프 로 젝 트
결 과 보 고 서
- 스마트 주차장 이상행동 감지 시스템 개발 프로젝트 -

학 과
컴퓨터공학과
과 목 명
종합설계프로젝트
담 당 교 수
권중장 교수님
조 원
8조 곽민규 (2020748002) 김준협 (2020548013) 김제민 (2020848010) 김형모 (2022961012)
제 출 일 자
2025년 6월 00일

목 차

1. 서론(프로젝트 개요)

1.1 프로젝트 추진 배경 및 필요성

최근 스마트시티의 발전과 더불어, 주차장과 같은 도시 인프라 공간에서의 안전성 확보 및 효율적인 관리의 중요성이 점차 강조되고 있습니다. 이미 많은 주차장에서는 IoT 기반의 시스템을 도입하여 공간 활용의 효율성을 높이고 있으나, 주차장 내에서 발생할 수 있는 위험 행동이나 범죄를 사전에 탐지하고 예방할 수 있는 기술은 여전히 부족한 실정입니다.

저희 조는 이러한 문제의식에서 출발하여, 범죄로 이어질 수 있는 이상행동을 실시간으로 탐지할 수 있는 시스템의 필요성을 인식하게 되었습니다. 기존의 주차장 CCTV는 단순 녹화 기능에 그치고 있어, 관리자가 직접 영상을 수시로 확인하지 않는 이상 이상행동을 즉각적으로 인지하고 대응하는 데에는 한계가 존재합니다.

이에 본 프로젝트에서는 기존 **CCTV 인프라를 그대로 활용하면서도, 영상처리 및 인공지능 기술을 통해 자동으로 이상행동을 감지할 수 있는 스마트 이상행동 감시 시스템을 구현하는 것을 목표**로 하였습니다.

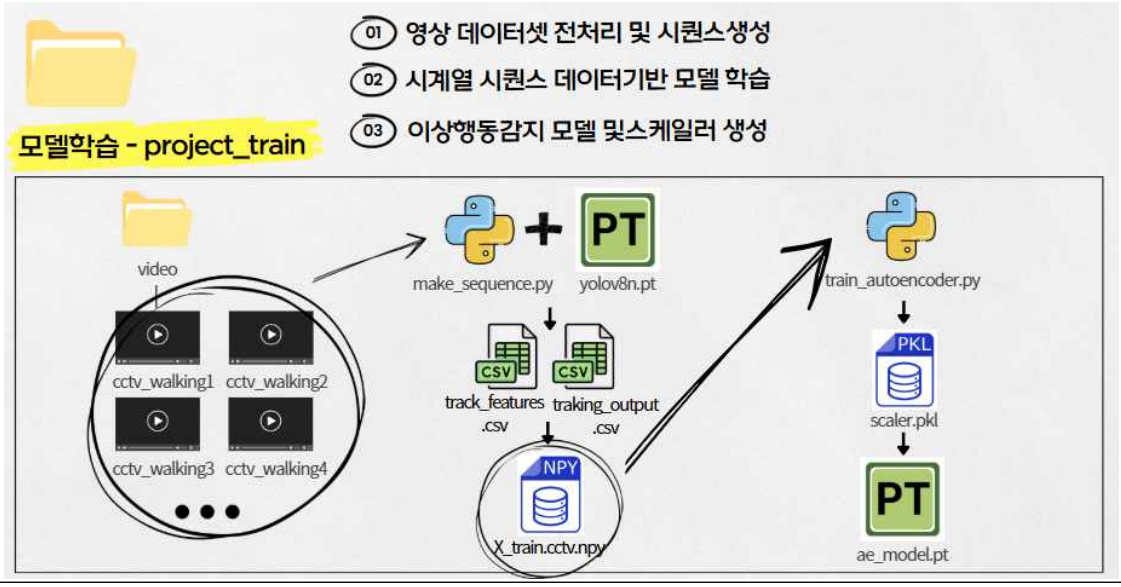
1.2 프로젝트 목표

해당 프로젝트에서 이상행동을 포착하는 방식으론 LSTM AutoEncoder모델을 이용하여 주차장에서의 정상행동을 학습후, 모델이 CCTV영상을 확인하고 오차값을 계산해서 정규화된 점수가 80점이상 3번반복되면 이상행동을 탐지하는 것으로 목표합니다. 또한 복원값(오차값)을 활용하여 객체의 움직임을 일반,배회,뜀으로 구분하여 탐지하도록 목표합니다.

2. 설계(요구분석 및 구조)

본 프로젝트는 전체 과정을 학습 단계와 탐지 단계로 구분하여 설계하였습니다. 실시간 CCTV 영상을 빠르게 처리하고 이상행동을 감지하기 위해, 사전에 정상 행동 데이터를 기반으로 모델을 학습하였으며, 실시간 처리 단계에서는 해당 모델이 입력된 행동을 분석하고 이상 여부만을 판단하도록 구성하였습니다. 전체 시스템의 구성도는 아래 그림과 같습니다.

[학습단계]



학습을 진행하는 project_train 디렉토리에는 다음과 같은 파일들로 구성되어있습니다.

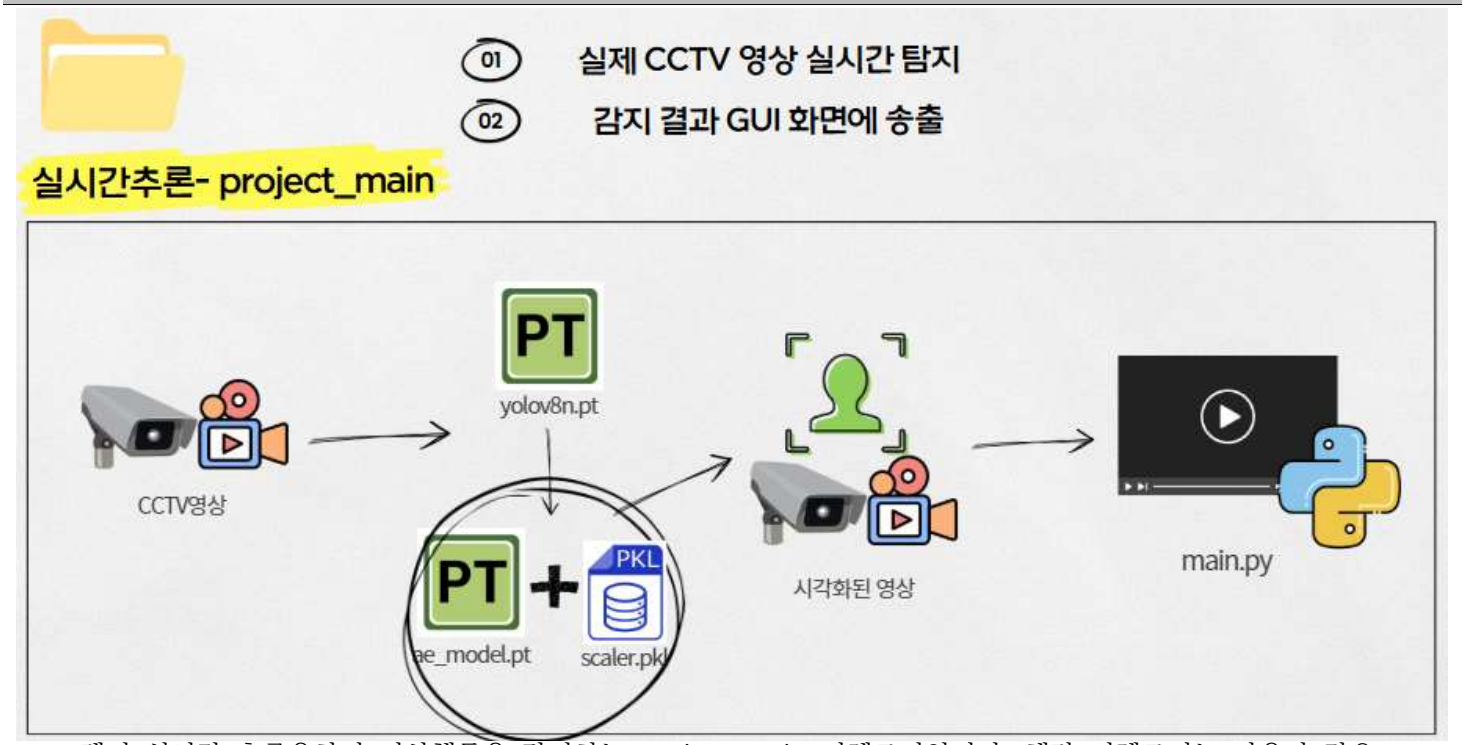
video : 학습에 사용할 영상 데이터가 저장된 디렉토리입니다.

make_sequence.py : video 디렉토리 내 모든 영상을 yolov8n.pt 모델로 분석하여, **객체별 로그(.csv) 및 행동 특징 파일을 생성**하고, 이를 기반으로 학습용 **시퀀스 데이터(.npy)를 생성**합니다.

train_autoencoder.py : 생성된 시퀀스 데이터를 활용해 이상행동 감지 모델을 학습합니다. 학습 전 이상 점수 정규화를 위한 **스케일러 파일(scaler.pkl)을 생성**하여 활용하며, **최종 학습된 모델은 ae_model.pt로 저장**됩니다.

yolov8n.pt : 객체 인식에 사용되는 YOLOv8 모델 파일입니다.

[탐지단계]



모델이 실시간 추론을하여 이상행동을 탐지하는 project_main 디렉토리입니다. 해당 디렉토리는 다음과 같은 파일들로 구성되어있습니다.

main.py : 실시간 탐지와 GUI 출력을 수행하는 메인 프로그램입니다. 사용자는 컴퓨터에 연결된 카메라, RTSP 스트림 주소, 또는 사전 준비된 영상 중 하나를 선택할 수 있으며, 재생 버튼을 클릭하면 실시간 영상 위에 이상행동 탐지 결과가 출력됩니다.

yolov8n.pt : 실시간 객체 인식을 위해 사용되는 YOLOv8 사전학습 모델입니다.

scaler.pkl : 학습단계에서 생성한 이상점수 정규화용 스케일러 파일로, 실시간 추론 시 사용됩니다.

ae_model.pt : 학습된 이상행동 감지 모델로, 실시간 입력 데이터의 이상 여부를 판단합니다.

즉, 위 모든기능은 모두 main.py를 통해 통합되어 실행되며 실시간 영상입력, 객체인식, 이상행동 판단, GUI 출력까지 한 흐름으로 작동합니다.

2.2 개념설계

■ 핵심 아이디어 및 접근방식

프로젝트 초기 기획 단계에서는 '차량 문 반복 개폐 시도', '창문 파손' 등 명확한 범죄 시나리오를 직접 탐지하는 지도학습(Supervised Learning) 기반의 접근법을 우선적으로 고려했습니다. 하지만 이러한 방식은 다음과 같은 현실적인 제약사항이 있었습니다.

- **학습을 위한 데이터 확보의 한계** : 지도 학습 모델을 학습 시키기 위한 실제 범죄상황이 담긴 CCTV영상은 개인정보 및 보안 이슈로 저희가 확보하기 현실적으로 힘들었습니다.
- **구현의 복잡성** : 방대한 유형의 이상행동들을 모두 정의하고 모델링 하는 것은 AI 프로젝트 경험이 터무니없이 부족한 학부생 수준에서 완성도가 극히 떨어질것같다고 판단했습니다.

이러한 제약사항을 해결하기 위해 프로젝트 방향을 비지도 학습(Unsupervised Learning) 기반의 이상탐지 방식으로 전환했습니다. 해당 접근법은 “정상행동”의 패턴만 학습하고, 이 패턴에서 크게 벗어나는 경우를 “이상행동” 으로 간주합니다. 이러한 방식은 앞서말한 지도학습 방식의 제약사항을 다음과 같이 해결할수있었습니다.

- **데이터 확보의 용이성** : 주차장에서의 “정상행동” CCTV영상은 저희가 충분히 촬영해서 확보할수있었습니다.
- **확장성 및 유연성** : 저희가 고려하지못한, 학습되지 않은 새로운 유형의 이상행동도 “정상행동”이 아니라는 기준으로 탐지할 수 있어 더 유연한 대응이 가능했습니다. 또한 “정상”학습만을 학습하면되기 때문에 프로젝트의 난이도는 낮아지고 구현 가능성은 더욱 높아졌습니다.

비지도학습으로 프로젝트 완성도를 높힌후 데이터 확보의 제약사항을 해결하게 되면 지도학습을 결합하여 사용하는 하이브리드구조도 충분히 고려할수있기 때문에 우선 비지도 학습 기반의 이상탐지 방식을 채택하였습니다.

■ 핵심 기술 스택 선정

이러한 접근법을 구현하기 위해, 저희는 YOLOv8 과 LSTM Autoencoder 모델을 조합하는 파이프라인을 설계했으며, 이유는 다음과 같습니다.

- **실시간 객체 탐지(YOLOv8)** : 객체 탐지 분야에는 다양한 사전 학습된 모델이 존재하며, 각각 정확도, 속도, 경량성 측면에서 장단점을 가지고 있습니다. YOLOv8은 모든 면에서 최고의 성능을 보이진 않지만, 여러 객체 탐지 모델을 비교한 연구들[1][2]에 따르면, 실시간 처리 속도와 정확도의 균형 면에서 본 프로젝트의 목표인 '실시간 이상행동 탐지'에 가장 적합하다고 판단하였습니다.
- **시계열 패턴 학습 및 이상탐지** : LSTM은 순서가 있는 데이터, 즉 시간에 따른 영상 프레임의 변화(행동 패턴)를 학습하는 데 탁월한 성능을 보입니다. AutoEncoder 구조는 정상적인 행동 패턴을 입력받아 원본과 유사하게 복원하도록 학습합니다. 만약 정상 범주에서 벗어난 행동이 입력되면, 원본과 큰 차이가 나는 결과물(높은 복원 오차, ReconstructionError)을 출력하며 이를 통해 이상 여부를 정량적으로 판단할 수 있습니다.

결론적으로, YOLOv8과 LSTM AutoEncoder의 조합은 실시간성, 구현 용이성, 그리고 데이터 확보의 현실성이라는 세 가지 측면을 모두 만족시키는 구조이며, 이러한 이유로 본 프로젝트의 핵심 기술 스택으로 선정하였습니다.

2.3 상세설계

■ 데이터셋 설계

본 프로젝트에서는 공개 데이터셋이 아닌, 직접 촬영한 CCTV영상을 기반으로 학습 및 실험을 진행했습니다.

- **데이터 수집**
  - ▶ **촬영장소** : 경성대학교 8호관 주차장
  - ▶ **촬영환경** : 모델의 기본적인 학습 가능성을 검증하기위해 낮 시간대에 고정된 CCTV시야각
  - ▶ **촬영내용** : 주차장에서 발생할 수 있는 “정상행동”으로 일반적으로 사람이 보행하는 영상
  - ▶ **촬영장비** : iPhone13~16 조원의 스마트폰을 활용
  - ▶ **데이터 구성 방식** : 의미 있는 행동 단위로 클립을 분할하여 데이터셋 구성

● **데이터 전처리 및 라벨링**

▶ **라벨링** : 본 프로젝트는 비지도 학습에 기반하므로, 별도의 이상행동 라벨링은 필요하지 않았습니다. 학습 데이터셋에 포함된 모든 행동을 “정상 클래스”로 간주하고 모델을 학습 시켰습니다.

▶ **데이터 전처리 파이프라인** : 수집한 데이터를 LSTM AutoEncoder 모델이 학습 가능한 시계열 특징 벡터 데이터로 가공하기위해, 다음과 같은 순서로 파이프라인을 구축했습니다

1. **객체 추적 및 프레임 추출** : OpenCV 라이브러리를 활용하여 영상에서 프레임을 추출하고, YOLOv8을 활용하여 객체를 탐지했습니다. 이를 통해 영상에 등장하는 각 사람 객체마다 고유ID를 할당하고, ID별로 연속된 프레임 이미지들을 그룹화하여 분석의 대상을 명확히 분리했습니다.
2. **행동 특징 벡터 추출** : 각 프레임에서 객체의 행동을 효과적으로 나타내는 수치적 특징(위치, 속도변화량, 바운딩 박스 좌표등)을 추출하여 시계열 벡터로 변환했습니다.
3. **시퀀스 생성 및 정규화** : 추출된 특징 벡터들을 시퀀스로 구성하여, 최종 모델의 입력데이터를 생성했습니다. 각 시퀀스는 .npy 파일 형식으로 저장하여 학습 효율을 높였습니다. 이후 모델의 안정적인 학습을 통해 MinMaxScaler를 사용하여 모든 특징값을 0과1사이로 정규화했습니다. 학습에 사용된 스케일러는 .pkl 파일로 저장하여 추후 테스트 및 실제 추론시에도 동일한 기준으로 데이터를 변환하는데 사용합니다.

■ **AI 모델 아키텍쳐**

본 프로젝트에서 사용된 딥러닝 모델은 객체 탐지 기능을하는 YOLOv8, 이상 탐지 기능을 하는 LSTM AutoEncoder 이다.

● **객체탐지 모델 - YOLOv8**

▶ **모델 구조** : Ultraytics에서 제공하는 YOLOv8n(nano버전)을 사용하였으며, 사람 클래스만 필터링하여 적용

▶ **가중치** : 사전 학습된 COCO 데이터셋 기반 yolov8n.pt 가중치를 활용

▶ **출력 형식** : 각 프레임별 객체위치(Bounding Box), 클래스, 신뢰도(confience score)

▶ **활용 목적** : 실시간 영상 내 사람 객체 탐지 및 추적에 사용.

● **이상행동 탐지 모델 - LSTM AutoEncoder**

▶ **입력 형태** : (30,N) 크기의 시퀀스 데이터 (각 객체 ID별로 30프레임 동안의 행동 feature가 축적된 .npy배열)

▶ **모델 구조**

LSTM Encoder : 32 -> 16차원

LSTM Decoder : 16 -> 32차원

출력 : 원본 시퀀스를 재구성한 feature배열

▶ **손실함수** : MSE(Mean Squared Error)입력 시퀀스와 복원 시퀀스 간의 평균 제곱 오차를 계산하여 손실로 사용.

▶ **이상 판단 기준** : 재구성 오차값이 임계 값(0.05) 이상일 경우 이상행동으로 간주

2.4 모듈별 설계

■ **H/W 개발 환경**

본 프로젝트의 딥러닝 모델 학습 및 실시간 추론 시스템 개발에 사용된 하드웨어 사양은 일반 사양의 개인 노트북 환경에서 진행되었습니다.

● **실제 개발 환경 및 성능분석**

항목	내용
CPU	Intel(R) Core(TM) i5-10210U
GPU	GeForce MX250
RAM	8GB
저장장치	SSD 256GGB
카메라	iPhone13~16 내장카메라

해당 환경에서 모델학습 및 추론은 주로 CPU를 기반으로 실행되었습니다. 그 결과, 시스템의 기능 구현은 가능했으나 **실시간 영상 스트리밍 처리시 프레임 지연 및 드롭 현상이 관찰**되었습니다. 해당 현상은 실시간 이상행동 탐지 시스템의 실시간성을 떨어뜨리는 문제로 **실제 서비스 환경에서는 GPU 연산 가속 및 충분한 메모리의 사양이 필수적**임을 확인했습니다.

■ S/W 개발 환경

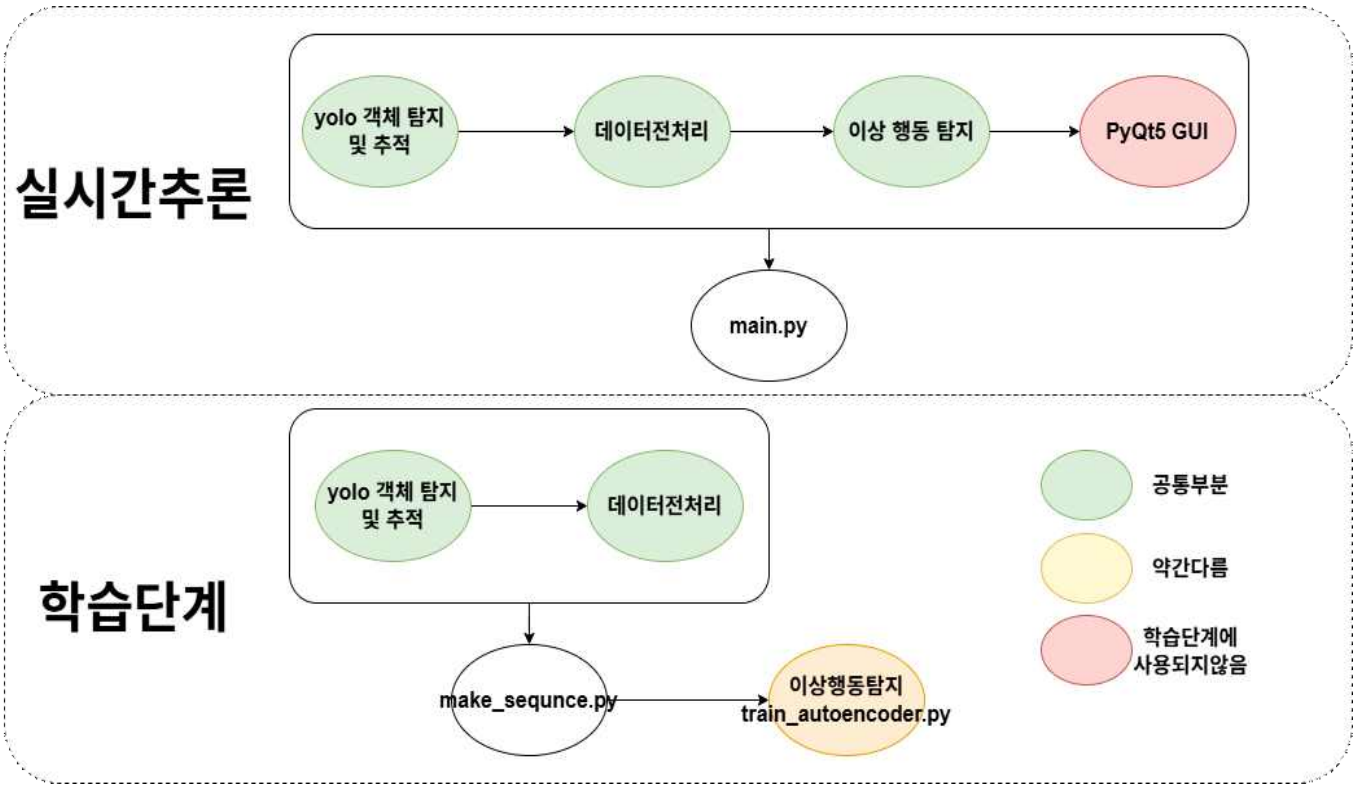
본 프로젝트 개발에 사용된 주요 소프트웨어 및 라이브러리 버전은 다음과같습니다.

● **프로젝트 주요 라이브러리 버전 및 용도**

항목	버전	용도 설명
Python	3.9.13	전체 개발 환경 기반
PyTorch	2.6.0	LSTM AutoEncoder 모델 구현 및 추론
Ultralytics YOLOv8	8.3.109	YOLOv8 객체 탐지 모델
OpenCV	4.11.0.86	영상 프레임 추출, 영상 처리
NumPy	1.26.4	수치 연산, 시퀀스 구성 등
Pandas	2.2.3	데이터 구조화 및 처리 보조
joblib	1.4.2	모델 저장/불러오기, 정규화 파라미터 관리 등
PyQt5	5.15.11	GUI 구성 및 시각화
scikit-learn	1.6.1	데이터 정규화(StandardScaler), 일부 전처리
torch.nn	(PyTorch 내부)	AutoEncoder 모델 구성 (LSTM 포함)

■ 시스템 모듈별 상세 기능

본 프로젝트의 실제 구현은 하나의 main.py 파일에 통합되어 있지만 다음과 같이 4개의 논리적 모듈로 명확히 구분하여 설계되었습니다. 각 모듈은 파이프라인 형태로 데이터를 처리하며 순차적으로 작동합니다.



각 모듈들은 각 단계에서 공통적인 기능을 하기도 하고, 조금씩 다른 역할을 하는 부분이 있기 때문에 각 모듈별로 공통기능, 각 단계에서의 역할을 구분서 설계했습니다.

- **YOLO 객체 탐지 및 추적 모듈**

  공통 기능 : 입력 프레임에서 '사람' 객체를 탐지하고 Bounding Box 정보 추출 및 ID부여 후 연속 추적

  학습 단계 : .mp4 영상 → 사람 객체 위치 추출 및 객체 기준 특징 추출을 위한 데이터 생성

  실시간 추론 : 실시간 프레임마다 사람 탐지 → ID 별 행동 분석용 시퀀스관리

- **데이터 전처리 모듈**

  공통 기능 : Bounding Box 정보로 행동특징 벡터생성 및 정규화

  학습 단계 : 행동 특징벡터를 추출하고 해당 벡터들을 모델 학습용 시퀀스 데이터셋 구축

  실시간 추론 : 추적중인 각 객체에 대한 특징벡터를 정규화하고 이후 시퀀스를 구성하여 이상행동 분석모듈로 전달.

- **이상 행동 탐지 모듈**

  공통 기능 : LSTM AutoEncoder 모델을 기반으로 시퀀스 재구성 오차 계산.

  학습 단계 : 데이터 전처리 모듈이 생성한 시퀀스 데이터셋을 입력받아 AutoEncoder 모델에 학습을 진행, 이후 모델의 가중치를 저장.

  실시간 추론 : 학습된 모델을 로드하여, 실시간으로 입력되는 시퀀스의 재구성오차를 계산후 임계값 초과여부를 판단하여 이상행동을 판단.

- **PyQt5 GUI 모듈**

  학습 단계 : 학습 단계에서 사용되지않음.

  실시간 추론 : 모든 처리 결과를 사용자에게 보여주며, 영상 로딩 및 시각화된 영상을 스트림으로 출력하고 Bounding Box를 표시, 이상행동 판단시 붉은색 경고 메시지로 강조

2.5 실험 및 테스트계획

■ 테스트 시나리오

모델이 다양한 실제 상황에서 어떻게 반응하는지 검증하기위해, 네 가지 핵심 시나리오를 다음과같이 설계했습니다.

시나리오	상세설명	주요 검증 목적
배회	단일 인물이 특정 구역을 느린속도로 반복적으로 이동	배회(Loitering) 패턴에 대한 탐지 성능확인
질주	단일 인물이 빠른속도로 화면을 가로 질러이동	질주(Running)와 같이 속도 기반의 이상행동 탐지 성능 확인
일반 행동	촬영된 다수의 정상행동(사람의 보행)	정상 상황을 이상으로 판단하는 오탐지 발생 여부확인
이상 행동	주차장에서의 임의의 수상한행동	이상행동으로 판단하는지여부
다중 객체	다수의 인물이 동시에 각기 다른 행동을 하는 상황	각 객체를 개별적으로 추적하고 독립적으로 행동을 분석하는지 확인
폭행	객체의 끼리의 폭행상황	폭행 상황을 이상 판단 발생여부 확인

■ **테스트 결과 기반 모델 개선 계획**

초기 테스트 결과에 따라 발생할 수 있는 주요 문제점을 정의하고, 이에 대한 해결방안은 다음과 같습니다.

- **높은 오탐지율 발생시** : 예측 범위를 조금만 벗어나는 행동도 이상행동으로 판단하는 경우.
  - ▶ **개선 방안** : 다양한 정상행동 데이터를 학습 데이터셋에 추가, 보강하여 재학습 진행.
- **높은 미탐지율 발생시** : 이상행동의 패턴과 정상행동의 특징적인 차이가 적어서, 모델이 구분하지 못하는경우
  - ▶ **개선 방안** : LSTM AutoEncoder의 재구성 오차 임계값을 조정하여 탐지 민감도를 조정한다.

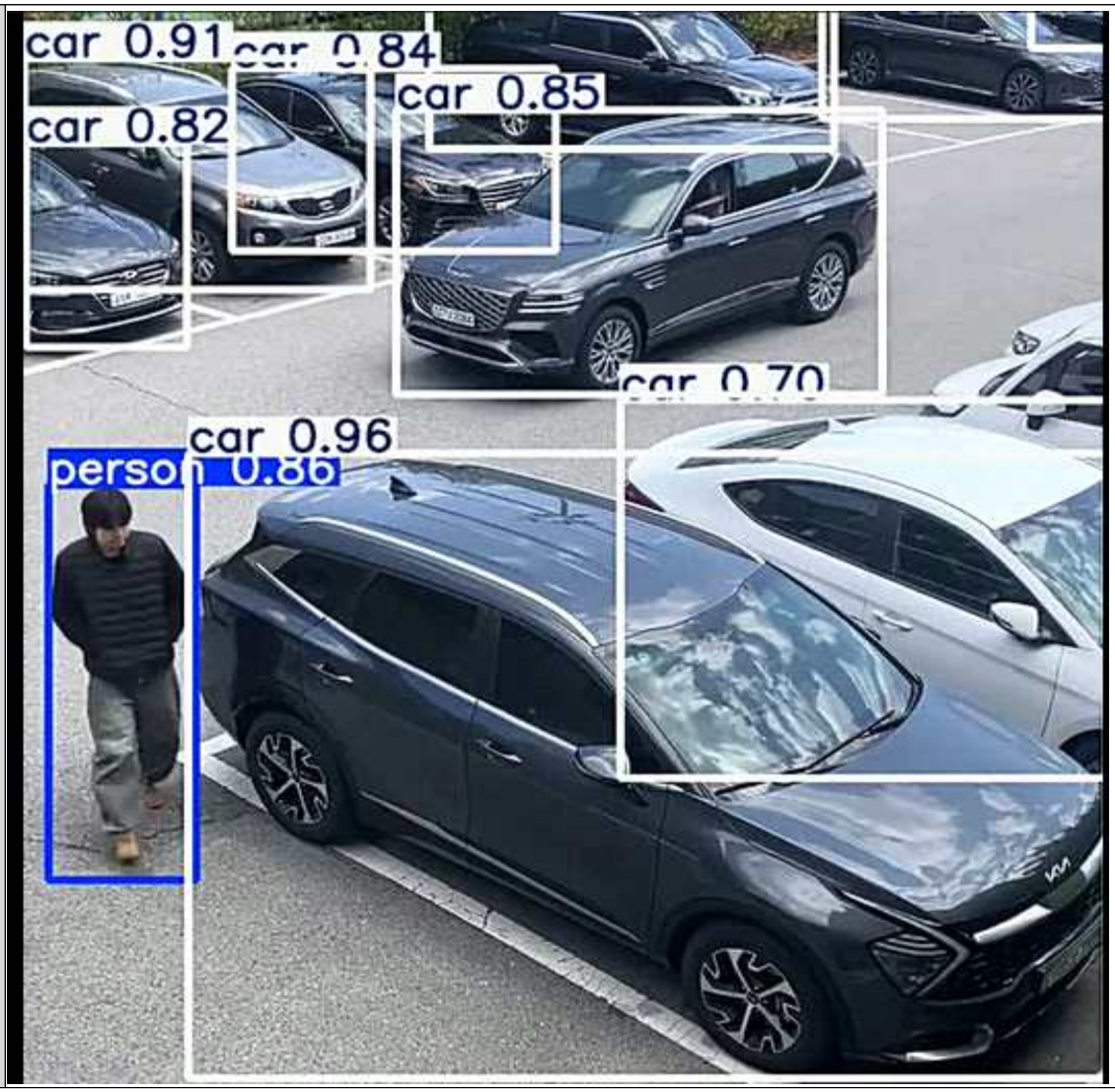
3. 구현

본 프로젝트에선 이상행동을 탐지하는 소프트웨어 시스템을 집중적으로 구현했습니다. 소프트웨어의 최종모델은 아래의 과정을 거쳐서 최종 구현되었습니다.

■ **1차 모델 (객체탐지)**

- **목표 : 입력받은 영상에서 YOLOv8을 사용해 영상에 나오는 객체 탐지 및 시각화**
- **테스트 영상** : 경성대학교 8호관 주차장에서 고정된 CCTV각도로 촬영한영상
- **결과** : 영상에나오는 객체들을 정상 탐지, 탐지한 객체에는 클래스와 신뢰도를 나타내는 Bounding Box를 시각화했음. 시각화된 영상은 같은 디렉토리에 output_annotated.avi로 저장함.

결과 사진



[첨부파일 1차모델 실행결과 사진]

■ 2차 모델 (ID 부여 및 로그 생성)

- **목표 : YOLOv8로 탐지된 객체에 ID를 부여 후 프레임별로 ID 기반 로그를 추출 및 시각화**
- **테스트 영상** : 경성대학교 8호관 주차장에서 고정된 CCTV각도로 촬영한영상
- **구현 과정 및 문제점** : 객체에 ID를 부여하고 해당 객체를 추적하기위해 DeepSORT객체추적 프레임워크를 사용하였다. 그러나 테스트중 다음과 같은 문제가 발생함
  1. 사람은 한명인데 여러 개의 ID가 중복생성됨.
  2. 동일 객체에 대해 중복된 박스가 표시됨.
  3. 로그 파일에도 같은 사람이지만 서로 다른 ID가 추적된 기록 존재함.
- **해결과정** : 다음과 같은 방법들을 시도했으나 완전한 해결에 실패함.
  ▶ DeepSORT의 max_age, n_init 파라미터값 조정
  ▶ ID병합로직, IoU 기반 박스 중복 제거등 후처리 로직

  최종적으로 **클러스터링 기반 로직을 직접 구현**하여 문제를 해결함. 핵심원리는 다음과같다.
  1. YOLO의 탐지 결과 중 클래스가 person인 객체만 필터링
  2. 중심 좌표 간의 거리가 일정 값 이하인 박스들을 하나의 클러스터(그룹)로 간주
  3. 각 클러스터에서 confidence(신뢰도가 가장 높은 박스 하나만 남김
  4. 차량 등 기타 객체는 별도 처리하지 않고 그대로 유지
- **결과 : 동일 인물에 중복되지 않는 ID가 정상부여, 불필요한 중복박스를 제거하여 시각화 및 ID기반 로그를 정상적으로 생성하였다.**

문제상황	정상 실행 결과

frame	time	track_id	class	x_center	y_center
1	0.03	1	person	107	425
1	0.03	2	car	419	562
1	0.03	3	car	123	149
1	0.03	4	car	254	136
1	0.03	5	car	60	192
1	0.03	6	car	414	86
1	0.03	7	car	630	64
1	0.03	8	car	563	432
1	0.03	9	car	664	296
2	0.07	1	person	110	425
2	0.07	2	car	419	562
2	0.07	3	car	123	148
2	0.07	4	car	254	136
2	0.07	5	car	61	192
2	0.07	6	car	414	86
2	0.07	7	car	630	64
2	0.07	8	car	564	432
2	0.07	9	car	664	295

[tracking_output.csv]

[첨부파일 2차모델 실행결과 사진]

■ 3차모델 (시퀀스기반 이상행동감지)

해당 과정이후부터 각 기능을 하는 모듈들로 분리하여 개발을 진행했음. 개발한 모듈과 기능은 다음과 같다.

2_tracking_cctv.py : 기존 모델인 cctv.py파일명을 변경.

3_extract_motion_features.py : 행동특징을 추출.

4_make_sequence.py : 행동특징들을 모델이 학습할 수 있는 시퀀스단위 파일로 변경

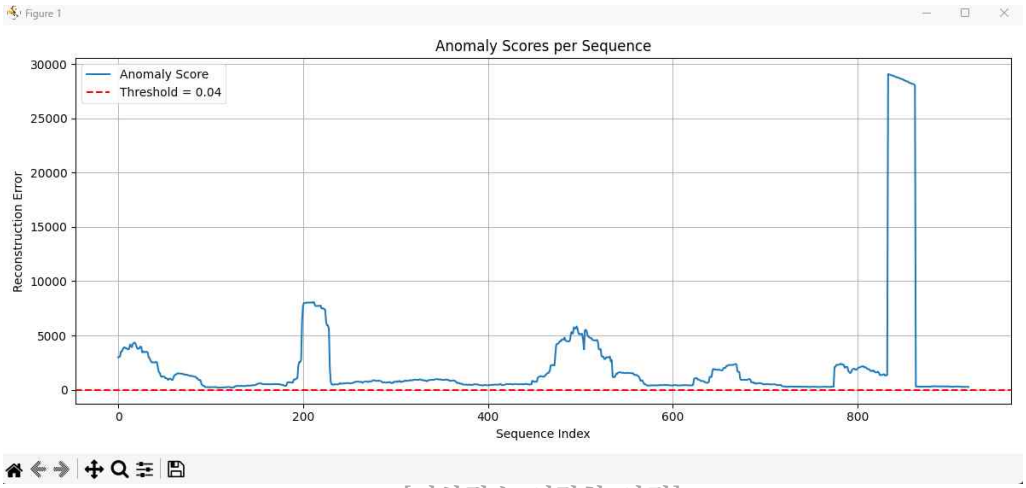
5_anomaly_detection_lstm.py : 시퀀스 데이터를 학습, 이상점수 계산 및 시각화

6_visualize_anomalies.py : 원본영상에 이상행동 탐지 시각화

- **목표 : 추적로그에서 객체의 행동특징을 추출 및 시퀀스화, 이상행동 탐지 모델 구성**
- **테스트 영상** : 경성대학교 8호관 주차장에서 고정된 CCTV각도로 촬영한영상 및 YouTube에서 확보한 일반적인 테스트동영상
- **구현 과정** : 객체추적로그(tracking_output.csv)로부터 이동 속도, 프레임 간 거리, 방향 변화량등을 계산해서 행동특징파일(track_features.csv)로 추출하였으며 추출된 특징은 모델이 .npy형식의 시퀀스데이터(X_train.npy)로 저장. 이후 LSTM AutoEncoder에 시퀀스 데이터를 입력하여 재구성오차를 계산하고 이를 기반으로 이상행동을 판단.

- **결과 및 문제점** : 모델을 구성하고, 시퀀스 데이터를 학습시켜 재구성 오차값을 기반으로 이상행동을 탐지할 수 있는 학습 및 추론 파이프라인을 구축하였다. 하지만, 해당 과정에서는 아래와 같은 문제점들이 있었다.
  1. 현재 단일 X_train.npy 파일만으로 학습 및 탐지를 수행함으로써 모델의 학습이 부족하다. 때문에 현재 오탐률은 100%로 이상행동을 판단할 수 없다.
  2. 이상행동 발생시 해당결과가 CCTV 전체 화면에 박스로 표시되어 어떤 객체가 이상행동을 보였는지 식별이 불가능하다.
  3. 일관되지않은 유튜브에서 확보한 테스트영상을 사용함.
- **개선방안**
  1. 다양한 시퀀스 데이터를 확보하고, 모델 학습시 가중치를 누적 저장할 수 있도록 구조를 개선한다.
  2. 이상행동 발생 시, 해당 객체의 Bounding Box 색상을 변경하는 방식으로 이상행동이 발생한 객체를 시각적으로 구분할 수 있도록 한다.
  3. 일관된 장소와 고정된 각도의 테스트영상을 촬영해서 데이터를 확보한다.
- **실행결과 및 해석**

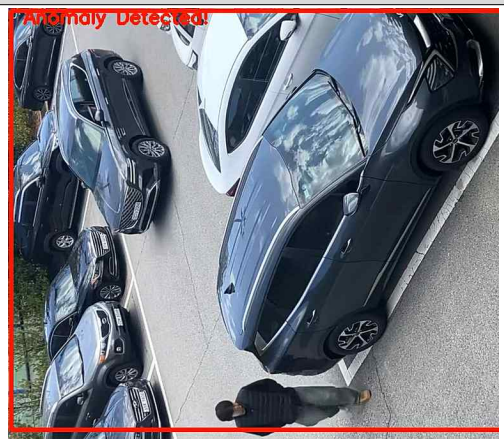
해당 그림은 단일 시퀀스 파일로 모델을 학습한뒤, 테스트 영상을 입력하여 추론한 이상점수 시각화 결과이다.



[이상점수 시각화 사진]

1. 파란색 실선은 각 시퀀스에 대한 재구성 오차값이며, 빨간색 점선은 이상행동 판단을 위한 임계값(Threshold = 0.04)이다.
2. 전체 시퀀스 중 특정 구간에서 재구성 오차가 비정상적으로 높게 폭등하는 현상이 관찰되며, 이는 실제 이상행동 여부와 무관하게 발생하였다.
3. 일부 정상 행동 시퀀스에서도 높은 이상 점수(최대 30,000 이상)가 출력되고 있으며, 이는 모델이 정상과 이상을 구분하지 못하고 있다는 신호다.

시각화 실행결과



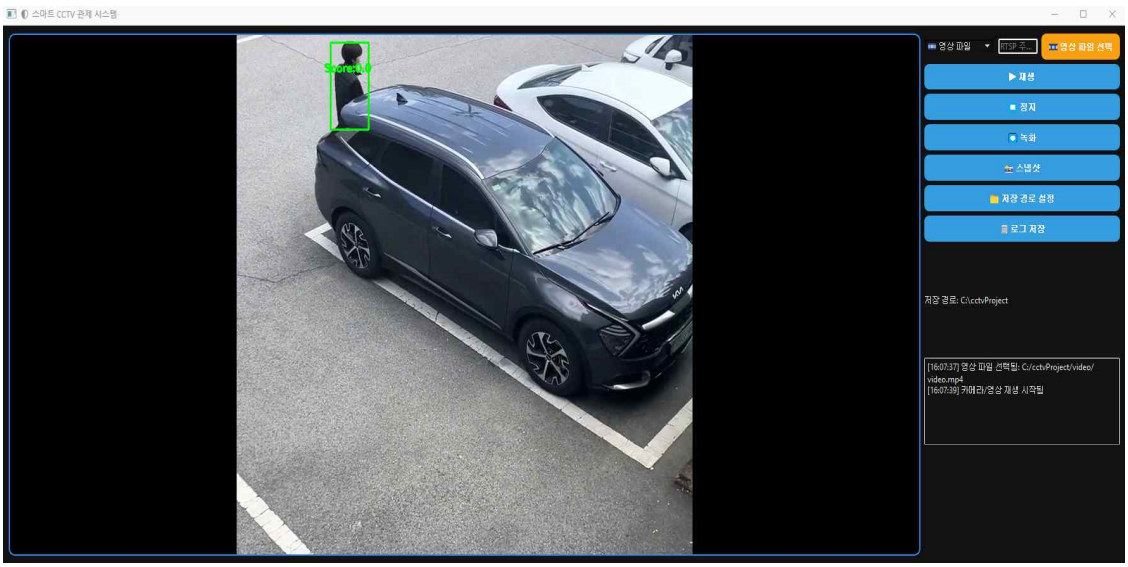
[첨부파일 3차모델 실행결과 사진]

■ 4차모델 (모델학습 분리 + GUI적용)

- **목표 : 모델의 학습 기능과 추론기능을 분리하여 실시간 탐지성능을 최적화하고 김준협 조원에게 받은 GUI화면 코드에 실시간 기능 코드를 병합한다.**
- **테스트 영상** : 경성대학교 8호관 주차장에서 고정된 CCTV각도로 촬영한영상
- **구현 과정** : 3차 모델 구현이후 YOLO객체탐지 -> 객체로그추출 -> 행동특징 추출 -> 시퀀스화 -> 모델 학습 -> 모델 추론 까지의 파이프라인을 하나의 main.py 파일로 통합시켰다. 해당 파일의 구조는 전체 파이프라인이 동시 실행되어 처리 시간이 과도하게 발생하여 실제 CCTV의 시간과 재생되는 시각화된 영상의 시간차가 벌어지는 문제가 발생하였다. 해당 과정속에서 문제를 해결하기위해 모델을 다음과같이 학습 단계와 추론단계로 분리하였다.
  - ▶ **학습단계** : train.autoencoder.py 모듈을 별도로 구성하여 시퀀스데이터(X_train.npy)를 입력받고, 학습을 진행후 모델 가중치(ae_model.pt)를 저장하는 방식으로 분리하였다.
  - ▶ **추론단계** : 저장된 ae_model.pt 파일을 로드하여 실시간 추론 기능만 진행하는 main_fps_synced.py 모듈을 별도로 구성하였다.

  이후 김준협 조원이 개발한 PyQt 기반의 GUI코드(demo_gui.py)에 실시간 추론기능을 통합하여 영상 선택후 재생시 이상행동 탐지가 실시간으로 시각화되는 GUI프로그램을 완성하였다.
- **문제점** : 새로운 파이프라인을 구성하고 실행환경을 구현했지만 모델학습을 추가로 진행하지않았음. 때문에 이상행동탐지 기능은 미흡하다.

결과 사진



[첨부파일 4차모델 demo_gui.py 실행결과 사진]

■ 5차모델 (최종모델)

- **목표 : 다량의 정상행동 데이터를 기반으로 학습 안정성과 이상행동 분류 정밀도를 개선하고, 기초적인 이상행동을 구분할 수 있는 로직을 적용한다.**
- **테스트 영상** : 경성대학교 8호관 주차장에서 고정된 CCTV각도로 촬영한영상. 이전보다 다량의 데이터영상을 확보하였고 테스트할 수 있는 이상행동영상들도 확보하였다.
- **구현 과정** :
  - ▶ **학습 데이터 확장** : 다양한 시나리오로 구성된 정상행동 데이터를 대량 확보했으며 해당 영상들중에서도 YOLO객체인식이 잘진행되는 영상들로 우선적으로 활용했다.
  - ▶ **시퀀스 학습 구조 개선** : 이전 모델들까지 단일 시퀀스만 학습할 수 있는 구조였지만 여러 시퀀스파일을 한번에 학습할수있도록 개선하였으며 학습과정 중에서 재구성 오차값이 비정상적으로 큰수치로 출력되는 현상이 반복되어서 학습이 불안정한 문제를 정규화 로직을 도입하여 해결하였다.

시퀀스 학습 구조 개선 핵심코드

```
# === 모든 .npy 시퀀스 파일 불러오기 ===
npy_files = glob.glob("*.npy")
X_all = []

for file in npy_files:
    try:
        data = np.load(file)
        if data.ndim == 3 and data.shape[2] == 2:
            X_all.append(data)
            print(f"✅ 로드됨: {file} → shape: {data.shape}")
        else:
            print(f"⚠️ 건너뜀(형식불일치): {file}")
    except Exception as e:
        print(f"❌ 오류 발생: {file} → {e}")

# === 병합 및 정규화 ===
X = np.concatenate(X_all, axis=0)
X_reshaped = X.reshape(-1, 2)

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_reshaped)
X_scaled = X_scaled.reshape(X.shape)

# === 정규화 스케일러 저장 ===
joblib.dump(scaler, "scaler.pkl")
print("✅ 정규화 스케일러 저장 완료 → scaler.pkl")
```

[핵심 동작코드설명]

1. glob.glob("*.npy")
X=np.concatenate(X_all,axis=0)
: 다수의 시퀀스파일(.npy)을 병합하여 하나의 데이터셋 으로 구성
2. scaler=MinMaxScaler()
: 전체 특징 데이터를 0~1로 범위로 정규화
3. joblib.dump(scaler, "scaler.pkl")
: 정규화시킨 데이터를 실시간 추론시 재사용하기위해 .pkl 파일로저장.

학습 구조 개선후 정상행동을 이상행동으로 탐지하는 오탐률은 약 5% 이하로 낮아졌다. 그러나 실제 이상행동에

결과 사진

```
✅ 로드됨: X_train_cctv_걷기8.npy → shape
✅ 로드됨: X_train_cctv_걷기9.npy → shape
✅ 정규화 스케일러 저장 완료 → scaler.pkl
Epoch 1 Loss: 3.162907
Epoch 2 Loss: 1.229172
Epoch 3 Loss: 1.162208
Epoch 4 Loss: 1.040672
Epoch 5 Loss: 0.885656
Epoch 6 Loss: 0.774097
Epoch 7 Loss: 0.694455
Epoch 8 Loss: 0.653750
Epoch 9 Loss: 0.597950
Epoch 10 Loss: 0.571970
✅ 모델 저장 완료 → ae_model.pt
```

모델 학습 이후 재구성 오차값이 전반적으로 유의미하게 감소하였으며,
학습이 진행될수록 손실 값(Loss) 또한 지속적으로 감소하는 경향을 보였다.
이는 모델이 정상행동의 패턴을 안정적으로 학습하고 있으며, 학습 과정이 정상적으로 수렴하고 있음을 의미한다.

대한 탐지율은 여전히 낮은수준으로 실질적인 위험을 감지하지는 못하는 한계가 발생하였다. 이를 보완하기 위해 기초적인 이상행동을 식별할 수 있는 간단한 룰 기반 탐지로직을 작성하였다.

▶ **기초적인 이상행동 탐지 로직** : 이상행동을 탐지하는 오탐률이 극히 떨어짐과 동시에 진짜 이상행동을 탐지해야하는 률도(단어생각이안남) 떨어졌다. 해당 문제상황을 해결하기위해 간단히 기본적인 이상행동(배회,뜀)정도를 탐지할 수 있는 로직을 작성하였다.

기초적인 이상행동 탐지 로직

```
if len(buf) >= 30:
    seq = np.array([[b[2], b[3]] for b in buf], d
    seq_scaled = self.scaler.transform(seq)
    x = torch.tensor(seq_scaled.reshape(1, 30, 2)

    dist_seq = seq_scaled[:, 0]  # 정규화된 distan
    speed_seq = seq_scaled[:, 1]  # 정규화된 speed

    avg_speed = np.mean(speed_seq)
    total_dist = np.sum(dist_seq)
    high_speed_count = np.sum(speed_seq > 0.3)

    if avg_speed < 0.1 and total_dist < 0.2:
        label = "Loitering"
        color = (0, 165, 255)  # Orange
    elif high_speed_count >= 2:
        label = "Running"
        color = (255, 100, 0)  # Blue-Orange
```

[핵심 동작코드설명]

len(buf)
: 각 객체별 최근 30프레임 중심 좌표, 속도정보 저장.'
2. scaler.transform(seq)
: 학습 시 저장한 MinMaxScaler를 사용해 거리/속도값 정규화
3. high_speed_count
: 정규화된 속도가 0.3을 넘는 프레임 수 카운팅 (뜀 판단근거)
4. avg_speed... , high_speed_count
: 객체가 거의 움직이지않고 제자리 근처를 반복 -> Loitering(배회)로 간주.
빠르게 이동한 프레임이 2카운트 이상이면 Running(뜀)으로 간주.

● **최종모델 핵심코드**

1, 2, 3, 4차 모델을 통해 단계별로 개선한결과 최종모델은 **시퀀스데이터추출, 모델학습, 실시간추론과** 같은 3가지 핵심파트로 통합구현되었으며 해당부분의 핵심코드는 아래와같다.

▶ **시퀀스추출 - make_sequence.py**

해당 모듈은 YOLO기반 객체추적(run_tracking()), 행동 특징 추출(extract_features()), 시퀀스 구성 및 저장(make_sequences()) 세가지 핵심 기능들로 구성되어있다. 다음은 해당기능들의 핵심코드와 설명이다.

1. **YOLO기반 객체추적 - run_tracking()**

YOLOv8과 DeepSORT를 활용하여 영상 속 사람 객체를 탐지 및 추적한 후, 프레임단위로 객체 ID, 시간, 클래스, 중심 좌표 정보를 CSV로 저장한다.

run_tracking() 함수 일부

```
results = model.predict(frame, conf=0.3, classes=[0])[0]
boxes = results.boxes.xyxy.cpu().numpy()
classes = results.boxes.cls.cpu().numpy()
confs = results.boxes.conf.cpu().numpy()

clustered_persons = cluster_and_filter(boxes, classes, confs)
filtered = [(box, conf, int(classes[idx])) for idx, conf, box

for i in range(len(boxes)):
    cls = int(classes[i])
    if cls == 2:
        x1, y1, x2, y2 = boxes[i]
        if x2 - x1 >= 30 and y2 - y1 >= 60:
            filtered.append((boxes[i], confs[i], cls))

input_dets = [([x1, y1, x2 - x1, y2 - y1], conf, cls) for (x1,
tracks = tracker.update_tracks(input_dets, frame=frame)

with open(output_csv, "a", newline='') as f:
    writer = csv.writer(f)
    for track in tracks:
        if not track.is_confirmed():
            continue
        raw_id = track.track_id
        cls = track.det_class
        x1, y1, x2, y2 = map(int, track.to_ltrb())
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

        if raw_id not in id_alias_map:
            matched = False
            for aid, (px, py, lf) in last_seen.items():
                if frame_count - lf < FRAME_GAP and euclidean(
                    id_alias_map[raw_id] = aid
                    matched = True
                    break
            if not matched:
                id_alias_map[raw_id] = next_alias_id
                next_alias_id += 1

        alias_id = id_alias_map[raw_id]
        last_seen[alias_id] = (cx, cy, frame_count)

        writer.writerow([frame_count, round(frame_count / fps
```

YOLO로 사람만 필터링 감지

중복 박스 제거를 위해 cluster_and_filter()사용

DeepSORT로 ID 추적

시간, 추적ID, 클래스명, 중심 X, 중심 Y작성 결과는 tracking_output_영상이름.csv 로저장.

2. **행동 특징 추출 - extract_features()**

extract_features() 함수

```
def extract_features(track_csv, output_csv):
    df = pd.read_csv(track_csv)
    df = df[df["class"] == "person"]
    results = []
    for track_id, group in df.groupby("track_id"):
        group = group.sort_values("frame").reset_index(drop=True)
        group["dx"] = group["x_center"].diff()
        group["dy"] = group["y_center"].diff()
        group["distance"] = np.sqrt(group["dx"]**2 + group["dy"]**2
        group["dt"] = group["time"].diff().replace(0, np.nan)
        group["speed"] = group["distance"] / group["dt"]
        group.fillna(0, inplace=True)
        results.append(group)
    feature_df = pd.concat(results, ignore_index=True)
    feature_df.to_csv(output_csv, index=False)
```

각 사람 객체의 프레임단위 중심좌표를 기반으로 이동거리, 속도등 행동 특징 벡터 생성

프레임 간 좌표 차이 → 거리계산

프레임 간 시간 차이로 속도 추정

결과는 track_features_영상 이름.csv 으로 저장

3. 시퀀스 구성 및 저장 - make_sequences()

make_sequences() 함수

```python
def make_sequences(feature_csv, npy_path, sequence_length=30):
    df = pd.read_csv(feature_csv)
    sequences = []
    for _, group in df.groupby("track_id"):
        group = group.sort_values("frame").reset_index(drop=True)
        for i in range(len(group) - sequence_length + 1):
            seq = group.iloc[i:i + sequence_length][["distance", "speed"]].values
            sequences.append(seq)
    X_train = np.array(sequences)
    np.save(npy_path, X_train)
```

각 객체의 distance/speed 시계열 벡터를
길이 30짜리 시퀀스로 나눠 .npy 로저장.

▶ 모델학습 - train_autoencoder.py

이 모듈은 LSTM AutoEncoder 모델을 정의하고, 시퀀스 데이터를 정규화하여 안정적인 모델 학습이 가능하도록 구성한다. 정규화 기능은 최종 모델 구현 과정에서 이미 소개하였으므로 이 항목에서는 모델 정의 및 학습에 관한 핵심 코드만을 다룬다.

1. LSTM AutoEncoder 모델정의

class LSTMAutoEncoder 정의 부분

```python
# === LSTM AutoEncoder 모델 정의 ===
class LSTMAutoEncoder(nn.Module):
    def __init__(self, input_dim=2, hidden_dim=16):
        super().__init__()
        self.encoder = nn.LSTM(input_dim, hidden_dim, batch_first=True)
        self.decoder = nn.LSTM(hidden_dim, input_dim, batch_first=True)

    def forward(self, x):
        _, (h, _) = self.encoder(x)
        repeated = h.repeat(x.size(1), 1, 1).permute(1, 0, 2)
        out, _ = self.decoder(repeated)
        return out
```

입력된 행동 시퀀스(x)전체를
단 하나의 핵심 요약벡터 h로 압축.

핵심 요약 벡터 h 정보만을 가지고
복원 하는부분
이 결과와 원본의 차이가 재구성 오차값(이상점수)

2. 학습 수행

모델이 학습 하는부분

```python
for epoch in range(10):
    total_loss = 0
    for i in range(0, len(X_tensor), 32):
        batch = X_tensor[i:i+32]
        output = model(batch)
        loss = criterion(output, batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1} Loss: {total_loss:.6f}")
```

전체 데이터(X_tensor)를 10번 반복해서 학습시킴
데이터가 너무 크기때문에
32개씩 작은 묶음(batch)으로 나누어학습.

모델이 복원한결과(output)과
원본데이터 (batch) 의 오차값을 계산.
loss(오차값)이 높을수록 복원이 잘안된것

!! 실제 학습이 일어나는 순간
손실값(loss)을 보고, 원인을 분석(backword)
다음에 틀리지않도록 수정(step)

이후 모델의 가중치는 ae.model.pt로 저장.

▶ 실시간 추론 및 GUI - main.py

이 모듈은 카메라 또는 영상 파일을 입력으로 받아, 사전에 저장된 모델 가중치를 로드하여 실시간으로 추론을 수행하는 기능을 담당한다. 특히, <u>실시간 추론의 핵심 기능인 update_frame 함수</u>는 다음과 핵심적인 파이프라인을 따라 동작한다.

update_frame

1. 객체 탐지 및 추적 및 행동특징 추출

```
def update_frame(self):
    if not self.cap: return
    ret, frame = self.cap.read()
    if not ret: return

    if self.rotate_checkbox.isChecked():
        frame = cv2.rotate(frame, cv2.ROTATE_90_CLOCKWISE)

    results = self.yolo.predict(frame, conf=0.5)[0]
    boxes = results.boxes.xyxy.cpu().numpy()
    classes = results.boxes.cls.cpu().numpy()
    confs = results.boxes.conf.cpu().numpy()

    clustered = cluster_and_filter(boxes, classes, confs)
    filtered = [(box, conf, int(classes[idx])) for idx, co
    input_dets = [([int(x1), int(y1), int(x2), int(y2)], co
    tracks = self.tracker.update(input_dets)

    for tid, (cx, cy) in tracks:
        buf = self.track_buffers[tid]
        if buf:
            px, py, _, _ = buf[-1]
            dist = np.hypot(cx - px, cy - py)
            speed = dist / 0.033
        else:
            dist = speed = 0
        buf.append((cx, cy, dist, speed))
```

YOLOv8 모델이 프레임 안의 사람 객체를 찾고 ID를 부여

객체의 행동특징을 계산하여 저장.
모델이 실시간 추론할 데이터들을 저장하는역할.

2. 이상행동 분석

이 단계는 미리 학습된 AI 모델이 실질적인 판단을 내리는 과정이다.

```
self.model.load_state_dict(torch.load(resource_path( ae_model.pt )
self.model.eval()
```

update_frame 함수 외부 프로그램 시작시 ae_model.pt 에 저장된 학습된 가중치를 모델에 로드함.

```
with torch.no_grad():
    recon = self.model(x)
    score = torch.mean((x - recon) ** 2).item()
    scaled = 100 * (score - self.MIN_SCORE) / (self.MAX_SCORE - self.MI
```

생성된 특징 시퀀스 x를 모델에 입력하여, 모델이 생각하는 정상적인 복원된 결과(recon)을 추출
재구성 오차를 score로 계산
해당 값을 정규화.
즉 원본과의 차이 score는 이상행동을 나타내는 점수.

3. 최종 판단 및 시각화

이 단계는 미리 학습된 AI 모델이 실질적인 판단을 내리는 과정이다.

```
if self.persistence[tid] >= 3:
    label = "Anomaly"
    color = (0, 0, 255)  # Red

cv2.putText(frame, f"{label} ({scaled:.1f})", (cx - 50, cy - 2
cv2.rectangle(frame, (cx - 30, cy - 60), (cx + 30, cy + 60), c
```

이상 점수가 3프레임 연속 높게 유지될때 최종적으로 Anomaly 라고판단.
해당로직은 순간적인 오류로 오탐을 방지하는 안전장치.

결과를 프레임위에 텍스트와, 사각형으로 시각화.

3. 실험 결과 및 분석

최종 완성된 모델을 설계 단계에서 계획한 테스트 시나리오에 따라 테스트하고 결과를 분석합니다. 최종 모델은 [최종모델 학습영상] 디렉토리의 정상적인 걷기영상 14개 항목으로 학습을 진행했습니다.

■ 배회

● 테스트 영상 : [영상데이터셋 - 결과시연테스트영상 - 배회1.mp4, 배회2.mp4]

● 결과 확인 동영상 및 결과

1. **배회1.mp4** : [결과시연동영상.mp4 0분53초 ~ 1분29초 부분]
   - ▶ **영상내용** : 정상적으로 걷다가 한자리에 주저앉고 잠시후 다시 걷는 영상.
   - ▶ **결과** : 객체가 주저앉는 순간에 배회(Loitering) 탐지
2. **배회2.mp4** : [결과시연동영상.mp4 1분30초 ~ 2분03초 부분]
   - ▶ **영상내용** : 객체가 술에취한 듯 배회하듯 걷는영상.
   - ▶ **결과** : 객체가 꼬불꼬불 배회하듯 걷지만 배회(Loitering) 탐지를 못함.

배회1.mp4 결과 결과사진	배회2.mp4 결과 결과사진
	

■ 질주 및 이상행

● 테스트 영상 : [영상데이터셋 - 결과시연테스트영상 - 이상행동(급질주).mp4, 이상행동(뛰+배회).mp4]

● 결과 확인 동영상 및 결과

3. **이상행동(급질주).mp4** : **[결과시연동영상.mp4 0분40초 ~ 0분50초 부분]**
   - ▶ **영상내용** : 정상적으로 걷다가 급질주후 반대 방향으로 질주.
   - ▶ **결과** : 질주(Running)은 탐지 못함. 이후 마지막 카메라에 사라지기전에 이상행동(Anomaly) 탐지.
4. **이상행동(뛰+배회).mp4** : **[결과시연동영상.mp4 2분06초 ~ 2분36초 부분]**
   - ▶ **영상내용** : 객체가 임의의차량을 향해 질주, 차량주변을 서성이다가 다시 걸어나옴.
   - ▶ **결과** : 질주와 이상행동 탐지 두상황 모두 탐지 하지못하였으며, 차량 사이에 들어 가는순간부터 객체 인식이 제한됨.

이상행동(급질주).mp4 결과사진	이상행동(뛰+배회).mp4 결과사진
	

■ 폭행

● 테스트 영상 : [영상데이터셋 - 결과시연테스트영상 - 폭력행위.mp4]

● 결과 확인 동영상 및 결과

5. 폭력행위.mp4 : [결과시연동영상.mp4 2분37초 ~ 2분43초 부분]

▶ **영상내용** : 정상적으로 걷다가 사람을 폭행함.

▶ **결과** : 폭행 행위를 이상행동으로 탐지하지못함.

폭력행위.mp4 결과사진

■ 일반행동, 다중객체

● 테스트 영상 : [영상데이터셋 - 결과시연테스트영상 - 걷기1.mp4, 걷기2.mp4, 걷기3.mp4, 걷기4.mp4]

● 결과 확인 동영상 및 결과

6. 걷기1~4.mp4 : [결과시연동영상.mp4 0분0초 ~ 0분39초 부분]

▶ **영상내용** : 다중객체 혹은 단일객체가 정상적으로 걷는영상.

▶ **결과** : 일반적으로 걷는상황을 이상행동으로 탐지하지않음. 객체가 두명있는상황에 객체인식 및 이상행동판단을 진행함.

걷기1~4.mp4 결과사진

■ **결과분석**

최종 완성된 모델의 성능을 검증하기 위해, '정상적인 보행' 시나리오 영상 14개를 추가로 학습한 후 테스트를 진행했습니다. 그 결과 두가지 주요사항을 확인할수있었습니다.

첫 번째, 오탐지율 개선 부분입니다. 최종 모델 이전의 모델들에서 문제가 되었던 오탐지율에 눈에 띄게 개선되었습니다. 실제 테스트결과 영상 9개 항목에서는 오탐지율 0%를 기록하며, 모델이 정상적인 상황을 안정적으로 판단함을 확인했습니다.

두 번째, 미탐지 문제 발생입니다. 특정 이상행동을 탐지하는 부분에서는 한계를 확인할수있었습니다. 자체적으로 정의한 "배회"와 "질주" 탐지 기능, 그리고 별도로 테스트한 "폭행" 상황에대한 이상행동탐지가 정상적으로 작동하지 않았습니다. 또한 일부 상황에서는 객체 탐지 자체의 불안정성도 관찰되었습니다. 테스트 과정에서 발생한 미탐지 문제들을 분석한결과는 아래와같습니다.

● **"배회" 및 "질주" 탐지 실패 원인**

핵심적인 실패원인을 설명하자면 해당 두가지 기능은 LSTM AutoEncoder가 아닌, 사전에 정의된 규칙에 기반하여 작동하기 때문이며 상세한 두가지 문제가있습니다.

▶ **고정 임계값(Threshold)의 한계** : 코드에 설정된 '평균속도', '이동거리' 등의 기준값은 카메라각도, 촬영 거리등에 매우 민감합니다. 예를 들어, 멀리서 뛰는 사람의 이동속도는 가까이서 걷는 사람보다 낮을 수있어, 고정된 값으로는 다양한 상황에 대처하기 어려웠습니다.

● **"폭행" 탐지 실패원인**

'폭행'은 저희가 규칙적으로 정의하지 않았으므로, LSTM AutoEncoder가 'Anomaly'로 판단해야하는 영역이었습니다. 탐지 실패원인은 다음과같이 분석했습니다.

▶ **단순한 특징 벡터의 한계** : 저희 모델은 객체의 Bounding Box 위치, 크기, 속도를 특징으로 사용합니다. 이는 객체의 이동속도, 이동방향 등은 설명할수있지만 팔이나 다리의 움직임 같은 '폭행'의 핵심적인 의미를 담기에는 정보가 너무단순합니다. 특징 추출의 제한성은 프로젝트 진행 중 인지하였으나, 모델의 검증과 테스트를 우선시하는 과정에서 이를 충분히 개선하지 못한 점이 주요한 한계로 작용했습니다.

● **특정 상황에서의 "객체탐지" 실패원인**

YOLOv8 모델 자체의 성능 문제라기보다는, 사람이 다른 사람이나 사물에 의해 가려지거나 어두운 환경에서 탐지율이 급격히 저하되는 현상이 주요 원인으로 분석되었습니다. 이러한 문제를 개선하기 위한 방안으로는 데이터 증강 기법을 적용하거나, MediaPipe 또는 OpenPose와 같은 관절 추정 모델을 YOLOv8과 결합하여 사용하는 방법이 있습니다.

실제로 프로젝트 진행 중 김제민 조원의 모델 제안으로 MediaPipe를 알게 되었고, YOLOv8과 MediaPipe를 결합하여 테스트한 결과도 존재합니다. 그러나 두 모델을 결합하여 정보를 통합·관리하는 과정에서 시스템의 복잡도가 급격히 증가하였고, 이로 인해 최종 구현 단계에서는 YOLOv8 단독으로 객체 탐지만 수행하였습니다.

실제 YOLOv8 + MediaPipe 결합 실행결과

MediaPipe 모델을 활용해서 객체의 관절 데이터들을 활용할수있었지만
시스템의 복잡도, 프로젝트의 난이도가 급격히 증가하여 결합사용을 고려만 하고있었음.

5. 종합 결과 및 고찰

■ 프로젝트 요약

본 프로젝트는 주차장 내 CCTV 영상을 실시간으로 분석하여 잠재적 위험 상황을 사전에 감지하고 대응하는 '주차장 이상행동 감지 시스템' 구축을 목표로 진행되었습니다. 이를 위해 최신 객체 탐지 모델 YOLOv8과 시계열 데이터 처리에 특화된 LSTM AutoEncoder를 활용했습니다. 먼저, YOLOv8로 객체의 움직임을 포착하고, LSTM AutoEncoder가 '정상 행동' 패턴을 학습하는 파이프라인을 구축하였습니다.
성능 평가 결과, 정상 행동을 이상 행동으로 오탐하는 비율은 0%에 근접하여, 모델이 정상 상태를 안정적으로 식별함을 확인했습니다. 하지만 학습된 정상 패턴과 편차가 매우 큰 일부 행동 외에, 실제 위협이 되는 명확한 이상행동(예: 폭행 등)은 탐지하지 못하는 한계를 보이며 프로젝트가 마무리되었습니다.

■ 한계점 및 개선방안

● 기술적 한계 및 개선방안

앞서 말한 본 프로젝트의 가장 명확한 기술적 한계는 이상행동에 대한 낮은 탐지성능입니다. 이러한 해당 한계점을 극복하기 위한 개선방안은 YOLOv8 과 특정 행동 인식 모델을 하이브리드 방식의 형태로 도입하는 것을 고려할수있었습니다. 또한, 기존의 부족한 행동특징들을 풍부하게 추출할 수 있는 Mediapipe와 같은 자세 추정 모델을 결합하여 사용할 수 있었습니다.

● 프로젝트 수행 과정의 한계점 분석

기술적인 한계점 외에도 프로젝트 수행 과정 전반에서 다음과 같은 한계점들이 발견되었습니다. 이는 향후 유사한 프로젝트를 진행할 때 반드시 개선되어야 할 부분입니다.

1. 명확한 목표정의와 구체성 부재

프로젝트 초기, '주차장에서의 이상행동'이라는 목표가 다소 주관적이고 추상적이었습니다. 탐지할 이상행동(예: 폭행, 배회, 기물 파손, 넘어짐)을 명확하게 정의하고 시작하지 못해 프로젝트의 방향성이 흔들리는 주요 원인이 되었습니다. 또한, 성능 평가를 위한 명확한 수치적 기준(Evaluation Metric)이 부재하여 테스트 및 검증 과정이 체계적으로 이루어지지 못하는 결과를 초래했습니다.

2. AI 프로젝트에대한 이해도 부족

AI 프로젝트의 성공에 있어 양질의 대규모 데이터셋 확보가 최우선 과제임을 뒤늦게 인지했습니다. 모델 구조 설계 및 파이프라인 구축, 기능코드 작성에 집중한 나머지, 최종 학습 및 검증 단계에서 데이터의 양적·질적 부족이라는 본질적인 문제에 직면했습니다. 이는 AI 프로젝트의 전체 파이프라인에 대한 경험 부족에서 기인한 것으로, 최종 모델의 완성도를 떨어뜨리는 결정적 요인이 되었습니다.

3. 체계적인 프로젝트 계획의 부재

프로젝트의 계획성 또한 부족했습니다. 이는 AI 프로젝트에 대한 경험 부족에서 비롯된 문제로 볼 수 있습니다. 결과보고서에 포함된 개념 설계, 상세 설계, 테스트 계획 등의 항목은 실제 프로젝트 진행 과정에서는 순차적으로 이루어지지 않았으며, 해당 문서들을 사전에 직접 작성한 적도 없었습니다. 실제로 프로젝트는 체계적인 설계 기반보다는 단순한 '의식의 흐름'에 따라 진행되었습니다. 만약 프로젝트 초기에 이러한 설계 문서들이 구체적으로 작성되어 있었다면, 진행 중 방향성이 흔들릴 때마다 프로젝트의 목표와 구조를 재확인하며 보다 일관성 있게 수행할 수 있었을 것입니다.
그러나 실상은 중간중간 프로젝트의 주제가 변경되거나 사용할 모델이 명확히 결정되지 못해 많은 시간이 소요되었고, 그 결과 전체적인 완성도에 부정적인 영향을 미쳤습니다.

■ 고찰 및 느낀점

학부 과정에서 처음으로 수행한 AI 프로젝트였던 만큼, 많은 시행착오를 겪었습니다. 결과적으로 프로젝트는 '부분 완성'이라기에도 부족한 수준에서 마무리되어, 실패로 평가될 수 있습니다. 그러나 이러한 경험은 값진 학습의 과정이었다고 생각합니다.

특히 프로젝트 막바지에 이르러서야 양질의 데이터셋 확보의 중요성과 전체적인 AI 파이프라인의 구조를 온전히 이해하게 되었지만, 그 시점에 프로젝트를 마무리해야 했던 점은 큰 아쉬움으로 남습니다.

그럼에도 불구하고, 본 프로젝트를 통해 AI 및 딥러닝의 이론을 실전적으로 적용해보는 기술적 성취 외에도 다양한 역량을 얻을 수 있었습니다. 실체가 있는 프로젝트를 직접 수행하고, 결과보고서를 작성하며 전 과정을 되짚는 과정 속에서 문제의 원인을 객관적으로 분석하고 해결 방안을 모색하는 능력이 향상되었습니다. 또한 세 차례의 발표 경험을 통해 기술적 내용을 명확히 설명하고, 프로젝트의 성과와 한계를 청중에게 효과적으로 전달하는 커뮤니케이션 방법과 책임감 있는 자세를 기를 수 있었습니다.

6. 사용 매뉴얼

본 프로젝트의 사용 매뉴얼은 개발자용 매뉴얼과 사용자용 매뉴얼로 구분되어 있습니다.

또한, 해당 프로그램은 실시간 이상행동 탐지를 위해 YOLO 및 LSTM 기반 딥러닝 모델의 연산이 포함되어 있어, 고성능 GPU 환경에서의 실행을 권장합니다.

따라서 원활한 영상 재생 및 탐지를 위해 필요한 권장 PC 사양은 다음과 같습니다.

항목	권장 사양
운영체제	Windows 10 (64bit) 이상
CPU	Intel i7 이상 또는 동급 AMD Ryzen
RAM	16GB 이상
GPU(그래픽)	NVIDIA RTX 3060 이상(CUDA지원)
저장장치	SSD (프로그램 실행속도 확보용)

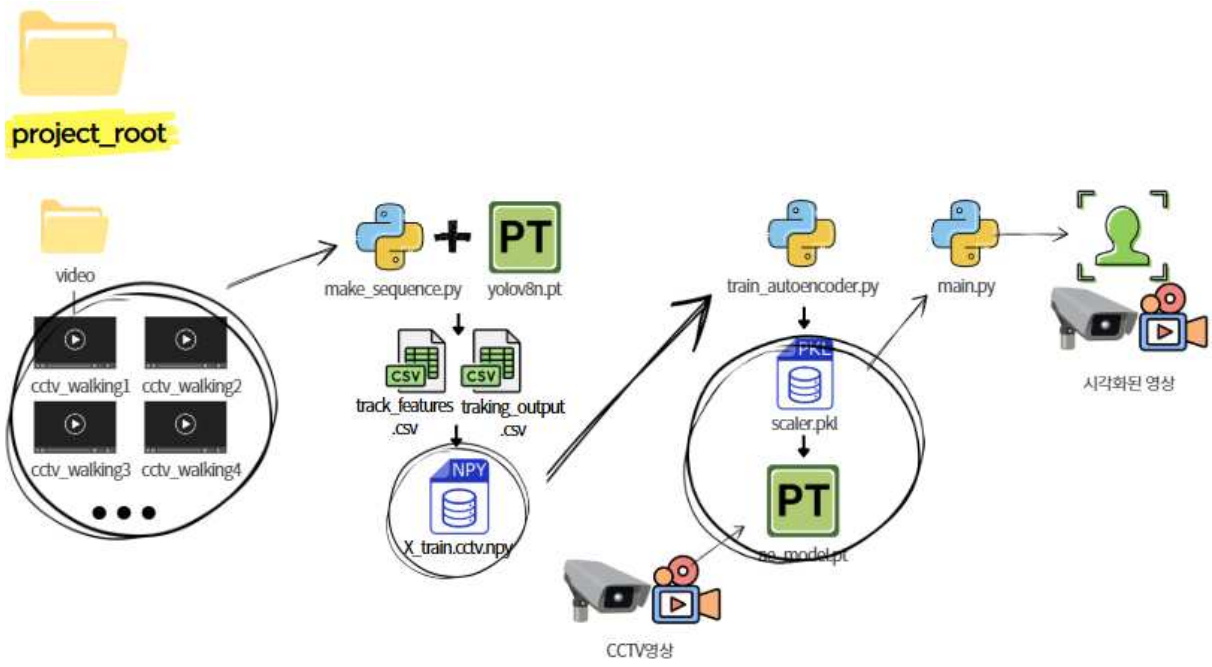
※ 위 사양은 원활한 실시간 추론과 영상 재생을 위한 권장 사양일 뿐이며, 해당 사양을 충족하지 않더라도 실제 실행 환경에서는 프로그램이 정상적으로 작동함을 확인하였습니다.

■ 개발자 매뉴얼

● 디렉토리 구조 및 흐름

결과보고서에 포함되어있는 project_root 디렉토리를 사용합니다. 해당 디렉토리의 구조는 다음과같습니다.

project_root 구조 및 흐름



● **라이브러리 설치**

해당 프로젝트는 torch, opencv-python, numpy, pandas, ultralytics, scikit-learn, joblib, deep_sort_realtime, PyQt5 등의 라이브러리를 기반으로 구현되었습니다. 따라서 프로젝트 실행 전 아래의 명령어를 통해 필수 라이브러리를 설치해야 합니다.

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118 && \

pip install opencv-python PyQt5 numpy pandas scikit-learn joblib matplotlib ultralytics deep_sort_realtime
```

또한 본 프로젝트는 Python 3.9 이상 버전에서 개발되었으므로, 프로젝트 실행 전 해당 버전 이상의 Python이 설치되어 있어야 합니다.

● **사용 방법 및 흐름**

1. 학습 영상 준비

video 디렉토리에 학습할 영상 파일(.mp4등)을 넣습니다.

2. 특징 추출 및 시퀀스생성

project_root 디렉토리에서 다음 명령을 실행합니다.

python make_sequence.py

실행후 root 디렉토리에 .csv 파일 및 .npy 파일이 정상생성 되었는지 확인합니다.

3. 모델학습

project_root 디렉토리에서 다음 명령을 실행합니다.

python train_autoencoder.py

실행후 root 디렉토리에 ae_model.pt 파일이 정상생성 되었는지 확인합니다.

4. 실시간 이상행동 감지실행

project_root 디렉토리에서 다음 명령을 실행합니다.

python main.py

GUI 프로그램이 실행되며 “영상선택” -> “재생” 버튼 클릭 후 탐지결과가 출력됩니다.

■ **사용자 매뉴얼**

사용자의 편의성을위해 pyinstaller을 활용해 .exe 배포본으로 배포했으며 main.exe 실행시 GUI프로그램이 실행되며 “영상선택” -> “재생” 버튼 클릭시 탐지결과가 출력됩니다.

■ **문제사항 발생시**

개발자 매뉴얼 실행중 오류 발생시 아래 항목들을 순차적으로 확인하십시오.

1. **입력 영상 경로확인** : 시퀀스 데이터 생성시 영상 경로는 project_root/video 디렉토리안의 영상의 시퀀스데이터만 추출됩니다.
2. **시퀀스 데이터 파일 위치** : 모델학습시 시퀀스데이터 경로는 project_root/X_train.npy 로 root디렉토리에 위치해야합니다.
3. **필수 라이브러리 설치 여부 확인** : 대부분의 실행오류는 라이브러리 미설치 또는 버전 불일치로 발생합니다. 아래 명령어를 입력하여 설치된 라이브러리 버전을 확인하십시오.

pip list

항목	버전
Python	3.9.13
PyTorch	2.6.0
Ultralytics YOLOv8	8.3.109
OpenCV	4.11.0.86
NumPy	1.26.4
Pandas	2.2.3
joblib	1.4.2
PyQt5	5.15.11
scikit-learn	1.6.1
torch.nn	(PyTorch 내부)

7. 프로젝트 공헌도

본 프로젝트는 팀원 4인이 각자의 역할을 바탕으로 진행 되었으며 아래와 같이 기여도를 분류하였습니다.

■ 공통 기여

팀원 전원은 주차장 CCTV 기반 영상 촬영, 주제 선정 토론, 데이터셋 확보를 위한 영상탐색 등 프로젝트 초기 기획 및 데이터 수집단계에 공동으로 참여함.

■ 1순위 : 곽민규

- **전체 파이프라인 설계 및 모든 핵심 기능 코드 구현 : YOLO 기반 객체탐지, 객체로그 추출, 객체 행동 특징 추출, 데이터 시퀀스화, AutoEncoder 기반 모델 학습 및 추론**
- **프로젝트 제안발표, 중간 발표, 최종 발표 ppt 제작**

■ 2순위 : 김준협

- **발표자 역할 수행 : 제안 발표, 중간 발표, 최종 발표**
- **GUI 구현 코드작성**
- **프로젝트 공정표 작성 및 제안서 작성 일부지원**

■ 3순위 : 김제민

- **다양한 모델 활용에 대한 기술적 아이디어제시 : MediaPipe, OpenPose, 제로샷 기법**
- **연구 노트 종합 주도, 제안서 작성 주도**

■ 4순위 : 김형모

- **프로젝트 초기 단계에서 기본 시스템 흐름도 의견제시**
- **프로그램 성능 테스트를 위한 고성능 GPU 컴퓨터제공**

8. 참조문헌

[1]Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification (한글 번역: 실시간 알약 인식에 대한 RetinaNet, SSD, YOLO v3 비교 연구)https://doi.org/10.1186/s12911-021-01691-8

[2]A Comparative Study on Object Detection Models: YOLOv3, SSD and Faster R-CNN(한글 번역: YOLOv3, SSD, Faster R-CNN 객체 탐지 모델 비교 연구)https://www.researchgate.net/publication/340138261_A_Comparative_Study_on_Object_Detection_Models_YOLOv3_SSD_and_Faster_R-CNN