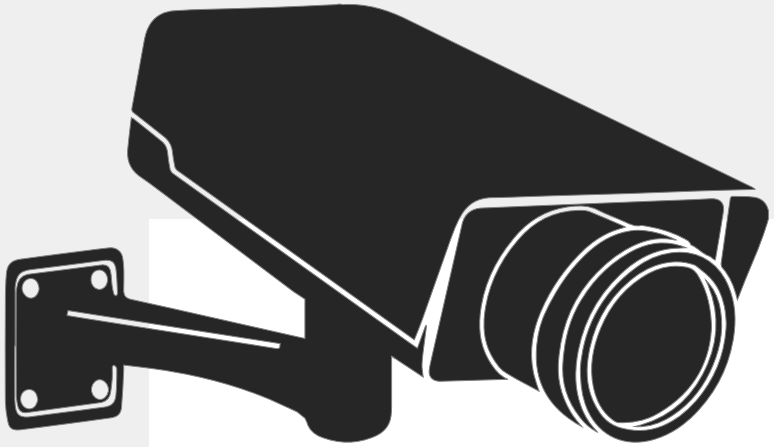


LSTM을 활용한 이상행동 탐지

수상하 조
(평일 오후 3조)
권태윤 배선용 한지은

CONTENT

1 주제 선정

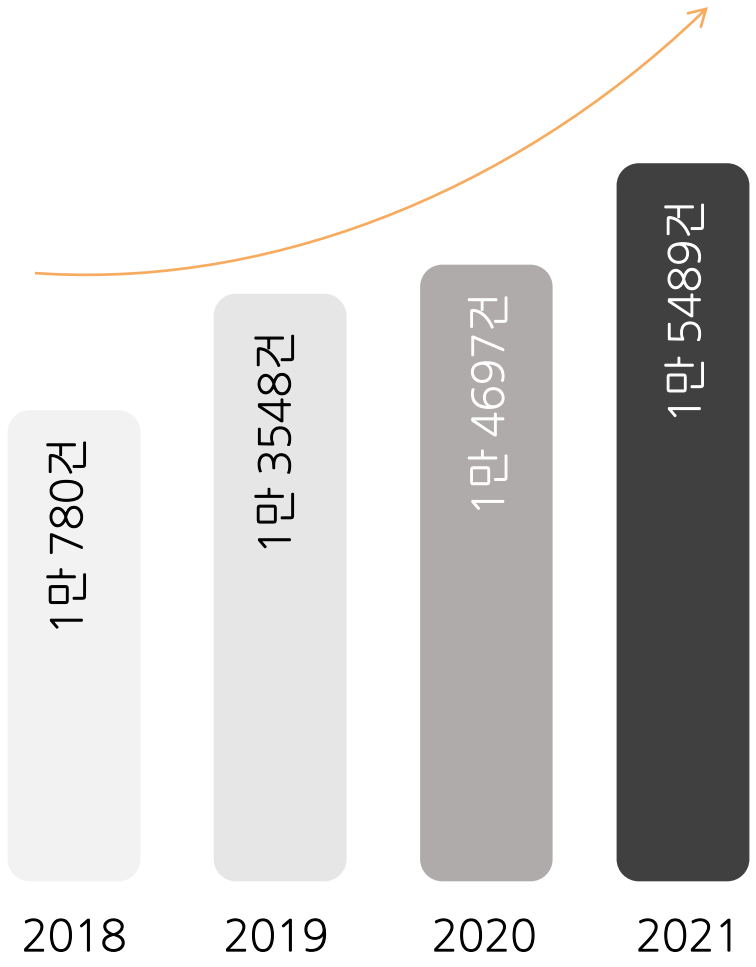
프로젝트 주제

실내 **이상행동(절도) 탐지** 모델 개발

Part 1

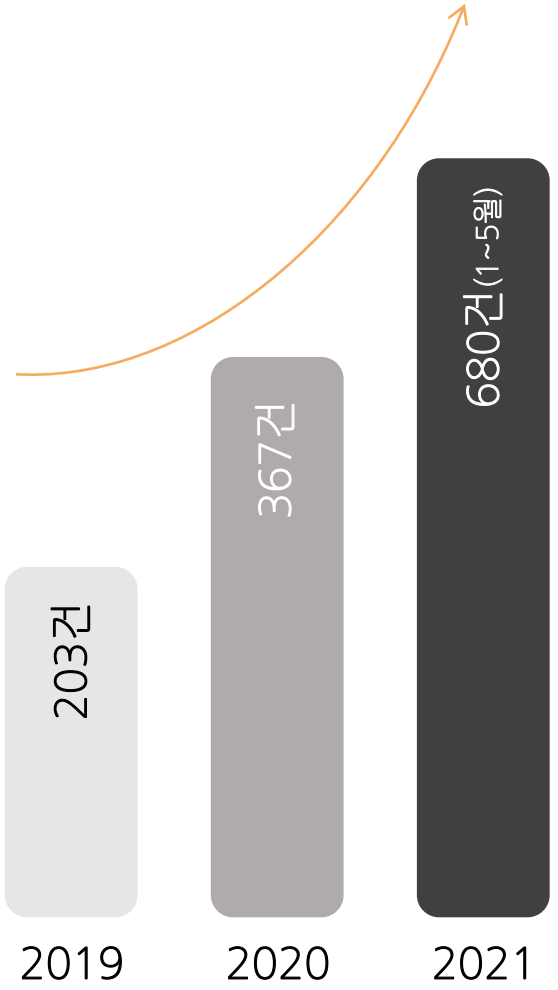
주제 선정 배경

편의점 내 범죄 발생 현황



출처: 데이터솜

무인 점포 절도 범죄

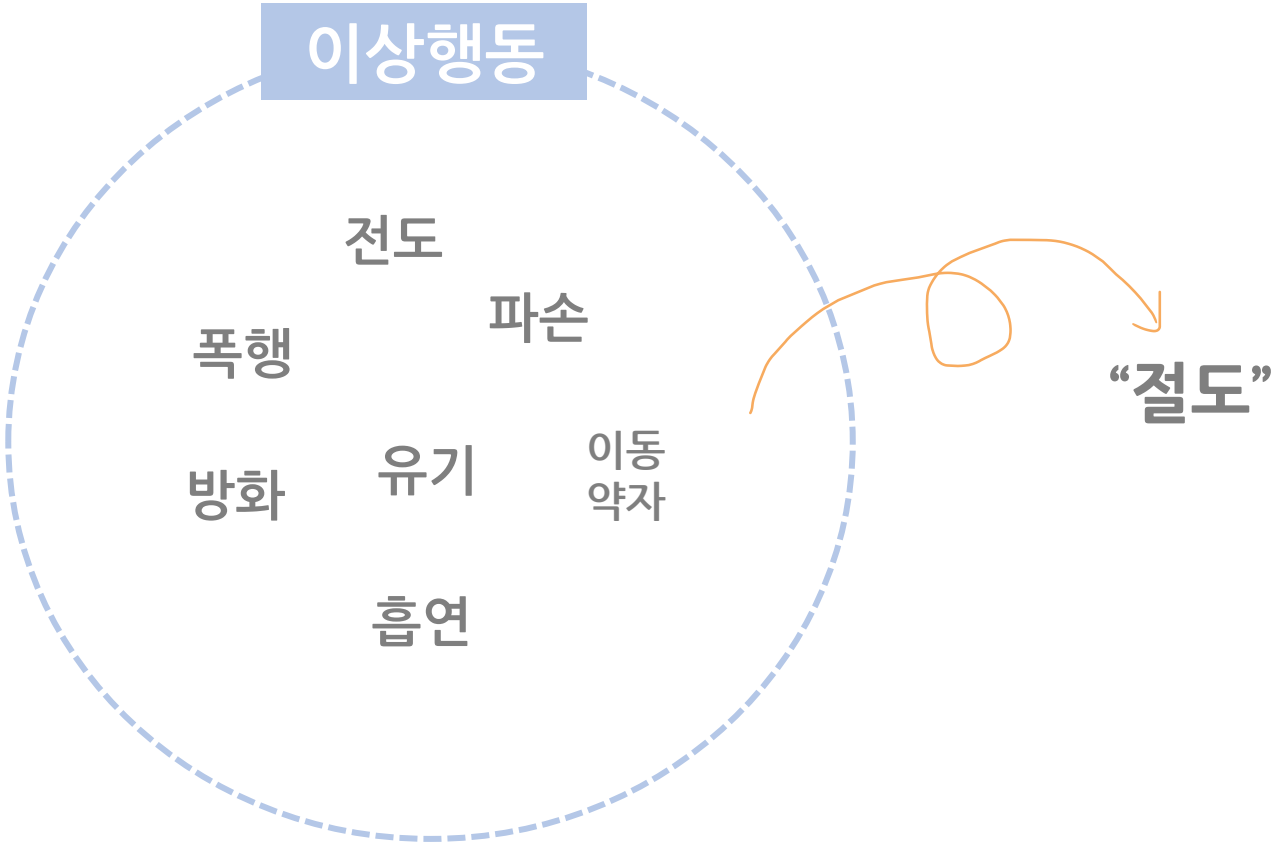


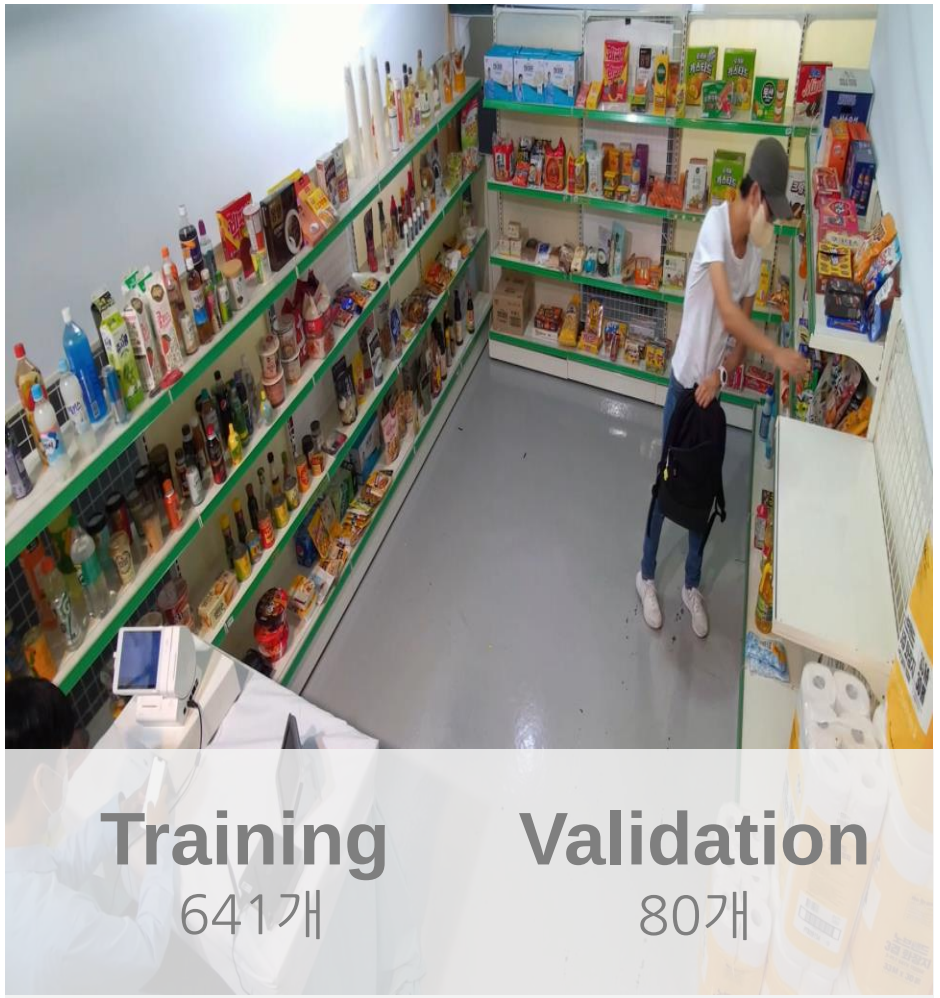
출처: 경찰청

2 데이터 전처리

데이터 소개

실내(편의점, 매장) 사람 이상행동 데이터





출처: AI - Hub

데이터 전처리(1) - Clip 추출

평균 재생시간 : 1분 (180 fps)

NORMAL : 10초 (30 fps)
ABNORMAL : 10초 (30 fps)

NORMAL : 10초 (30 fps)
ABNORMAL : 10초 (30 fps)

NORMAL : 10초 (30 fps)
ABNORMAL : 10초 (30 fps)

데이터 추출 비율
NORMAL : ABNORMAL
50 : 50

Part 2

데이터 전처리(1) - Clip 추출

001_normal_30

타입 : 정상 행동
재생시간 : 10초
총 프레임 수 : 30

001_abnormal_30

타입 : 절도
재생시간 : 10초
총 프레임 수 : 30

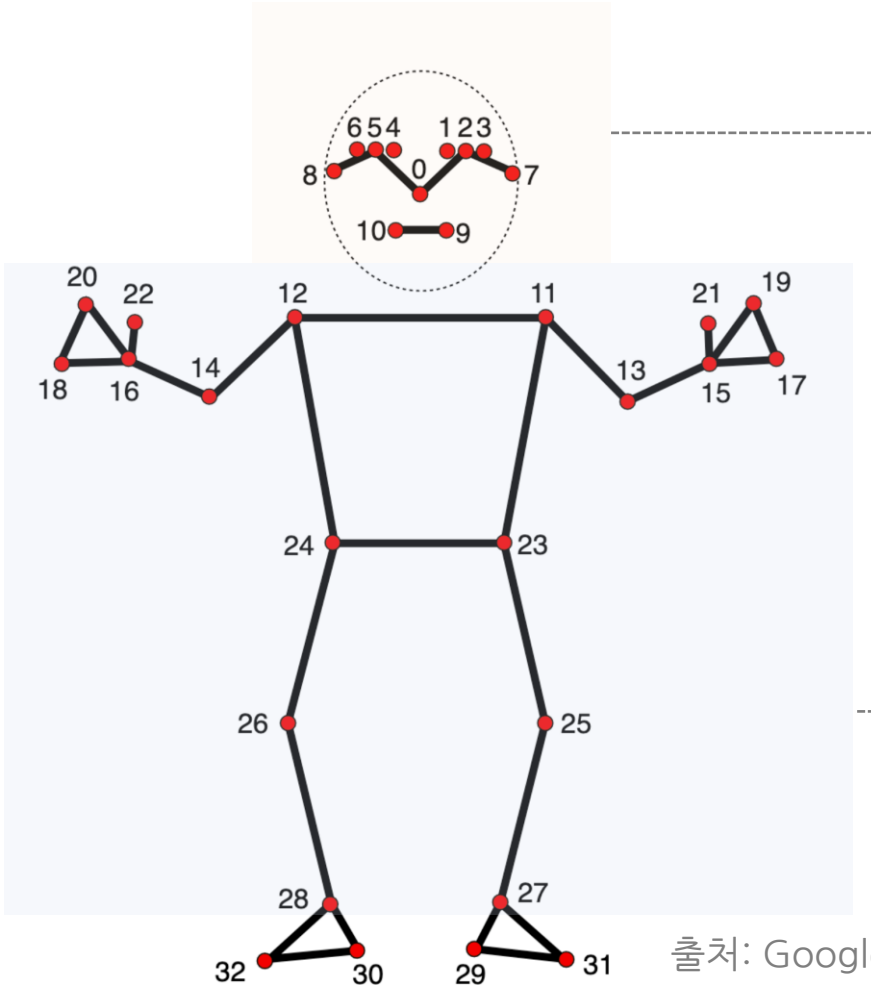
데이터 전처리(2) - 추출한 Clip 병합

NORMAL 10초 (30 fps)
+
NORMAL 10초 (30 fps)
+
NORMAL 10초 (30 fps)

ABNORMAL 10초 (30 fps)
+
ABNORMAL 10초 (30 fps)
+
ABNORMAL 10초 (30 fps)

데이터 전처리(3) - MediaPipe

Pose landmark detection



출처: Google

얼굴

0 - nose
1 - left eye (inner)
2 - left eye
3 - left eye (outer)
4 - right eye (inner)
5 - right eye
6 - right eye (outer)
7 - left ear
8 - right ear
9 - mouth (left)
10 - mouth (right)

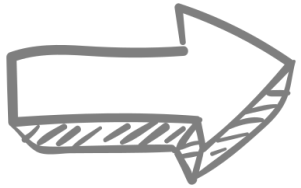
몸

11 - left shoulder
12 - right shoulder
13 - left elbow
14 - right elbow
15 - left wrist
16 - right wrist
17 - left pinky
18 - right pinky
19 - left index
20 - right index
21 - left thumb
22 - right thumb
23 - left hip
24 - right hip
25 - left knee
26 - right knee
27 - left ankle
28 - right ankle
29 - left heel
30 - right heel
31 - left foot index
32 - right foot index

Part 2

데이터 전처리(3) - MediaPipe

적용 전



적용 후

3 모델링

사용한 프레임 워크

사용환경

딥러닝 프레임 워크

MediaPipe

PyTorch

YOLOv5

OpenCV

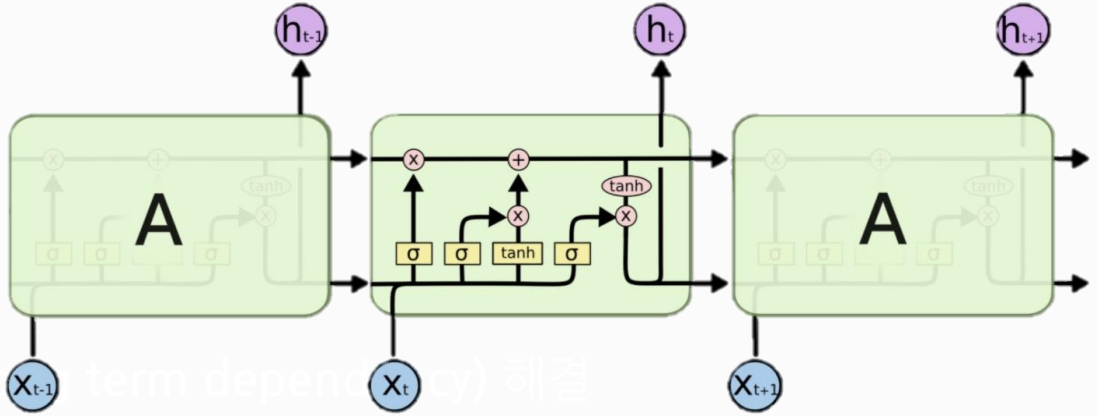
모델 선정 - LSTM(Long Short-Term Memory)

시간 순서에 따른 패턴 학습을 위한 순환신경망 필요

순환 신경망의 종류:

- ☑ LSTM
- ☑ Simple RNN
- ☑ GRU

LSTM



☑ Simple RNN에서 **Cell state**를 추가
→ **장기 의존성(Long term dependency)** 해결
→ Time step을 가로지르며 **셀 상태가 보존**

시간 순서에 따른 패턴을 학습하고 적용하기 위한 모델로 적정

모델 설계 - Input size, Loss function, Optimizer

하이퍼파라미터 (Hyperparameter)

```
class skeleton_LSTM(nn.Module):
    def __init__(self):
        super(skeleton_LSTM, self).__init__()
        self.lstm1 = nn.LSTM(input_size=36, hidden_size=128, num_layers=1, batch_first=True)
        self.lstm2 = nn.LSTM(input_size=128, hidden_size=256, num_layers=1, batch_first=True)
        self.lstm3 = nn.LSTM(input_size=256, hidden_size=512, num_layers=1, batch_first=True)
        self.dropout1 = nn.Dropout(0.1)
        self.lstm4 = nn.LSTM(input_size=512, hidden_size=256, num_layers=1, batch_first=True)
        self.lstm5 = nn.LSTM(input_size=256, hidden_size=128, num_layers=1, batch_first=True)
        self.lstm6 = nn.LSTM(input_size=128, hidden_size=64, num_layers=1, batch_first=True)
        self.dropout2 = nn.Dropout(0.1)
        self.lstm7 = nn.LSTM(input_size=64, hidden_size=32, num_layers=1, batch_first=True)
        self.fc = nn.Linear(32,2)
```

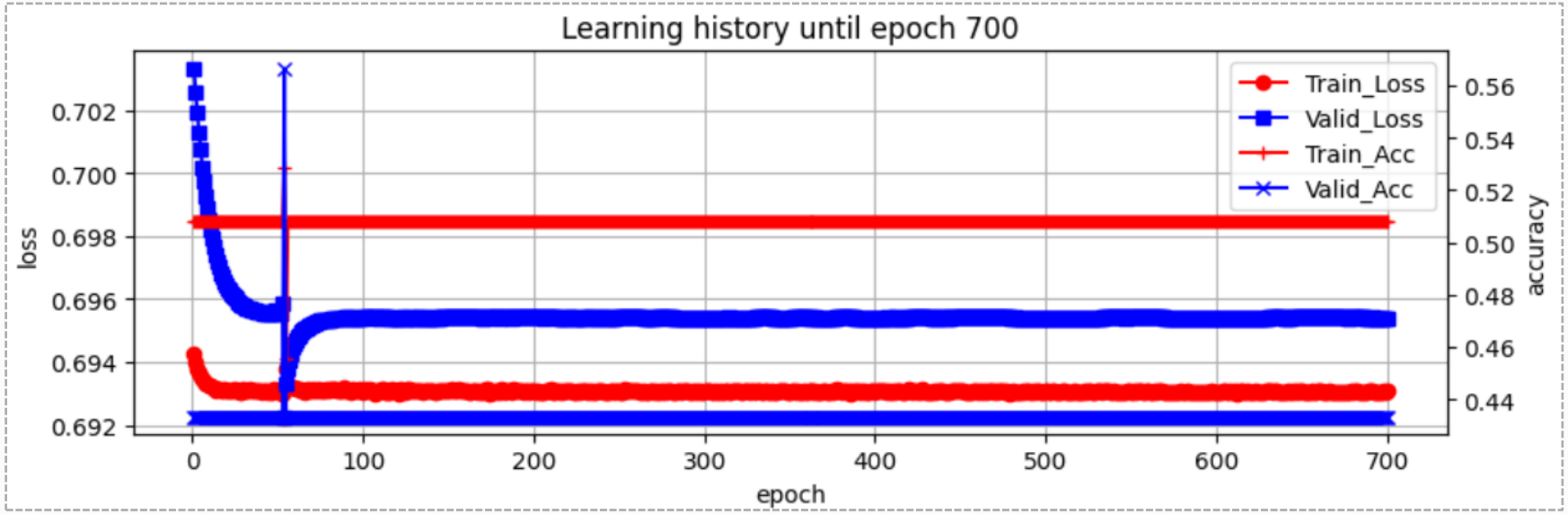
```
def forward(self, x) :
    x, _ = self.lstm1(x)
    x, _ = self.lstm2(x)
    x, _ = self.lstm3(x)
    x = self.dropout1(x)
    x, _ = self.lstm4(x)
    x, _ = self.lstm5(x)
    x, _ = self.lstm6(x)
    x = self.dropout2(x)
    x, _ = self.lstm7(x)
    x = self.fc(x[:,-1,:])
    return x
```

- ✓ **Loss function : CrossEntropyLoss**
- ✓ **Optimizer : Adam**

모델 설계 - Num layers

Num layers = 1로 설정한 이유

Num layers = 2 ?

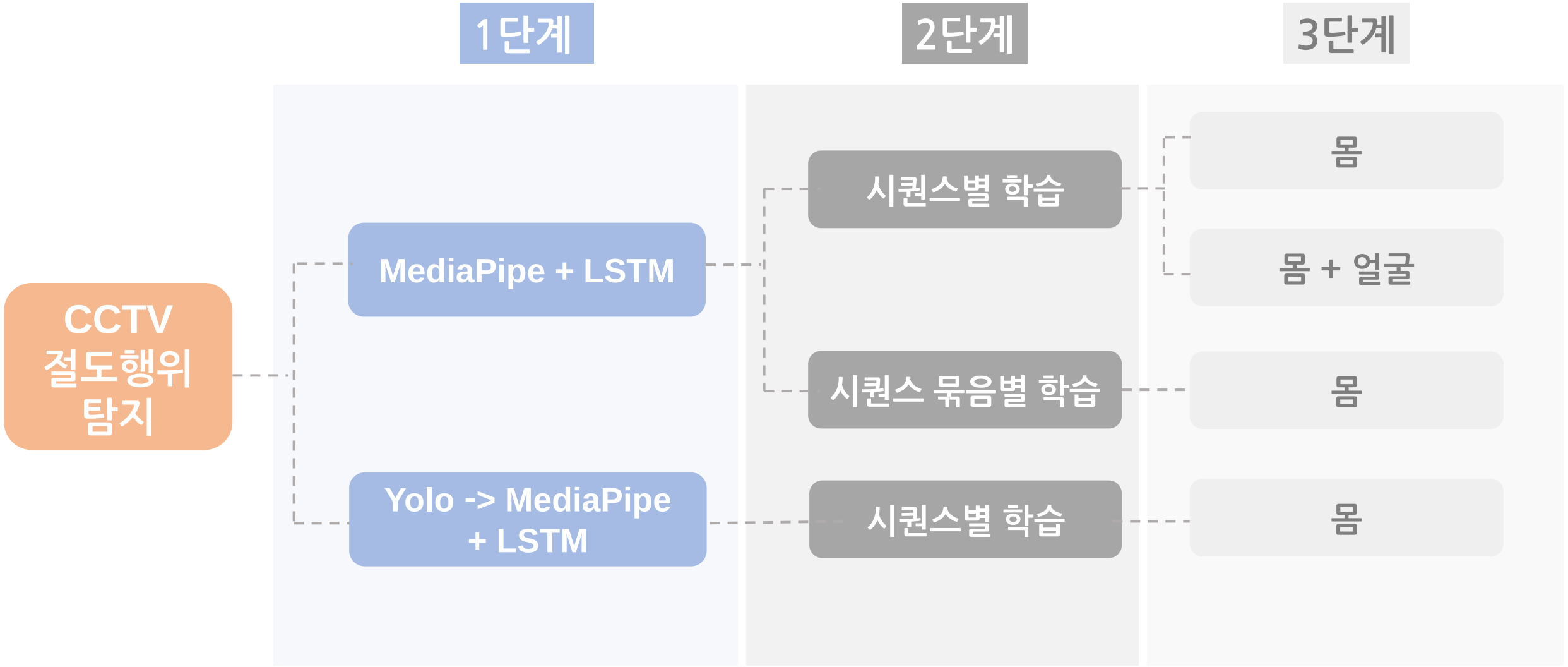


Epoch	Train_Loss	Train_Acc	Valid_Loss	Valid_Acc
700	0.693	0.508	0.695	0.433

4 결과

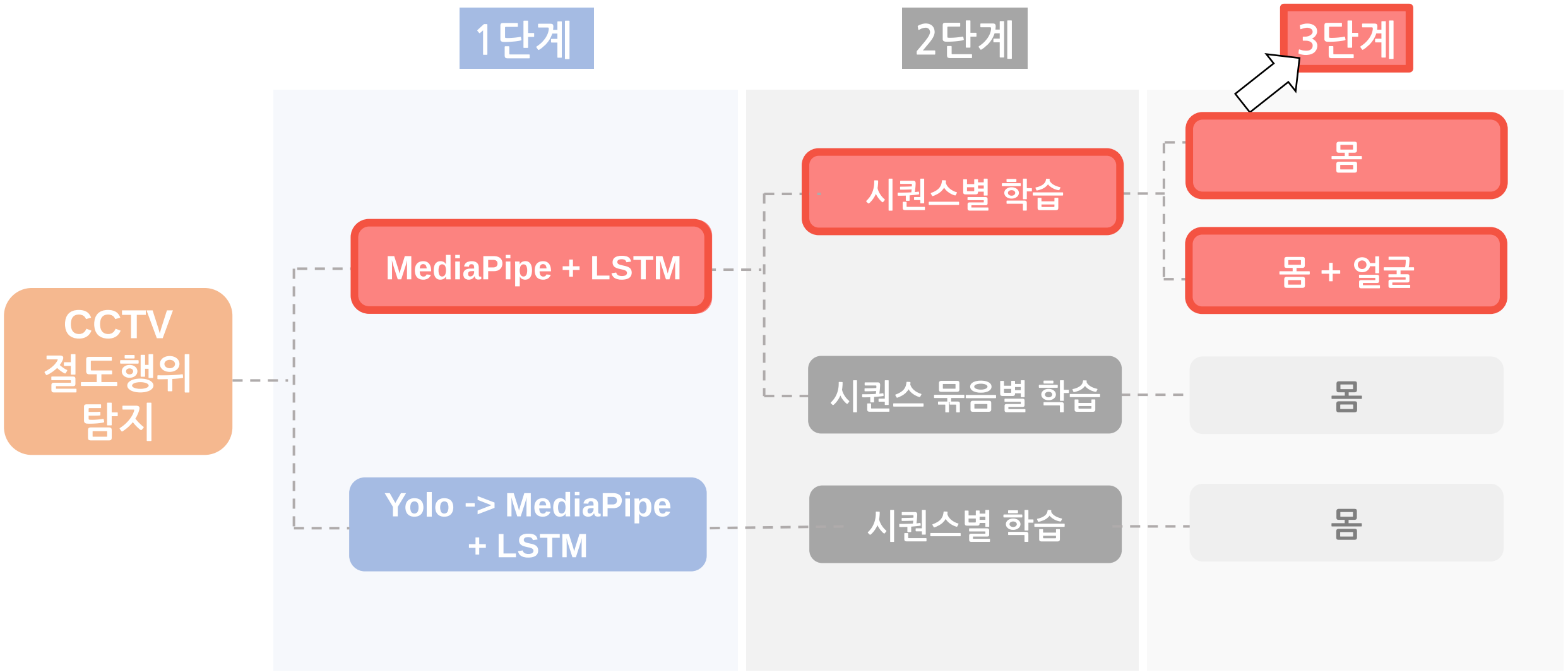
Part 4

단계별 결과 비교 분석



Part 4

단계별 결과 비교 분석 - 결과 1



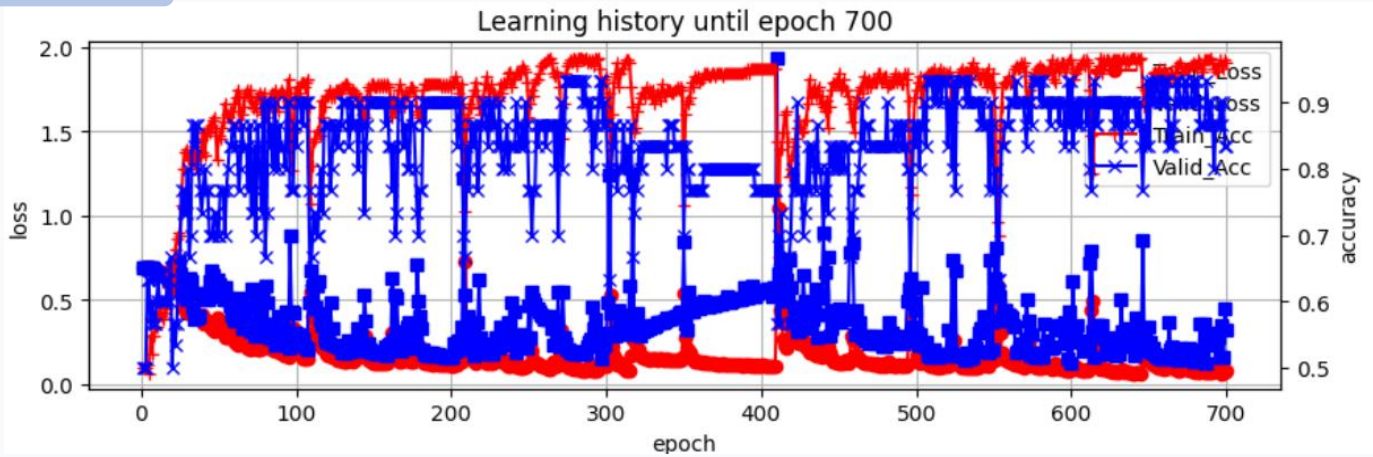
Part 4

결과 1 - 몸 vs 몸 + 얼굴

Epoch 700 | Batch size 6 | Learning rate 0.0001

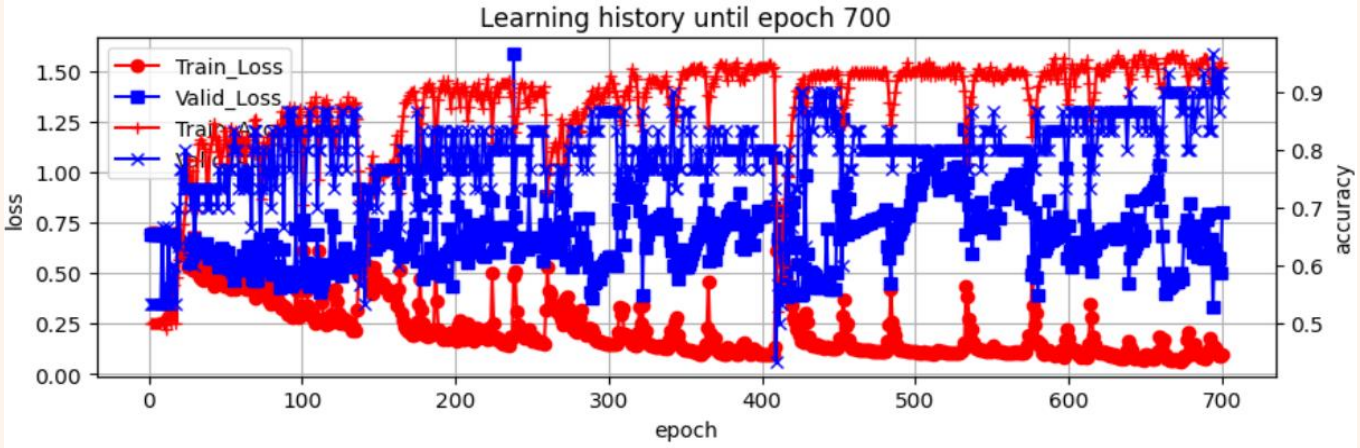
Train_Acc Train_Loss Valid_Acc Valid_Loss

몸



Train_Loss	Train_Acc
0.084	0.959
Valid_Loss	**Valid_Acc**
0.322	0.833

몸 + 얼굴



Train_Loss	Train_Acc
0.096	0.943
Valid_Loss	**Valid_Acc**
0.804	0.9

Part 4

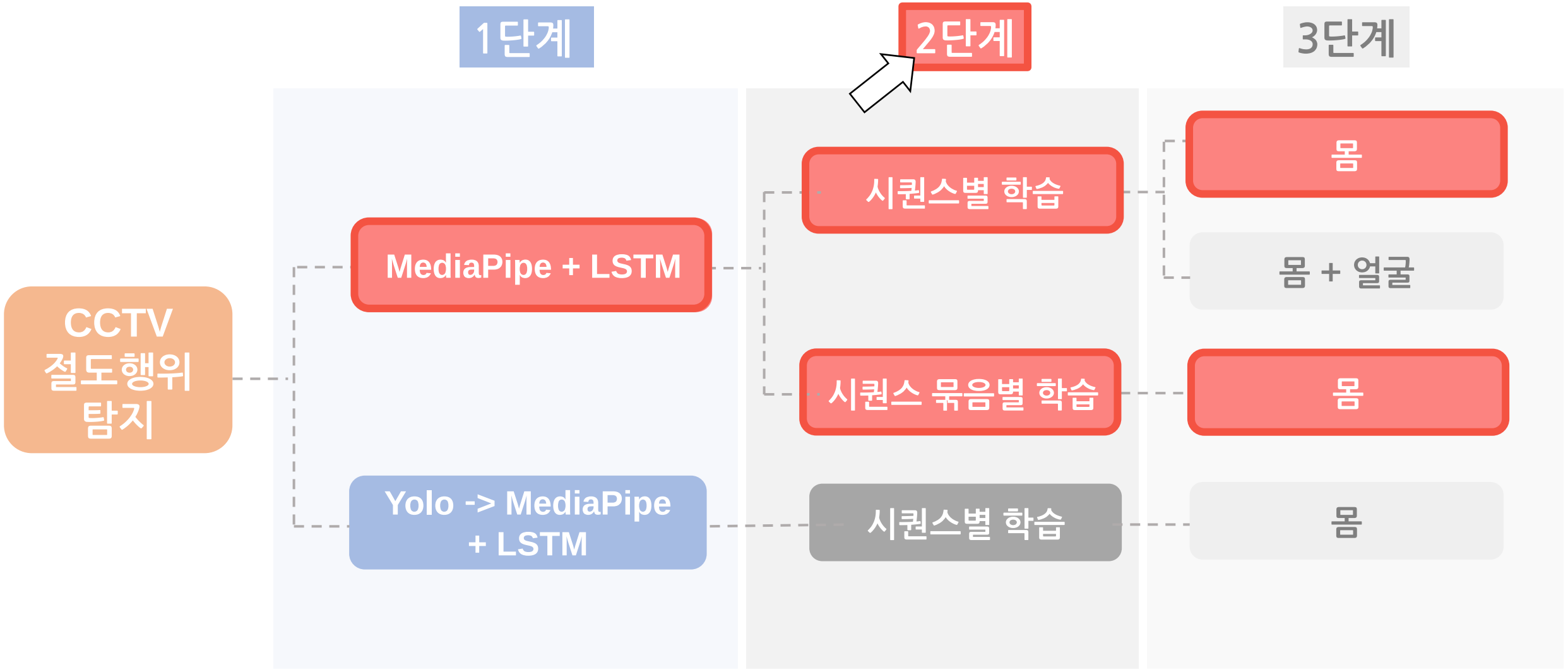
결과 1 - 몸 vs 몸 + 얼굴

몸

몸 + 얼굴

Part 4

단계별 결과 비교 분석 - 결과 2



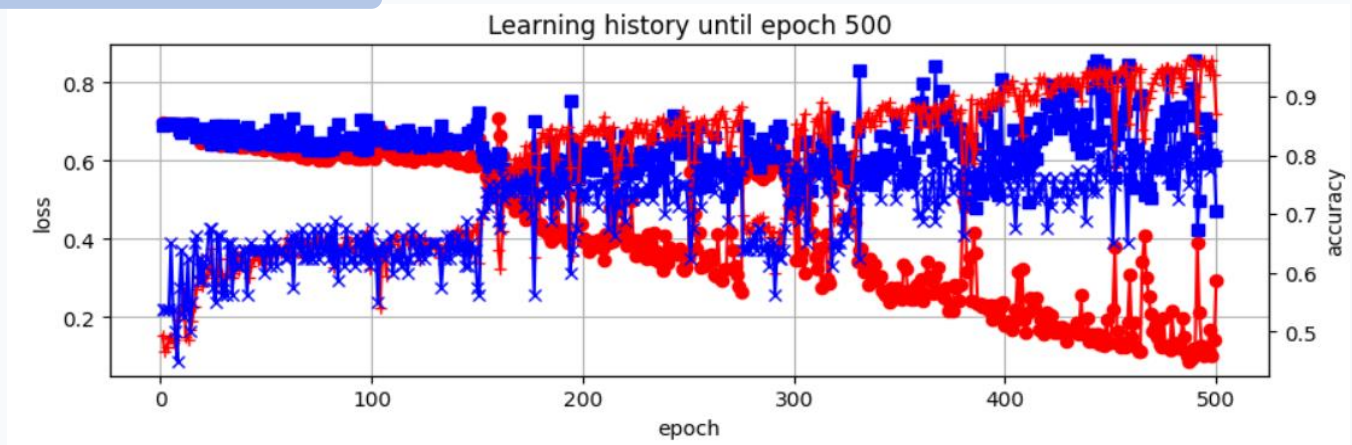
Part 4

결과 2 - 시퀀스별 vs 시퀀스 묶음별

Batch size 5 | Learning rate 0.0001

Train_Acc Train_Loss Valid_Acc Valid_Loss

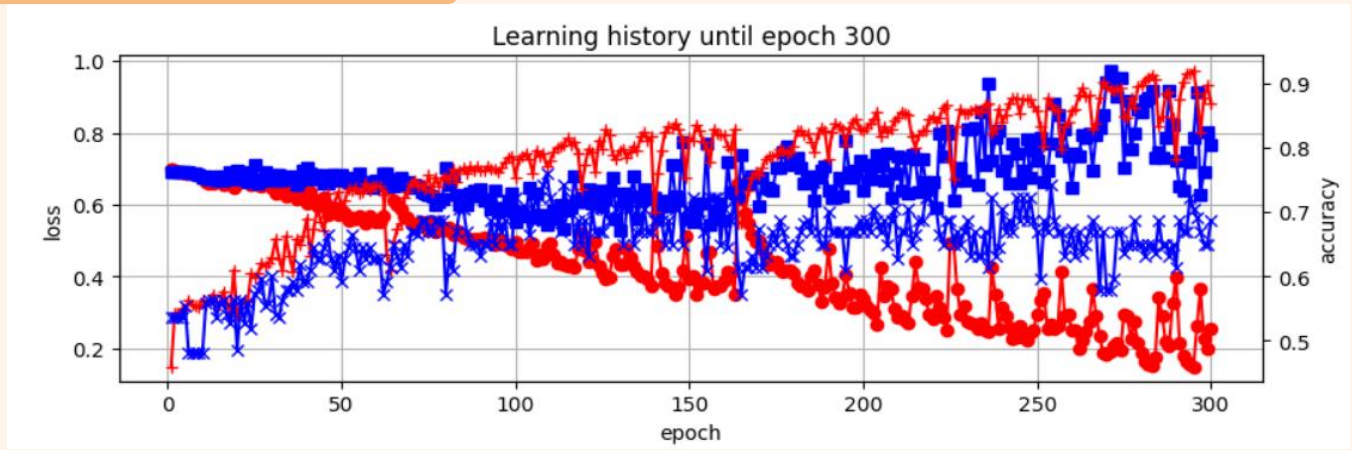
시퀀스별 학습



Epoch 500

Train_Loss	Train_Acc
0.293	0.87
Valid_Loss	**Valid_Acc**
0.47	0.8

시퀀스 묶음별 학습



Epoch 300

Train_Loss	Train_Acc
0.256	0.869
Valid_Loss	**Valid_Acc**
0.765	0.686

Part 4

결과 2 - 시퀀스별 vs 시퀀스 묶음별 (편의점)

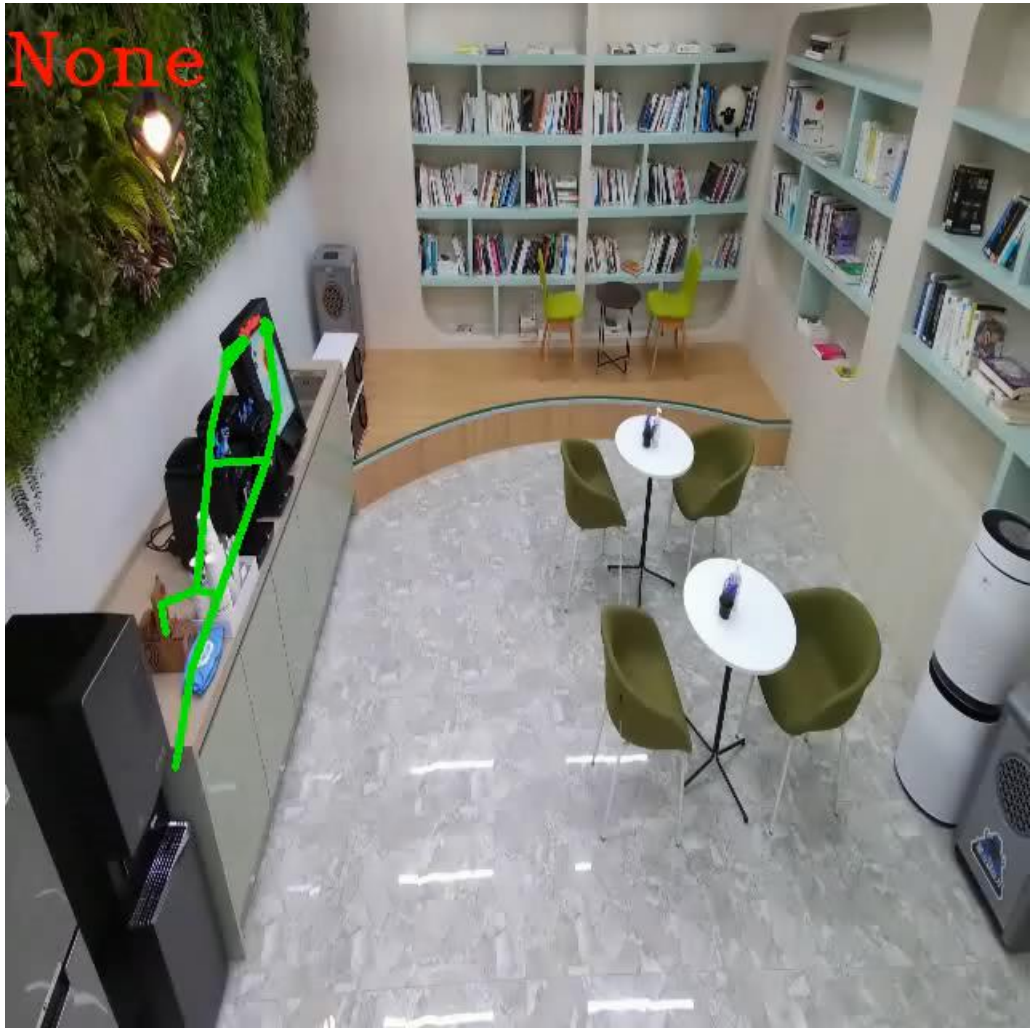
시퀀스별 학습

시퀀스 묶음별 학습

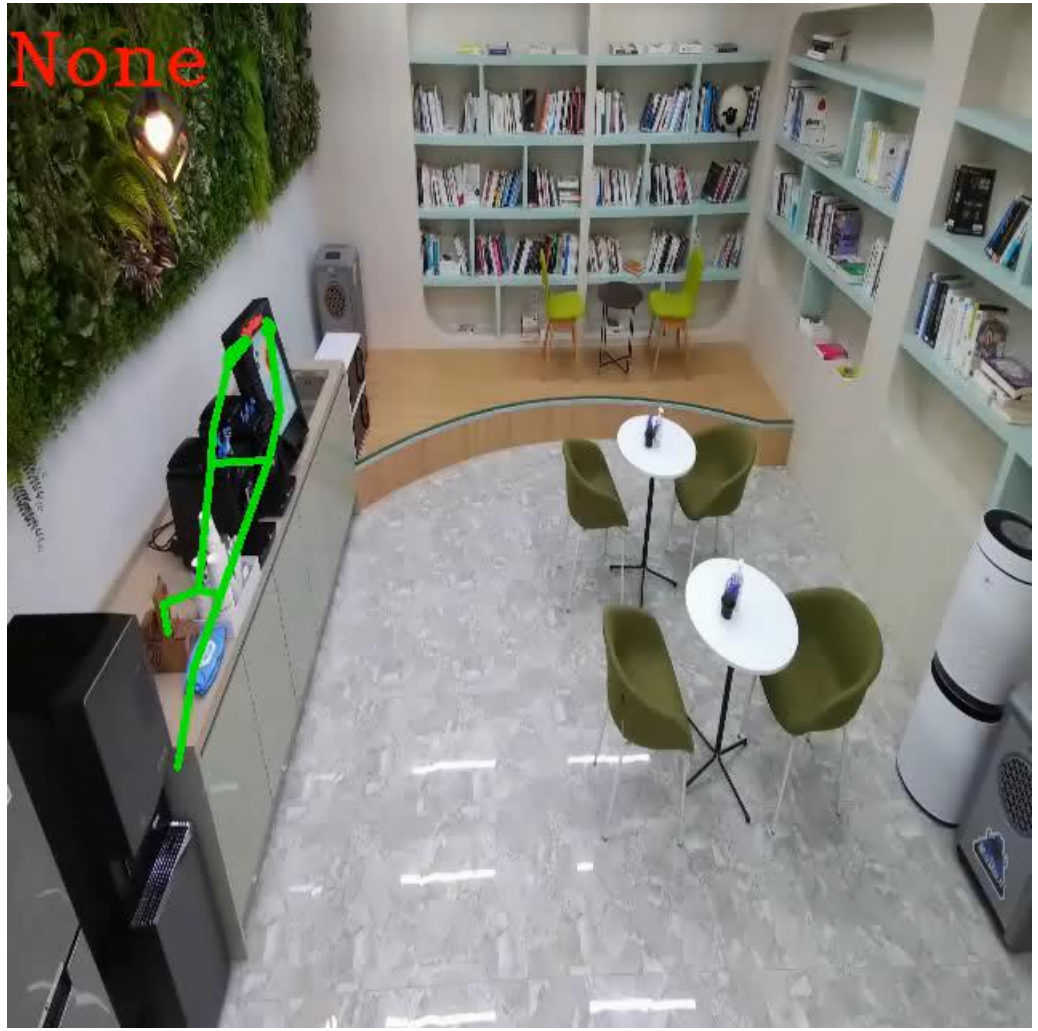
Part 4

결과 2 - 시퀀스별 vs 시퀀스 묶음별 (무인점포)

시퀀스별 학습

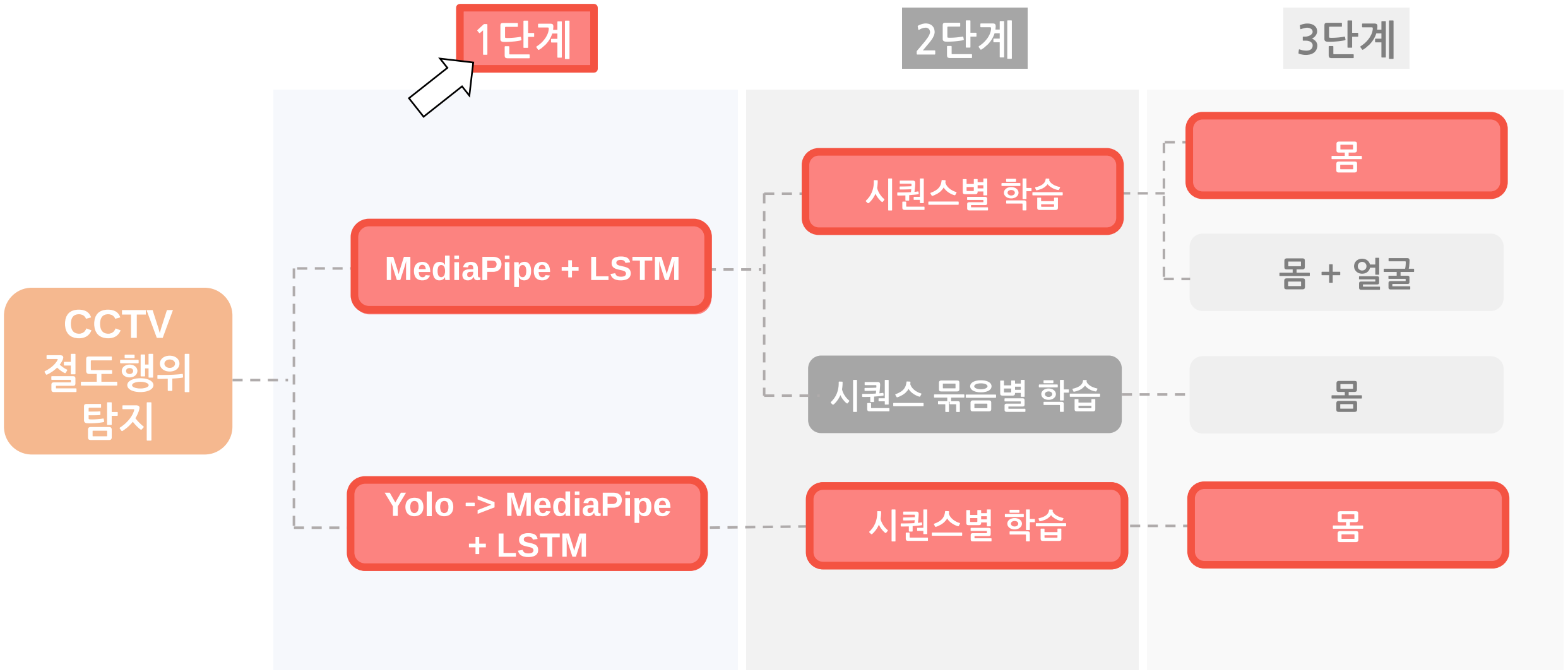


시퀀스 묶음별 학습



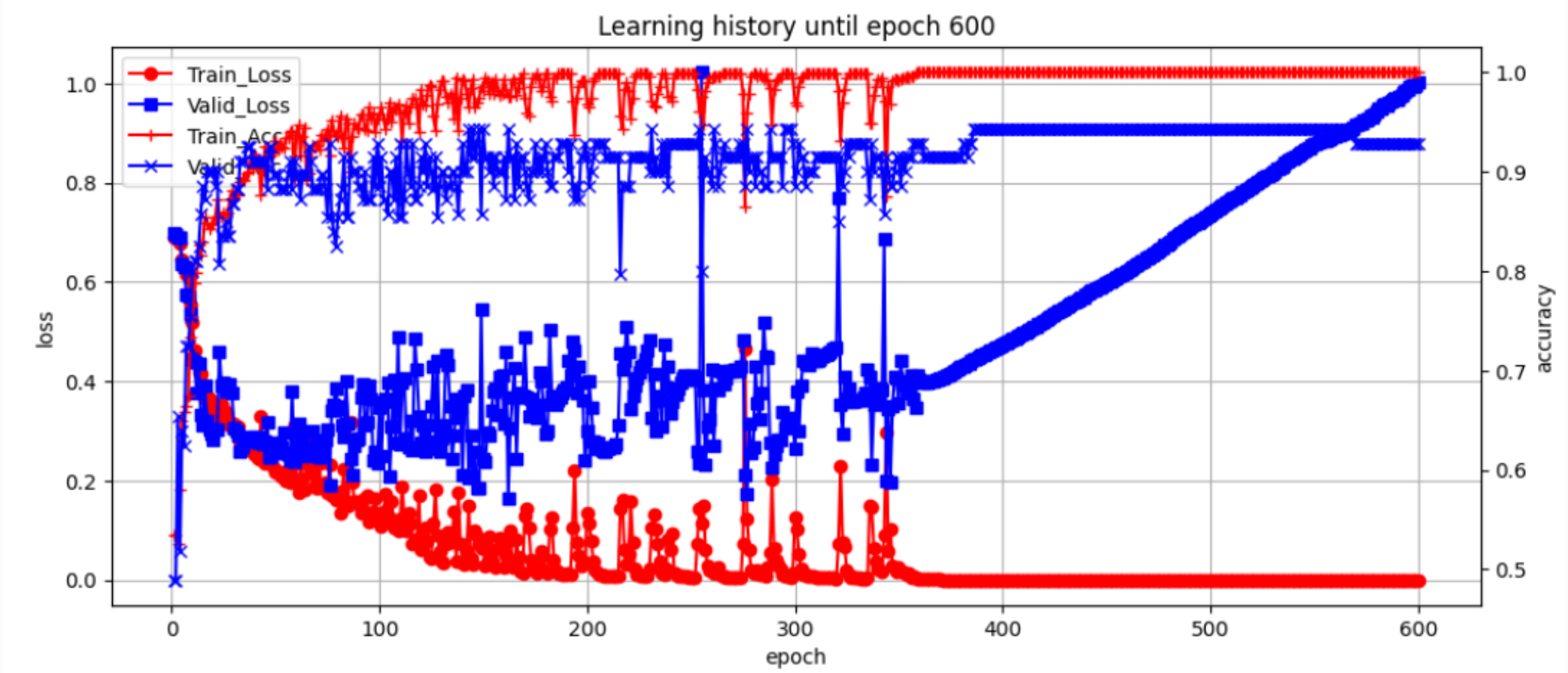
Part 4

단계별 결과 비교 · 분석



결과 3 - Yolo 로 객체 탐지 후 이상탐지

Epoch 600 | Batch size 6 | Learning rate 0.0001



Train_Loss	Train_Acc	Valid_Loss	Valid_Acc
0.0	1.0	1.002	0.929

Part 4

결과 3 - Yolo 로 객체 탐지 후 이상탐지 (편의점)

Yolo 미적용

Yolo 적용

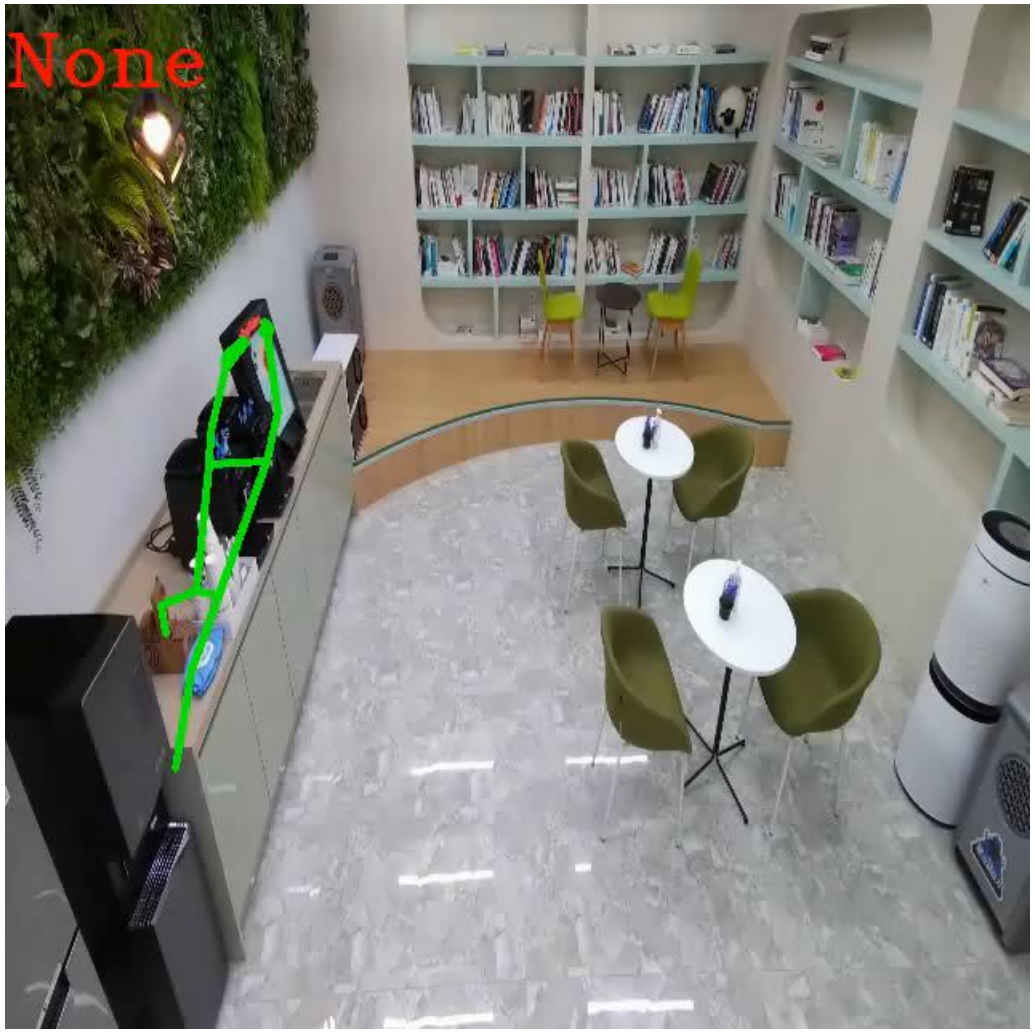
결과 3 - Yolo 로 객체 탐지 후 이상탐지 (무인점포 1)

Yolo 미적용

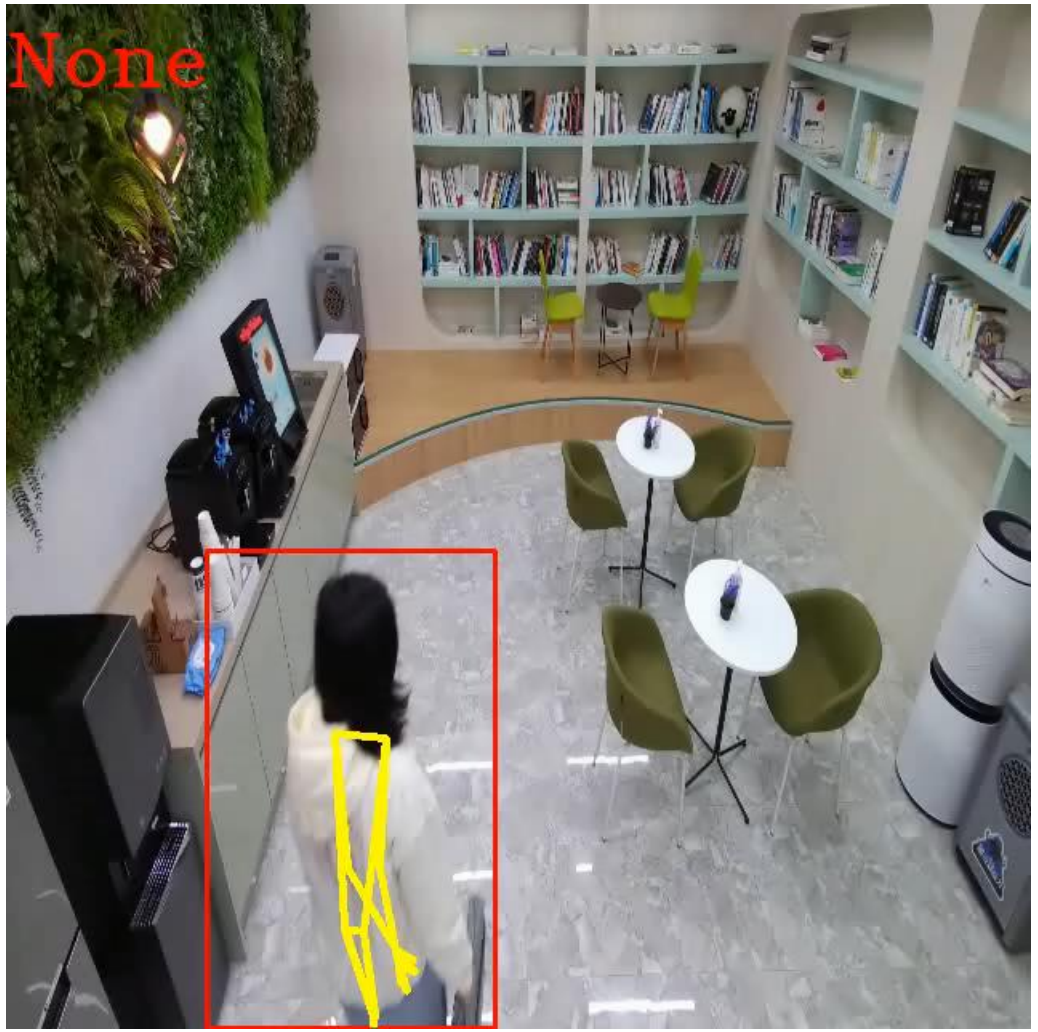
Yolo 적용

결과 3 - Yolo 로 객체 탐지 후 이상탐지 (무인점포 2)

Yolo 미적용



Yolo 적용



최종 결과

편의점

MediaPipe + LSTM

시퀀스별 학습

몸

무인점포

Yolo -> MediaPipe + LSTM

시퀀스별 학습

몸

5 기대 효과

Part 5

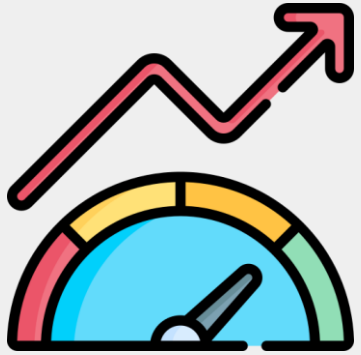
기대 효과

탐지 자동화로 보안 강화

실시간 탐지로 신속한 조치

보안 인력 최소화로 비용 절감

매장 이용에 대한 서비스 품질 향상

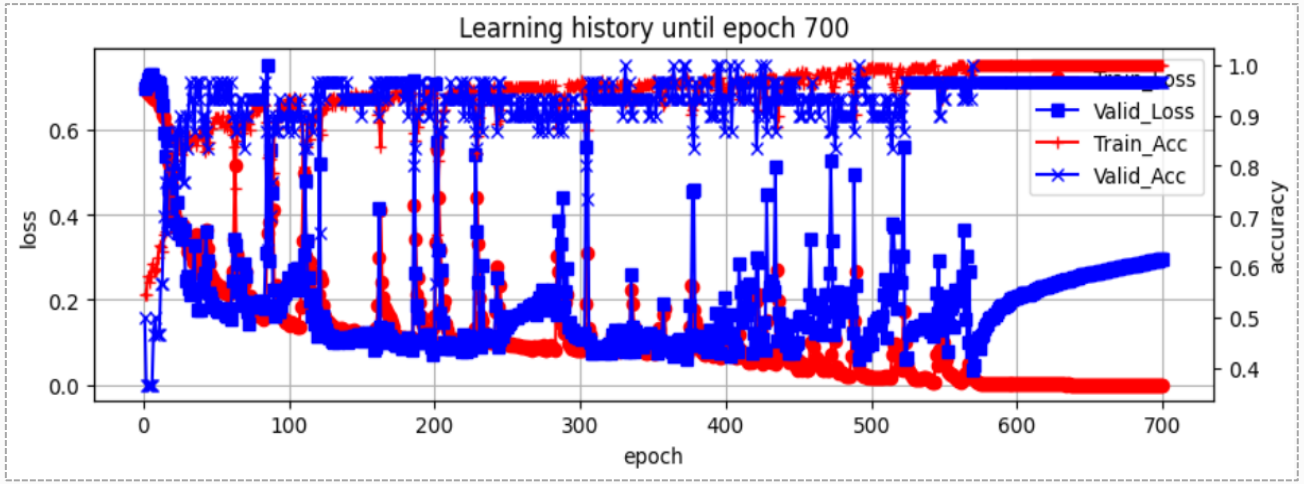


학습량 누적에 따라 탐지 성능 향상

6 아쉬운 점

아쉬운 점 1

GRU



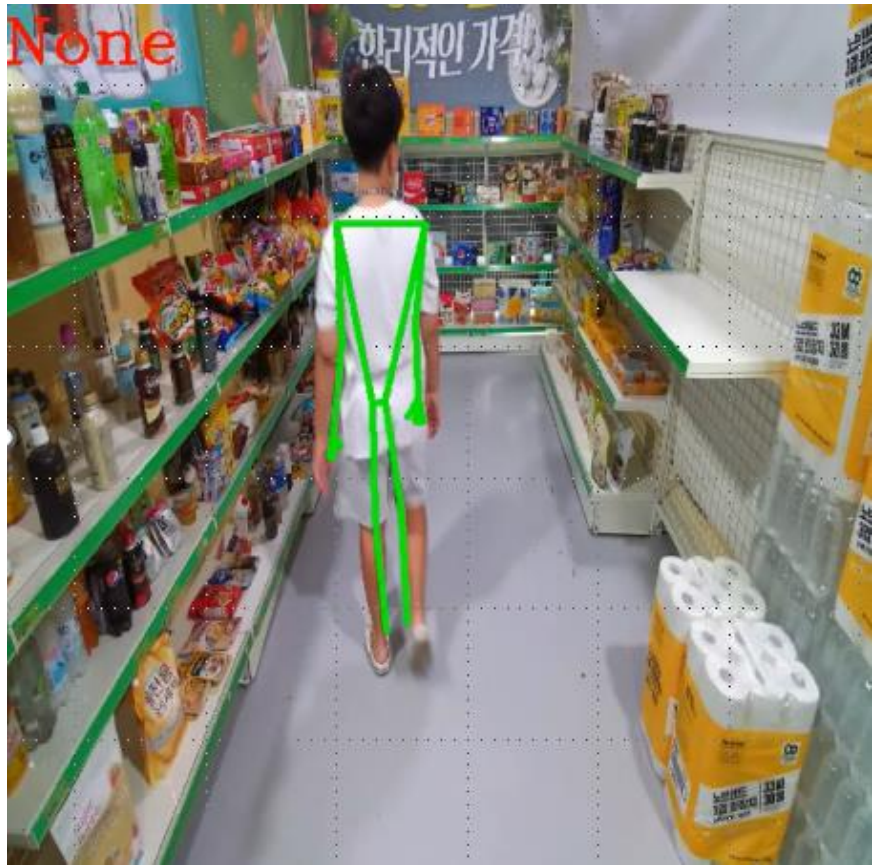
Epoch	Train_Loss	Train_Acc	Valid_Loss	Valid_Acc
700	0.0	1.0	0.296	0.967

순환 신경망(RNN) 계열 다른 모델(예. **GRU**)과

추가로 비교 분석했다면 더욱 **다양한 결과 도출** 가능

✓ 예시영상

아쉬운 점 2



다중 객체 탐지 한계

단일 객체의 이상행동 데이터를 기준으로 모델 개발

다중 객체의 이상행동을 탐지하려면 한계 존재

감사합니다