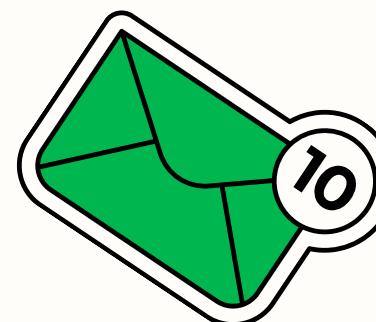
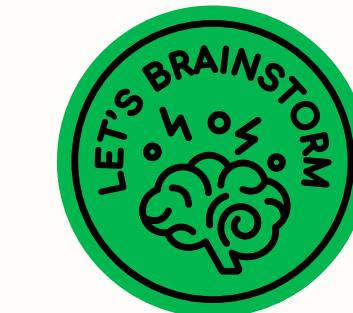


캡그룹 디자인
6포

기각 장애인



보행 보고 장애



금쪽이들

최호수 - 장진우 - 정대휴 - 손승하

목차

01

기획배경

02

1안 – 클라우드 기반 시각장애인용 AI 객체 감지 및 안내 시스템

03

2안 – 라즈베리파이 기반 실시간 YOLOv5 객체 탐지 및 보행자 안전 알림 시스템



LARANA COMPANY

기획 배경

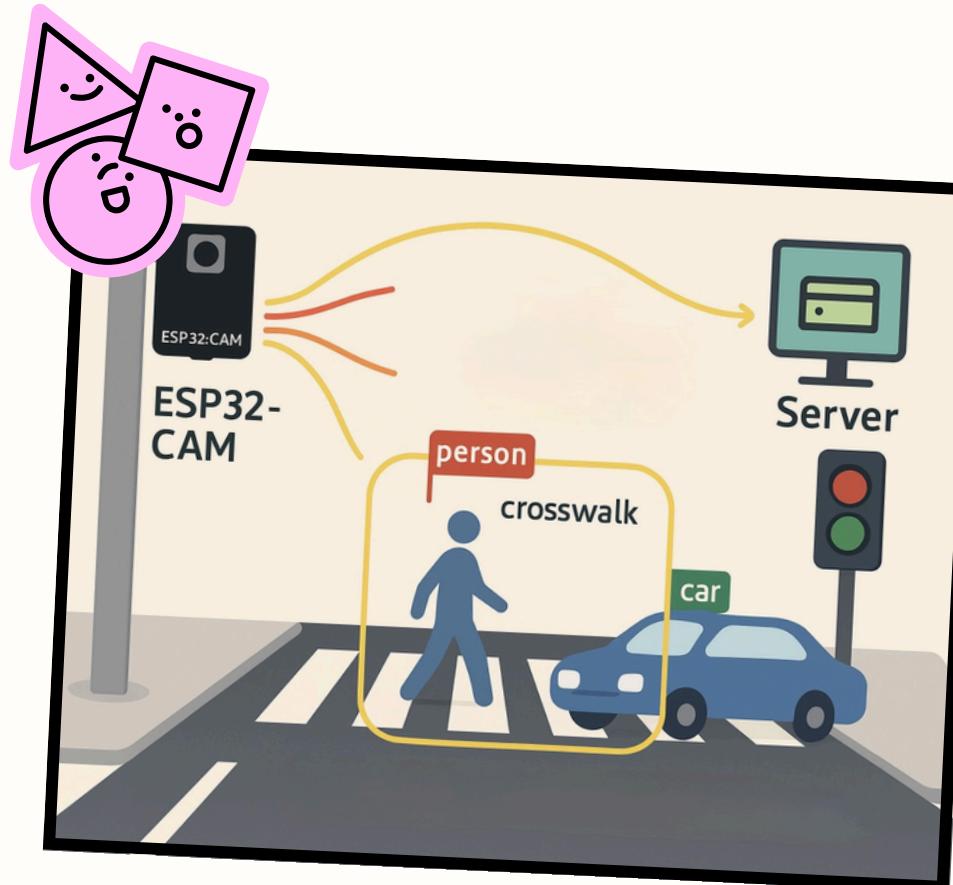
✓ 횡단보도 중앙에서의 위험 경험

보건복지부 조사 결과, 시각장애인의 60% 이상이
‘횡단보도 중앙에서 보행 중’ 위험을 경험한 장소로 꼽음



✓ 짐작틀록 부작정 설치율

정지 블록이 제대로 설치된 경우는 매우 드물(4.0%)
대부분은 위치나 방향이 틀렸거나 아예 없는 경우이다

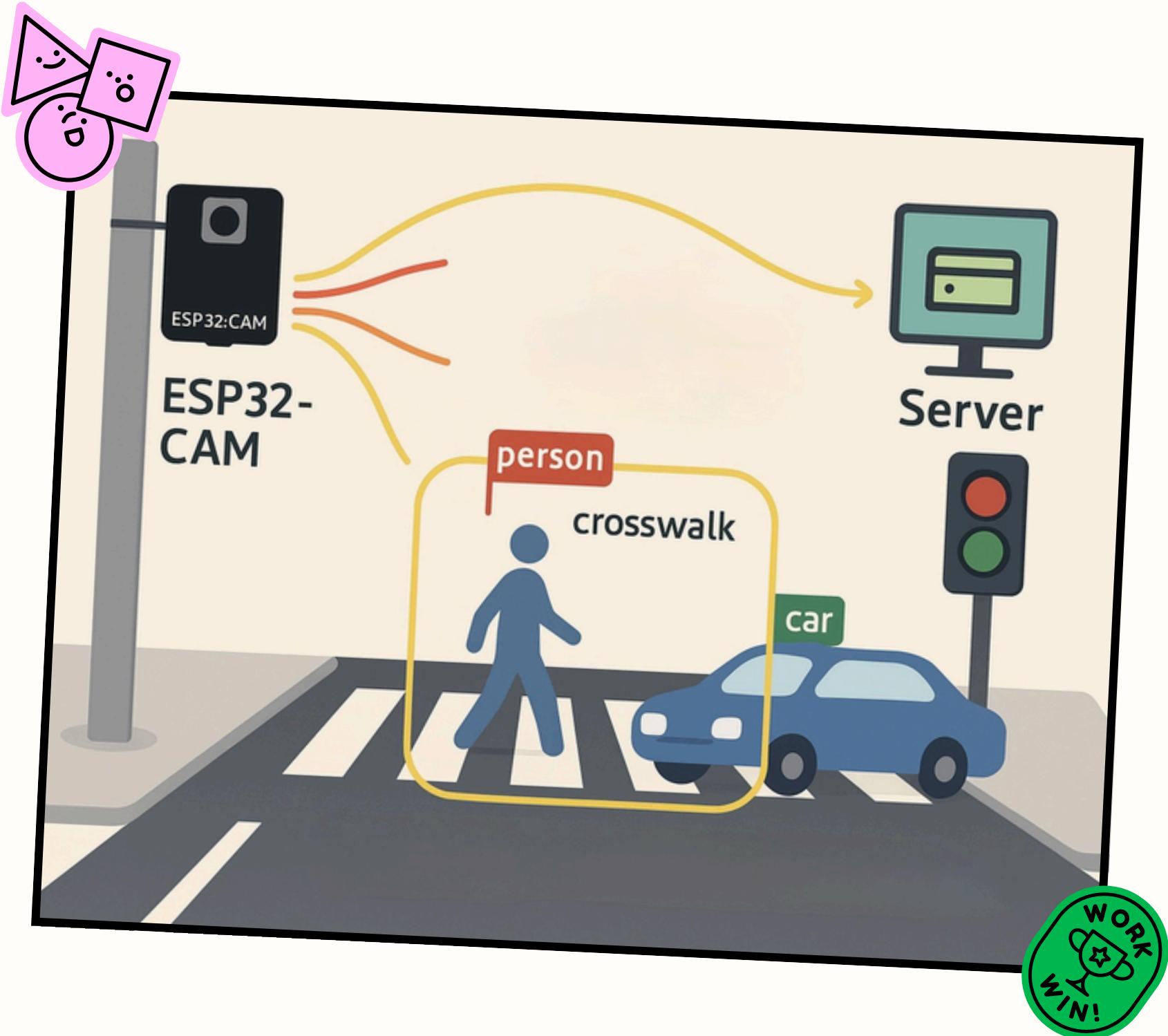


✓ 껴비용·고효율 시그널의 필요성

시각장애인의 안전한 보행을 위해, 최소 비용으로도
높은 효율을 갖춘 실시간 객체 인식 시스템이 필요하다

✓ 기존 시그널의 한계

대부분의 보행 보조 시스템은 고가 장비나 복잡한 서버가 필요해,
실제 현장 보급에 큰 장벽



1
안

클라우드 기반 기각장애인용 AI 객체 감지 및 안내 시스템

발표자: 정대희



1안) 시스템 구성 및 기술

1 하드웨어

- ESP32-CAM
- CH340 Micro B

2 소프트웨어

- YOLOv8 기반 실시간 객체 인식
- 서버(Ngrok를 통해서 제작되었으며, 응답속도가 느린것을 대체하기 위해 다중모델 사용)
- ESP32-CAM : 아두이노 예제코드를 기반으로 작성

3 흐름

- ESP32-CAM이 Wifi를 통해서 Server에 3초마다 찍은 사진을 전달
- Server는 받은 사진을 AI모델이 인식 후 결과 값으로 TTS를 생성
- 생성된 TTS파일은 Server의 웹에서 출력
- 웹에서 출력 되는 소리(TTS파일)을 듣기



DEVELOPMENT

1안) 모드코드 흐름 순서도

ESP32_CAM → “실제 장치가 이미지 전송을 어떻게 하는지”



FastAPI 기반 서버 코드 → “서버가 어떻게 구성되었는지”



API 라우터 → “이미지를 받아서 분석하고 음성까지 만드는 과정”



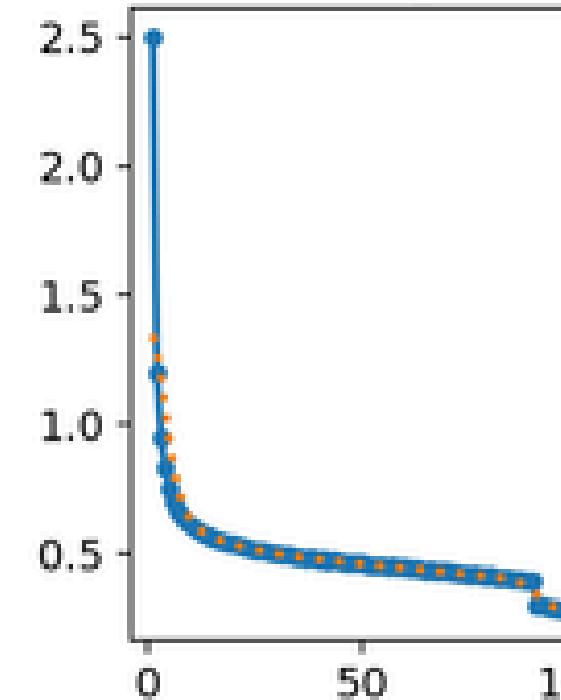
YOLOv 분석코드 → “실제 AI 분석이 어떻게 돌아가는지”



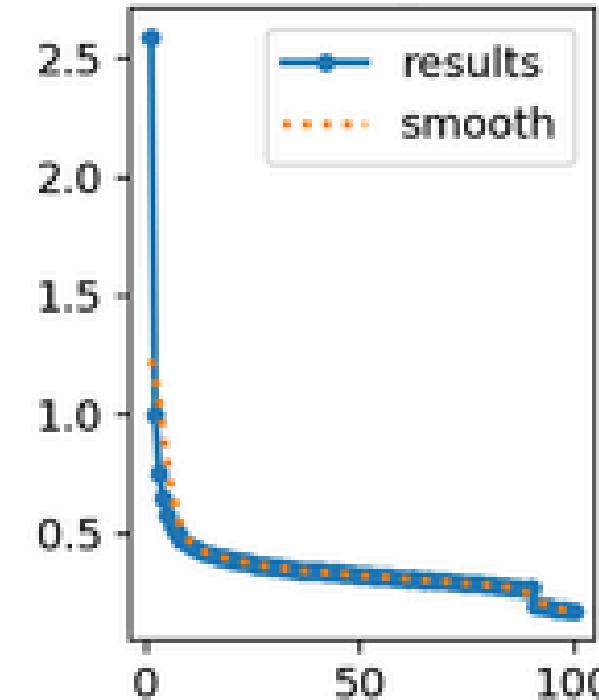
AI 모델 결과 값 출력 웹 페이지 → “사용자가 이 결과를 확인하는 모습”

Yolov8 Traffic & 기타 객체 모델 - 횡단보도 인식 전용 모델

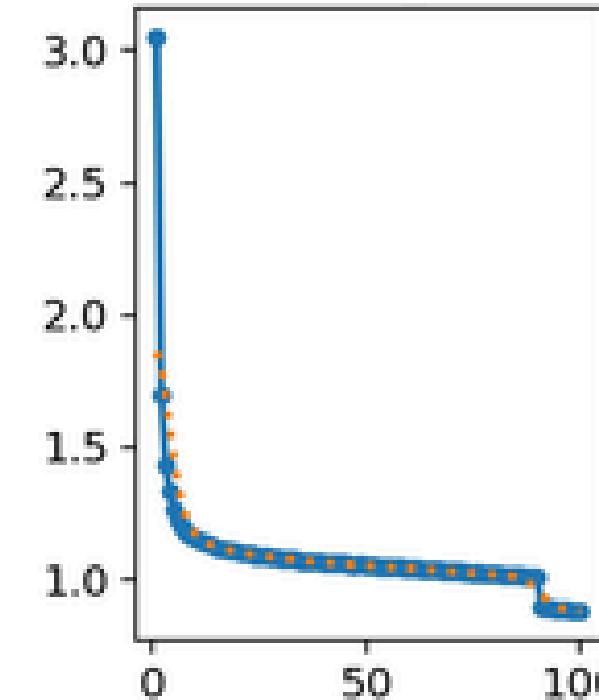
train/box_loss



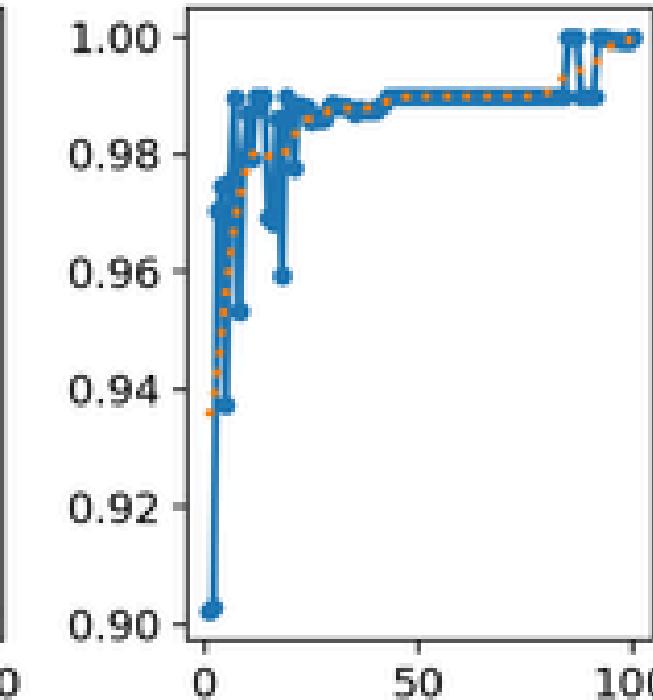
train/cls_loss



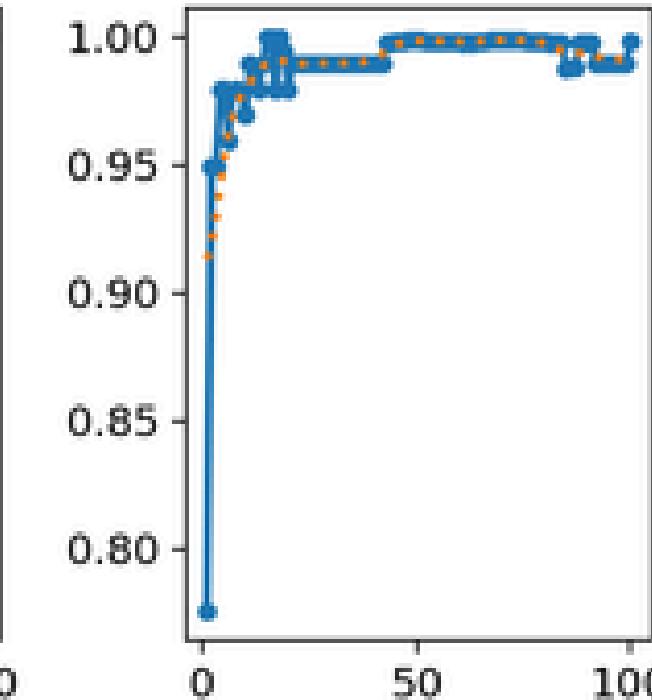
train/dfl_loss



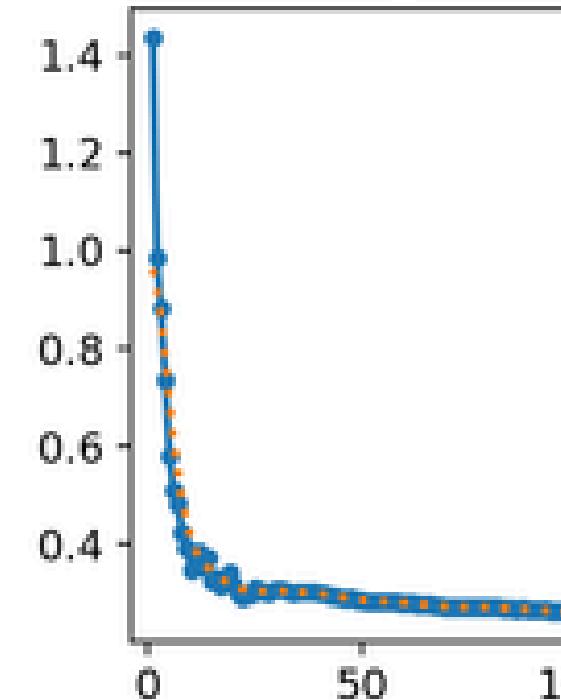
metrics/precision(B)



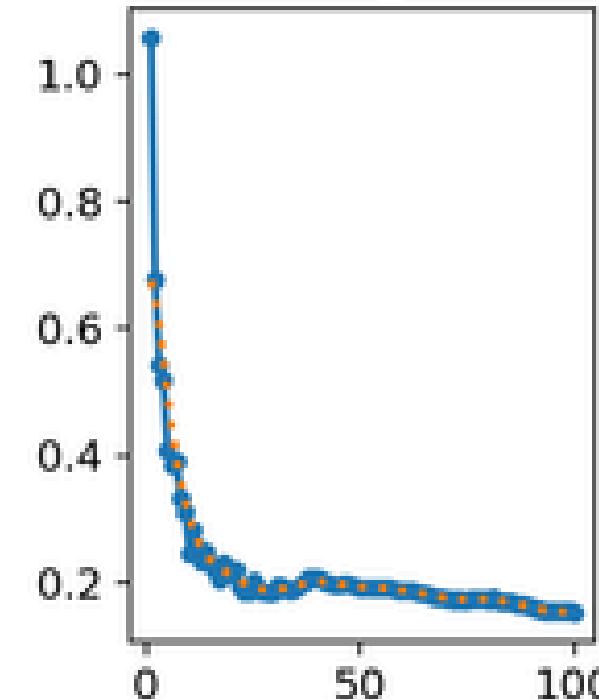
metrics/recall(B)



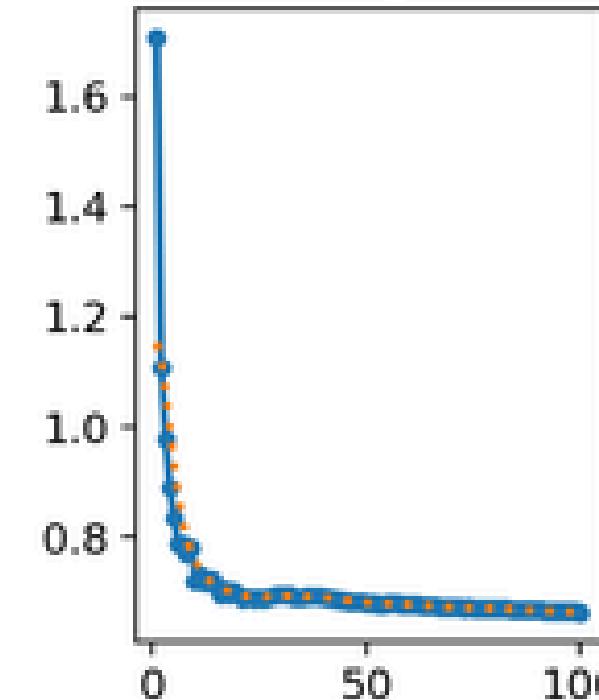
val/box_loss



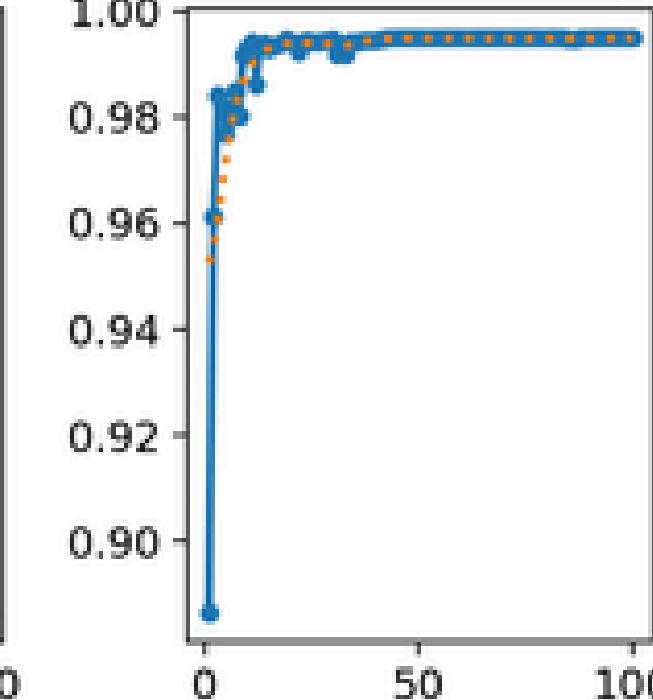
val/cls_loss



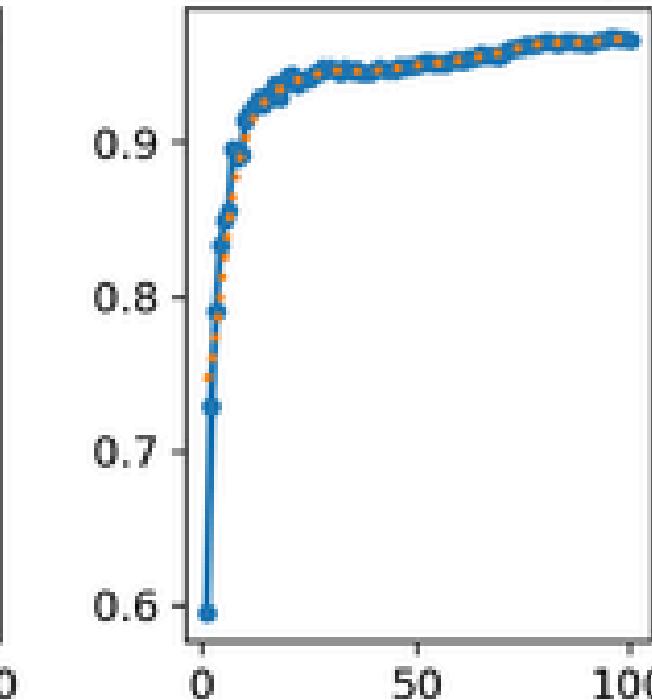
val/dfl_loss



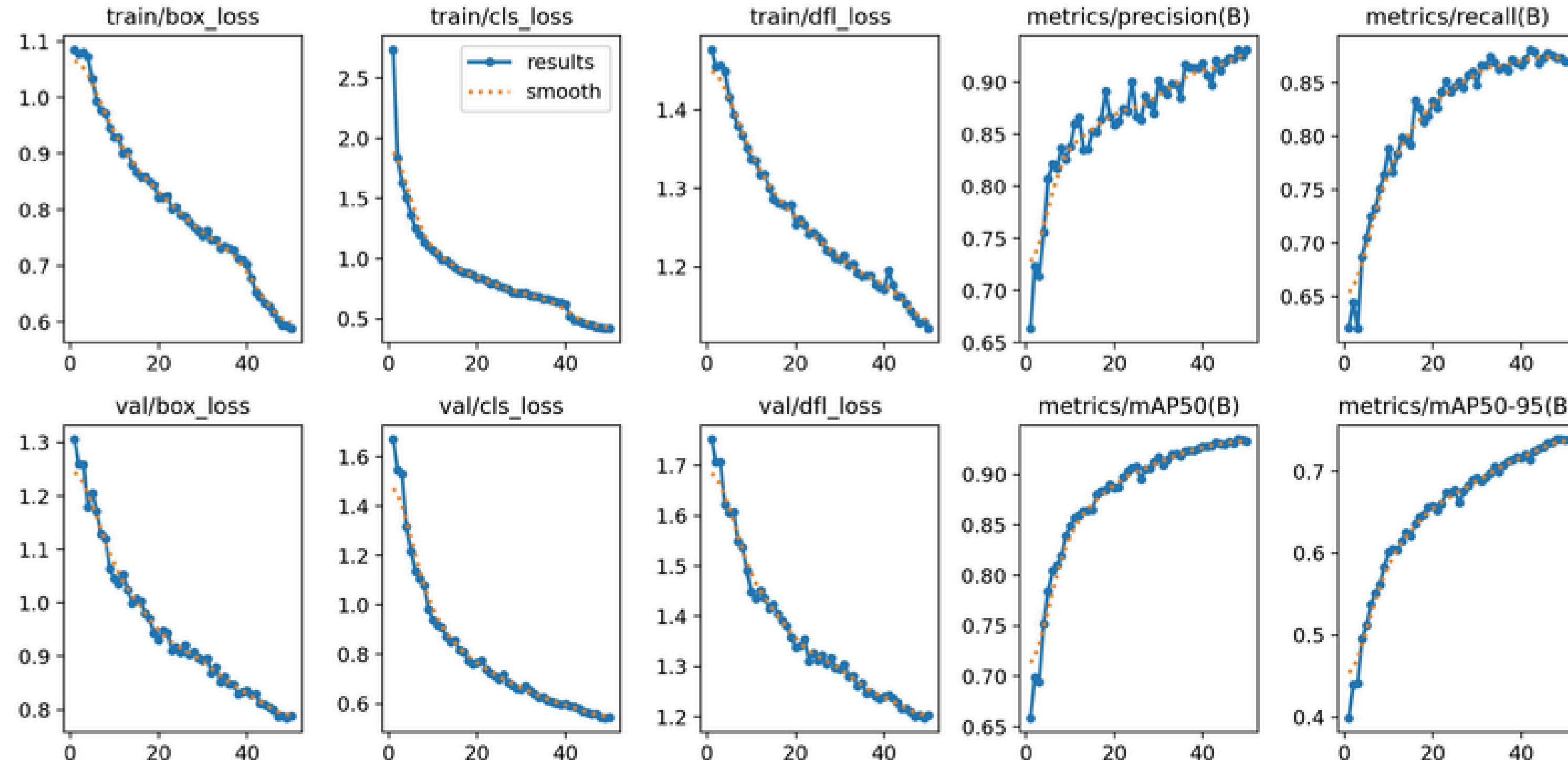
metrics/mAP50(B)



metrics/mAP50-95(B)



Yolov8 Traffic & 기타 객체 모델 - 사람, 자동차, 전동킥보드 등 다양한 교통 객체 탐지



제작 품 구성



1 **ESP32-CAM**

객체 인식



2 **CH340 micro B
업로드 보드**

SB to Serial(USB → UART) 변환을 해주는 부품





LARANA COMPANY



29

라끄베리파이 기반 실시간 YOLOv5 객체 탐지 및 보행자 안전 알림 시스템

발표자 : 손승하



LARANA COMPANY

시각장애인의 보행 속도



① 시각장애인 보행 속도 :

비장애인 대비 약 40~66% 정도로, 보행 속도가 느림



시야 정보 부족으로 인해 주변 물체·장애물을 인식하거나 대응하는데 시간이 더 걸려서, 전체적으로 보행 속도가 줄어듬



2안) 프로그램 구성 및 기술

1 하드웨어

- 라즈베리파이5
- LUAZ-1080P 카메라(영상 인식)
- 휴대형 USB 스피커_SP001 (음성 출력)
- 라즈베리파이5 18650 UPS 5V 5A 충전 모듈 및 리튬배터리 3.7V 2600mAh 2개

2 소프트웨어

- YOLOv5s 기반 실시간 객체 인식
- OpenCV를 활용해 카메라에서 수신된 영상을 실시간으로 처리
- 실제 환경의 조명·크기·각도 변화에 대응하기 위해 YOLOv5s의 하이퍼파라미터(학습률, weight decay, 데이터 증강) 등을 튜닝

3 흐름

- LUAZ-1080P 카메라가 보도블럭 인식 후 횡단보도를 인식
- 라즈베리파이로 영상 전송
- YOLOv5s 기반 분석
- 스피커로 음성 출력("전방에 횡단보도")



2안) 모그코드

YOLOv5 학습 코드



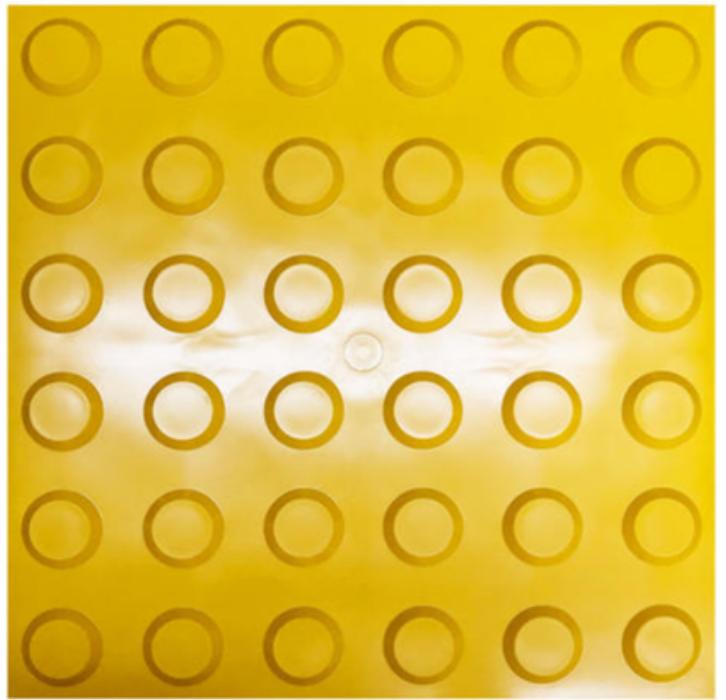
YOLOv5 학습용 하이퍼파라미터



라즈베리파이 실시간 객체 인식 코드

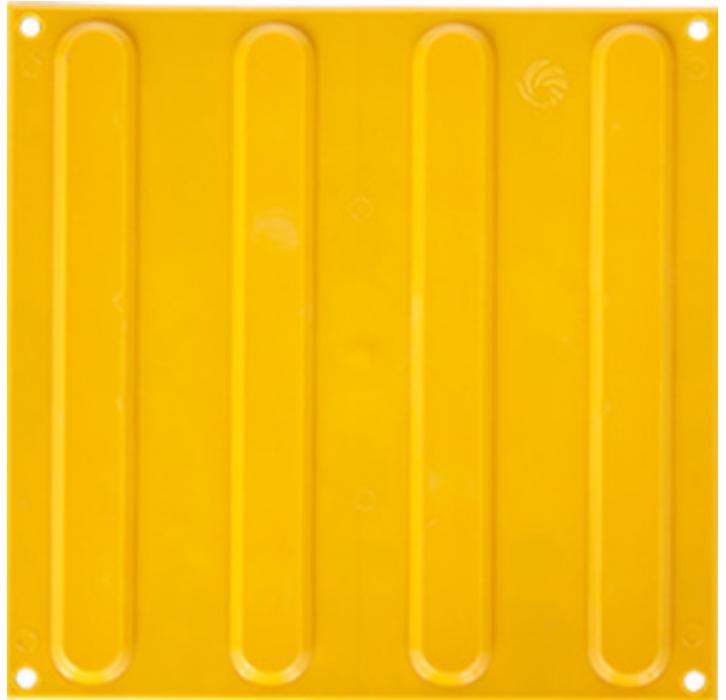


점짜블럭의 광류



점형 점짜블록

&



선형 점짜블록

- 점형 점자블럭 : “위험!”을 알려주는 경고용, 정지를 알림
- 선형 점자블럭 : “안전한 경로는 이쪽!” 같이 안내하는 역할



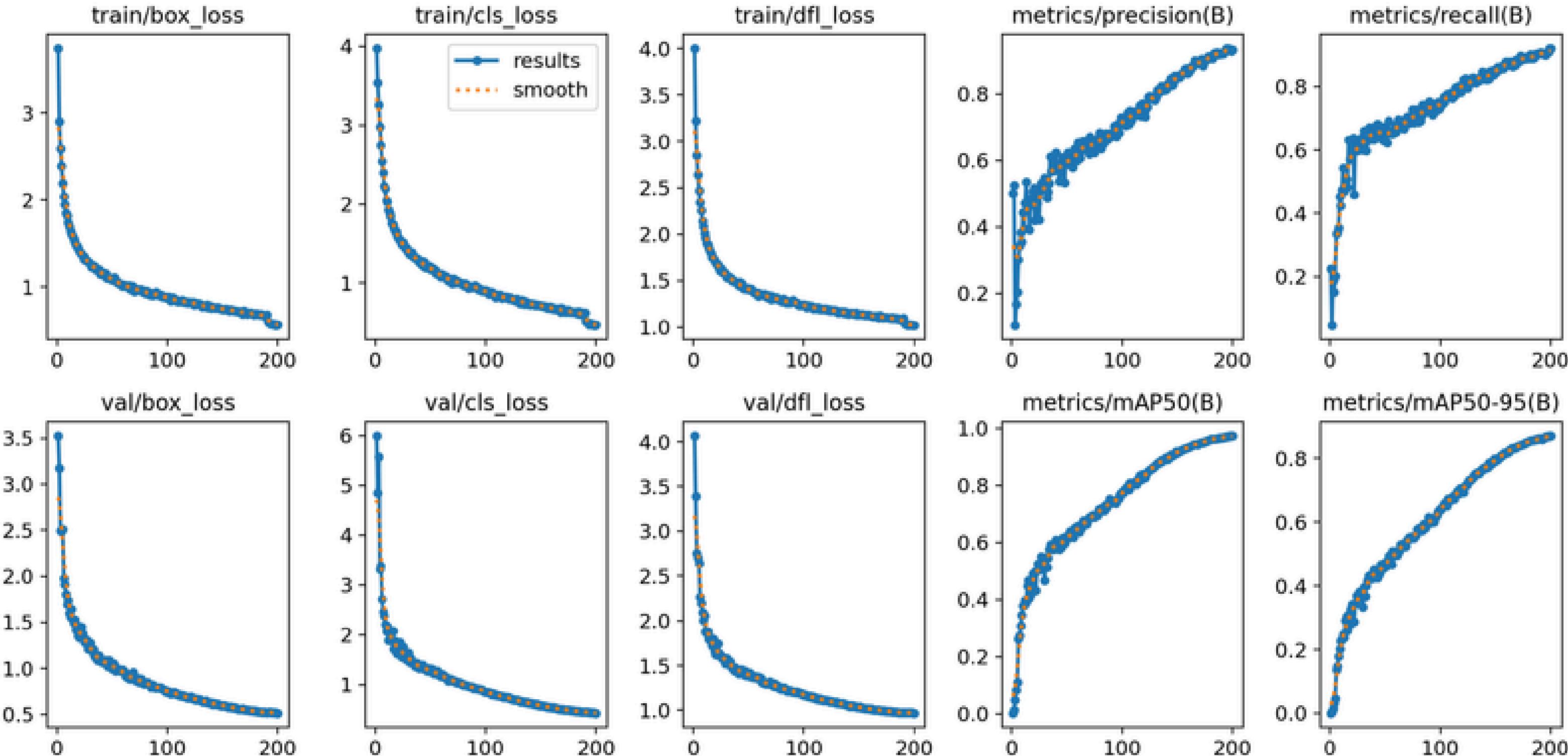
DEVELOPMENT

점자블럭의 종류



횡단보도에서의 점자블럭의 모양이 바뀌는 이유

각계 인식 모델의 학습 안정성과 성능 향상을 보여주는 시각화



기계품 구성



1 그피커

“전방에 횡단 보도” 출력

2 카메라

점자블럭 및 횡단보도 인식

3 라즈베리파이

서버와의 연동 실험 및 구동 환경 테스트에 사용



시각장애인 보조기기 제품 비교 분석



시각장애인을 위한
보조공학기기

안경/썬글라스에 장착하여 사용하는
장애물 감지 전동밀림기기
초음파 공간스캐너 SS-02A

이름 : 시크 (CEEK)
형태 : 웨어러블 디바이스
가격 : 약 108만원

이름 : 아이즈 쉐어 넷
형태 : 장착하여 사용하는 디바이스
가격 : 50만원 ~ 100만원



이름 : We Walk(스마트지팡이)
형태 : 지팡이 형태의 보조기기
가격 : 80만원 ~ 120만원



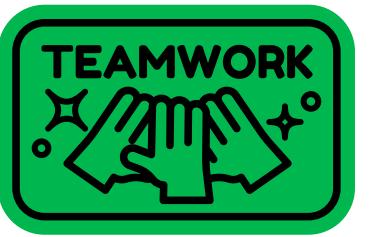
강점

1. 압도적 가격 경쟁력 :
(1안 - 약 5만원 / 2안 - 약 20만원)
2. 편의성 :
조끼 형태는 손이 자유롭고, 센서 배치가 안정적이며 착용 편의성과 외형 부담까지 줄여줌 → 2안
3. 실시간 AI비전 기반 탐지 :
YOLOv를 사용해 횡단보도, 점자블럭, 차량, 보행자, 신호등 등 다양한 시각 정보 탐지 가능
4. TTS 음성 출력 내장 :
탐지 결과를 실시간으로 한국어 음성 안내, 시각장애인의 행동 결정에 직접적인 도움 제공

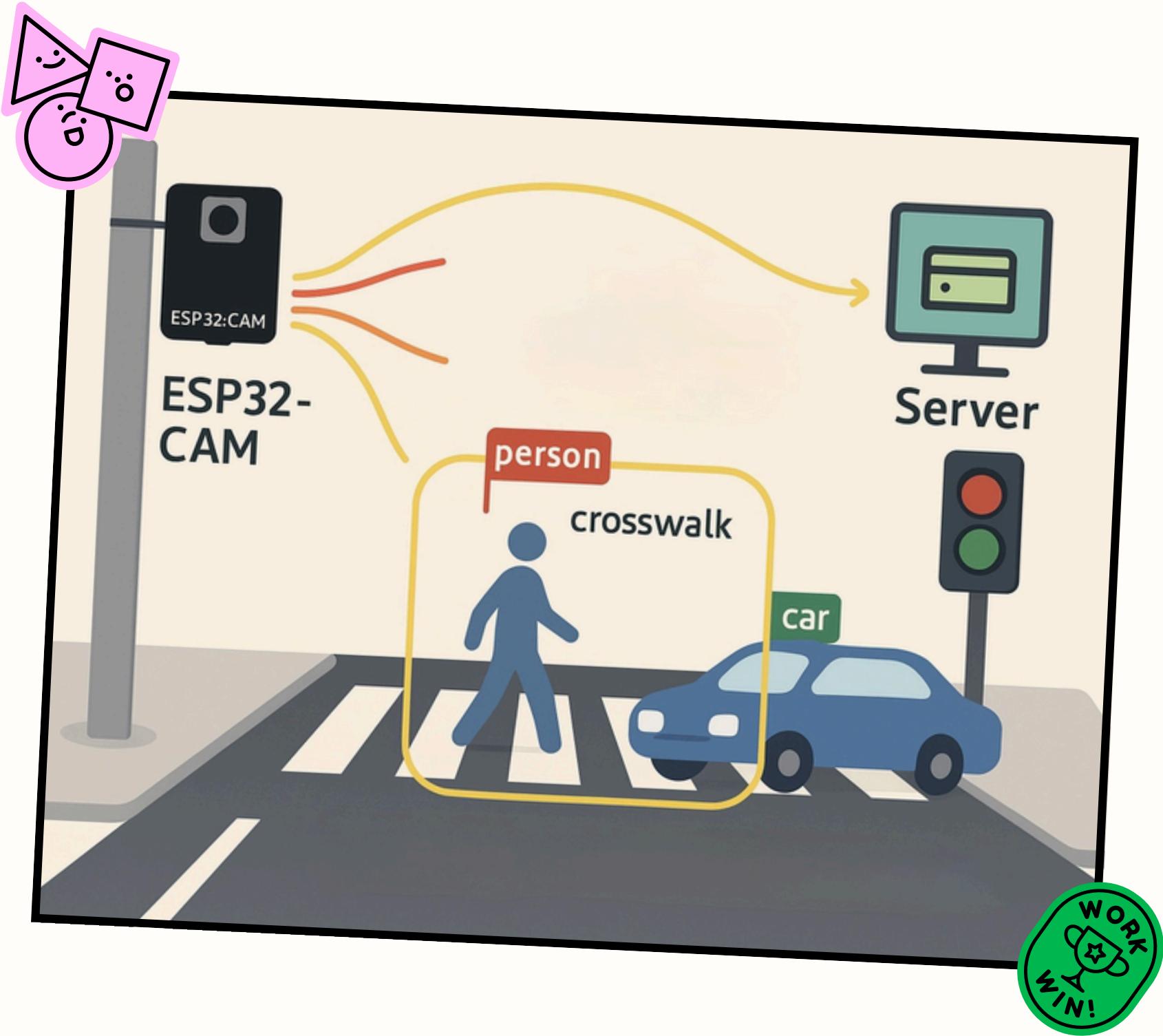


A horizontal row of seven icons: a large letter 'T', a left arrow, a bookmark, a magnifying glass, a square thumbnail with a mountain image, a share icon, and a speech bubble icon.

QUESTION & ANSWER



감사합니다



194

제작 및 작성코드

1. ESP32_CAM - (1) 라이브러리 설치 및 임포트

```
// ESP32-CAM Arduino Code
// 설치 및 설정 안내
// - ch340 드라이버 설치
// - ArduinoJson 라이브러리 설치 (저자: Benoit Blanchon)
// - Arduino IDE Board Manager URL: "https://dl.espressif.com/dl/package\_esp32\_index.json"
#include "esp_camera.h"          // ESP32-CAM 카메라 기능 라이브러리
#include <WiFi.h>                // Wi-Fi 연결을 위한 라이브러리
#include <WiFiClientSecure.h> // HTTPS 통신을 위한 라이브러리
#include <HTTPClient.h>           // HTTP 통신을 위한 라이브러리
#include <ArduinoJson.h>          // JSON 파싱 라이브러리
```

- ESP32-CAM이 카메라 영상을 캡처해 서버로 전송하는 구조를 구현
- 이를 통해 서버는 실시간으로 영상을 받아 객체 인식 및 음성 안내 가능

1. ESP32_CAM - (2) 모듈 핀 설정 및 Wi-Fi 연결 및 서버 URL을 설정

```
#define PWDN_GPIO_NUM      32
#define RESET_GPIO_NUM     -1
#define XCLK_GPIO_NUM       0
#define SIOD_GPIO_NUM      26
#define SIOC_GPIO_NUM      27
#define Y9_GPIO_NUM        35
#define Y8_GPIO_NUM        34
#define Y7_GPIO_NUM        39
#define Y6_GPIO_NUM        36
#define Y5_GPIO_NUM        21
#define Y4_GPIO_NUM        19
#define Y3_GPIO_NUM        18
#define Y2_GPIO_NUM         5
#define VSYNC_GPIO_NUM     25
#define HREF_GPIO_NUM      23
#define PCLK_GPIO_NUM      22
```

```
const char* ssid          = "SSID";
const char* password      = "Pw";
const char* serverUrl    = "https://YourDomain/analyze_image";
```

- 무선 네트워크 SSID, 비밀번호, 그리고 이미지 전송 대상 서버 URL을 설정
- ESP32-CAM 모듈의 각 핀을 명시적으로 정의

1. ESP32_CAM – (3) 카메라 초기화 함수

```

void initCamera() {
    camera_config_t config;
    // 핀맵 설정
    config.pin_pwdn      = PWDN_GPIO_NUM;
    config.pin_reset     = RESET_GPIO_NUM;
    config.pin_xclk      = XCLK_GPIO_NUM;
    config.pin_sscb_sda  = SIOD_GPIO_NUM;
    config.pin_sscb_scl  = SIOC_GPIO_NUM;
    config.pin_d7         = Y9_GPIO_NUM;
    config.pin_d6         = Y8_GPIO_NUM;
    config.pin_d5         = Y7_GPIO_NUM;
    config.pin_d4         = Y6_GPIO_NUM;
    config.pin_d3         = Y5_GPIO_NUM;
    config.pin_d2         = Y4_GPIO_NUM;
    config.pin_d1         = Y3_GPIO_NUM;
    config.pin_d0         = Y2_GPIO_NUM;
    config.pin_vsync      = VSYNC_GPIO_NUM;
    config.pin_href       = HREF_GPIO_NUM;
    config.pin_pclk       = PCLK_GPIO_NUM;
    config.ledc_timer     = LEDC_TIMER_0;
    config.ledc_channel   = LEDC_CHANNEL_0;
    config.xclk_freq_hz  = 20000000;
    config.pixel_format   = PIXFORMAT_JPEG;
    config.frame_size     = FRAMESIZE_XGA;
    config.jpeg_quality   = 10;
    config.fb_count       = 1;

    // 카메라 초기화
    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK) {
        Serial.printf("X Camera init failed: 0x%x\n", err);
        while (true) { delay(1000); }
    }

    // 색상 및 기본 설정
    sensor_t *s = esp_camera_sensor_get();
    s->set_whitebal(s, 1);
    s->set_awb_gain(s, 1);
    s->set_brightness(s, 0);
    s->set_saturation(s, 0);
    s->set_contrast(s, 0);
    s->set_gainceiling(s, (gainceiling_t)4);
}

```

- 카메라의 각종 핀 설정과 기본 화질/밝기 등의 세팅을 통해, ESP32-CAM이 영상을 안정적으로 촬영할 수 있도록 초기화하는 함수

1. ESP32_CAM - (4) 라이브 및 Wi-Fi 초기화

```
void setup() {
    Serial.begin(115200);
    Serial.println("\n📸 ESP32-CAM 시스템 시작");

    initCamera();
    Serial.println("✓ Camera initialized");

    WiFi.begin(ssid, password);
    Serial.print("📶 Connecting to Wi-Fi");
    int tries = 0;
    while (WiFi.status() != WL_CONNECTED && tries < 20) {
        delay(500);
        Serial.print(".");
        tries++;
    }
    if (WiFi.status() == WL_CONNECTED) {
        Serial.print("\n✓ Wi-Fi connected, IP: ");
        Serial.println(WiFi.localIP());
    } else {
        Serial.println("\n✖ Wi-Fi connect failed, rebooting...");
        ESP.restart();
    }
}
```

- Wi-Fi를 연결하는 함수로, 연결 실패 시 재부팅을 수행합니다. Wi-Fi 연결이 성공하면 장치의 IP 주소를 시리얼로 출력

1. ESP32_CAM - (5) 이미지 전송 함수 (서버로 HTTPS POST)

```
void sendImageToServer() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("⚠️ Wi-Fi disconnected, retrying...");
        WiFi.reconnect();
        return;
    }

    camera_fb_t* fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("❌ Camera capture failed");
        return;
    }

    WiFiClientSecure client;
    client.setInsecure();
    HTTPClient http;
    http.begin(client, serverUrl);
    http.addHeader("Content-Type", "application/octet-stream");
    int code = http.POST(fb->buf, fb->len);
    esp_camera_fb_return(fb);
}
```

```
if (code <= 0) {
    Serial.printf("❌ HTTP POST failed, code=%d\n", code);
    http.end();
    return;
}

String resp = http.getString();
http.end();
Serial.println("👉 Response: " + resp);

DynamicJsonDocument doc(256);
if (deserializeJson(doc, resp)) {
    Serial.println("⚠️ JSON parse error");
    return;
}
if (doc.containsKey("object")) {
    Serial.println("🎯 Detected: " + String(doc["object"].as<const char*>()));
}
}
```

- ESP32-CAM으로부터 사진을 캡처하여 서버에 HTTPS POST로 전송하는 역할
- 서버로부터 응답을 받아 JSON 형식으로 결과를 파싱하고, 감지된 객체를 시리얼 모니터에 출력

1. ESP32_CAM - (6) 루프 함수

```
void loop() {
    sendImageToServer();
    delay(3000); // 3초 간격
}
```

3초마다 이미지를 반복해서 전송

2. FastAPI 기본 구조 코드

```
from fastapi import FastAPI # FastAPI 웹 프레임워크 라이브러리 (API 서버 제작에 사용)
from fastapi.middleware.cors import CORSMiddleware # CORS: 다른 도메인의 요청 허용을 위한 미들웨어
from fastapi.staticfiles import StaticFiles # 정적 파일 (HTML/CSS/JS) 제공을 위한 모듈
from fastapi.responses import FileResponse # 파일을 응답으로 보내기 위한 클래스 (index.html 반환)
from app.api_routes import router as api_router # 실제 API 엔드포인트를 정의한 router 불러오기

# ✓ 1) FastAPI 앱 객체 생성 (앱 이름, 설명, 버전) / Create FastAPI app object
app = FastAPI(
    title="Vision Aid with Web UI", # 프로젝트 이름
    description="ESP32-CAM → 서버 → 웹 대시보드 / ESP32-CAM → Server → Web Dashboard",
    version="1.0.0" # 버전 정보
)

# ✓ 1) CORS 미들웨어 등록: 모든 도메인에서 접속 허용 / Register CORS middleware (Allow all origins)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # 모든 출처(origin) 허용
    allow_methods=["*"], # GET, POST, PUT 등 모든 HTTP 메소드 허용
    allow_headers=["*"], # 모든 요청 헤더 허용
)

# ✓ 2) /static 경로에 정적 파일(HTML, CSS, JS) 연결 / Mount static directory to /static
# static 폴더 내부에 있는 파일들은 "/static" 경로로 접근할 수 있게 됨
app.mount("/static", StaticFiles(directory="static"), name="static")

# ✓ 3) 루트(/)로 접속 시 index.html 반환 / Serve index.html at root URL
@app.get("/", include_in_schema=False) # API 문서(Swagger)에 표시되지 않도록 설정
def serve_root():
    return FileResponse("static/index.html", media_type="text/html") # index.html을 반환

# ✓ 4) API 라우터 등록 (ESP32-CAM에서 분석 결과 제공) / Include API router (detection endpoints)
# API 라우터는 /analyze_image 등 실제 분석 기능 제공
app.include_router(api_router, prefix="", tags=["detection"])
```

- ESP32-CAM에서 전송된 이미지 데이터를 처리하는 웹 서버를 FastAPI로 구현한 구조
- 사용자 인터페이스와 API를 통합해 실시간 분석과 결과 출력을 한 화면에서 제공할 수 있도록 구성

3. API 라우터 – (1) 환경 및 전역 변수 정의

```
import os
import uuid
from fastapi import APIRouter, Request, HTTPException
from fastapi.responses import JSONResponse, FileResponse
from gtts import gTTS
from app.ai_model import analyze_image # AI 모델 분석 함수

#📌 FastAPI 라우터 생성
router = APIRouter()

#📌 업로드 및 TTS 파일 저장 경로 설정
UPLOAD_DIR = "static/uploads" # 이미지 업로드 저장 폴더
TTS_DIR = "static/tts" # 음성 출력(TTS) 파일 저장 폴더
os.makedirs(UPLOAD_DIR, exist_ok=True) # 디렉토리 없으면 생성
os.makedirs(TTS_DIR, exist_ok=True)

#📌 최근 감지 결과를 메모리 저장소로 관리
stored_result: dict = {}
```

필요한 라이브러리 불러오고, 업로드·음성 파일 저장소 경로와 최근 감지 결과를 관리

3. API 라우터 - (2) 이미지 분석 + TTS 생성 API 엔드포인트

```
# 🔴 이미지 분석 및 TTS 파일 생성
@router.post("/analyze_image")
async def analyze_and_tts(request: Request):
    # 🔴 요청 헤더 content-type 확인 (이미지 바이너리 전송)
    ctype = request.headers.get("content-type", "")
    if not ctype.startswith("application/octet-stream"):
        raise HTTPException(status_code=415, detail="Unsupported Media Type")

    # 🔴 이미지 데이터 수신
    body = await request.body()
    if not body:
        raise HTTPException(status_code=400, detail="Empty body")

    # 🔴 이미지 파일로 저장
    img_name = f"{uuid.uuid4().hex}.jpg"
    img_path = os.path.join(UPLOAD_DIR, img_name)
    with open(img_path, "wb") as f:
        f.write(body)

    try:
        # 🔴 AI 모델로 이미지 분석
        result = analyze_image(img_path)
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Model error: {e}")

    # 🔴 결과에서 주요 값 분리
    traffic_label = result.get("traffic_label", "")
    objects = result.get("objects_on_crosswalk", [])
    has_crosswalk = result.get("has_crosswalk", False)
```

카메라로 보낸 사진을 받아서 어떤 상황인지 분석하고, 안내 문구를 음성파일로 만듬

3. API 라우터 - (3) 최근 감지 결과 조회 API

```
tts_lines = [] # TTS 음성 문장 리스트

# ↗ 신호등 상태에 따른 기본 안내 음성
if traffic_label:
    if traffic_label == "초록불":
        tts_lines.append("초록불입니다. 횡단하세요.")
    elif traffic_label == "빨간불":
        tts_lines.append("빨간불입니다. 건너지 마세요.")

# ↗ 상황별 경고 음성 추가
if traffic_label == "초록불" and any(o in ["car", "truck", "bus", "obstacle"] for o in objects):
    tts_lines.append("차량 또는 장애물이 횡단보도를 막고 있습니다. 후진해주세요.")
elif traffic_label == "빨간불" and "person" in objects:
    tts_lines.append("사람이 횡단보도 위에 있습니다. 건너지 마세요.")

# ↗ 신호등 없고 횡단보도만 감지된 경우
if not traffic_label and has_crosswalk:
    tts_lines.append("횡단보도를 감지했습니다.")

# ↗ 최종 TTS 문장 생성
tts_message = " ".join(tts_lines) if tts_lines else "감지된 객체가 없습니다."
tts_name = None

# ↗ gTTS로 음성 파일 생성 (필요할 때만)
if tts_lines:
    tts_name = f"{uuid.uuid4().hex}.mp3"
    tts_path = os.path.join(TTS_DIR, tts_name)
    try:
        tts = gTTS(text=tts_message, lang="ko")
        tts.save(tts_path)
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"TTS error: {e}")

# ↗ 결과 및 TTS 파일 이름을 저장
stored_result['result'] = result
stored_result['tts'] = tts_name

# ↗ 응답 데이터 준비 및 반환
response = {'result': result}
if tts_name:
    response['tts_url'] = '/latest_tts'
return JSONResponse(response)
```

마지막에 처리된 결과를 다시 물어보면 그 내용을 간단히 정리되어 출력

3. API 라우터 – (4) TTS 파일 제공 API

```
# ❤ 최근 감지 결과 요청 처리
@router.get("/latest_detection")
async def latest_detection():
    if 'result' not in stored_result:
        raise HTTPException(status_code=404, detail="No detections yet")
    resp = {'result': stored_result['result']}
    if stored_result.get('tts'):
        resp['tts_url'] = '/latest_tts'
    return JSONResponse(resp)

# ❤ TTS 음성 파일 요청 처리 (개별 파일)
@router.get("/tts/{filename}")
def serve_tts(filename: str):
    path = os.path.join(TTS_DIR, filename)
    if not os.path.exists(path):
        raise HTTPException(status_code=404, detail="TTS file not found")
    return FileResponse(path, media_type="audio/mpeg")

# ❤ 가장 최근 TTS 음성 파일 요청 처리
@router.get("/latest_tts")
def latest_tts():
    tts_file = stored_result.get('tts')
    if not tts_file:
        raise HTTPException(status_code=404, detail="No TTS available yet")
    path = os.path.join(TTS_DIR, tts_file)
    if not os.path.exists(path):
        raise HTTPException(status_code=404, detail="TTS file not found")
    return FileResponse(path, media_type="audio/mpeg")
```

만들어진 음성 안내 파일(mp3)을 바로 들을 수 있도록 파일을 보여주는 기능

4. YOLOv 분석 코드 - AI 모델 추론

```

from ultralytics import YOLO  # YOLOv8 라이브러리 불러오기
import cv2                      # OpenCV: 영상 처리 라이브러리
import numpy as np                # NumPy: 배열/수치 계산

# 1 YOLOv8 모델 경로 설정 (점자블럭, 횡단보도, 신호등)
MODEL_PATHS = {
    "braille": "../models/braille.pt",
    "crosswalk": "../models/crosswalk.pt",
    "traffic": "../models/traffic.pt",
}
# YOLO 모델 객체 생성 (모델을 불러와서 딕셔너리에 저장)
models = {k: YOLO(v) for k, v in MODEL_PATHS.items()}

# 2 B↔R 채널 스왑 함수 (ESP32-CAM에서 RGB 색상이 반전되어 나오는 이슈 보정)
def swap_blue_red_channel(img_bgr):
    # BGR 이미지를 B와 R 채널을 뒤바꿔서 RGB로 보정
    return img_bgr[:, :, ::-1].copy()

```

```

# 3 이미지 분석 함수: 이미지 경로를 받아서 횡단보도, 신호등, 객체를 분석
def analyze_image(image_path):
    # 1 이미지 로드
    img = cv2.imread(image_path)

    # 2 색상 반전 대신 B↔R 채널을 교환 (ESP32-CAM 문제 보정)
    img = swap_blue_red_channel(img)

    # 3 YOLO 입력용 RGB 형식으로 변환
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # 4 횡단보도와 신호등 모델로 각각 예측 실행
    crosswalk_result = models["crosswalk"](img_rgb)
    traffic_result = models["traffic"](img_rgb)

    # 5 신호등 결과 이미지를 저장 (디버깅용)
    traffic_result[0].save("output.jpg")

    # 6 횡단보도 감지 결과 확인
    boxes = crosswalk_result[0].boxes  # 탐지된 횡단보도 박스들
    has_crosswalk = len(boxes) > 0      # 하나라도 있으면 True

```

4. YOLOv 분석 코드 - AI 모델 후론

```

crosswalk_area = None
if has_crosswalk:
    # 7 가장 큰 횡단보도 박스를 선택
    biggest_box = max(
        boxes,
        key=lambda b: (b.xyxy[0][2] - b.xyxy[0][0]) * (b.xyxy[0][3] - b.xyxy[0][1])
    )
    crosswalk_area = biggest_box.xyxy[0].cpu().numpy() # 박스 좌표를 numpy 배열로 저장

# 8 신호등 감지 결과의 클래스명 추출 (소문자로 변환)
traffic_classes = [
    traffic_result[0].names[int(c)].lower()
    for c in traffic_result[0].boxes.cls.cpu().numpy()
]
print("🔥 감지된 클래스:", traffic_classes)

# 9 신호등 상태 판단
has_red = any("traffic light-red-" in cls for cls in traffic_classes) # 빨간불 감지 여부
has_green = any("traffic light-green-" in cls for cls in traffic_classes) # 초록불 감지 여부

# 10 최종 신호등 상태 라벨 결정
if has_red and has_green:
    traffic_label = "⚠️ 빨간불과 초록불이 동시에 감지됨"
elif has_red:
    traffic_label = "빨간불"
elif has_green:
    traffic_label = "초록불"
else:
    traffic_label = "" # 감지된 신호등 없음
print("🚦 판단된 신호등 상태:", traffic_label)

# 11 횡단보도 위 객체 탐지
objects_on_crosswalk = []
if crosswalk_area is not None:
    x1, y1, x2, y2 = crosswalk_area # 횡단보도 영역 좌표
    for box in traffic_result[0].boxes: # 신호등 탐지 결과 박스들 중
        cls = traffic_result[0].names[int(box.cls[0])].lower() # 클래스명
        if cls in ["person", "car", "truck", "bus"]:
            # 객체 중심점 좌표 계산
            bx1, by1, bx2, by2 = box.xyxy[0].cpu().numpy()
            cx, cy = (bx1 + bx2) / 2, (by1 + by2) / 2
            # 객체 중심이 횡단보도 내부에 있으면 추가
            if x1 <= cx <= x2 and y1 <= cy <= y2:
                objects_on_crosswalk.append(cls)

# 12 최종 분석 결과 딕셔너리로 반환
return {
    "has_crosswalk": has_crosswalk, # 횡단보도 감지 여부
    "traffic_label": traffic_label, # 신호등 상태
    "objects_on_crosswalk": objects_on_crosswalk # 횡단보도 위 객체 목록
}

```

- YOLOv 모델을 사용하여 주어진 이미지에서 횡단보도, 신호등, 횡단보도 위의 객체를 감지·분석하는 기능을 구현
- 특히 ESP32-CAM의 색상 채널 문제를 보정하는 함수도 포함

5. AI 모델 결과 감지 출력 웹 페인트

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="utf-8" />
  <title>Vision Aid 실시간 감지</title>
</head>
<body>
  <h1>Vision Aid 실시간 감지</h1>
  <button id="startBtn">시작</button>
  <p id="status">대기 중...</p>
  <audio id="player"></audio>
  <p id="ttsStatus">🔇 음성 없음</p>

  <!-- 감지 결과 표시 -->
  <div>
    <div>신호등 상태: <span id="trafficLabel">-</span></div>
    <div>횡단보도: <span id="crosswalk">-</span></div>
    <div>횡단보도 위 객체: <span id="onCrosswalk">-</span></div>
  </div>

  <script>
    const statusEl = document.getElementById("status");
    const player = document.getElementById("player");
    const ttsStatus = document.getElementById("ttsStatus");
    const btn = document.getElementById("startBtn");

    const trafficEl = document.getElementById("trafficLabel");
    const crosswalkEl = document.getElementById("crosswalk");
    const onCrossEl = document.getElementById("onCrosswalk");
  </script>

```

```
// 오디오 끝나면 상태 초기화
player.onended = () => {
  ttsStatus.textContent = "🔇 음성 없음";
};

// 서버에서 감지 결과 요청하고 TTS 재생
async function pollAndPlay() {
  try {
    const res = await fetch("/latest_detection");
    const j = await res.json();
    const result = j.result;

    trafficEl.textContent = result.traffic_label || "없음";
    crosswalkEl.textContent = result.has_crosswalk ? "있음" : "없음";
    onCrossEl.textContent = (result.objects_on_crosswalk || []).join(", ") || "없음";
    statusEl.textContent = `${trafficEl.textContent}, 횡단보도: ${crosswalkEl.textContent}`;

    if (j.tts_url) {
      player.src = `${j.tts_url}?t=${Date.now()}`;
      await player.play();
      ttsStatus.textContent = "🔊 음성 출력 중...";
    }
  } catch {
    statusEl.textContent = "🚫 오류 발생";
    ttsStatus.textContent = "🔇 음성 없음";
  }
}

// 버튼 누르면 5초마다 감지 반복
btn.addEventListener("click", () => {
  btn.disabled = true;
  statusEl.textContent = "시작됨: 5초마다 감지 중...";
  pollAndPlay();
  setInterval(pollAndPlay, 5000);
});
</script>
</body>
</html>
```

버튼 클릭 시 서버로부터 5초마다 감지 결과를 받아서 웹페이지에 표시하고, 음성이 있으면 재생



LARANA COMPANY



**Electronic
Assistive
Device for
Visually
Impaired**

2원

학습 코드 및 간접인식 코드

1. YOLOv5 학습 코드

```
!python train.py \
--img 640 \
--batch 32 \
--epochs 200 \
--data /content/drive/MyDrive/cross_block/data.yaml \
--cfg models/yolov5s.yaml \
--weights yolov5s.pt \
--hyp data/hyps/hyp.scratch-low.yaml \
--name crosswalk_dotted_train_ep200 \
--project runs/train \
--save-period 10 \
--patience 30

# YOLOv5의 학습 스크립트 실행
# 입력 이미지 크기를 640x640으로 설정
# 배치 사이즈: 32장씩 동시에 학습
# 총 학습 반복 횟수: 200회
# 데이터셋 정보가 들어있는 YAML 파일 경로
# 사용할 YOLOv5 모델 구성 파일 (YOLOv5s: 소형 모델)
# 사전 학습된 가중치(Pretrained weights) 파일 경로
# 하이퍼파라미터 설정 파일 경로
# 학습 결과를 저장할 디렉토리 이름
# 결과가 저장될 프로젝트 루트 디렉토리
# 10 에폭마다 모델 가중치를 저장
# 30 에폭 연속으로 성능 개선이 없으면 학습 조기 종료
```

- YOLOv5s 기반으로 횡단보도/점자블럭 인식 모델을 학습하기 위한 전체 파라미터 세트를 명시
- 추가적으로 하이퍼파라미터, 데이터 구성 방식에 따라 세밀한 성능 조정이 가능

2. YOLOv5 학습용 하이퍼파라미터 설정

```
box: 0.05          # Bounding box 회귀 손실의 가중치 (박스 정확도에 대한 중요도)
cls: 0.5           # 클래스 분류 손실의 가중치
cls_pw: 1.0         # 클래스 BCE 손실에서 양성 샘플의 가중치
obj: 1.0           # 객체ness(객체 존재 여부) 손실의 가중치
obj_pw: 1.0         # 객체ness BCE 손실의 양성 샘플 가중치
iou_t: 0.20         # IoU 트레이닝 임계값: 예측 상자와 GT 상자의 IoU가 이 값 이상일 때 객체로 판단
anchor_t: 4.0        # 앵커-배율 임계값: 같은 앵커로 간주할 크기 비율의 임계값
# anchors: 3          # 출력 레이어당 앵커 수 (0으로 두면 기본값 사용)
fl_gamma: 0.0        # Focal Loss의 감마 값 (0이면 focal loss 비활성화)

hsv_h: 0.015        # 색상(Hue) 변화 범위 비율
hsv_s: 0.7           # 채도(Saturation) 변화 범위 비율
hsv_v: 0.4           # 명도(Value) 변화 범위 비율
degrees: 0.0          # 이미지 회전 범위 (+/- 도)
translate: 0.1         # 이미지 이동 범위 (+/- 비율)
scale: 0.5            # 이미지 스케일 조정 범위 (+/- 비율)
shear: 0.0             # 이미지 왜곡(Shear) 범위 (+/- 도)
perspective: 0.0       # 원근 변환 범위 (+/- 비율)
flipud: 0.0            # 상하 반전 확률
fliplr: 0.5            # 좌우 반전 확률
mosaic: 1.0            # 모자이크 증강을 적용할 확률
mixup: 0.0              # mixup 증강을 적용할 확률
copy_paste: 0.0          # 이미지 일부 복사-붙여넣기 증강 확률
```

2. YOLOv5 학습용 하이퍼파라미터 설정

```
# Hyperparameters for low-augmentation COCO training from scratch - 하이퍼파라미터
```

```
lr0: 0.01          # 초기 학습률 (기본적으로 SGD=1e-2, Adam=1e-3 정도의 추천값)
lrf: 0.01          # 최종 학습률 비율 (OneCycleLR에서 초기 학습률 대비 마지막 학습률의 비율)
momentum: 0.937    # 모멘텀 (SGD에서 사용), 모델이 빠르게 최적화되도록 돋는 역할
weight_decay: 0.0005 # 가중치 감쇠 (L2 정규화) 계수, 과적합 방지를 위해 가중치 크기를 줄임
warmup_epochs: 3.0  # 웜업 에포크 수: 학습 초반에 서서히 학습률을 높이기 위한 단계
warmup_momentum: 0.8 # 웜업 중 모멘텀 시작값
warmup_bias_lr: 0.1 # 웜업 중 바이어스(편향) 파라미터의 학습률
```

- YOLOv5를 처음부터 학습할 때(사전학습 없이) 사용할 수 있도록 구성된 하이퍼파라미터 집합
- COCO 데이터셋을 대상으로 적은 증강만으로도 안정적이고 효율적인 학습이 가능하게끔 최적화된 값

3. 라프베리파이 실시간 객체 인식 코드 - (1) 라이브러리 & 모델 로딩

```
# OpenCV: 카메라와 영상처리를 위한 라이브러리
import cv2
# PyTorch: AI 모델을 불러오고 처리하기 위한 라이브러리
import torch
# time: 시간 관련 함수 (클타임 체크, FPS 계산 등)
import time
# os: 시스템 명령어(예: 음성 출력) 실행을 위한 라이브러리
import os
# warnings: 불필요한 경고 메시지를 숨기기 위한 라이브러리
import warnings
warnings.filterwarnings("ignore") # 경고 메시지 숨김

# 모델 경로와 로딩
model_path = 'models/crosswalk_dotted_best.pt'
model = torch.hub.load('ultralytics/yolov5', 'custom', path=model_path, device='cpu', verbose=False)
model.conf = 0.15 # 탐지 신뢰도 임계값
```

- OpenCV, PyTorch 등 객체 탐지와 영상처리를 위한 필수 라이브러리들을 임포트하고, 불필요한 경고는 제거
- 사전에 학습된 YOLOv5 모델을 로드해 탐지할 준비를 하며, 탐지 정확도를 설정해 과도한 탐지를 방지

3. 라프베리파이 실시간 객체 인식 코드 - (2) 카메라 및 상태 변수 초기화

```
# 카메라 초기화 (기본 0번 카메라)
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
cap.set(cv2.CAP_PROP_FPS, 30)

if not cap.isOpened():
    print("Camera open failed")
    exit()

# 탐지 상태 초기화
dotted_detected = False # 점자블럭 탐지 상태
crosswalk_detected = False # 횡단보도 탐지 상태
cooldown = False # 쿨타임 상태
last_trigger_time = 0 # 마지막 음성 출력 시작

start_time = time.time()
frame_count = 0
```

- 카메라를 열어 영상 입력을 받고, 해상도·프레임 등 영상을 처리하기 위한 기본 설정을 끝냄
- 점자블럭·횡단보도 탐지 상태와 음성출력 쿨타임 등을 관리하기 위한 변수들도 함께 초기화

3. 라프베리파이 실시간 객체 인식 코드 – (3) 메인 루프 (실시간 탐지 및 음성 출력)

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        continue

    # YOLOv5 모델을 통해 탐지 수행
    results = model(frame)
    labels = results.pandas().xyxy[0]['name'].tolist()
    rendered = results.render()[0].copy() # 탐지 결과 렌더링
    now = time.time()
    frame_count += 1

    # FPS 계산 및 출력 (10프레임마다)
    if frame_count % 10 == 0:
        fps = frame_count / (now - start_time)
        print(f"fps:{fps:.2f}")

    # 점자블럭 탐지 여부 확인
    if 'dotted' in labels and not dotted_detected:
        print("점자블럭 탐지")
        dotted_detected = True
        last_dotted_time = now

    # 횡단보도 탐지 여부 확인
    if 'crosswalk' in labels and not crosswalk_detected:
        print("횡단보도 탐지")
        # 최근에 점자블럭 탐지되었고 0.5초 이내라면 무시 (증복 안내 방지)
        if dotted_detected and (now - last_dotted_time < 0.5):
            pass
        else:
            crosswalk_detected = True
```



장기 쿨타임 30초 설정 이유

1 사용자 관점

점자블럭+횡단보도 인식이 반복될 때마다 음성이 계속 나오는 건 오히려 혼란을 줄 수 있어서,
30초 동안은 다시 안내하지 않도록 차단(쿨타임)을 설정



기술적 근거

30초는 시각장애인이 실제로 횡단보도를 건너거나 벗어나는데 충분한 시간.
보행 속도(비장애인 대비 40~66%)를 감안하면, 짧지도 길지도 않은 중간 정도의 여유를 주는 타이밍



객관식인식 0.5초 고건의 역할

2 사용자 관점

점자블럭과 횡단보도를 거의 동시에 접하면 사용자는 이를 하나의 상황으로 인식할 수 있음



기술적 근거

0.5초는 약 15프레임 정도의 짧은 순간으로, 같은 상황으로 보기에 적절한 간격

3. 라즈베리파이 실시간 객체 인식 코드 - (3) 메인 루프 (실시간 탐지 및 음성 출력)

```
# 점자블럭+횡단보도 둘 다 탐지 시 음성 출력 (쿨타임 중엔 비활성화)
if dotted_detected and crosswalk_detected and not cooldown:
    print("전방에 횡단보도")
    os.system('espeak -v ko -a 150 -s 100 "전방에 횡단보도"'') # TTS 음성 안내
    cooldown = True
    last_trigger_time = now
    dotted_detected = False # 상태 초기화
    crosswalk_detected = False

# 쿨타임 30초 경과 시 초기화
if cooldown and now - last_trigger_time > 30:
    cooldown = False

# 영상 디스플레이
cv2.imshow('Object Detection + FPS', rendered)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

- 실시간으로 카메라 프레임을 읽어 YOLOv5 모델로 객체를 탐지하고, 점자블럭과 횡단보도를 식별해 각각 상태를 갱신
- 두 객체가 모두 탐지되면 음성 안내를 출력하며, 30초간 쿨타임을 설정해 중복 출력을 방지

라즈베리파이 실시간 객체 인식 코드 – (4) 종료 및 리소스 정리

```
# 리소스 해제  
cap.release()  
cv2.destroyAllWindows()
```

- 탐지 루프를 빠져나오면 카메라를 안전하게 해제하고, 영상 출력 창도 닫아 리소스 누수를 방지
- 마지막까지 코드가 깔끔하게 종료되도록 정리해주는 단계



A horizontal row of seven icons: a large letter 'T', a left arrow, a bookmark, a magnifying glass, a square thumbnail with a mountain image, a share icon, and a speech bubble icon.

ON-SITE PHOTOS



2안) 짹풀기연 영상

⋮

T ⏪ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻

