = cv2.dnn.readNet(weights_path, cfg_path)
        self.model = cv2.dnn_DetectionModel(net)

        self.classes = []
        self.load_class_names()
        self.colors = np.random.uniform(0, 255, size=(80, 3))

        self.model.setInputParams(size=(self.image_size, self.image_size), scale=1/255)

    def load_class_names(self, classes_path="dnn_model/classes.txt"):

        with open(classes_path, "r") as f:
            for class_name in f.readlines():
                class_name = class_name.strip()
                self.classes.append(class_name)

        self.colors = np.random.uniform(0, 255, size=(80, 3))
        return self.classes

    def detect(self, frame):
        return self.model.detect(frame, nmsThreshold=self.nmsThreshold, confThreshold=self.confThreshold)
```

- YOLOv4 사전학습 가중치와 cfg를 사용
- load\_class\_names함수로 class 이름을 불러옴
- detect 함수로 모델을 사용할 수 있도록 함수 저장

## 4. 프로젝트 수행 결과

```
# Point current frame
center_points_cur_frame = []

# Detect objects on frame
(class_ids, scores, boxes) = od.detect(frame)
for box in boxes:
    (x, y, w, h) = box
    cx = int((x + x + w) / 2)
    cy = int((y + y + h) / 2)
    center_points_cur_frame.append((cx, cy))
    #print("FRAME N°", count, " ", x, y, w, h)

    cv2.circle(frame, (cx, cy), 5, (0, 0, 255), -1)
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Only at the beginning we compare previous and current frame
if count <= 2:
    for pt in center_points_cur_frame:
        for pt2 in center_points_prev_frame:
            distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])

            if distance < 20:
                tracking_objects[track_id] = pt
                track_id += 1
else:
    tracking_objects_copy = tracking_objects.copy()
    center_points_cur_frame_copy = center_points_cur_frame.copy()

    for object_id, pt2 in tracking_objects_copy.items():
        object_exists = False
        for pt in center_points_cur_frame_copy:
            distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
```

- 인식된 객체에 bounding box와 center point를 부여함
- center point를 기준으로 전프레임과 현재 프레임의 빗변 거리를 구하여서 거리가 20pixel 이하이면 같은 id를 부여함
- count를 2 이하로 지정한 이유는 초반에 감지된 객체가 없기 때문

## 4. 프로젝트 수행 결과

```
# Remove IDs lost
if not object_exists:
    tracking_objects.pop(object_id)

# Add new IDs found
for pt in center_points_cur_frame:
    tracking_objects[track_id] = pt
    track_id += 1

for object_id, pt in tracking_objects.items():
    cv2.circle(frame, pt, 5, (0, 0, 255), -1)
    cv2.putText(frame, str(object_id), (pt[0], pt[1] - 7), 0, 1, (0, 0, 255), 2)

print("Tracking objects")
print(tracking_objects)

print("CUR FRAME LEFT PTS")
print(center_points_cur_frame)

cv2.imshow("Frame", frame)

# Make a copy of the points
center_points_prev_frame = center_points_cur_frame.copy()

key = cv2.waitKey(0)
if key == ord('s'):
    break
```

- 화면에서 사라지는 객체는 추적이 멈춤
- 새로 탐지된 객체에게 계속 새로운 id를 부여하여 추적



## 5. 프로젝트 고찰

### 개선점:

- 추적 id가 같은 물체에게 새로 부여되는 경우가 있었음
  - 객체 추적과정에서 새로운 객체로 인식되는 과정을 더 관찰 필요
    - center point 거리계산 과정 또는 물체 전체 리 과정 예상
- 자동차가 아닌 얼굴에 추적 id 부여
  - 탐지 객체 면적을 지정하여 조정
- 더 새로운 모델을 사용하여서 객체 인지 정확도 향상
- 다양한 CCTV영상을 사용하여 모델 성능 확인
- 실시간 영상 성능의 mAP를 구하는 방법 고민

### 응용 기능:

- center point를 기준으로 프레임/초를 고려하여 자동차 시속 계산
- 영상 내에 ROI를 지정하여 관심 구역에 교통 또는 인구 혼잡도를 계산
  - 과하게 혼잡한 구역은 주요할 수 있도록 경고
- 번호판이 가려져있거나 없는 불량 차량 검출