쓰러진 나 대신!!

신고해'조'

2024.06.20

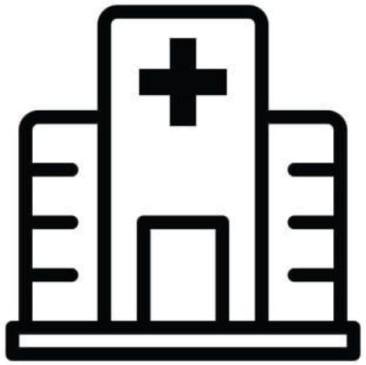
목차

01. 팀원 소개 및 역할

문영식	- Yolo 아키텍쳐 분석 - Yolo v1~v5 Finetuning 및 성능 테스트 - 알고리즘 작성 및 테스트
문예준	- Yolo v8 Finetuning 및 성능 테스트 - Yolo 아키텍쳐 분석 - 알고리즘 작성 및 테스트
이재우	- Yolo v10 Finetuning 및 성능 테스트 - Yolo pose모델 분석 - 알고리즘 작성 및 테스트
정호석	- 데이터 전처리 - Yolo v7 Finetuning 및 성능 테스트 - Yolo 아키텍쳐 분석 - 알고리즘 작성 및 테스트

02. 프로젝트 주제

객체인식을 통한 쓰러짐 탐지



병원, 요양원 등

독거 노인

등산로, 산책로 등

인적이 드문 곳에서 본인의 의지와 관계 없는 쓰러짐을 자동 탐지하여 신속한 인명 구조 수행

02. 프로젝트 주제

기존 대응 방식의 한계점

인적 자원을
활용한 감시

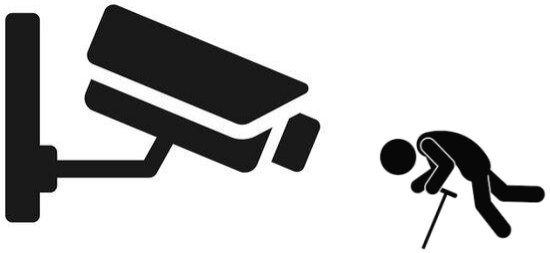
- 인력 부족으로 인한 시간적, 공간적 한계 존재
- 부주의, 자리 비움, 실수 등에 의한 공백 발생 가능성
- 인건비 상승 가능성에 따른 장기 플랜의 불확실성 및 리스크 존재

스마트기기를
활용한 탐지

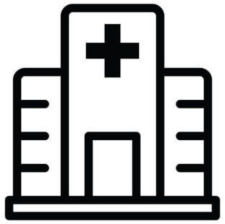
- 고장, 방전 등의 리스크 존재
- 사용자의 능동적인 관리 및 상시 착용의 불확실성 존재
- 감시 대상 인원수의 유동적인 변화에 유연한 대응 어려움

02. 프로젝트 주제

기존 방식의 한계점을 보완할 수 있는 효율적 수단 필요



- 상시 감시가 어려운 장소의 CCTV 영상을 활용한 객체 탐지 활용

		
복도, 인적이 드문 곳에서의 “눕는 행위” 자동 탐지 및 관리자 알람 전송	실내 공간에서 “비자발적 쓰러짐” 자동 탐지 및 119 구조 요청	실외 공간에서 “쓰러진 후 미동 없음” 자동 탐지 및 119 구조 요청
⬇	⬇	⬇
“주저앉거나 눕는 자세”에 대한 객체 탐지 기술 필요 → 특정 자세에 대한 객체 인식 필요	“주저앉거나 눕는 자세, 비자발적 쓰러짐”에 대한 객체 탐지 기술 필요 → 동일 객체에 대한 이동 속도 계산 필요	“주저앉거나 눕는 자세, 쓰러진 후 행동”에 대한 객체 탐지 기술 필요 → 동일 객체에 대한 이동 속도 계산 필요

03. 프로젝트 수행 절차

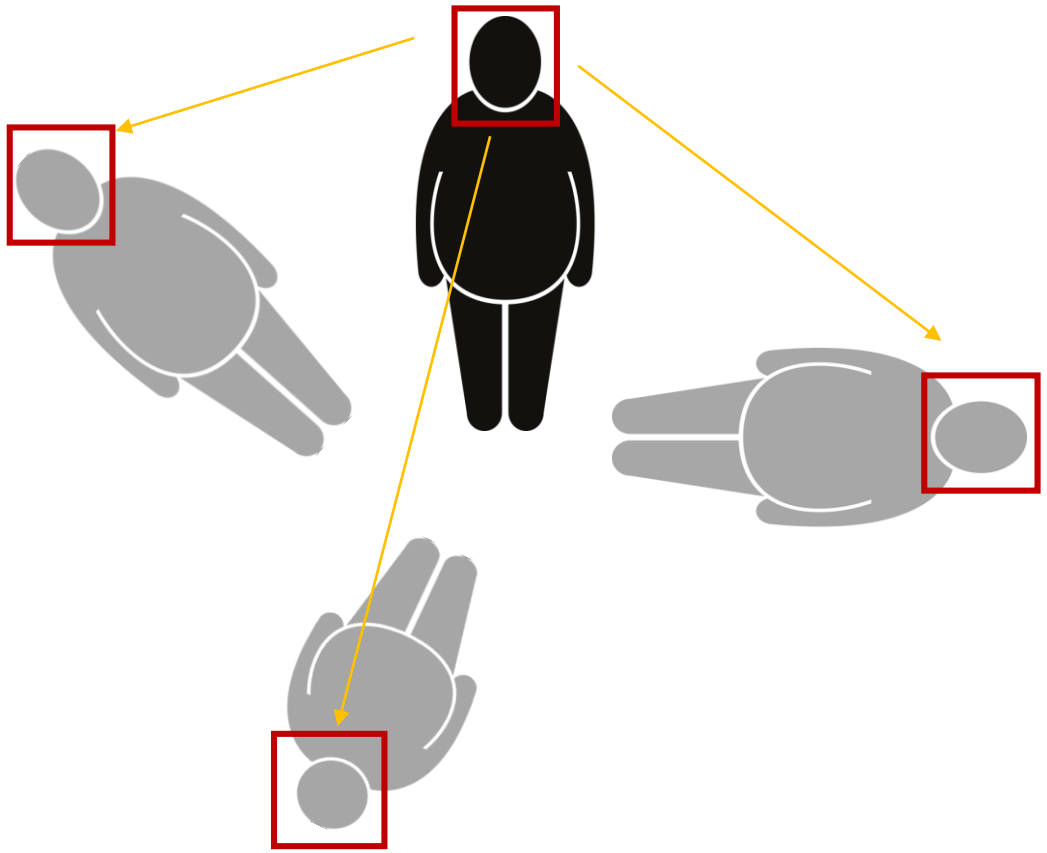
Yolo 객체 탐지모델을 활용한 “쓰러짐 탐지” 서비스 구축	
1. 파인튜닝	• 프로젝트 기간과 컴퓨팅 리소스 한계를 고려해 YOLO SOTA 모델만을 활용 • 버전 별 파인튜닝 및 성능 비교 (v5, v7, v8, v10) • 특정 자세를 감지할 수 있도록 “주저 앉거나 쓰러진 행동” 데이터를 이용한 파인튜닝 • 쓰러지거나 눕는 속도를 계산할 수 있도록 “머리” 데이터를 이용한 파인튜닝 (머리 감지)
2. 모델선정	• 쓰러짐 탐지에 가장 적합한 모델 버전 선정 • 정확성과 인식 속도를 기준으로 선정
3. 알고리즘 작성	• 각 프레임 별 동일 객체를 추적할 수 있는 알고리즘 작성 • 동일 객체에 대한 자세 변화의 속도를 측정할 수 있는 알고리즘 작성
4. 최종 테스트	• 테스트 영상 데이터를 사용한 테스트 진행 • 직접 촬영한 영상 데이터를 사용한 테스트 진행

03. 프로젝트 수행 절차


프로젝트 일정
6/3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
주제선정 및 데이터 수집
데이터검증 및 전처리
파인튜닝 및 성능 비교
알고리즘 작성 및 테스트
PPT 작성

04. 프로젝트 수행

데이터 소개 (머리 인식 파인튜닝)



눕는 자세

→ 머리가 땅에 닿음

→ 머리의 움직임이 가장 많음

→ 좌표값의 변화량이 가장 큼

비자발적 쓰러짐

→ 머리의 이동 속도가 빠를 가능성 高

→ 이동 속도 측정을 통한 탐지 시도

한계점

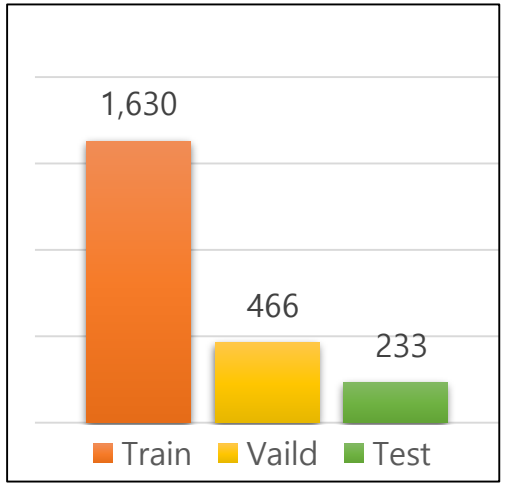
→ 다양한 경우의 수를 모두 충족시키기 어려움

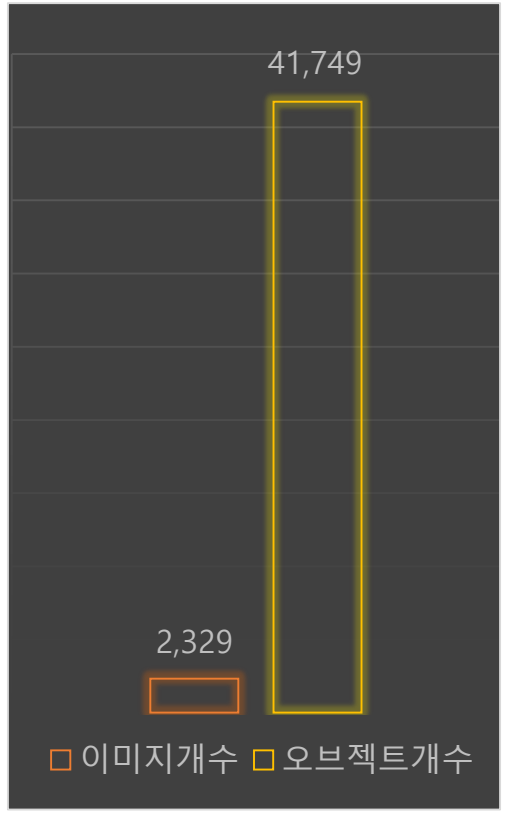
→ 자세(포즈) 탐지와 앙상블 필요

04. 프로젝트 수행

데이터 소개 (머리 인식 파인튜닝)

- 출처 : 로보플로우(Roboflow - head datasets)
- 데이터 : 총 2,329장
- Class : 1개 (head)
- 오브젝트(머리) 개수 : 41,749개
- 이미지 사이즈 : 다양함



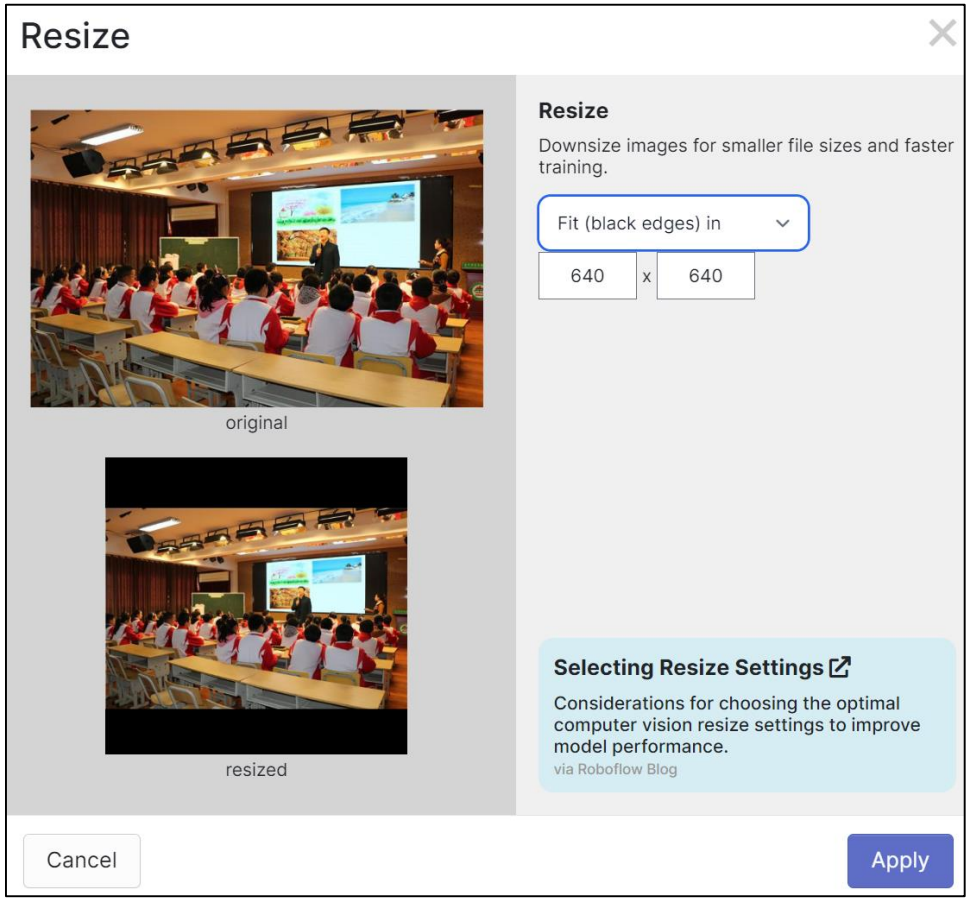


04. 프로젝트 수행

데이터 소개 (머리 인식 파인튜닝)

이미지 크기의 다양함
→ 사이즈 통일 및 패딩 필요
→ 라벨 좌표 변화 고려 必

- 리사이징 및 패딩 (640 x 640) → 로보플로우 전처리 기능 활용

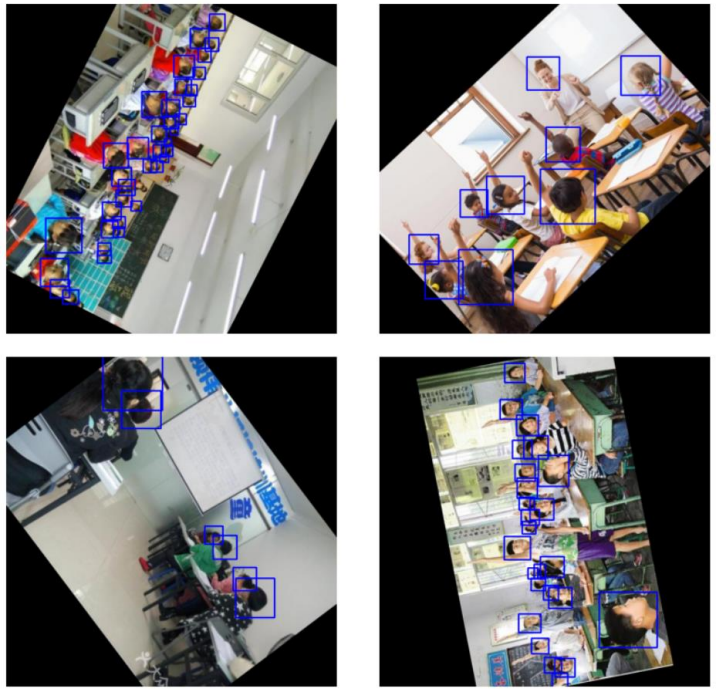


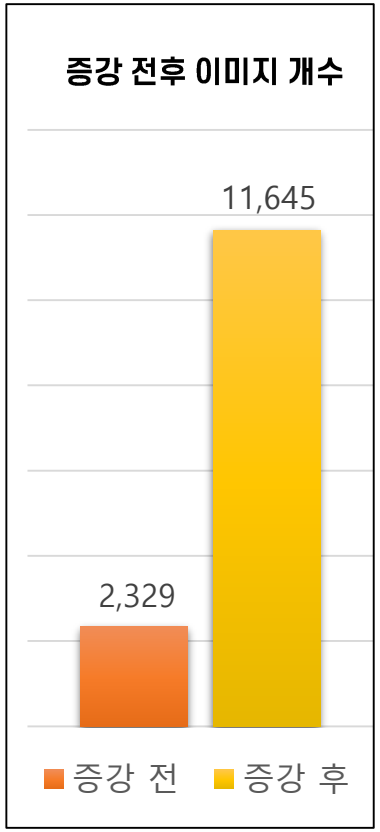
04. 프로젝트 수행

데이터 소개 (머리 인식 파인튜닝)

머리 방향이 대부분 같음
→ 다양한 자세에서 인식 어려움
→ 회전 증강 필요
→ 라벨 좌표 변화 고려 必

- 자체 코드 작성
  → 로보플로우 증강 0~45도, 90도 회전만 지원해 사용 불가
  → 라벨 좌표 변화를 고려한 증강 코드 작성
  → 30~330도 랜덤 회전, 각 사진 당 4장씩 증강

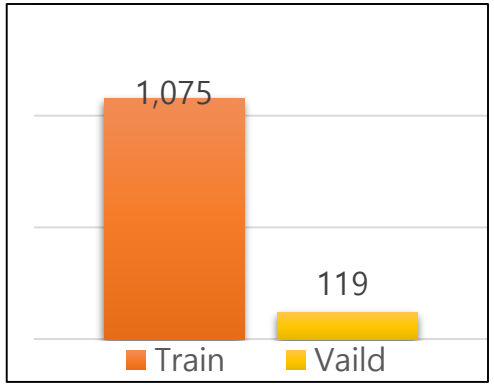




04. 프로젝트 수행

데이터 소개 (누운 자세 인식 파인튜닝)

- 출처 : AI허브(시니어 이상행동)
- 데이터 : 총 1,194장
- Class : 1개 (누운자세)
- 오브젝트(누운사람) 개수 : 총 1,194개
- 이미지 사이즈 : 3,840 x 2,160

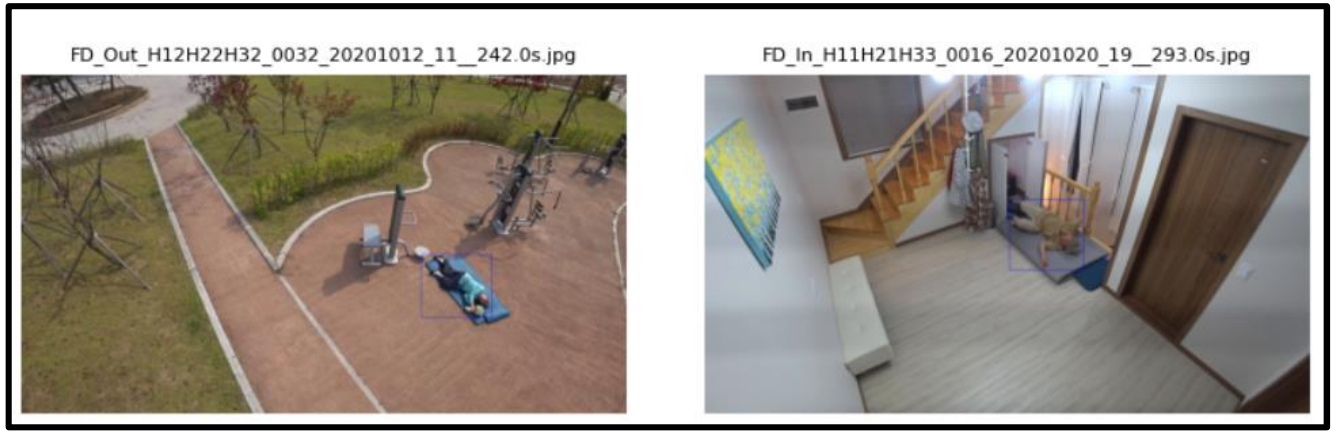


- 전처리 과정에서 좌표값 혼용 문제 발견

→ xmin, ymin, xmax, ymax

→ x_center, y_ceter, width, height

→ 패턴 파악을 통해 다른 전처리 알고리즘 적용

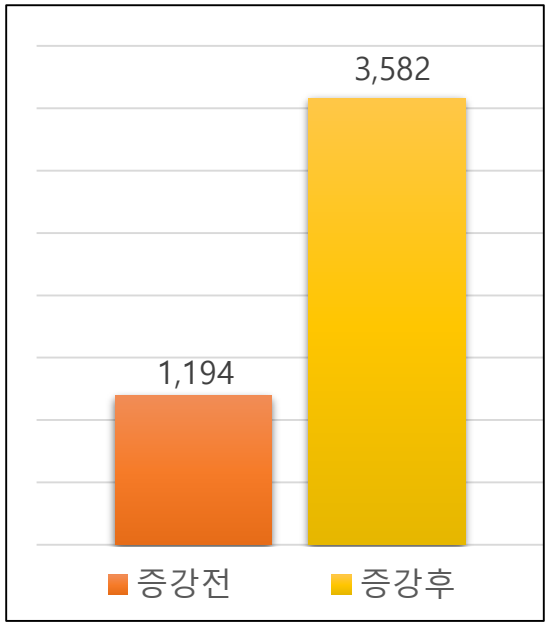


➔ 출처에 관계 없는 철저한 사전 데이터 검증의 중요성 인지

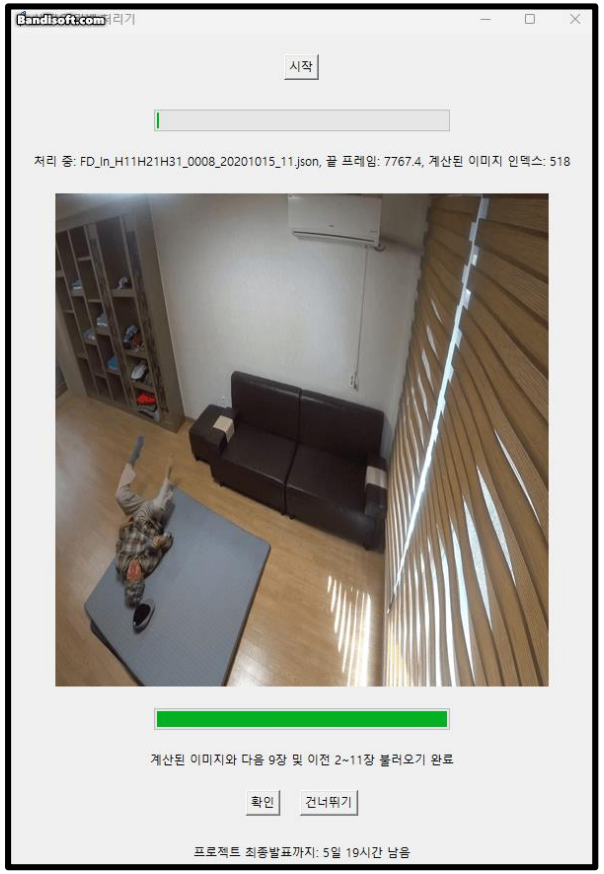
04. 프로젝트 수행

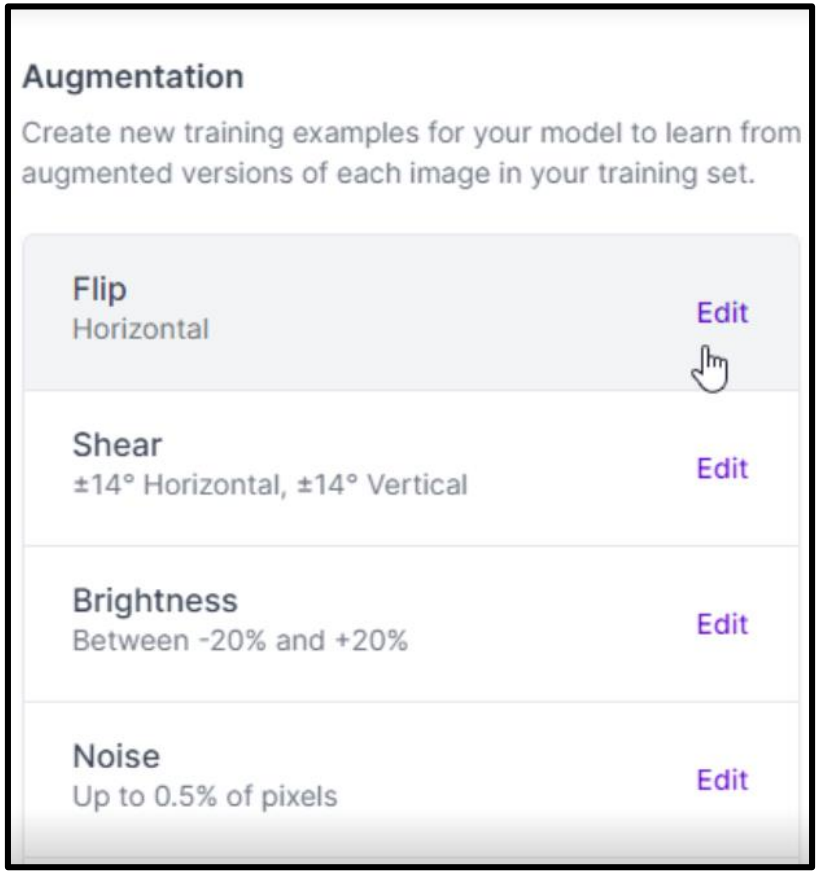
데이터 소개 (누운 자세 인식 파인튜닝)

* 증강(Augmentation)
  - 자세변화 : Horizontal Flip
  - 사진왜곡 : Shear
  - 밝기변화 : Brightness
  - 화질훼손 : Noise



- 데이터 수작업 검증 및 로보플로우를 이용한 4가지 증강 랜덤 적용





04. 프로젝트 수행

YOLO 최종 모델 선정

* Head Data Test set (증강 적용) : 640x640
* 속도 측정 기준 : 동일 로컬 환경 (CPU) 1장 기준

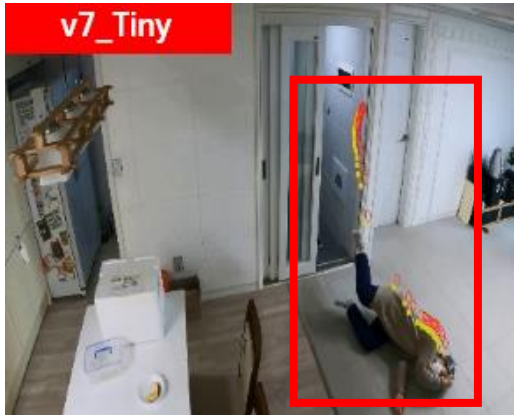
모델	버전	파라미터	속도 (inference, sec)	mAP (VOC)	mAP (COCO)	비고
v5	small	7 million	0.23436	0.938	0.557	* batch size :16, epoch : 30
	medium	21 million	0.55481	0.942	0.573	
v7	Tiny	6 millilon	0.22768	0.936	0.528	* batch size : 64, epoch : 30
	e6	110 million	1.76003	0.890	0.478	* batch size : 32, epoch : 30
v8	Nano	3 million	0.11399	0.932	0.567	* batch size : 64, epoch : 30
V10	Nano	2.7 million	0.15812	0.930	0.559	* batch size : 64, epoch : 30

→ 정확도는 v5-medium 모델이 가장 높으나, inference 속도가 느림

→ 훈련데이터(Train, Valid, Test)와 다른 별도 영상으로 최종 테스트 후 속도와 정확성을 고려한 모델 선정

04. 프로젝트 수행

YOLO 최종 모델 선정



추가 테스트 영상 기준

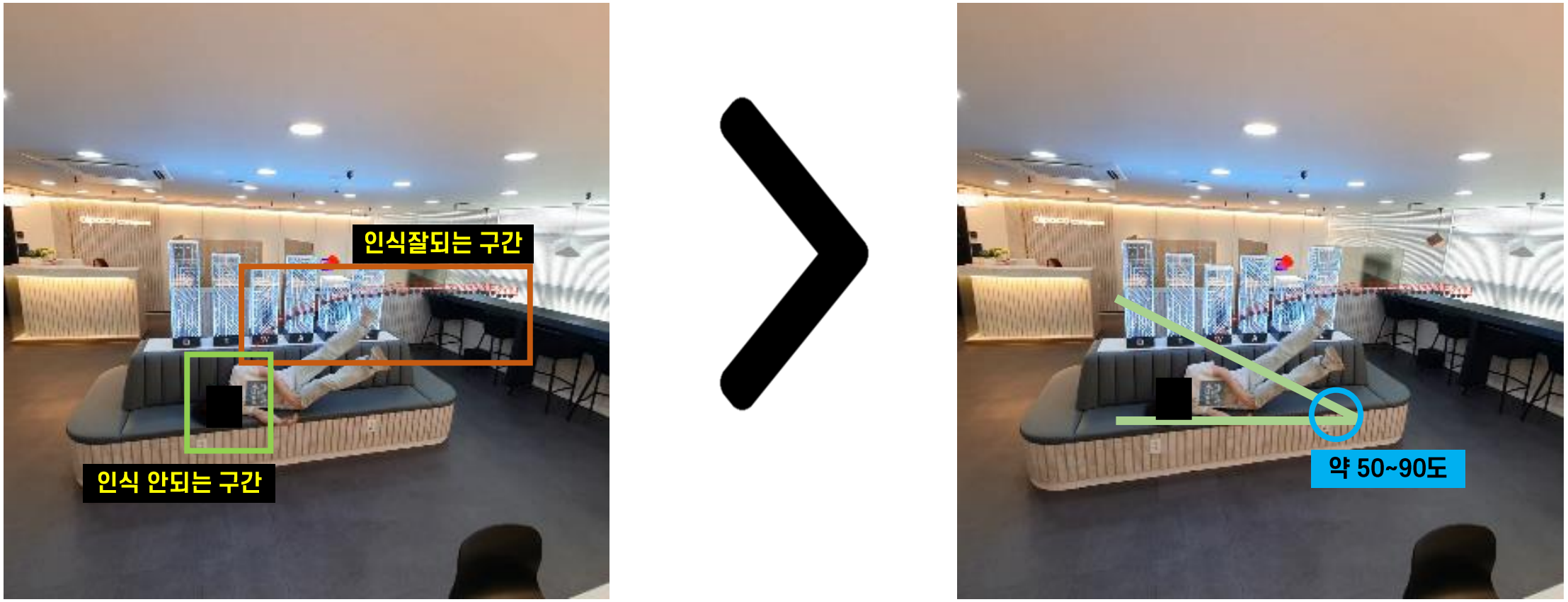
"V7-Tiny" 모델이 가장 적합한 것으로 판단

→ 인식 프레임 개수 최다

→ 인식 속도 高

04. 프로젝트 수행

V7 – Tiny 추가 증강(Augmentation) 적용 ver1



자체 제작 영상 및 테스트 영상에서 특정 각도 인식률이 낮은 문제 발생
→ “50~90”도 범위 랜덤 회전 증강 추가 적용 데이터(+각 3장)로 신규 학습 진행
→ 기존 증강 포함 데이터 (8,150) → 추가 증강 포함 데이터 (13,040)

04. 프로젝트 수행

V7 – Tiny 추가 증강(Augmentation) 적용 ver2

인식률이 소폭 증가했으나 불충분
→ “70~95”도 범위 랜덤 회전 증강 추가 적용 데이터(+각 5장)로 신규 학습 진행
→ 기존 증강 포함 데이터 (13,040) → 추가 증강 포함 데이터 (21,190)

04. 프로젝트 수행

V7 – Tiny 추가 증강(Augmentation) 결과

대규모 증강에도 인식률 변화 없음
→ 학습 데이터의 문제가 아님을 인지
→ 기본 모델 권장 사용법(일명 “딸깍 버전”)의 결과와 비교 시도

```
result = self.model(img)
p_class = 0
p_center, p_xyxy = None, None
```

실시간 알고리즘 적용을 위한 커스텀 버전


```
(alphaco_pj_04_yolo5) E:\02.공부\02.코딩\01.Python\01.Alphaco\02.코드\03.프로젝트\04.객체인식\02.Code\yolov7>py
thon detect.py --weights save_model\yolov7_tiny_final2_head_finetuned_64_30\weights\best.pt --conf 0.25 --img-s
ize 640 --source E:\02.공부\02.코딩\01.Python\01.Alphaco\02.코드\03.프로젝트\04.객체인식\02.Code\test_video\202
40612_173343_1.mp4
```

결과 저장만 가능한 기본 딸깍 버전

같은 “가중치” 정보를 가진 동일 모델의 인식률 차이가 발생
→ YOLO 라이브러리 코드 정밀 분석 진행

04. 프로젝트 수행

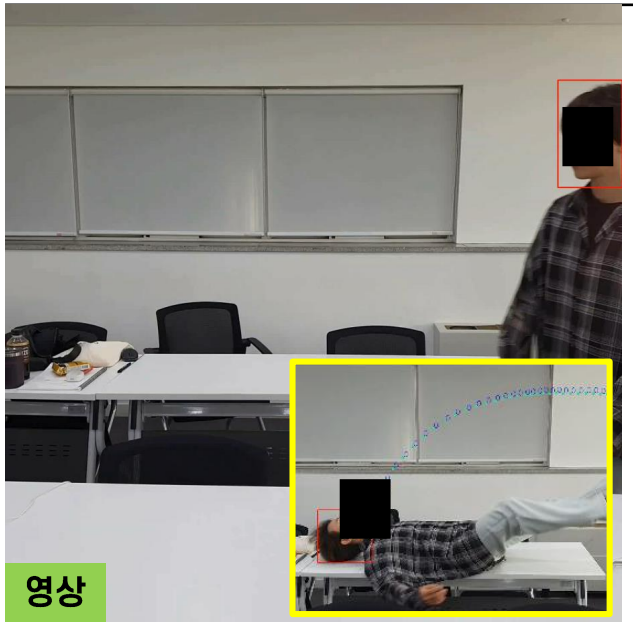
YOLO 인식률 저하 원인 분석

가설1	Confidence, IOU Trheshold 등 임계 설정값 적용의 차이 → NMS는 Confidence가 가장 높은 박스는 남기므로 인식이 되지 않는 문제와는 상관관계 無 → Confidence 수치 자체가 변하는 사실에서 임계값 문제는 아닌 것으로 판단
가설2	가중치 파일(.pt) 로딩 과정에서 아키텍쳐와 가중치 테이블의 불일치 가능성 → pt 파일의 경우 아키텍쳐 자체를 저장하므로 일치하지 않을 가능성 없음 → 만약 일치하지 않는다면 로딩 과정에서 에러가 발생할 것이므로 문제가 아닌 것으로 판단
가설3	모델 통과 전 데이터 전처리의 차이 → 코드 분석 결과 YOLO 버전 별 자체 전처리 시 “RGB”, “BGR” 이미지 채널 순서 기준이 다름을 확인 → open-cv 라이브러리로 영상을 편집하며 “RGB” 형식으로 모든 모델을 테스트한 것이 원인 → 모델 훈련과 다른 형식의 테스트 데이터 인풋 타입이 인식률 저하의 원인으로 확인 → 알고리즘 이해를 바탕으로 한 데이터 전처리 프로세스 파악의 중요성 인식

v5, v7 : 입력이미지 RGB 형식 / v8, v10 : 입력이미지 BGR 형식 (자체 알고리즘)

04. 프로젝트 수행

문제 해결 후 YOLO 최종 모델 선정



* 자체제작 테스트 영상 4개 기준 / Confidence Threshold 0.7 기준 인식된 프레임 개수

버전	v5 medium	v5 small	v7 e6	v7 tiny	v8 nano	v10 nano
falldown_small	112	88	53	74	53	61
falldown_large	117	116	100	114	110	114
sleep_small	147	147	146	147	118	138
sleep_large	161	161	158	160	156	158
합계	537	512	457	495	437	471

* 정확성 기준 : v5 medium

* 속도 기준 : v5 small

→ 열악한 성능 환경을 가정하여
v5 small 버전으로 최종 선정
(Inference 속도 차이 약 2배)

04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성

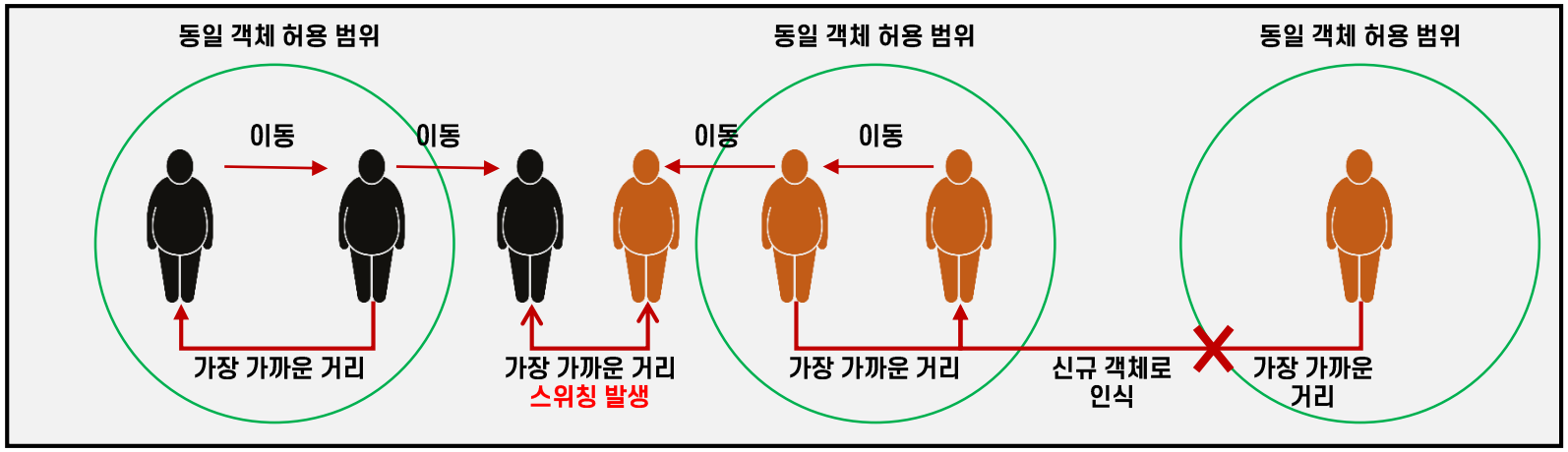
1. 동일 객체 인식 및 좌표 추적

- 중심 좌표가 이전 프레임에서 가장 가까울 경우 동일 객체로 인식

  → 가까운 객체가 범위 기준값 내에 없을 경우 새로운 객체로 판단 (기준값은 원근에 따라 차등 적용)

  → 추적 프레임의 기준 횟수를 정해 메모리 낭비 방지 (기준값 이전 프레임 좌표는 삭제)

  → 기준 횟수 이상 업데이트 되지 않는 객체는 화면에 없는 것으로 간주해 삭제해 메모리 낭비 방지



- 문제점

  → 두 객체의 좌표가 겹칠 때 객체 정보가 혼동되는 현상 발생(스위칭 등)

  → DeepSort 등의 객체 추적 모델 사용으로 완화할 수 있으나 인적이 드문 곳에서 사용할 서비스이기 때문에 큰 문제라고 볼 수 없으므로, 리소스 절약을 위해 적용하지 않음

쓰러짐 구분 알고리즘 작성

한적한 곳에서 동일 객체 인식 → 인식 잘 됨

한적한 곳에서 동일 객체 인식 → 스위칭 등 문제 발생

04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성

2. 머리의 움직임 속도 계산

- 기준값 내 프레임 범위에서 좌표 변화량 측정

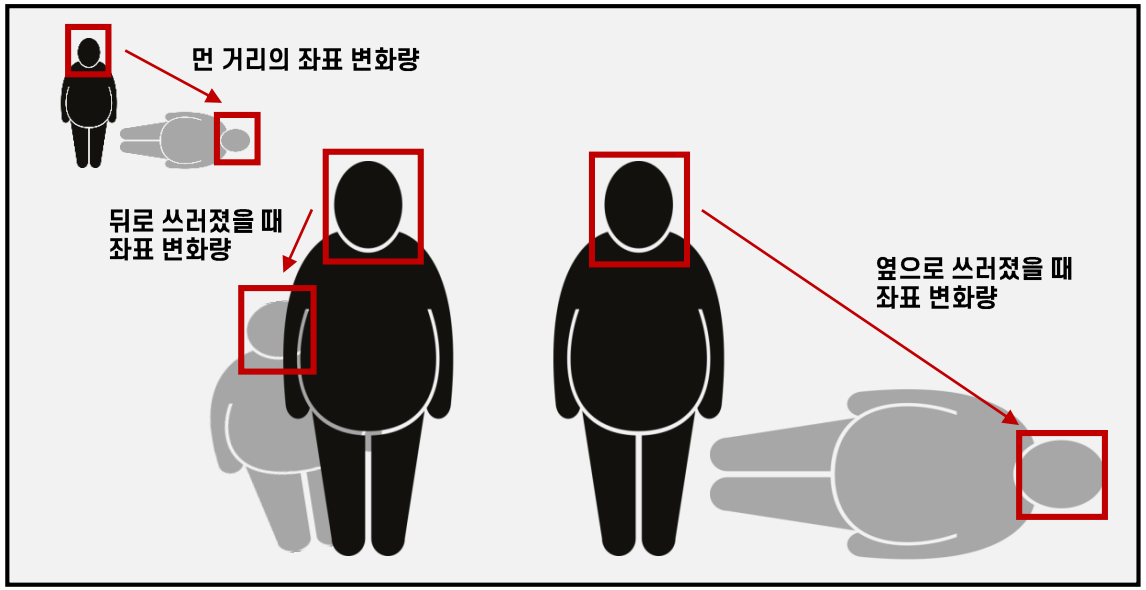
  → 범위 내 최종 이동 거리를 통한 측정 또는 가속도 계산을 통한 측정 시도

- 문제점

  → 원근에 따라 같은 속도임에도 좌표 변화량이 달라질 수 있음

  → 같은 원근에서도 쓰러지는 방향과 카메라 위치에 따라 좌표값 변화량이 다를 수 있음

  → 자발적으로 눕는 행위와 쓰러지는 것을 구분하는 기준값의 모호함



- 해결방안

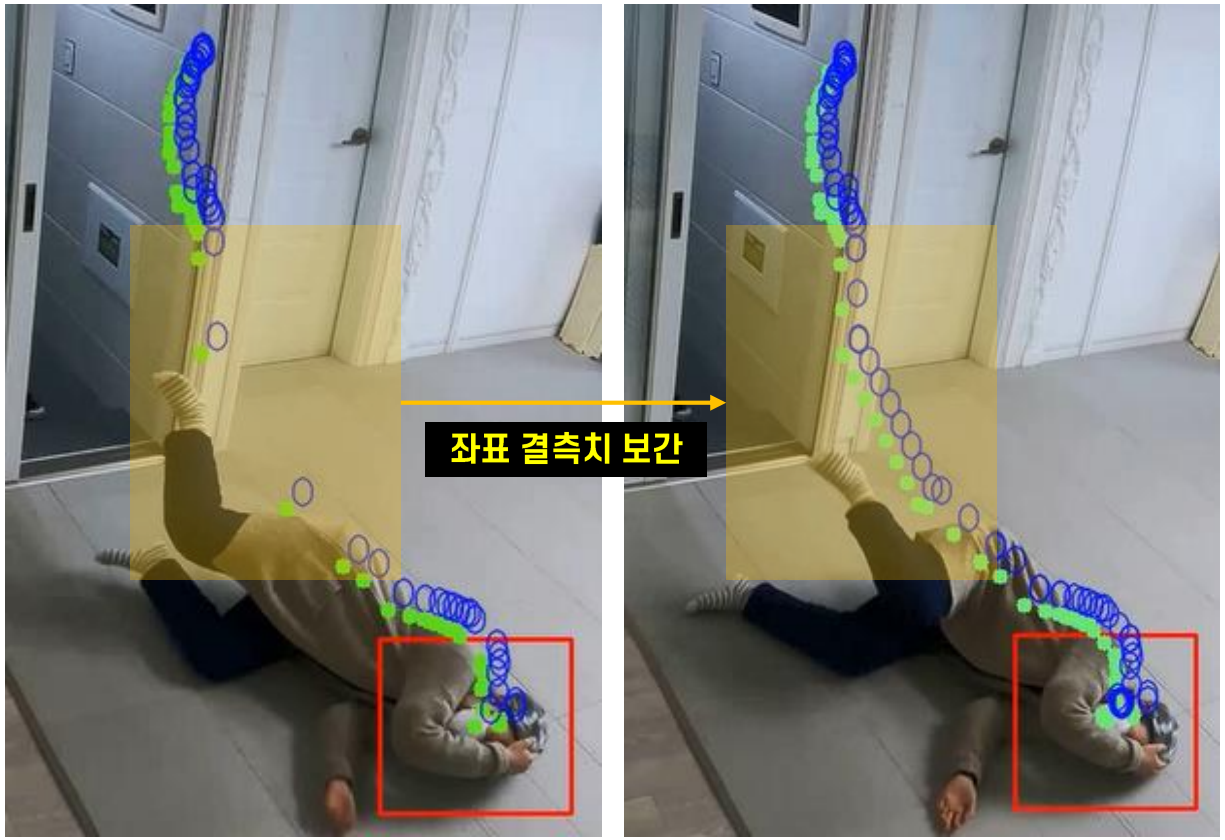
  → 실험을 통한 기준값 설정

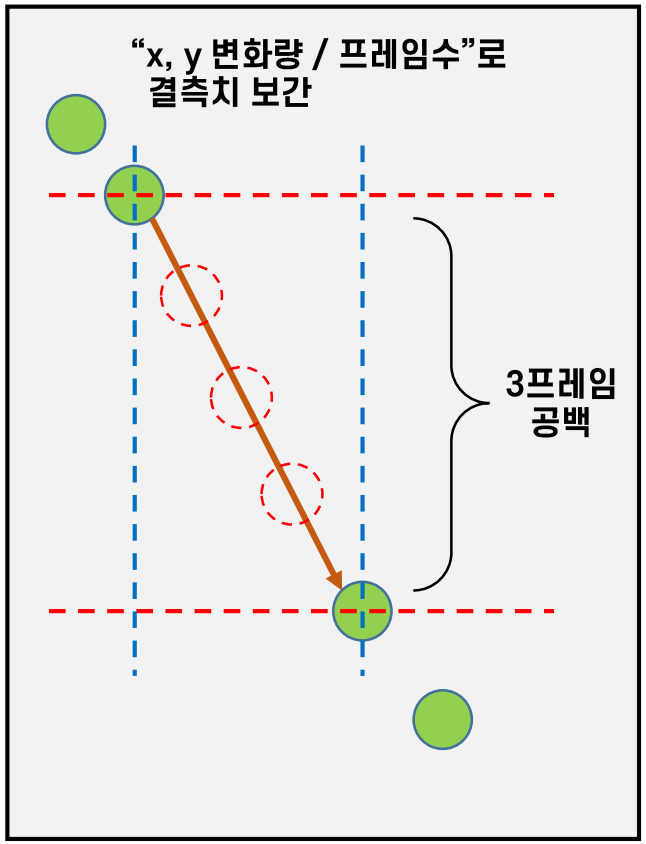
  → 자세 인식 모델과의 앙상블로 다양한 경우의 수에 대응

쓰러짐 구분 알고리즘 작성

2. 머리의 움직임 속도 계산

프레임 간 객체가 인식되지 않아 생긴 공백 프레임의 좌표 처리 (Interpolation)



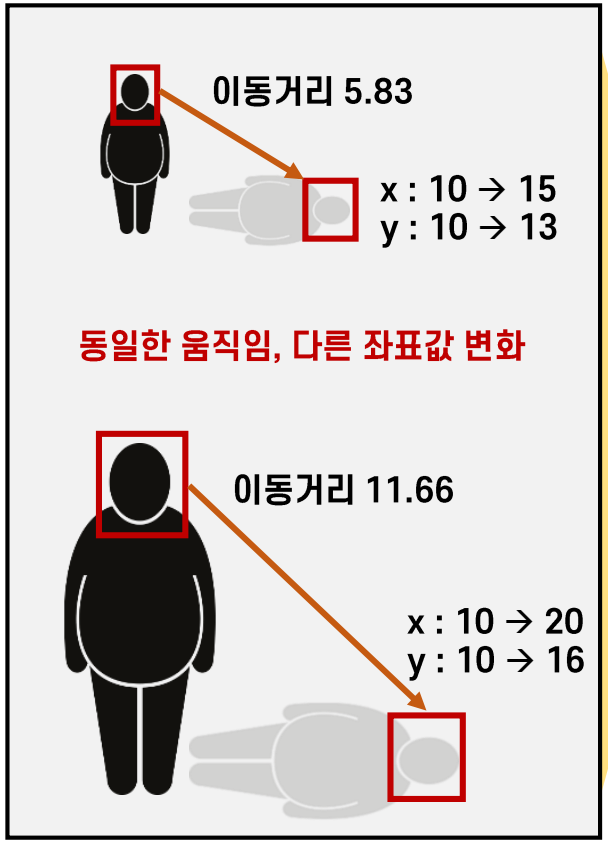


04. 프로젝트 수행

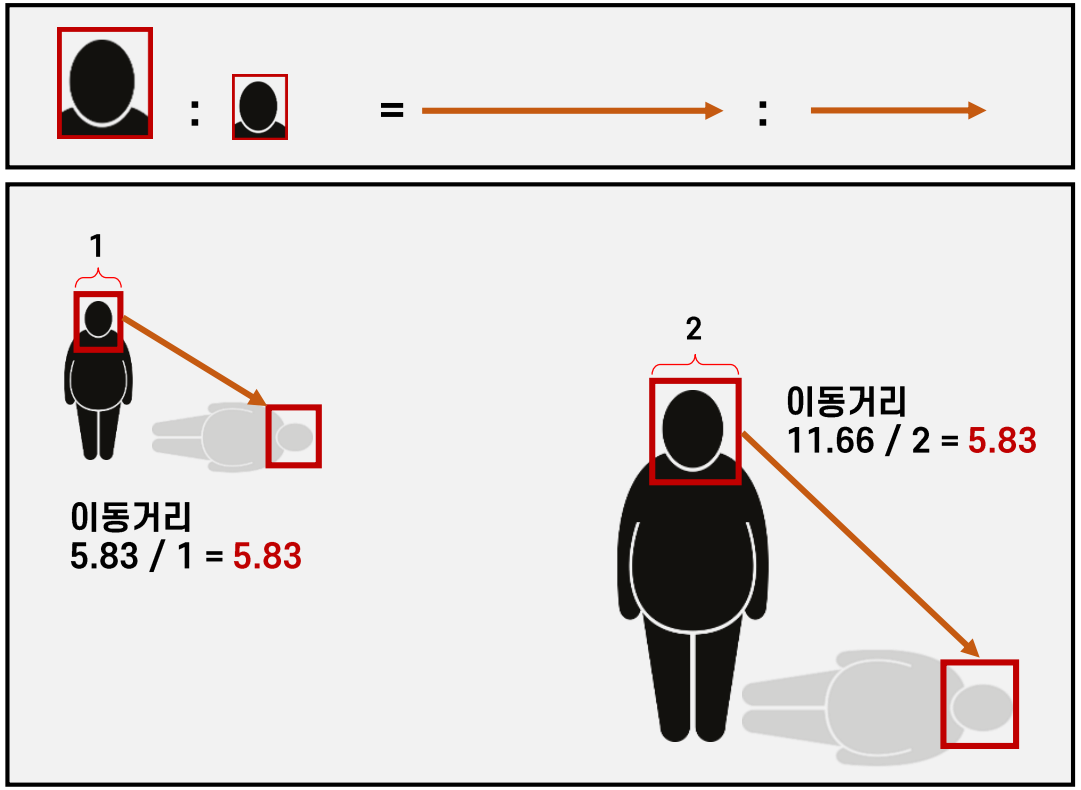
쓰러짐 구분 알고리즘 작성

2. 머리의 움직임 속도 계산

머리 크기 비율을 통한 좌표 이동거리의 정규화



머리 크기(width)로 이동거리를 나눠줌



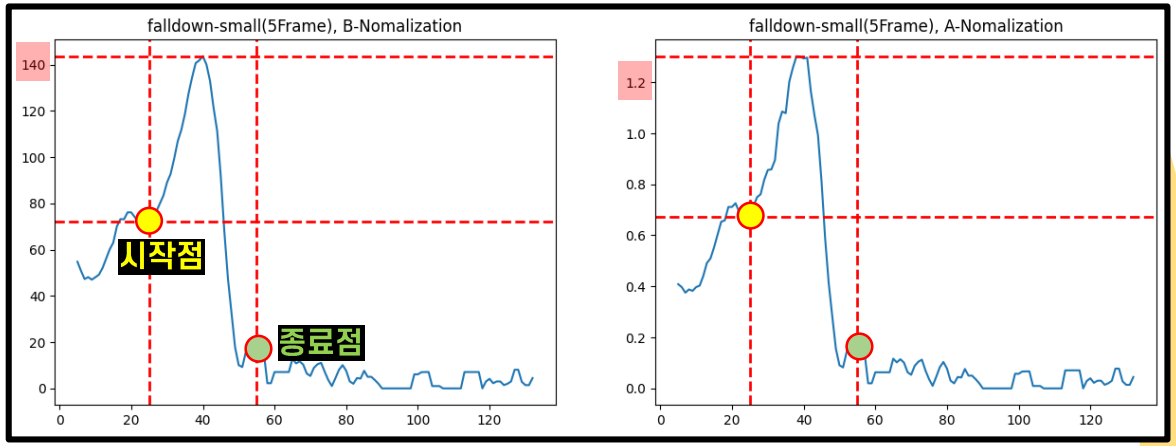
* 각 프레임 간 이동거리마다 머리크기(width)를 나눠서 정규화

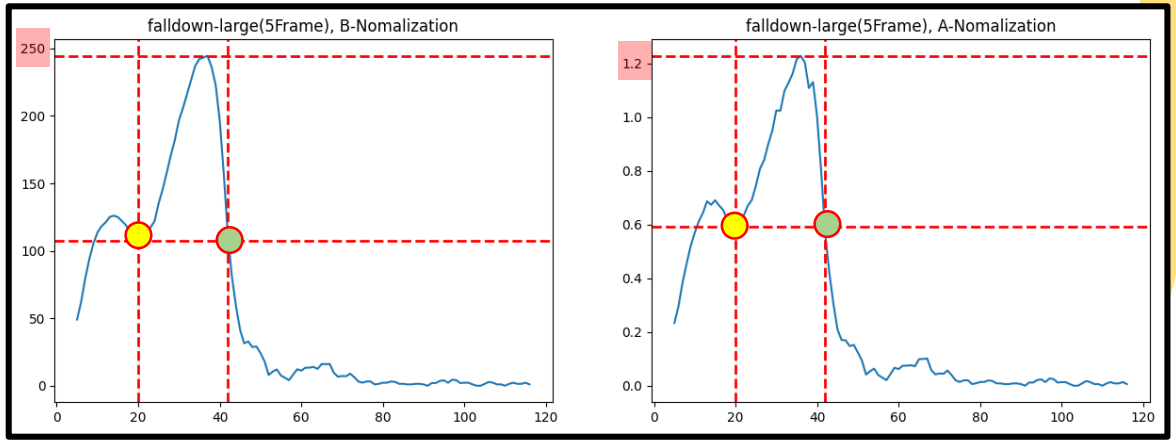
04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성

2. 머리의 움직임 속도 계산

- 쓰러짐 영상 이동 속도 정규화



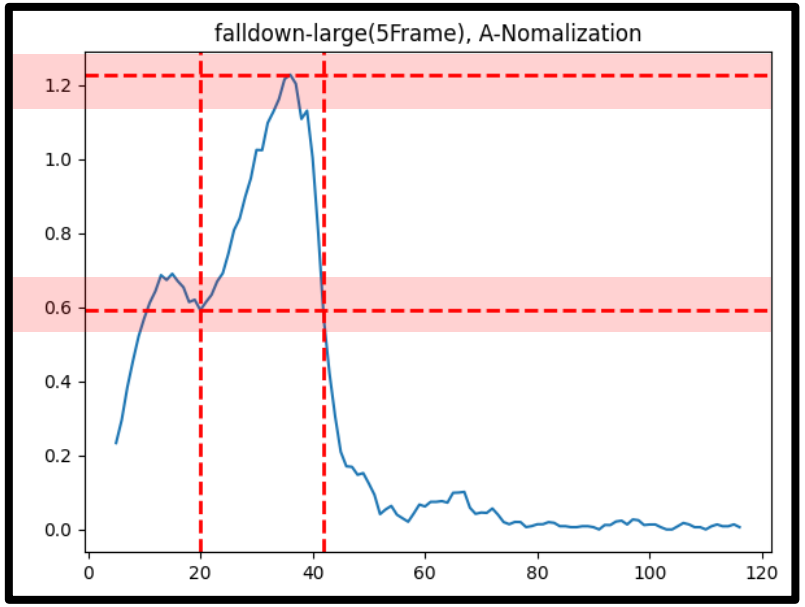
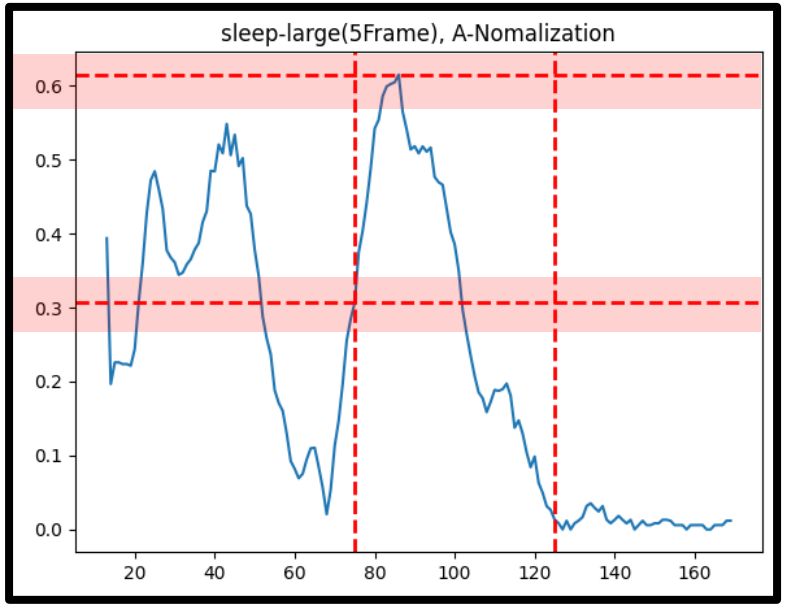


작은 배율 최고속도
- 정규화 전 : 143
- 정규화 후 : 1.3

큰 배율 최고속도
- 정규화 전 : 244
- 정규화 후 : 1.23

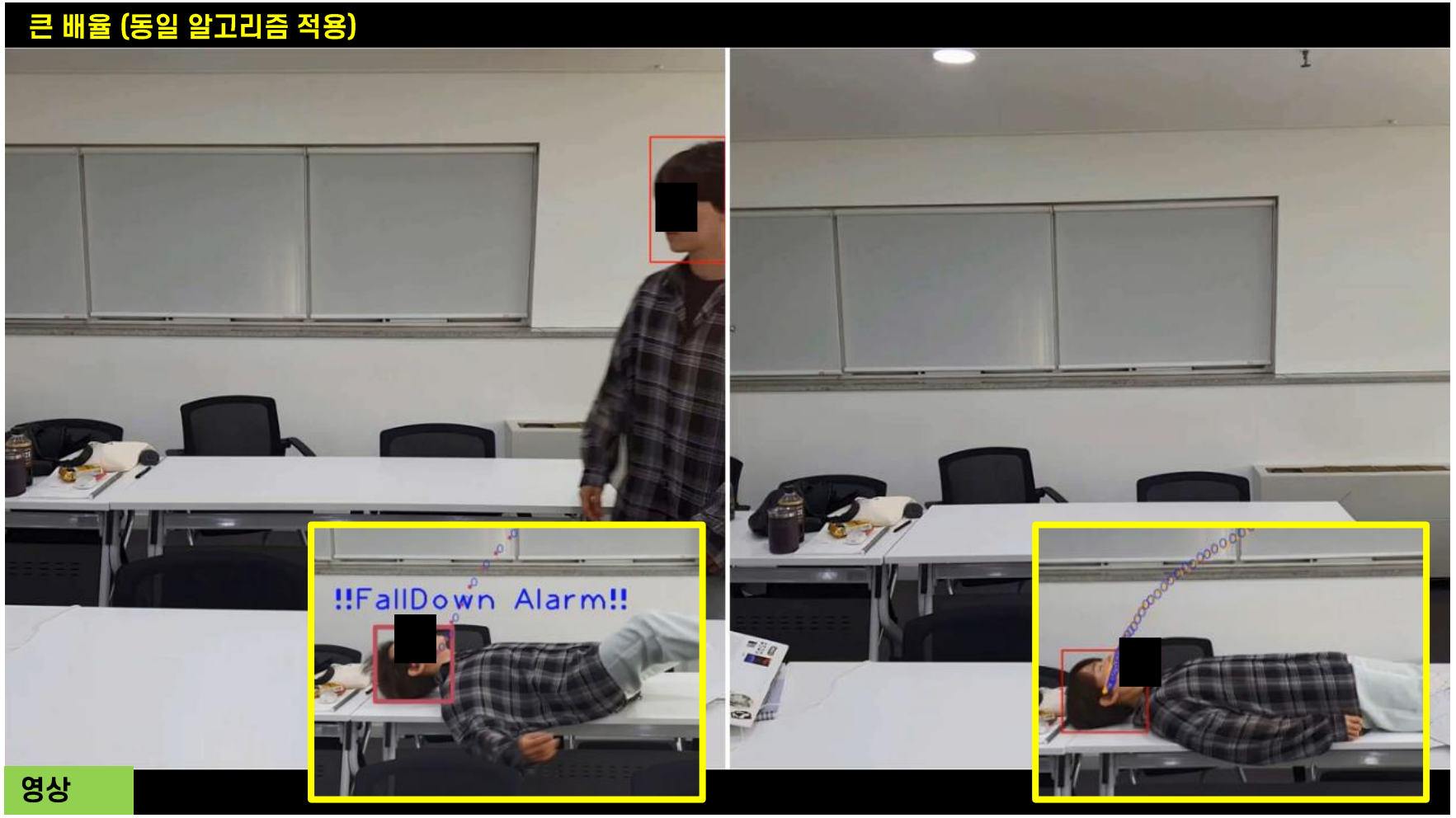
04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성

	쓰러짐 속도 (정규화)	누움 속도 (정규화)
2. 머리의 움직임 속도 계산		
	- 시작속도 : 약 0.6 - 최고속도 : 약 1.2	- 시작속도 : 약 0.3 - 최고 속도 : 약 0.6
	쓰러짐 구분 속도 임계점(Threshold) → 1.1 설정	

04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성



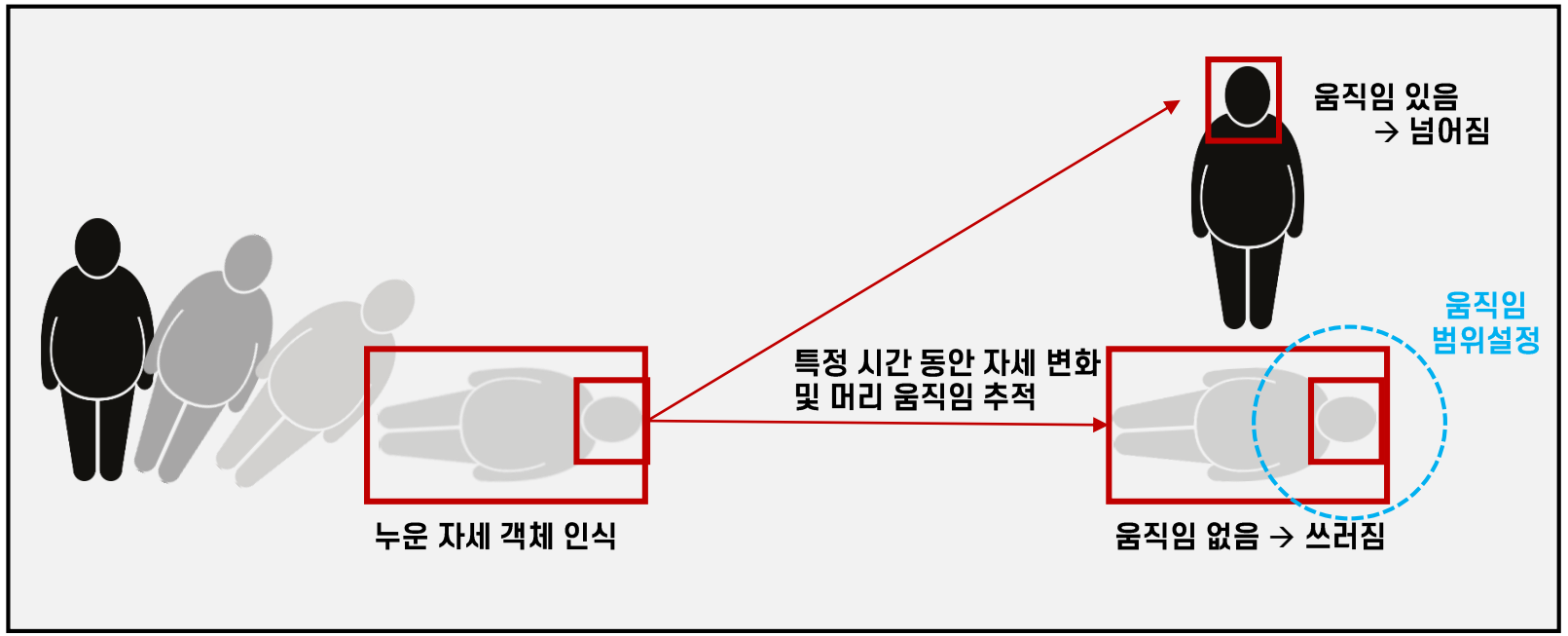
04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성

04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성

3. 쓰러진 후 움직임 변화 추적	• 특정 시간 동안 특정 범위내의 움직임이 없을 경우 → 넘어진 경우 바로 일어나겠지만 쓰러진 경우에는 움직임이 없을 확률 高 → 원근을 고려해 일정 시간 움직임이 없을 경우 쓰러진 것으로 인식 → 자세 인식 모델로 눕는 자세를 탐지했을 때 알고리즘이 적용되도록 함 (등산로 등에 적용)



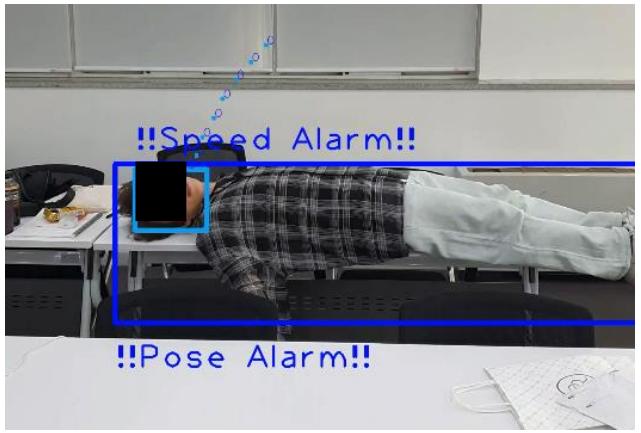
04. 프로젝트 수행

쓰러짐 구분 알고리즘 작성

3. 쓰러진 후 움직임 변화 추적

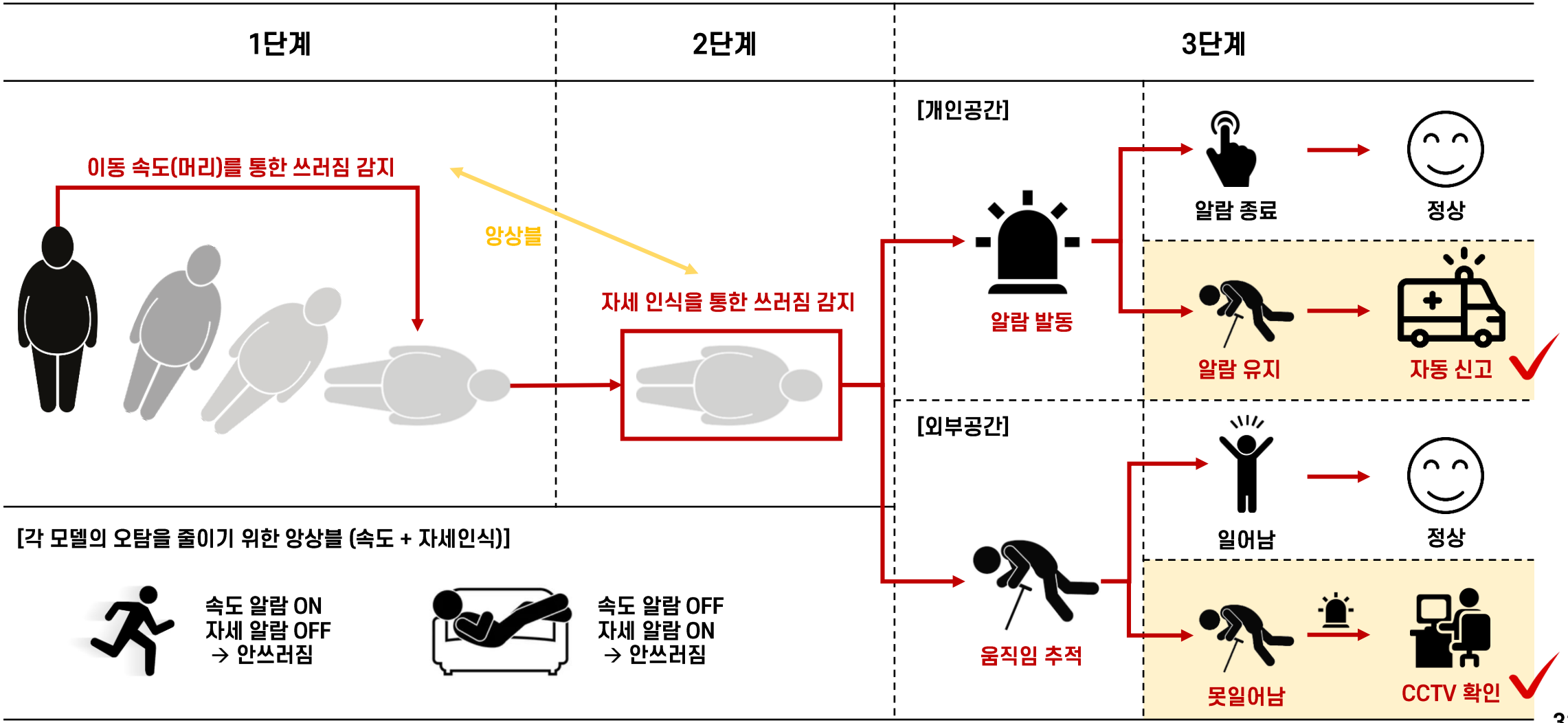
- 누운 자세 인식 테스트

쓰러짐 감지 (속도+자세)



04. 프로젝트 수행

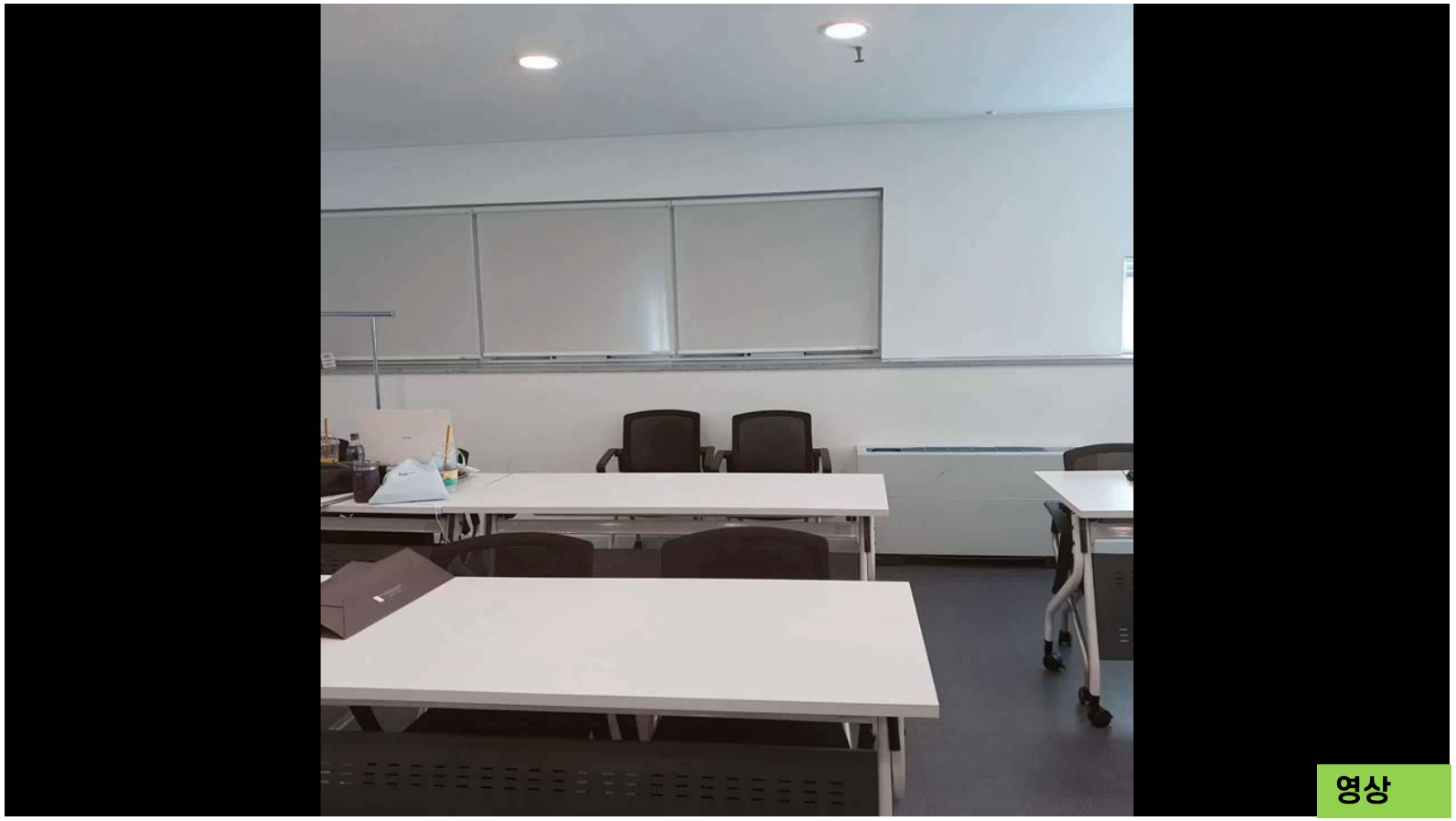
최종 서비스 알고리즘 요약



05. 최종 테스트 결과

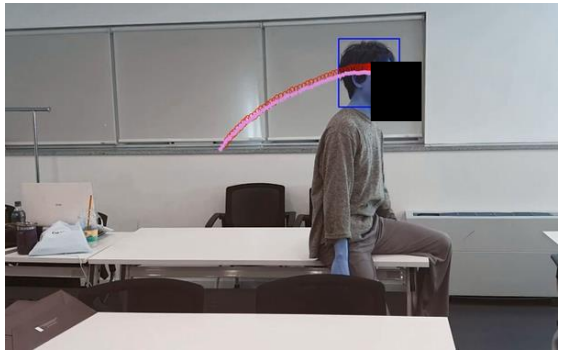
최종 서비스 테스트 시연

* 파란 얼굴은 의도된 연출임



쓰러짐 감지 (속도+자세)

쓰러짐 감지 해제



06. 자체평가 및 한계점 분석

긍정평가	- 작성한 로직이 자체 제작 테스트 영상에서 성공적으로 작동 - 기간 및 가용 리소스를 감안할 때 만족스러운 아웃풋 도출
개선사항	- 객체 추적(DeepSort) 등 추가적인 모델을 적용하면 사용 환경의 제약을 넓힐 수 있을 것으로 예상 - CCTV 각도 및 다양한 상황에서 테스트를 진행하면 보다 범용적인 서비스 로직을 완성할 수 있을 것 - 최고 속도 외 속도 변화(가속도)에 따른 쓰러짐 탐지 알고리즘 등을 추가하면 오탐을 줄이고 정확도를 더 높일 수 있을 것으로 예상

→ 약 2.5주의 짧은 기간과 가용 컴퓨팅 리소스의 부재로 인한 한계점 존재

Thank you