# AI Fitness Trainer :

## Developing a real-time exercise evaluation system
## 실시간 운동 평가 시스템 개발

TEAM 3조 CEREAL
김요셉, 박현상, 김은교, 최아리, 고정빈, 임태하

# 목차

# 프로젝트 개요

## 목표

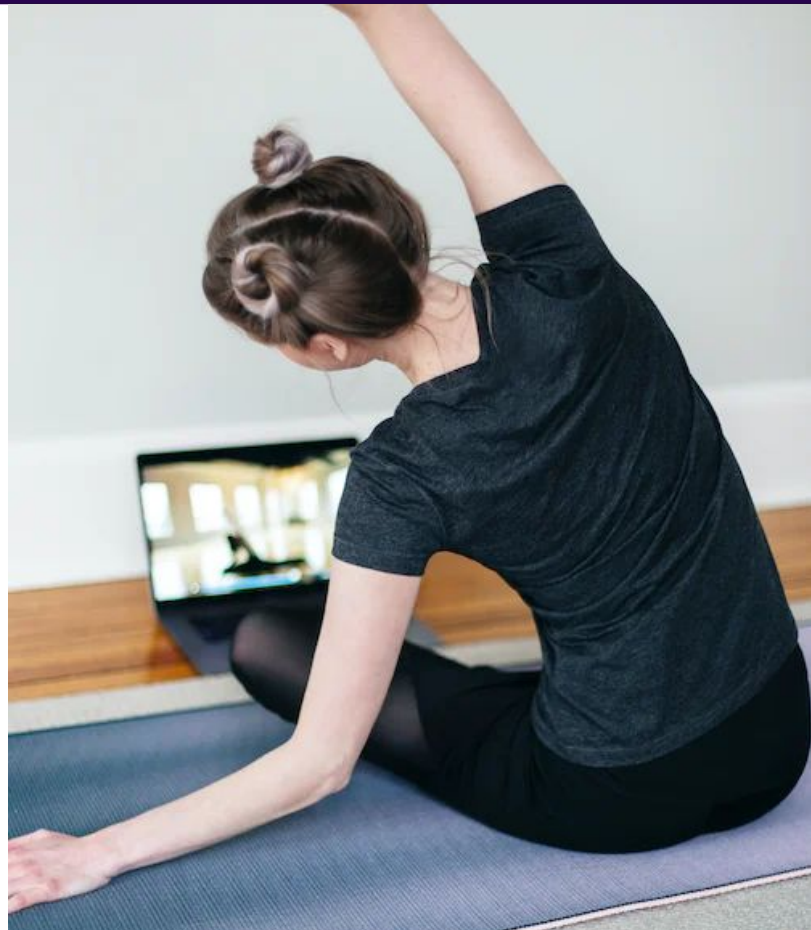
- 운동 자세의 실시간 평가 및 개선을 위한 피드백 제공 시스템 개발
  - 6가지 운동에 대하여 분류
  - 운동별 2가지 이상 자세 평가
  - 실시간 피드백 출력

## 배경

- 비대면 피트니스 시장의 성장과 전문가의 직접적인 피드백 부재 문제 해결 필요

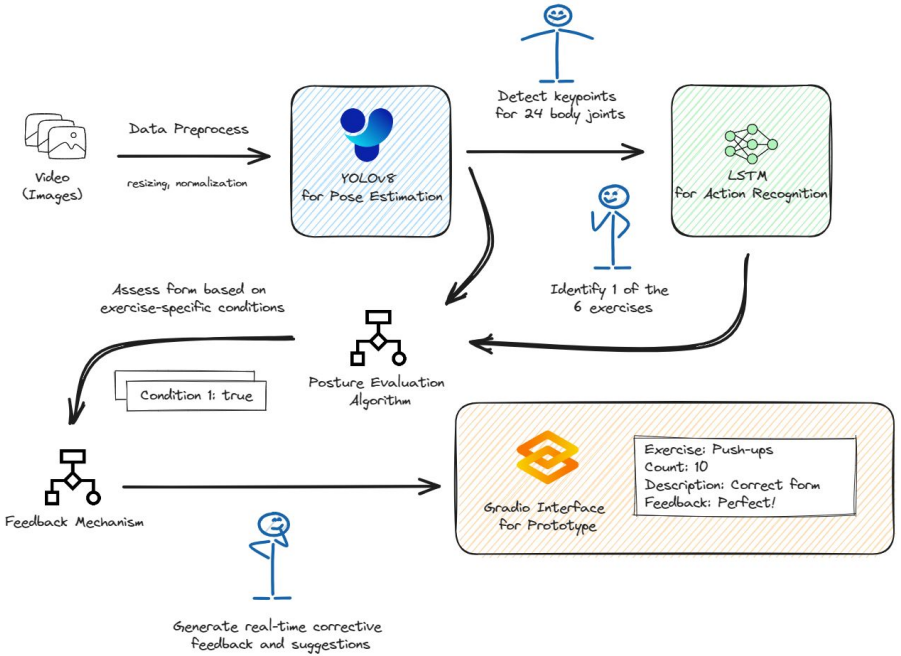
## 기대 효과

- 운동 효율성 증대 및 부상 방지를 위한 실시간 정확한 피드백 제공



# 접근 방법

1. **데이터 수집 및 처리:**
   a. 웹캠을 통해 실시간으로 사용자의 운동 영상을 수집하고 전처리
2. **포즈 추정:**
   a. YOLOv8을 기반으로 한 포즈 추정 모델을 사용하여 실시간으로 운동자의 키포인트 검출
3. **운동 패턴 분석:**
   a. LSTM 네트워크를 활용하여 수집된 시계열 키포인트 데이터로부터 운동 패턴을 분석하고 식별
4. **자세 평가 알고리즘:**
   a. 운동별 특정 조건들을 기반으로 각 자세의 정확성 평가
5. **피드백 메커니즘:**
   a. 정확성이 떨어지는 자세에 대해 개선점을 제시하는 피드백 시스템 설계




실시간 운동 평가 시스템 워크플로우 구조도

# 프로젝트 개발 환경

## 1. 버전 관리 및 코드 협업:

- Git
- GitHub

## 2. 개발 언어 및 통합 개발 환경(IDE):

- Python
- PyCharm
- Google Colab

## 3. 주요 라이브러리 및 프레임워크:

- Ultralytics
- TensorFlow
- Weights & Biases (W&B)

## 4. 모델 학습 및 실행 환경:

- YOLOv8
  - 용도: 객체 탐지 및 Keypoint Detection
- LSTM (Long Short-Term Memory)
  - 용도: 운동 분류를 위한 시계열 및 순차 데이터 처리

# 팀 역할

## 김요셉 (팀장)

### YOLOv8 모델링

- YOLOv8 모델 학습 및 최적화
- 키포인트 검출 향상
- 모델 추론 파이프라인 구축

## 임태하

### 시퀀스 모델링(LSTM)

- LSTM 모델 학습 및 최적화
- 시계열 데이터 전처리 파이프라인 구축
- 운동 패턴 인식 정확도 향상

## 박현상

### 데이터 엔지니어링

- 데이터 수집 및 관리
- AI HUB 데이터셋 관리
- 데이터 전처리 파이프라인 구축

## 고정빈

### 운동별 자세 평가 알고리즘 개발

- 운동별 테크닉 평가 기준 설정
- 알고리즘 설계 및 개발
- 평가 알고리즘 테스팅

## 최아리

### SW 개발 및 인터페이스 디자인

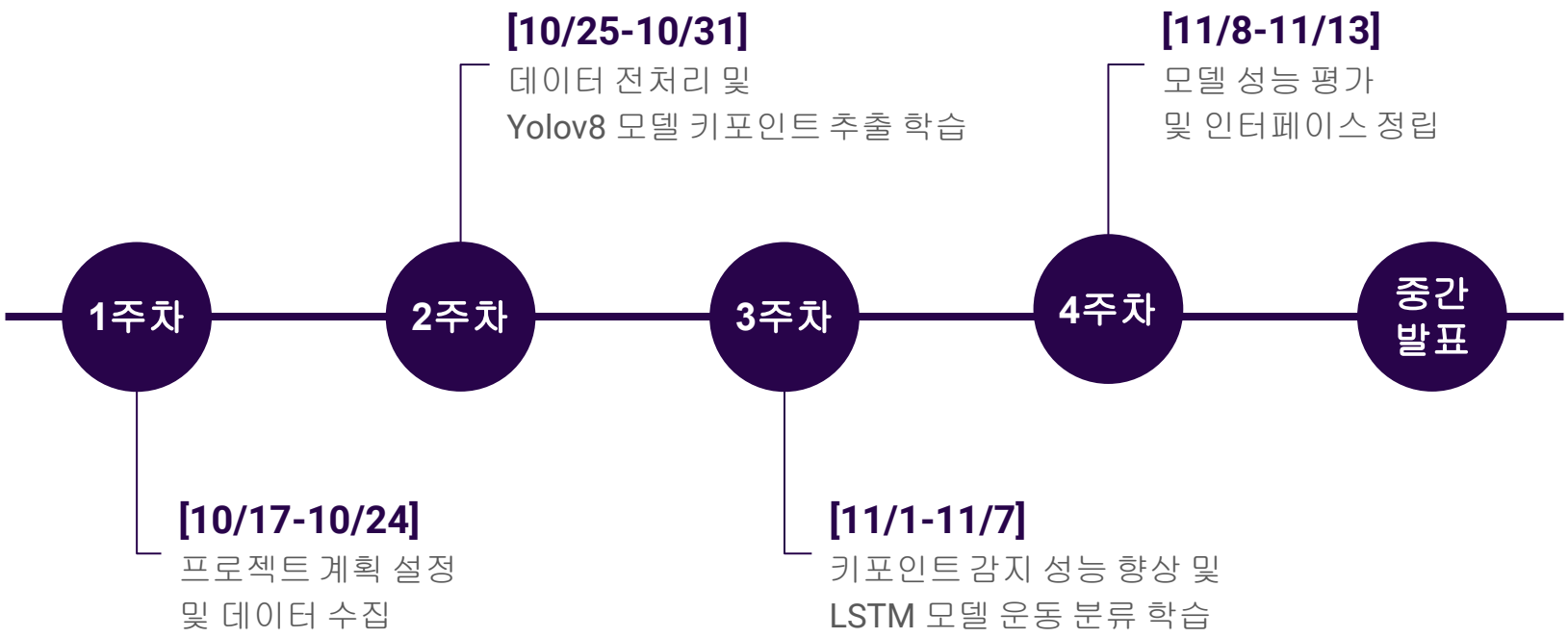
- UI/UX 디자인 및 개발
- Gradio 모델 서빙 및 프로토타입 개발
- 데이터 라벨링 작업

## 김은교

### 통합 시스템 엔지니어링

- 전체 시스템 아키텍처 설계 및 통합
- Notion, Github 및 결과물 통합 관리
- 데이터 라벨링 작업

# 프로젝트 일정


[10/25-10/31]
데이터 전처리 및
Yolov8 모델 키포인트 추출 학습
[11/8-11/13]
모델 성능 평가
및 인터페이스 정립
1주차
2주차
3주차
4주차
중간
발표
[10/17-10/24]
프로젝트 계획 설정
및 데이터 수집
[11/1-11/7]
키포인트 감지 성능 향상 및
LSTM 모델 운동 분류 학습

# 데이터셋

## AI HUB의 피트니스 자세 이미지 데이터 소스

- 다양한 자세와 체형을 가진 사람들의 홈트레이닝 이미지
- 구축 목적:
  - 국가 차원의 인공지능 경쟁력 강화를 위한 피트니스 데이터 구축.
  - 5G 기반 증강현실 컨텐츠 및 원격치료 서비스 개발에 기여.
- 데이터의 활용:
  - 본 프로젝트에서는 실시간 운동 자세 평가 및 개선 제안을 목표로 함.
  - 데이터의 다양성을 통해 모델의 정확도와 범용성 향상에 기여.

AI Hub


구축 내용 및 제공 데이터량

| 동작 | 수집피사체 | 촬영 Clip | 기본 정보 | 주요 특징 |
|---|---|---|---|---|
| 30종의 동작(5개의 운동상태) | 성별, 체형, 키크기 등 다양한 형태 | 200,000 만 Clip (5개 Multiview로 40,000만 Clip수집) | COCO 17 Skeleton keypoint | 정자세, 오류자세를 구분하여 취득 |

# 데이터셋

## 데이터 주요 특징

- 전신에 걸친 24개의 키포인트
- 5개의 다른 각도로 촬영된 운동 이미지
- 각 운동에 대한 바른 자세 데이터와 올바르지 않은 자세 데이터 존재
    - 본 프로젝트에서는 정자세 데이터 학습

HumanFit human keypoint set


A 우측후면


B 우측측면


C 정면


D 좌측측면


E 좌측후면


```
"type": "377",
"type_info": {
    "key": "377",
    "type": "바벨/덤벨",
    "pose": "선 자세",
    "exercise": "사이드 레터럴 레이즈",
    "conditions": [
        {
            "condition": "무릎 반동 없음",
            "value": true
        },
        {
            "condition": "어깨 으쓱 없음",
            "value": true
        },
        {
            "condition": "상관과 전완의 각도 고정",
            "value": true
        },
        {
            "condition": "손목의 각도 고정",
            "value": true
        },
        {
            "condition": "상체 반동 없음",
            "value": true
        }
    ],
    "description": "1 무릎반동 없음 2 어깨 으쓱 없이 3 상관과 전완의 각도 고정 4 손목의 각도 고정 5 상체 반동없이"
}
```

# YOLOv8을 활용한 Keypoint Detection

## 모델 개요

- YOLOv8-pose: **YOLO** 기반, **객체 검출** 및 동시에 실시간 **자세 추정**을 수행하는 모델

- YOLO: **한 번의 순전파**로 이미지를 여러 그리드로 나누고,
  각 그리드 셀에서 객체의 경계 상자와 클래스를 예측합니다.

- 장점: Yolov8-pose는 객체 검출과 자세 추정을 동시에 수행하며 실시간 환경에서 효과적으로 동작합니다.(실시간 자세 추정)

# YOLOv8을 활용한 Keypoint Detection

## 모델 개요

Pose - Ultralytics YOLOv8 Docs

- Pose Estimation: 이미지의 특정 지점 위치(**키포인트**) **식별** 작업
  키포인트: **관절**, **랜드마크** 또는 기타 **독특한 특징**과 같은 개체의 다양한 부분
  (2D [x, y] 또는 3D [x, y, visible] 좌표 집합)

- **모델의 출력**: **신뢰도 점수**와 **객체의 키포인트**를 나타내는 포인트 세트
  한 장면에서 개체의 특정 부분에서 서로 위치 관계를 식별해야 할 때 유용

# YOLOv8을 활용한 Keypoint Detection

**Keypoint detection 과정**

- 입력 이미지를 정사각형 이미지로 크기를 조정하고, 정규화를 통해 픽셀 값을 [0, 1] 범위로 조정합니다.

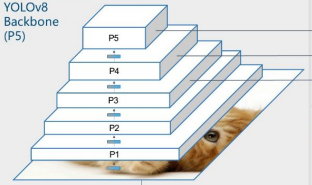
- 이미지 특징 추출 > PANet(Pyramid Attention Network)로 특징 피라미드 구성 > 정보를 효과적으로 전파 > 신뢰도(confidence)와 좌표 정보(x, y)를 예측

# YOLOv8

RangeKing

## Backbone

YOLOv8 Backbone (P5)




640×640×3

Conv k=3, s=2, p=1 — 0 — P1

320×320×64×w

Conv k=3, s=2, p=1 — 1 — P2

160×160×128×w

C2f shortcut=True, n=3×d — 2

160×160×128×w

Conv k=3, s=2, p=1 — 3 — P3

80×80×256×w

C2f shortcut=True, n=6×d — 4 — 80×80×256×w Stride=8

80×80×256×w

Conv k=3, s=2, p=1 — 5 — P4

40×40×512×w

C2f shortcut=True, n=6×d — 6 — 40×40×512×w Stride=16

40×40×512×w

Conv k=3, s=2, p=1 — 7 — P5

20×20×512×w×r

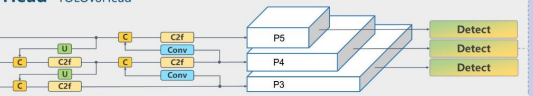
C2f shortcut=True, n=3×d — 8

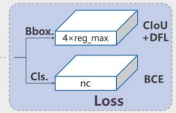
20×20×512×w×r

SPPF — 9 — 20×20×512×w×r Stride=32

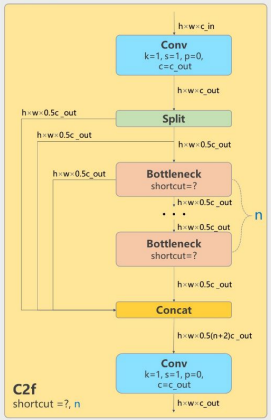
Note: height×width×channel

## Head YOLOv8Head

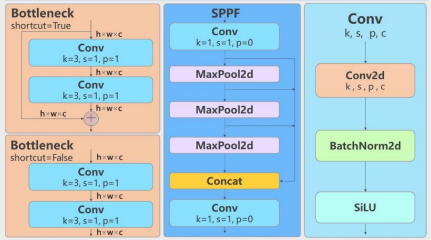


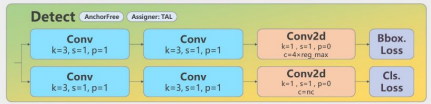




## Details




| model | d (depth_multiple) | w (width_multiple) | r (ratio) |
|---|---|---|---|
| n | 0.33 | 0.25 | 2.0 |
| s | 0.33 | 0.50 | 2.0 |
| m | 0.67 | 0.75 | 1.5 |
| l | 1.00 | 1.00 | 1.0 |
| x | 1.00 | 1.25 | 1.0 |




C2f shortcut=False, n=3×d — 15 — P3 — 80×80×256×w — Detect

80×80×768×w

Concat — 14

80×80×512×w

Upsample — 13

40×40×512×w

C2f shortcut=False, n=3×d — 12 — 40×40×512×w

40×40×512×w×(1+r)

Concat — 11

40×40×512×w×r

Upsample — 10

20×20×512×w×r

80×80×256×w

Conv k=3, s=2, p=1 — 16 — P3

40×40×256×w

Concat — 17

40×40×768×w

C2f shortcut=False, n=3×d — 18 — P4 — 40×40×512×w — Detect

40×40×512×w

Conv k=3, s=2, p=1 — 19

20×20×512×w

20×20×512×w×r

Concat — 20

20×20×512×w×(1+r)

C2f shortcut=False, n=3×d — 21 — P5 — 20×20×512×w×r — Detect

## Backbone

## Head

# YOLOv8을 활용한 Keypoint Detection

fitness_train.py × | fitness.yaml | tuner.py | augment.py | yolo_lstm.py | fitness_test.py | wel

```python
from ultralytics import YOLO

if __name__ == "__main__":
    model = YOLO('yolov8m-pose.pt')

    model.train(data="./fitness.yaml", epochs=10, batch=32, optimizer='AdamW', lrf=0.001, lr0=0.001,
                degrees=3, perspective=0.0001)
```

fitness_train.py | fitness.yaml × | tuner.py | augment.py | yolo_lstm.py | fitness_test.py | w

```yaml
train: ../fitness/train/images/
val: ../fitness/valid/images/

# Keypoints
kpt_shape: [24, 3]  # number of keypoints, number of dims (2 for x,y or 3 for x,y,visible)
flip_idx: [0, 2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11, 14, 13, 16, 15, 17, 19, 18, 20, 21, 23, 22]

# Class 'person'
# Classes 'burpees', 'cross_lunge', 'barbell_squat', 'side_lateral_raise', 'push_up', 'pull_up'
names:
  0: person
#  0 : burpees
#  1 : cross_lunge
#  2 : barbell_squat
#  3 : side_lateral_raise
#  4 : push_up
#  5 : pull_up
```

# YOLOv8을 활용한 Keypoint Detection

```python
# 키포인트 정보를 상대 좌표로 변환하고 텍스트 형식으로 저장합니다.
output_text = ''
xs = []
ys = []
for keypoint in pts.values():
    x, y = keypoint['x'], keypoint['y']
    xs.append(x)
    ys.append(y)
    # 상대 좌표로 변환합니다.
    relative_x = x / image_width
    relative_y = y / image_height
    # 텍스트 형식으로 저장합니다.
    output_text += f' {relative_x} {relative_y} 2'
w = (max(xs) - min(xs)) / image_width
h = (max(ys) - min(ys)) / image_height
cx = min(xs) / image_width + w/2
cy = min(ys) / image_height + h/2
output_text = f'0 {cx} {cy} {w} {h}' + output_text


# 결과를 텍스트 파일로 저장합니다.
with open(labels_dst, 'w', encoding='utf-8') as f:
    f.write(output_text)
```

```
boxes

cls: tensor([0.], device='cuda:0')  # class
conf: tensor([0.7611], device='cuda:0')  # confidence
data: tensor([[154.0000, 184.0000, 504.0000, 346.0000,   0.7611,   0.0000]], device='cuda:0')  # xyxy, conf, cls
id: None
is_track: False
orig_shape: (405, 720)
shape: torch.Size([1, 6])
xywh: tensor([[329., 265., 350., 162.]], device='cuda:0')   # cx, cy, w, h
xywhn: tensor([[0.4569, 0.6543, 0.4861, 0.4000]], device='cuda:0')  # n : 정규화
xyxy: tensor([[154., 184., 504., 346.]], device='cuda:0')   # x1, y1, x2, y2
xyxyn: tensor([[0.2139, 0.4543, 0.7000, 0.8543]], device='cuda:0')

----------------------------
keypoints

conf: tensor([[0.9994, ... ... 0.9988]], device='cuda:0')
data: tensor([[[498.2198, 206.8159,   0.9994],
                    x좌표, y좌표, conf
                       ... ... ...
               [154.1659, 305.3474,   0.9988]]], device='cuda:0')
has_visible: True
orig_shape: (405, 720)
shape: torch.Size([1, 24, 3])
xy: tensor([[[498.2198, 206.8159],
                   ... ...
               [154.1659, 305.3474]]], device='cuda:0')
xyn: tensor([[[0.6920, 0.5107],
                  ... ...
               [0.2141, 0.7539]]], device='cuda:0')
```

# YOLOv8을 활용한 Keypoint Detection

## Keypoint detection 과정

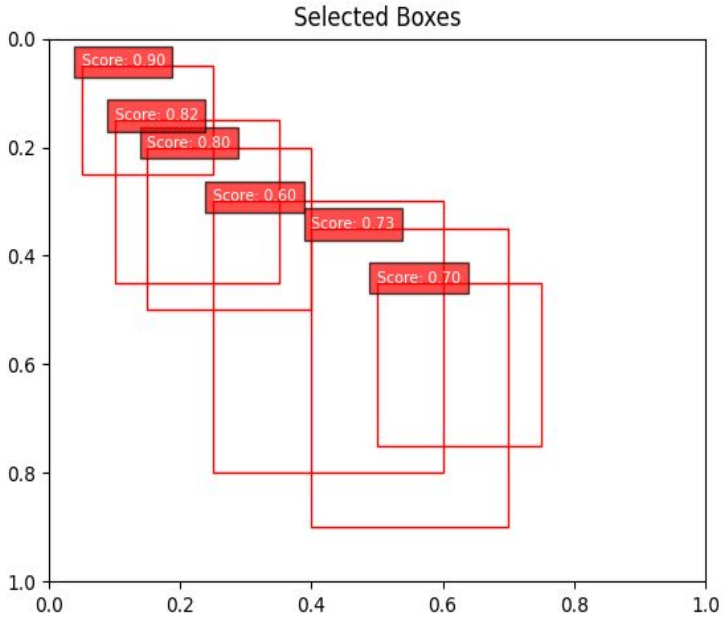
- 겹침 제거(Non-Maximum Suppression, NMS)을 사용하여 겹치는 예측 결과를 정리하고, 가장 확률이 높은 키포인트를 선택

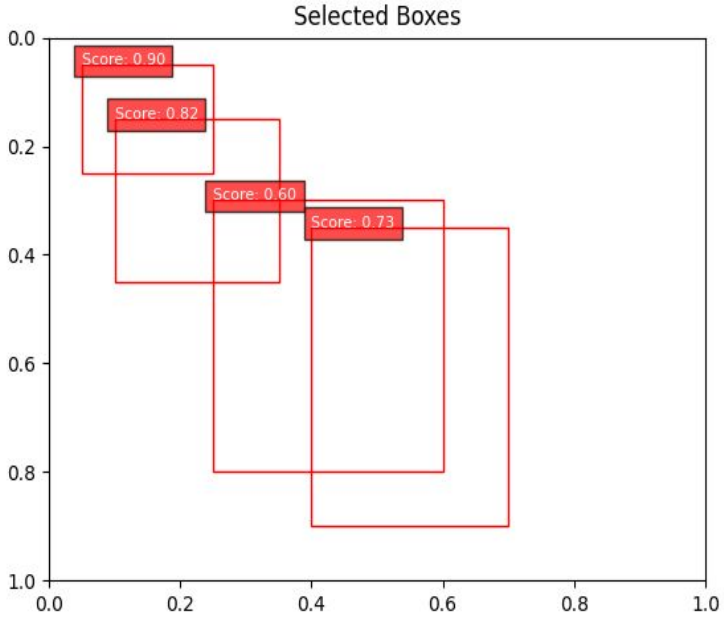
- 키포인트는 정규화된 상대 좌표로 나오기 때문에, 이를 절대 좌표로 변환하고 이미지 크기에 맞게 조정

# YOLOv8을 활용한 Keypoint Detection

## Keypoint detection 과정

- NMS



# YOLOv8을 활용한 Keypoint Detection

**Keypoint detection 과정**

- 6가지 운동에 대한 분류 모델 - 문제점

- 1. 장기 의존성(Long-Term Dependencies) 문제
  프레임 간의 자세 연속성을 학습하지 못한 경우, 연속적인 동작을 구분하기 어려움
  시퀀스 묶음으로 입력이 필요함 >> LSTM 모델 활용

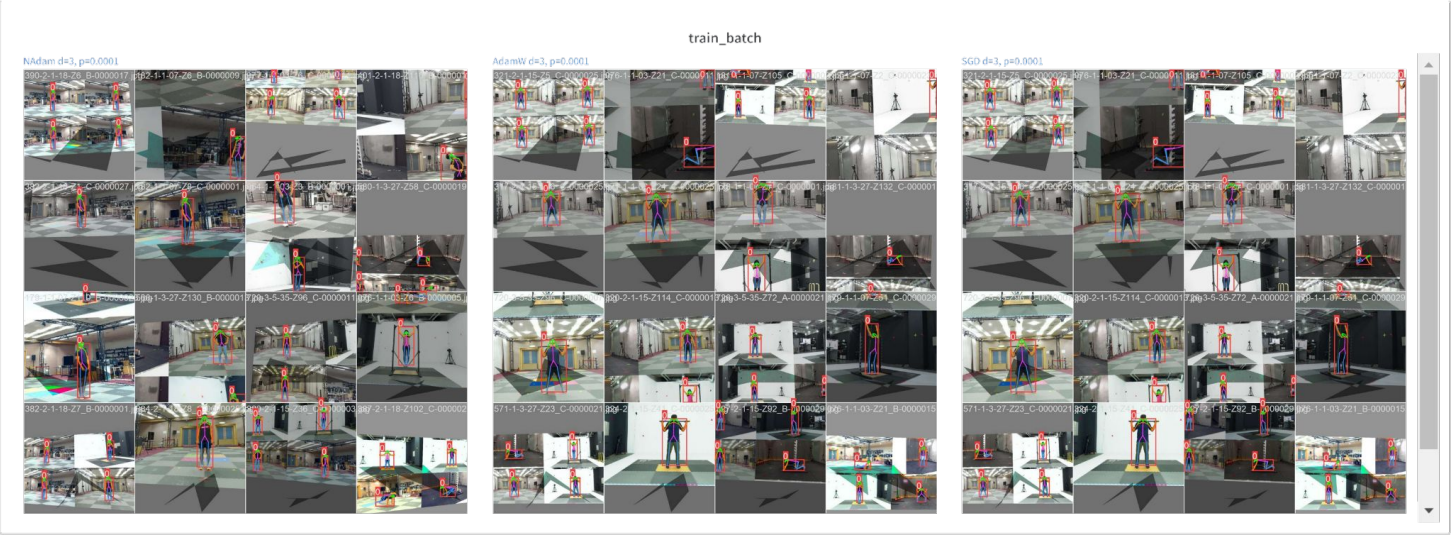
- 2. 연속적인 동작에 대한 정확한 라벨링이 어려움
  운동마다, 구분 동작마다 라벨링을 위한 충분한 데이터 확보의 어려움
  다른 운동이더라도 한 프레임 기준 인간도 판단 내리기 어려움 비슷한 자세 존재
  충분한 데이터 확보 시 학습 소요 시간 증가 문제 발생

- >>> yolo모델을 키포인트 추출, lstm모델을 동작 구분에 활용

# YOLOv8을 활용한 Keypoint Detection

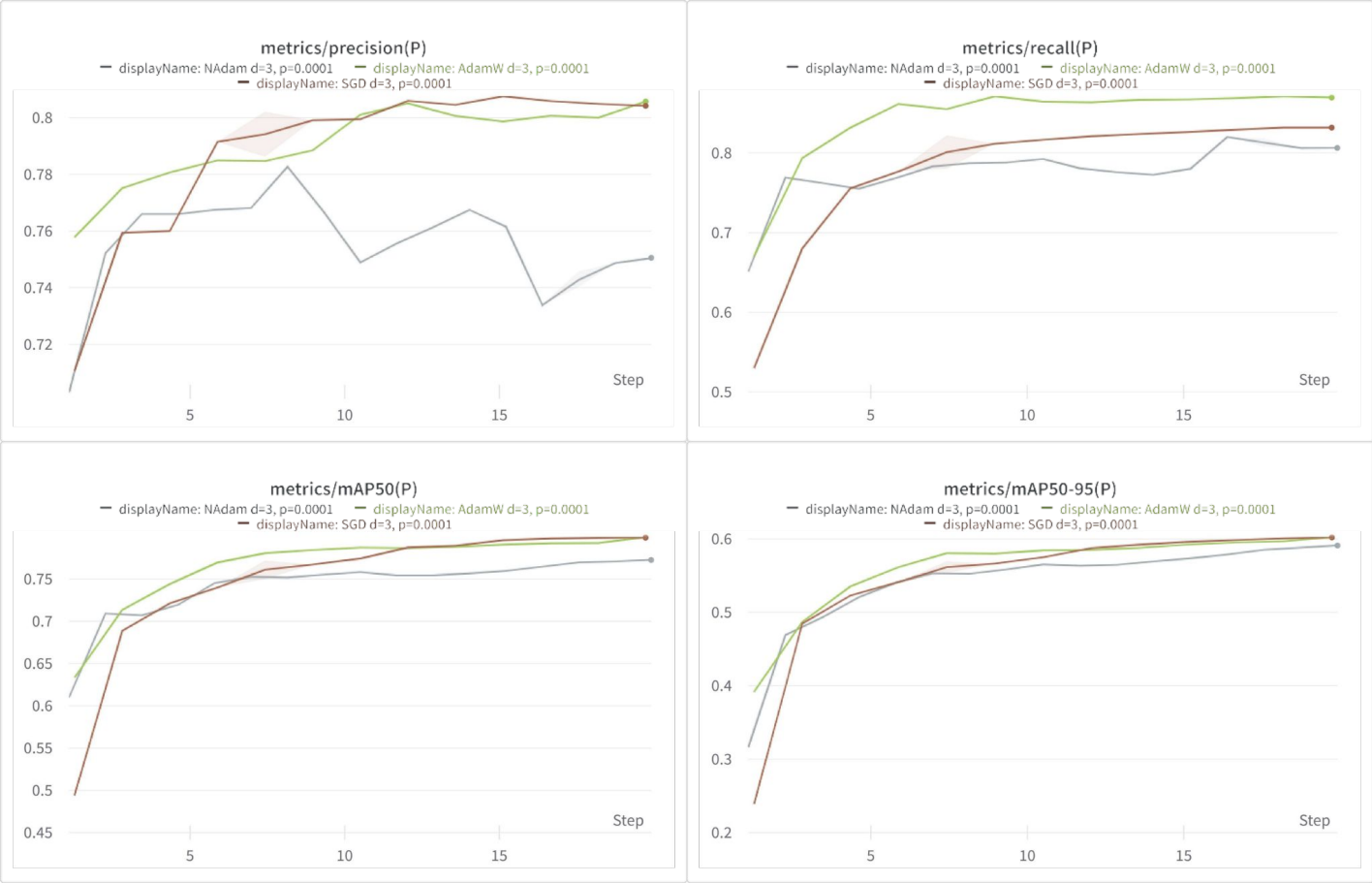
## Yolov8 Augment

- 모자이크 이미지(4장 또는 9장의 이미지를 하나의 모자이크 이미지로 결합)
- 좌우 Flip(p=0.5), 회전(degrees=3), Perspective(0.0001)



# YOLOv8을 활용한 Keypoint Detection

## 평가 매트릭스



# LSTM을 통한 운동 패턴 인식 및 분류

## Action Recognition 방법론 선택


각 프레임을 독립적으로 처리

그럼 연속적인 프레임에서 운동의 동작 패턴을 포착하는 최적의 방법은?

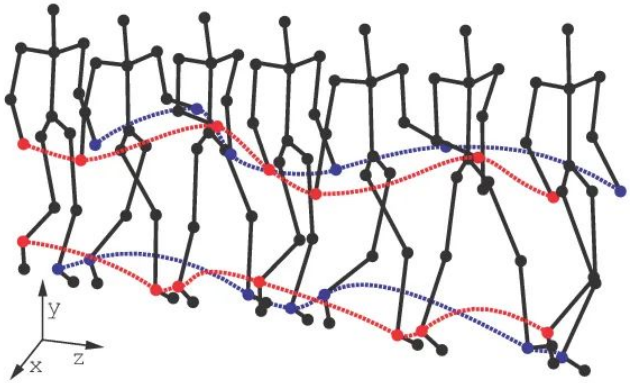
**Keypoint 데이터 활용**

**VS.**

이미지 데이터 활용

# LSTM을 통한 운동 패턴 인식 및 분류

## Keypoint 데이터를 활용한 LSTM



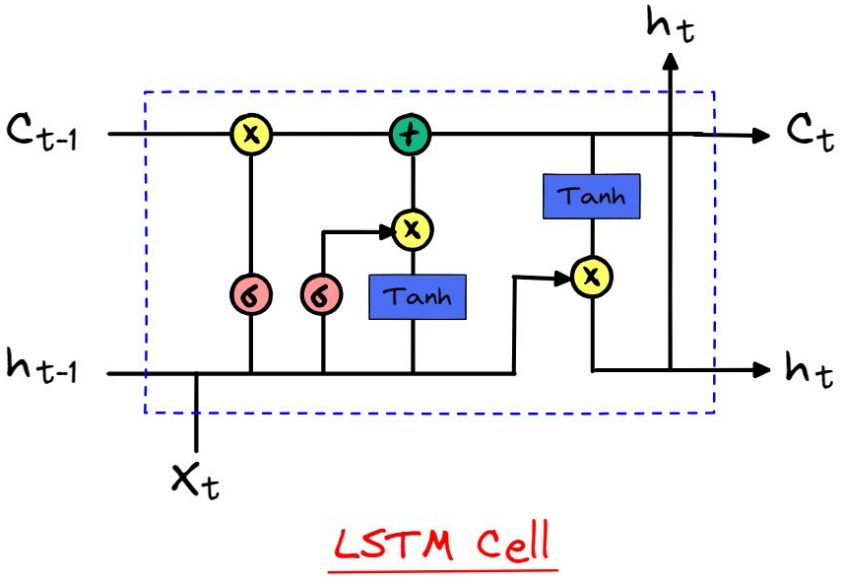

시간 경과에 따른 관절의 부드러운 움직임을
보여주는 모션 캡처 데이터

**제한된 데이터셋**과 **다양한 환경**에서의 운동 분류 문제에 대한 효과적인 성능을 **실시간 환경**에서 기대하기에 keypoint 데이터를 활용한 LSTM이 낫다고 판단.

- **데이터셋 한계 극복**
- **검증된 객체 탐지 및 포즈 추정 성능**
- **계산효율성 고려**

# LSTM을 통한 운동 패턴 인식 및 분류

## LSTM?




- RNN의 장기 의존성 문제(long-term dependencies)를 해결하기 위해서 나온 모델
- 시계열 데이터를 처리하는 데 사용
- 셀(cell)과 게이트(gate)로 구성됨
- 장점
  i. 장기 의존성 문제 해결 가능
  ii. 다양한 시퀀스 데이터 처리 가능
- 단점
  i. 계산 비용 비교적 높음
  ii. 모델 구조가 복잡함

# LSTM을 통한 운동 패턴 인식 및 분류

## 데이터 전처리(1) - JSON 데이터 처리 및 정규화


[384 rows x 1 column]

| | pose_id | x-axis | y-axis |
|---|---|---|---|
| 0 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.499479 | 0.315741 |
| 1 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.505729 | 0.306481 |
| 2 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.492708 | 0.306481 |
| 3 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.514583 | 0.309259 |
| 4 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.481771 | 0.312963 |
| .. | ... | ... | ... |
| 379 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.000000 | 0.000000 |
| 380 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.000000 | 0.000000 |
| 381 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.000000 | 0.000000 |
| 382 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.000000 | 0.000000 |
| 383 | /content/drive/MyDrive/exercise_data/json_v2/s... | 0.000000 | 0.000000 |

Data Extraction + Normalization

# LSTM을 통한 운동 패턴 인식 및 분류

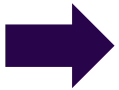
## 데이터 전처리(2) - 패딩 처리 및 시퀀스 추출

**Padding 처리**

- 처리 과정: 필요한 프레임 수(16)에 도달하지 않은 경우, 값을 추가하여 패딩함
- 예시: 데이터 프레임의 길이가 필요한 길이에 미치지 못할 경우, 나머지 부분은 0으로 채움

**Sequence 추출**

- 데이터 프레임에서 x축, y축 데이터만 추출하여 (16, 48) 크기의 시퀀스로 재구성
- NumPy 배열로 변환



```
Total sequences shape: (1, 16, 48)

[[0.49947917 0.31574074 0.50572917 0.30648148 0.49270833 0.30648148
  0.51458333 0.30925926 0.48177083 0.31296296 0.53125    0.37962963
  0.46979167 0.38148148 0.5609375  0.43981481 0.44270833 0.44907407
  0.57291667 0.52222222 0.43125    0.5287037  0.51822917 0.53148148
  0.48020833 0.53055556 0.53645833 0.63703704 0.46510417 0.63518519
  0.53489583 0.75555556 0.46458333 0.75740741 0.4994791  0.35
  0.57395833 0.53796296 0.43072917 0.54444444 0.4994791  0.42592593
  0.4994791  0.48240741 0.54010417 0.77777778 0.4578125  0.77777778]
 [0.4984375  0.30462963 0.50364583 0.29444444 0.49166667 0.2962963
  0.5140625  0.30185185 0.48125    0.30740741 0.53020833 0.37407407
  0.46875    0.37222222 0.56510417 0.42314815 0.4328125  0.42222222
  0.59375    0.48703704 0.40260417 0.475      0.51822917 0.52777778
  0.4796875  0.52777778 0.5359375  0.63611111 0.465625   0.63333333

...

  0.53489583 0.75648148 0.46458333 0.75740741 0.496875   0.3537037
  0.5484375  0.5537037  0.45260417 0.55648148 0.4984375  0.42685185
  0.4984375  0.48333333 0.53958333 0.77777778 0.45729167 0.77777778]
 [0.49635417 0.29537037 0.5015625  0.28425926 0.48958333 0.28611111
  0.51145833 0.28981481 0.4796875  0.29259259 0.52708333 0.35092593
  0.46822917 0.35648148 0.534375   0.43518519 0.46197917 0.44259259
  0.52708333 0.51851852 0.46770833 0.5212963  0.5171875  0.52685185
  0.4796875  0.52685185 0.5328125  0.63611111 0.46614583 0.63425926
  0.534375   0.75740741 0.46458333 0.75740741 0.49635417 0.3287037
  0.52552083 0.53518519 0.46927083 0.53981481 0.496875   0.40925926
  0.496875   0.46666667 0.5390625  0.77685185 0.4578125  0.77685185]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.        ]]
```

# LSTM을 통한 운동 패턴 인식 및 분류

## 데이터 전처리(3) - 데이터 결합, 레이블 데이터 준비, 원-핫 인코딩

**데이터 결합**

- 개별 시퀀스 데이터를 하나의 큰 데이터 세트로 결합
- -> 모든 시퀀스 데이터를 포함하는 `(253, 16, 48)` 크기의 배열 생성

**레이블 데이터 준비**

- 분류할 운동의 종류에 따라 각 시퀀스에 레이블 할당
- -> 각 시퀀스에 대한 레이블을 포함하는 배열 생성

**원-핫 인코딩**

- 레이블을 수치적 형태로 인코딩함
- 예시: '운동 A'는 `[1, 0, 0, 0, 0, 0]`, '운동 B'는 `[0, 1, 0, 0, 0, 0]` 등으로 변환됨
- -> 모든 레이블을 포함하는 `(253, 6)` 크기의 배열 생성

# LSTM을 통한 운동 패턴 인식 및 분류

## 모델 훈련에 사용된 데이터셋 구조

### 데이터셋 구조

- 데이터 형태: 16개의 프레임에 대한 x, y축 값 각각 24개
- 입력 데이터 차원: `X.shape = (253, 16, 48)`
    a. 253개 샘플, 각각 16개 프레임, 프레임당 48개 특징
- 출력 레이블 차원: `y_encoded.shape = (253, 6)`
    a. 253개 샘플, 6개 운동 분류

### 학습 및 검증 데이터셋 분할

- 분할 비율: 8:2

# LSTM을 통한 운동 패턴 인식 및 분류

## 모델 학습

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_16 (LSTM)              (None, 16, 64)            28928

 layer_normalization_8 (Lay  (None, 16, 64)            32
 erNormalization)

 lstm_17 (LSTM)              (None, 16, 128)           98816

 lstm_18 (LSTM)              (None, 16, 128)           131584

 layer_normalization_9 (Lay  (None, 16, 128)           32
 erNormalization)

 lstm_19 (LSTM)              (None, 64)                49408

 dense_12 (Dense)            (None, 64)                4160

 dense_13 (Dense)            (None, 32)                2080

 dense_14 (Dense)            (None, 6)                 198

=================================================================
Total params: 315238 (1.20 MB)
Trainable params: 315238 (1.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

**Run history:**


**Run summary:**

| | |
|---|---|
| epoch/categorical_accuracy | 0.9802 |
| epoch/epoch | 19 |
| epoch/learning_rate | 0.001 |
| epoch/loss | 0.07741 |
| epoch/val_categorical_accuracy | 1.0 |
| epoch/val_loss | 0.01677 |

# LSTM을 통한 운동 패턴 인식 및 분류

## 모델 추론

Make Predictions

```
[ ]  res = model.predict(X_test)

     2/2 [==============================] - 1s 27ms/step
```

```
[ ] # Actions that we try to detect
    actions = np.array(['burpee_test', 'pull_up', 'cross_lunge', 'side_lateral_raise', 'barbell_squat', 'push_up'])
```

```
[ ]  np.sum(res[0])

     0.99999994
```

```
[ ]  res.shape

     (51, 6)
```

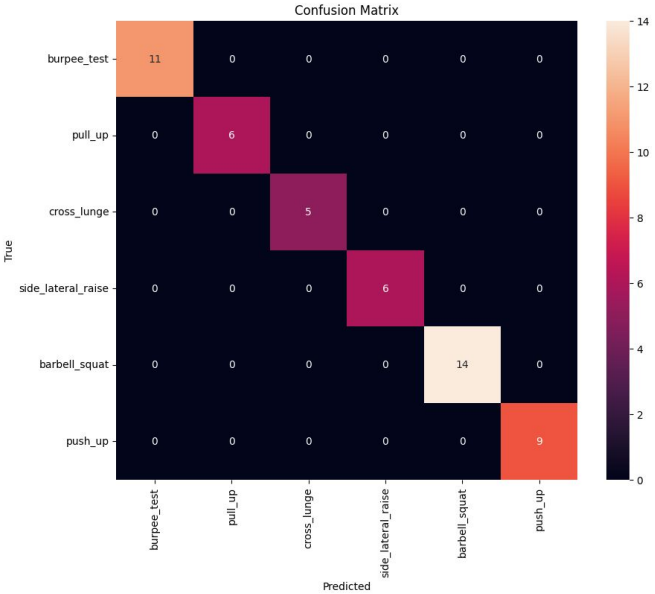
```
[ ]  np.argmax(res[0])

     4
```

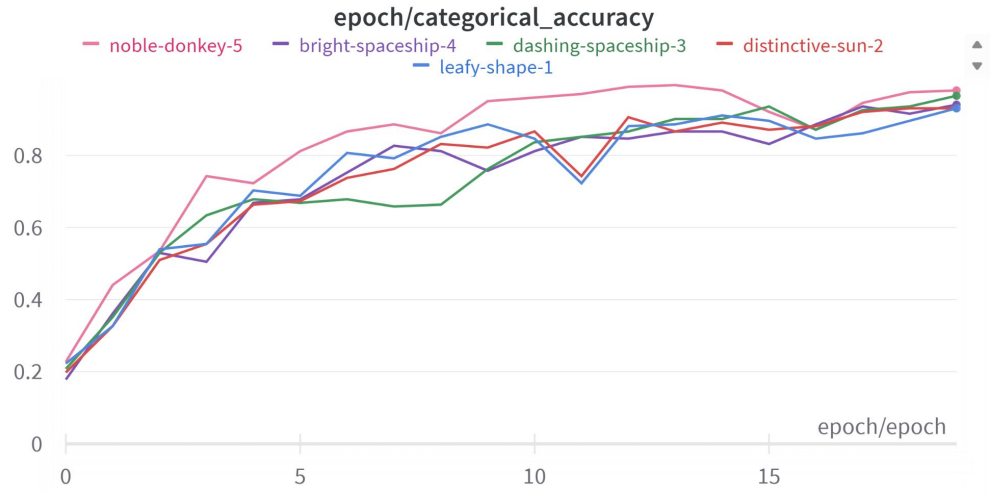
```
[ ]  actions[np.argmax(res[9])]

     'burpee_test'
```

```
[ ]  actions[np.argmax(y_test[9])]

     'burpee_test'
```

# LSTM을 통한 운동 패턴 인식 및 분류

## 평가 매트릭스와 벤치마킹 (1)



# LSTM을 통한 운동 패턴 인식 및 분류

## 평가 매트릭스와 벤치마킹 (2)

**과적합 발생.** 실제 다양한 조건과 시나리오에서는 모델의 일반화 능력이 떨어짐.

| 운동 유형 | 정확도 (실시간) | 정확도 (테스트셋) |
|---|---|---|
| 푸쉬업 | 90% | 100% |
| 크로스 런지 | 70% | 100% |
| 풀업 | 40% | 100% |
| 버피테스트 | 89% | 100% |
| 바벨 스쿼트 | 89% | 100% |
| 사이드 레터럴 레이즈 | 89% | 100% |

- 프레임 전처리의 일관성 부족.
  - 일관성을 유지하려면 운동별로 LSTM 학습시켜야함
- 실시간 처리의 복잡성.
- 데이터 품질과 다양성 부족.

# 운동 상태 평가 방법

## 상태 평가 및 횟수 측정 모듈 작성

- 기본적인 구분 동작 함수 구현
- 한 시퀀스 단위로 이전값과 비교하여 분산 값을 활용
- 데이터셋 json을 활용하여, 통계자료를 구축함으로써 기본적인 임계값(threshold)으로 설정
- Webcam으로 테스트 하면서 임계값(threshold)을 미세 조정

- 기준 상태 도달 유무 또는 기준 범위 이탈을 기준
- 운동 상태 구분의 성능 향상을 위해 오차범위 설정
- 정규화된 임계값(threshold) 설정

- LSTM에 입력하는 시퀀스 단위로 운동 상태 평가
- 운동별 카운트 함수 구현

# 운동 상태 평가 방법

```python
# 정규화된 키포인트
pt = np.random.uniform
data = np.array([[np.squeeze(np.array([[pt(0, 1), pt(0, 1)] for _ in range(24)])) for _ in range(16)]])
data = data.tolist()

# cos 법칙으로 각 ABC를 °(도) 단위로 구하는 함수
def cal_angle(A, B, C):
    if A == B or C == B:
        return 180
    A, B, C = map(np.array, (A, B, C))
    angle = degrees(arccos(min(max(dot(A - B, C - B) / (norm(B - A) * norm(C - B)), -1.0), 1.0)))
    return angle


# 두 점 사이의 거리를 구하는 함수
def cal_distance(A, B):
    distance = ((A[0] - B[0]) ** 2 + (A[1] - B[1]) ** 2) ** 0.5
    return distance


# 과거와 현재 값 비교해서 얼마나 차이가 나는 지 확인하는 코드
def past_current(past, current, error_message: set, message: str, threshold: float, mode=False):
    if past:
        if mode:
            if abs(current - past) < threshold:
                error_message.add(message)
        elif abs(current - past) > threshold:
            error_message.add(message)
    return current
```

# 운동 상태 평가 방법

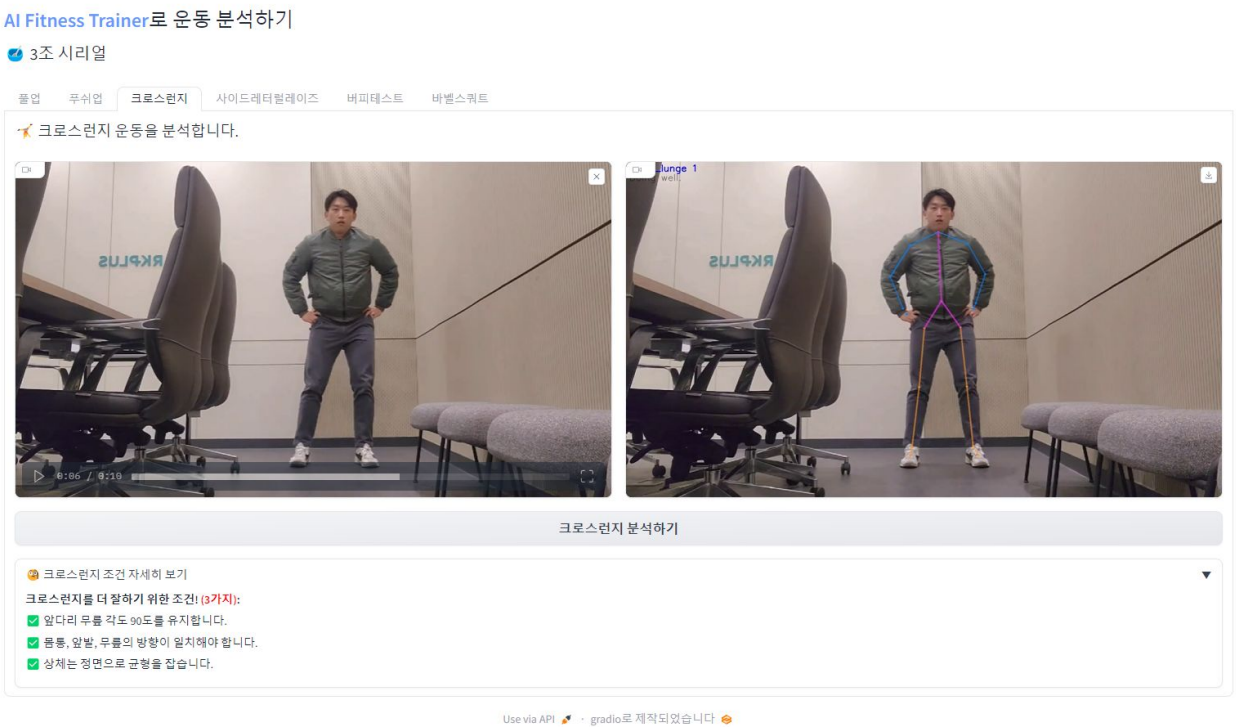
```
        Foot_L, Foot_R = pts[22], pts[23]

        # 눈-코의 중점과 귀의 중점의 y값(높이) 비교
        if ((Eye_L[1] + Eye_R[1]) / 2 + Nose[1]) / 2 > (Ear_L[1] + Ear_R[1]) / 2 + 0.011:
            error_message.add("Don't bend your upper body too much.")  # 상체를 너무 숙이지 마세요

        # 구부린 무릎의 최소 각의 크기 저장
        if Knee_L[1] < Knee_R[1]:  # 왼발이 앞
            Knee_mL = min(Knee_mL, cal_distance(Hip_L, Knee_L))
            if not (Foot_L[0] - 0.04 < Knee_L[0] < Foot_L[0] + 0.04):
                error_message.add("Make sure your knees and legs are straight.")  # 무릎과 다리가 일자가 되게 해주세요

        else:
            Knee_mR = min(Knee_mR, cal_distance(Hip_R, Knee_R))
            if not (Foot_R[0] - 0.04 < Knee_R[0] < Foot_R[0] + 0.04):
                error_message.add("Make sure your knees and legs are straight.")

        if Back[1] - (Shoulder_L[1] + Shoulder_R[1]) / 2 < 0.05:
            error_message.add("Please face your upper body straight ahead.")  # 상체를 정면으로 향하게 해주세요

    if Knee_mL > 0.115 and Knee_mR > 0.115:
        error_message.add("Bend your knees more.")  # 무릎을 더 구부리세요

if len(error_message):
    result.append(error_message)
else:
    result.append({"Doing well."})

return result
```

# 운동 상태 평가 방법

```python
# 크로스 런지
def count_cross_lunge(pts, flag):
    Hip_L, Hip_R = pts[11], pts[12]
    Knee_L, Knee_R = pts[13], pts[14]

    # 구부린 무릎의 최소 각의 크기 저장
    if Knee_L[1] < Knee_R[1]:  # 왼발이 앞
        Knee_dL = cal_distance(Hip_L, Knee_L)
        if Knee_dL > 0.16:
            flag = True
        if Knee_dL < 0.125 and flag:
            return 1, False

    else:
        Knee_dR = cal_distance(Hip_R, Knee_R)
        if Knee_dR > 0.16:
            flag = True
        if Knee_dR < 0.125 and flag:
            return 1, False

    return 0, flag
```

# Gradio를 활용한 UI 설계와 프로토타입 제작

gradio

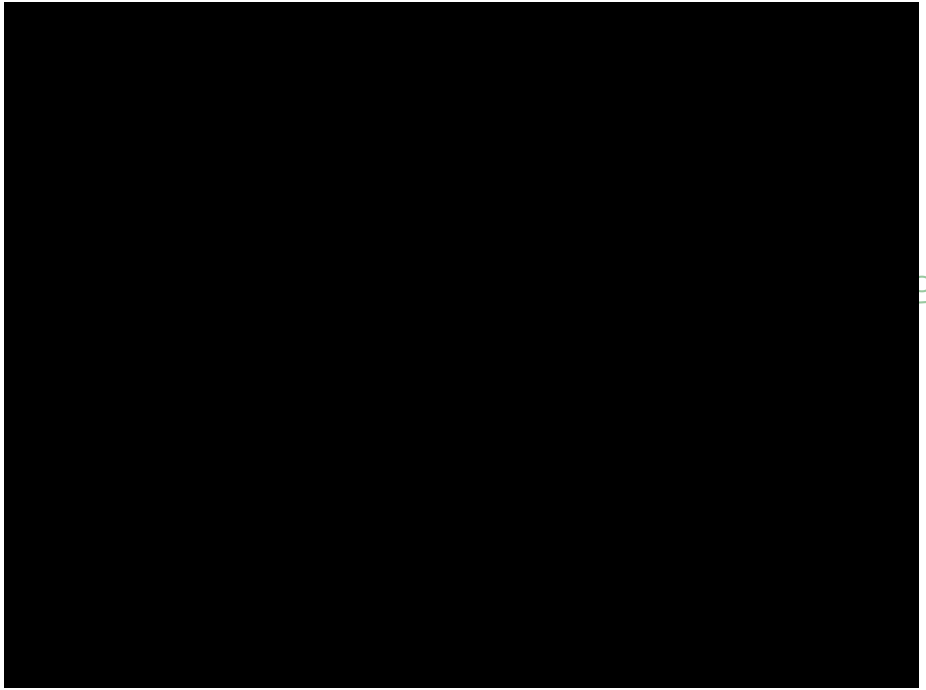


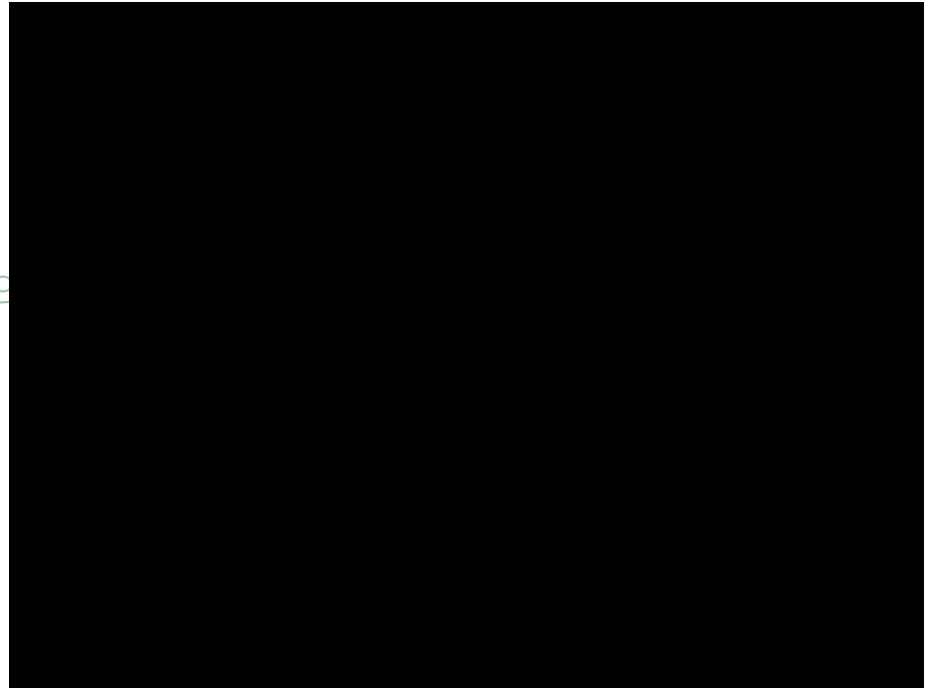
# Gradio를 활용한 UI 설계와 프로토타입 제작

1. **키 포인트 시각화 및 운동 분석**
   - YOLOv8로 키포인트 검출
   - LSTM을 사용한 운동 분류
2. **운동 평가 및 운동 횟수 측정 모듈 설계**
   - 조건문을 활용하여 구간별 운동 평가 및 횟수 측정
3. **Gradio 인터페이스 구성**
   - 6가지 운동 탭과 UI 구성
   - classify_video 함수를 통한 분석
4. **동영상 저장 및 결과 반환**
   - H.264 코덱 사용, 28fps 설정, 분석된 프레임 저장
   - 생성된 동영상 파일 경로 반환 및 Gradio UI 출력

# 결과 시연

Gradio를 통해 구현된 사용자 친화적인 인터페이스를 가진 프로토타입 제작





# 이슈사항과 해결방안

## 1. 운동분류 성능향상

- **문제 발견:**
  - 상황: AIHUB 데이터셋에 방향이 측면,사선,정면 등으로 구분되어 있음
  - 영향: 특정 방향에서 분류 인식 성능 저하
- **문제 해결을 위한 접근:**
  - 전략: 추가로 다양한 방향의 데이터셋 확보
  - 구현: CCL이 있는 동영상을 수집하여, AIHUB 데이터와 유사한 구조로 데이터셋 변경

# 이슈사항과 해결방안

## 2. LSTM을 통한 운동 분류 정확도 향상

- **문제 발견:**
  - 상황: YOLOv8을 활용한 운동 분류할 시, 유사한 준비 동작에서 잘못된 예측 발생.
  - 원인: 키포인트만을 기반으로 정적으로  운동 분류, 단일 모델로 연속 동작를 고려하지 못함.
- **문제 해결을 위한 접근:**
  - 전략 : 연속동작을 구분 할 수 있는 모델을 결합
  - 구현 : YOLOv8은 단일 클래스로 사람의 키포인트 감지 , LSTM은 키포인트의 시퀀스를 분석하여 연속 동작 인식.
- **성과:**
  - 운동 연속 동작 인식 성능 향상.
  - 운동 분류 시 오류 감소

# 이슈사항과 해결방안

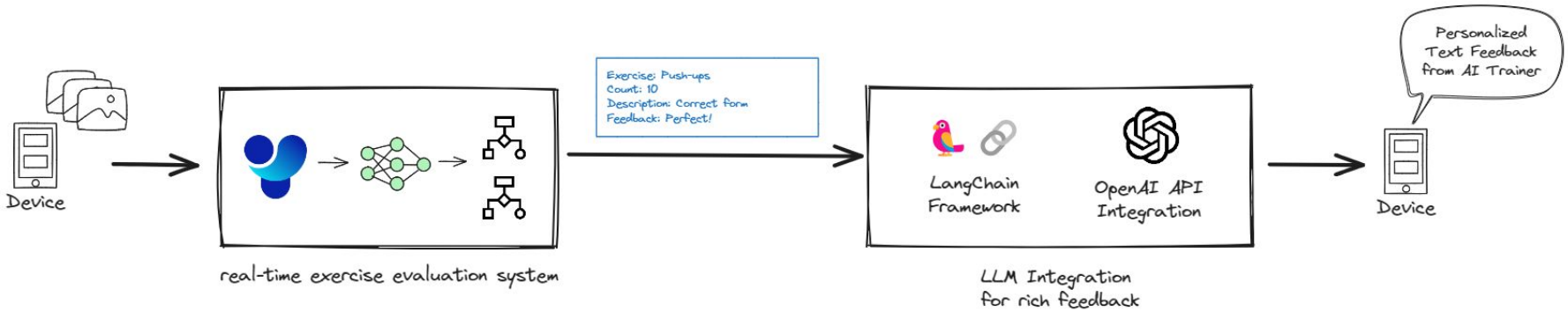
## 3. 프레임 정렬을 통한 LSTM 성능 향상

- **문제 발견:**
  - 상황: 운동별 1회당 프레임 수 차이 (3프레임 vs 9프레임) .
  - 영향: 불규칙한 시퀀스 길이가 LSTM 모델의 패턴 인식 능력 저하.
- **문제 해결을 위한 접근:**
  - 전략: 운동당 16프레임으로 시퀀스 길이 통일.
  - 구현: 빈 프레임에 대한 패딩(0) 적용으로 일관된 입력 시퀀스 길이 확보.
- **성과:**
  - LSTM 모델의 시간적 패턴 인식 능력 향상.

# 향후 도전과제

## 개인화된 AI Fitness Trainer 앱 개발(AI프로젝트)

- 과적합 해결 (모델의 일반화된 성능 향상)
  - AI HUB 데이터셋의 다양한 운동 활용
  - 다양한 환경의 데이터셋 구축
- LLM 통합 (OpenAI API)
- 실제 프로토타입 앱 형태로 배포



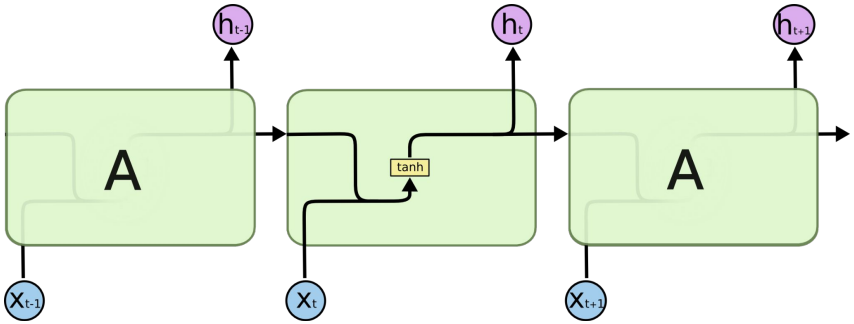
DETECTING KEYPOINTS & ACTIONS REAL-TIME

ASSESSING ACTIONS & PROVIDING FEEDBACKS

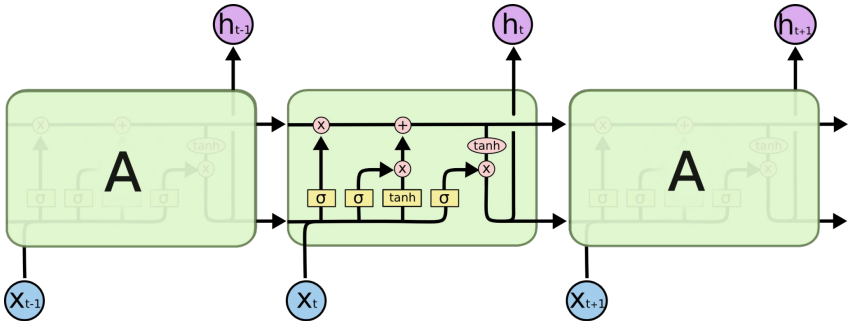
# Q&A

# Appendix

## LSTM이 궁금해요



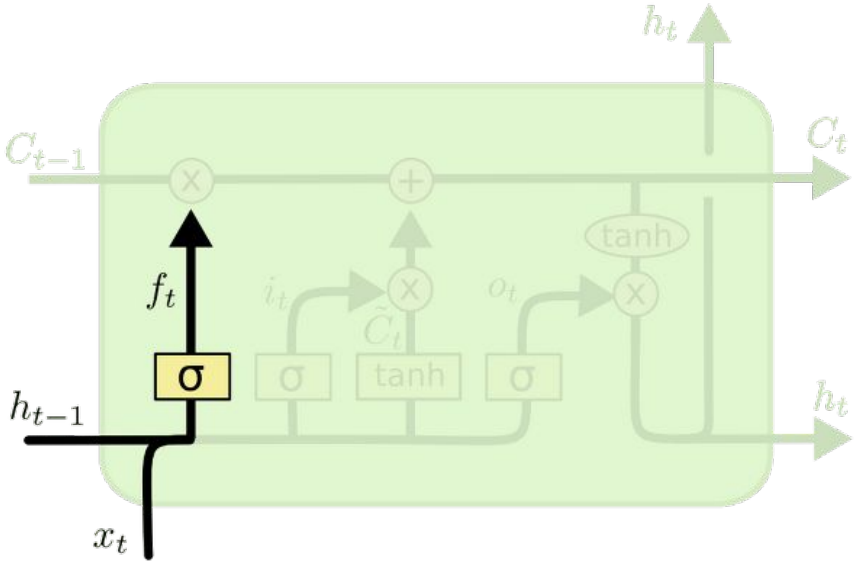

RNN 구조




LSTM 구조

# Appendix

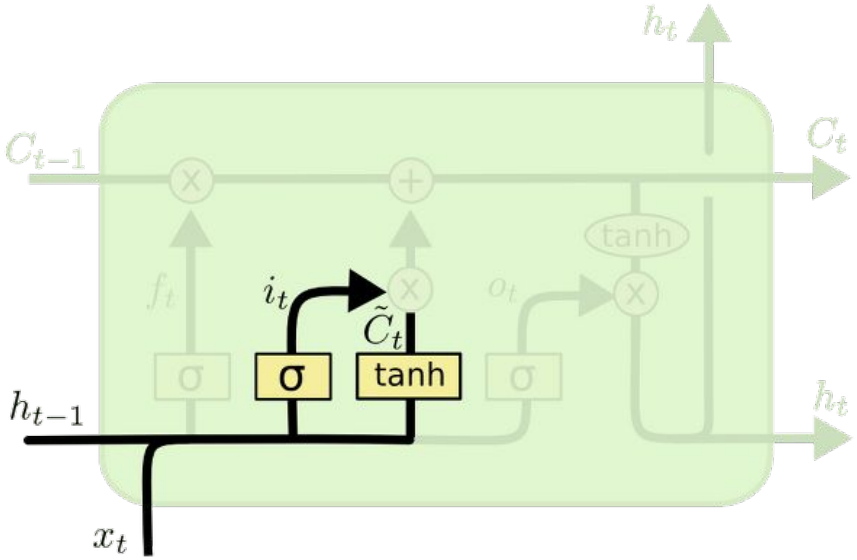
## LSTM이 궁금해요




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

# Appendix

## LSTM이 궁금해요

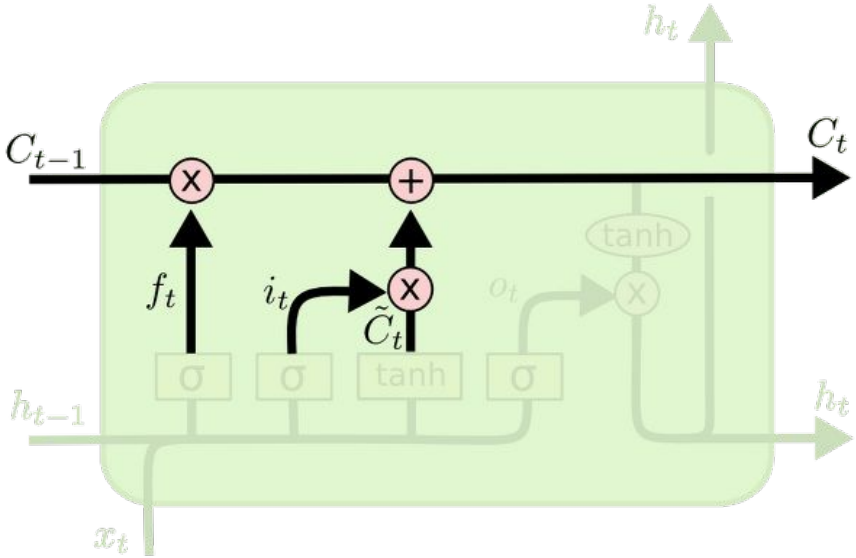



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

# Appendix

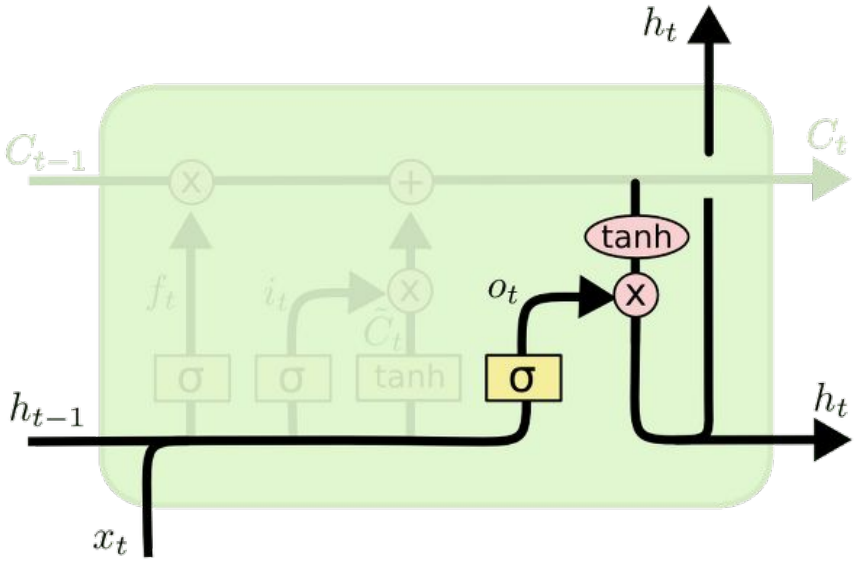
## LSTM이 궁금해요




$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

# Appendix

## LSTM이 궁금해요




$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$