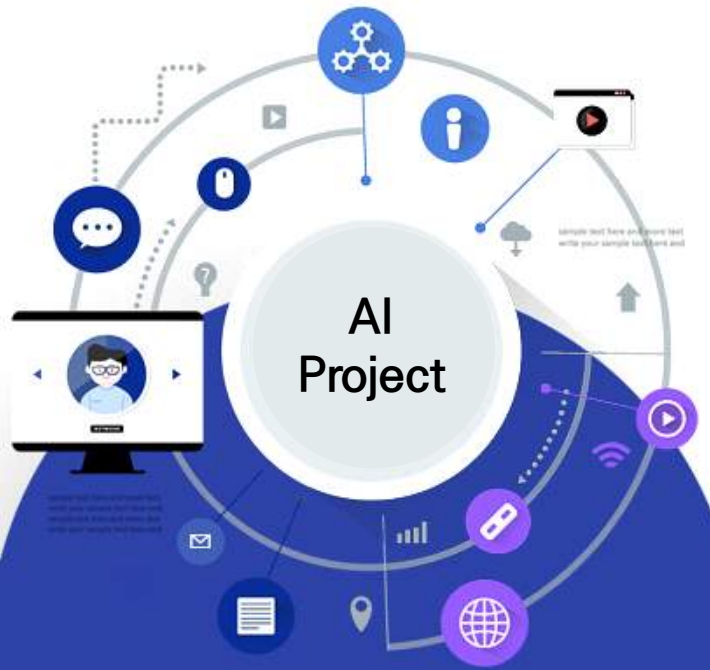
딥러닝 기반 당구 3쿠션 예측 시스템

Yolo, LSTM 기반 예측
RAG + OLLAMA

작성자 : 류건우



1. 개요 및 현황

	AI기반 3쿠션 예측모델 개발 및 시각화
추진배경 및 목적	. 득점 판정의 객관성 확보 : 당구는 육안에만 의존하기 때문에, 득점이 애매한 샷의 경우 득점의 기준이 애매해짐. . 사용자 학습 및 피드백 강화 : 자신의 샷을 복기하고 실력을 향상할 수 있는 기술적 토대 마련. . 디지털 트랜스포메이션(DX) : 기존 오프라인 중심의 당구 산업에 딥러닝 기술을 접목하여 경기 기록의 자동화 및 데이터 시각화 서비스 제공 가능

2. 과제 범위

과제구분		내용
AI	딥러닝 기반 3쿠션 판독 모델 구현	데이터 수집 및 데이터셋 구축
		데이터 전처리, 표준화, 상관관계 분석
		예측모델 선정 및 학습
		Accuracy, Loss 등 평가지표를 활용한 모델 성능 평가
		예측모델 웹기반 시스템 구축
		테스트

3. 과제 추진 방법 – AI 예측 분석모델 적용 대상

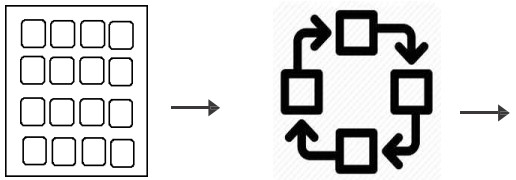
	수집데이터	예측모델인자(독립변수)	AI예측 분석 대상
시각화	• 3쿠션 경기 영상 데이터(.mp4 녹화 영상) • 영상 프레임별 객체 이미지(YOLO) • 객체 좌표 원시 데이터(X,Y)	• 위치변수 → 테이블 크기로 정규화된 X, Y 좌표 • 물리 변수 → 속도, 이동 각도, 벡터 변화량 • 시계열 변수 → 60프레임 단위의 연속된 궤적 데이터	• 경기 결과 판정 → 3쿠션 득점 성공 / 실패 여부 • 판독 신뢰도 산출 → 모델의 예측 확률값 • 궤적 시각화 → 수구의 이동 경로 및 판정 결과 오버레이

3. 과제 추진 방법 – AI 예측 분석모델 구축 프로세스

DATA IMPORTING

AUTO PREDICTION

USER INSIGHT



- 경기 영상 수집 및 객체 탐지
- mp4 경기 영상 로드 및 프레임 분할
- Yolov8 기반 당구공 객체 인식
- 프레임별 좌표 데이터 (X,Y) 추출

- 결측치 보간 및 노이즈 스무딩
- 물리 파생 변수 산출 : speed, angle, dx, dy
- 데이터 정규화 및 시퀀스(60 frame) 정렬

- 시계열 패턴 인식을 위한 LSTM
- 6채널(x, y, dx, dy, speed, angle) 입력층 구성
- 성공/실패 패턴 학습 및 최적 가중치 도출

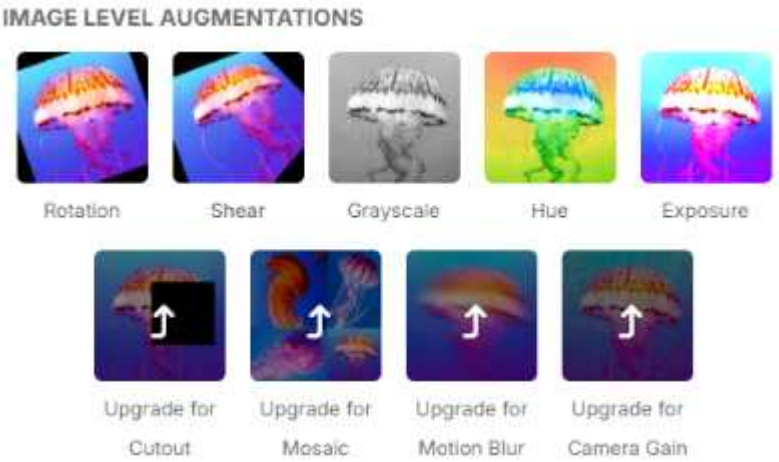
- 입력된 궤적에 대한 득점 확률 계산
- 판독 신뢰도 산출

- 경기 분석 시각화
- 원본 영상 위 판독 결과 및 궤적 오버레이
- 판정 결과 영상의 웹 렌더링

4. 주요 결과물 – 데이터셋 구축

데이터셋 구축

- 소스 : YouTube 3쿠션 경기(PBA) 영상 및 직접 촬영한 영상을 프레임 추출 (2천장)
- 라벨링 : Robflow를 활용한 당구공 바운딩 박스 처리
- 반복 학습 : 1차 라벨링 → 모델 학습 → 오인식 데이터 선별 → 2차 추가 라벨링
- 데이터 증강 : 회전, 밝기 조절 등 사용



4. 주요 결과물 – YOLOv8 객체 탐지 모델 학습

학습 모델 선정

- 속도와 정확도의 균형이 좋은 YOLOv8m (medium) 모델 채택
- 사전 학습된 모델에 당구공 데이터셋을 전이 학습

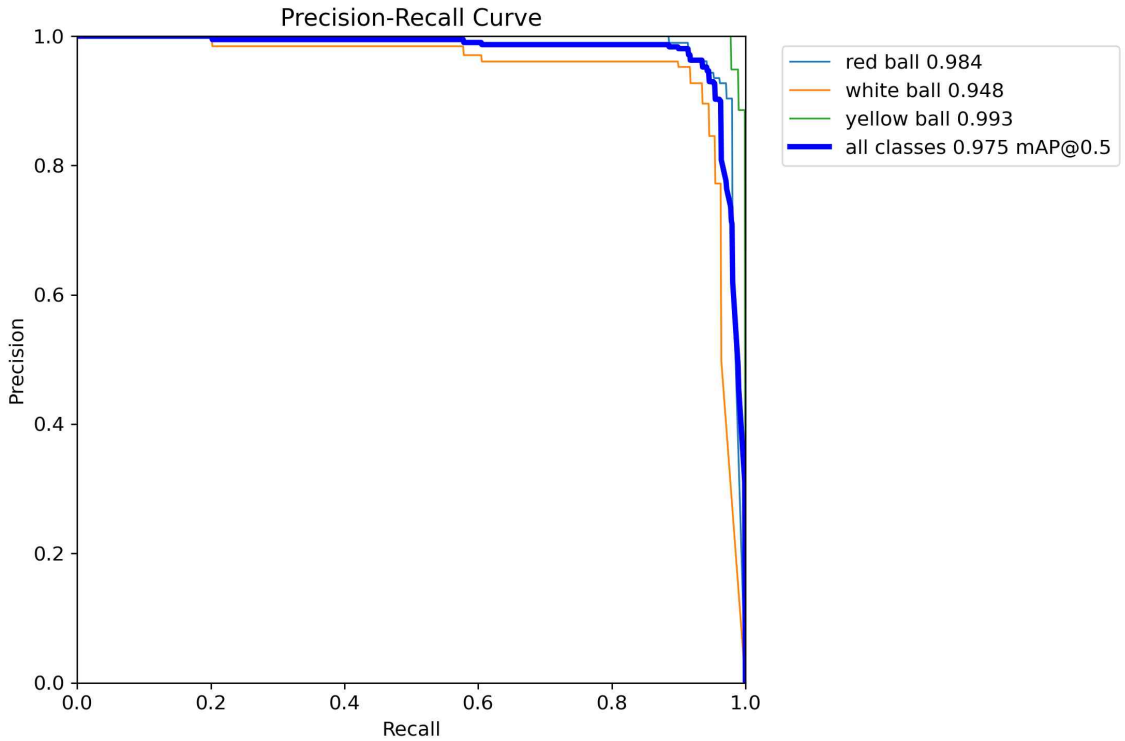
```
model = YOLO('yolov8m.pt')

yaml_path = r'C:\Users\kwr51\Downloads\--.v4i.yolov8\data.yaml'

model.train(
    data=yaml_path,
    epochs=100,
    imgsz=640,
    batch=16,
    device=0,
    project='C:/bill/runs',
    name='billiard_yolo',
    exist_ok=True
)
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     94/100      3.99G     0.7946     0.4284     0.8491         30        640: 100% ━━━━━━━━━━━━ 69/69 8.71it/s 8.0s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 7.11it/s 0.6s
                   all        100        307      0.961      0.953      0.972      0.844

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     95/100      4.01G     0.7989     0.4287     0.8414         34        640: 100% ━━━━━━━━━━━━ 69/69 8.61it/s 8.0s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 7.41it/s 0.5s
                   all        100        307       0.96      0.945      0.973      0.849

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     96/100      2.26G     0.7768     0.4258     0.8421         33        640: 100% ━━━━━━━━━━━━ 69/69 8.7it/s 8.0s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 7.01it/s 0.6s
                   all        100        307      0.957      0.949      0.973      0.843

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     97/100      2.26G     0.7812     0.4236     0.8488         31        640: 100% ━━━━━━━━━━━━ 69/69 8.71it/s 7.9s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 6.91it/s 0.6s
                   all        100        307      0.954      0.953      0.973      0.834

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100      2.26G     0.7793     0.4213     0.8424         35        640: 100% ━━━━━━━━━━━━ 69/69 8.71it/s 8.0s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 7.01it/s 0.6s
                   all        100        307      0.958      0.948      0.973      0.837

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      2.26G     0.7791      0.418     0.8401         34        640: 100% ━━━━━━━━━━━━ 69/69 8.71it/s 8.0s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 6.91it/s 0.6s
                   all        100        307      0.956      0.949      0.976      0.844

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      2.26G     0.7812     0.4228     0.8441         37        640: 100% ━━━━━━━━━━━━ 69/69 8.61it/s 8.0s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 7.11it/s 0.6s
                   all        100        307      0.955      0.945      0.975      0.852
```

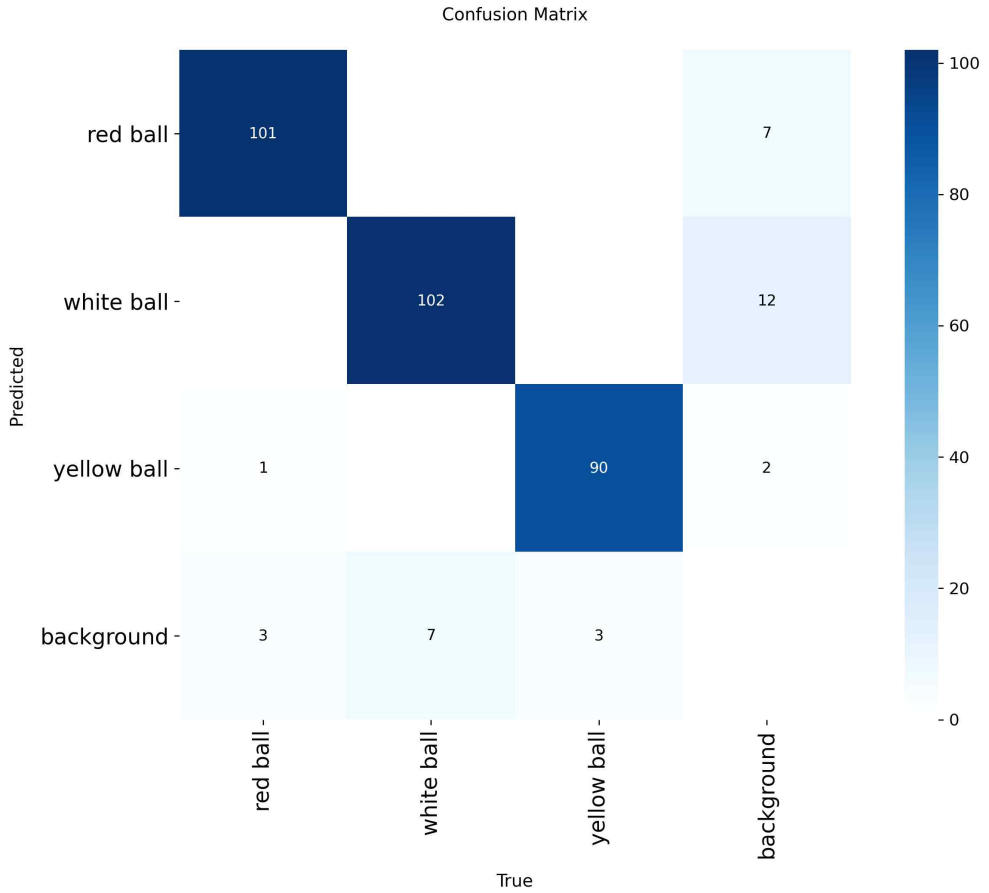
4. 주요 결과물 – YOLO 학습 결과

```
Model summary (fused): 72 layers, 3,006,233 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 4/4 3.7it/s 1.1s
                   all        100        307      0.968      0.946      0.975      0.851
              red ball         96        105      0.956      0.943      0.984      0.881
            white ball         95        109      0.952      0.917      0.948      0.785
           yellow ball         90         93      0.995      0.978      0.993      0.887
```

- mAP@0.5가 0.975를 기록함
- 객체 이미지를 거의 놓치지 않음을 확인함



4. 주요 결과물 – YOLO 학습 결과



- 전이 학습을 하면서 공 인식률이 점점 높아짐을 확인할 수 있음.
- 기존의 노란공을 흰색공으로 오탐지 하는 문제도 어느정도 해결

4. 주요 결과물 – 모델 선정

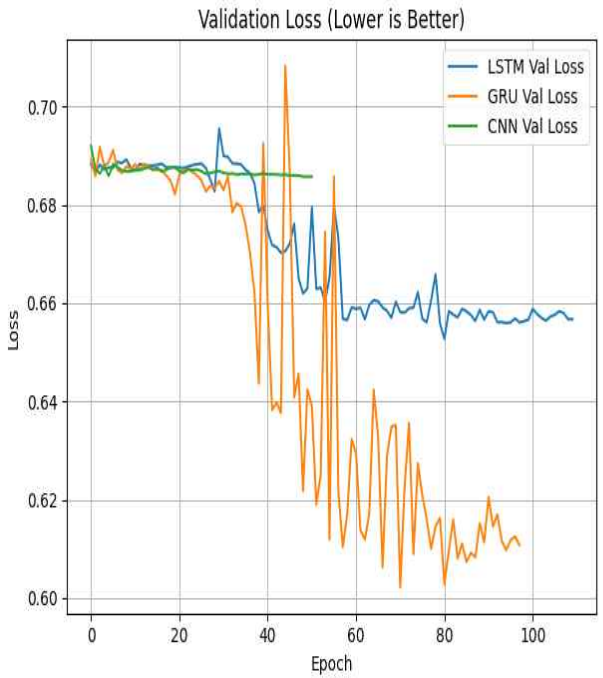
- 판독의 핵심 : 공이 움직이는 순서와 경로가 3쿠션 규칙에 맞는가?
- 후보 모델군

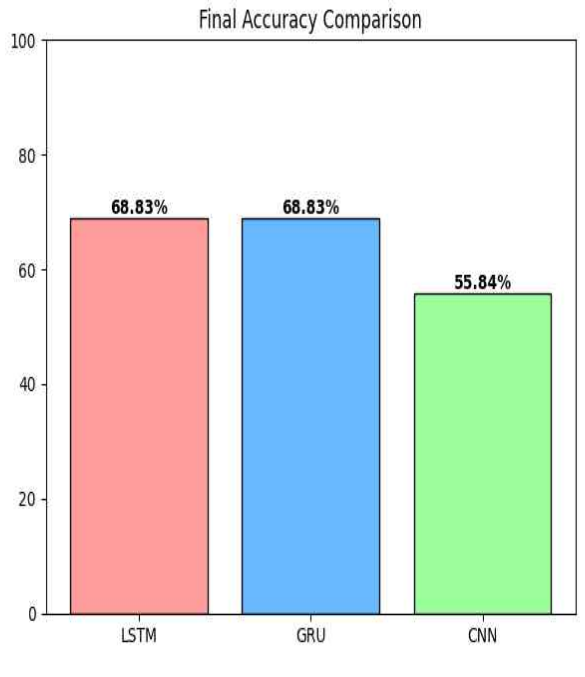
1. CNN
2. LSTM
3. GRU

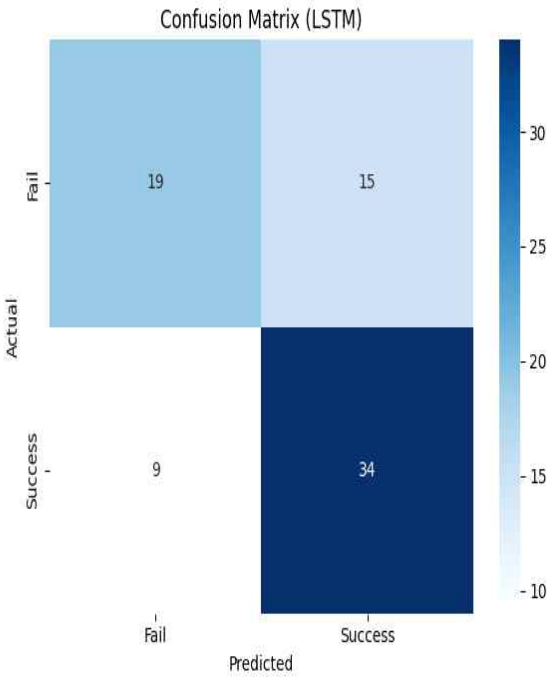
```python
class ModelLSTM(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(2, 64, 2, batch_first=True)
        self.fc = nn.Linear(64, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        out, _ = self.lstm(x)
        return self.sigmoid(self.fc(out[:, -1, :]))

class ModelGRU(nn.Module):
    def __init__(self):
        super().__init__()
        self.gru = nn.GRU(2, 64, 2, batch_first=True)
        self.fc = nn.Linear(64, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        out, _ = self.gru(x)
        return self.sigmoid(self.fc(out[:, -1, :]))

class ModelCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(2, 64, 3)
        self.pool = nn.MaxPool1d(2)
        self.conv2 = nn.Conv1d(64, 128, 3)
        self.gap = nn.AdaptiveAvgPool1d(1)
        self.fc = nn.Linear(128, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        x = x.permute(0, 2, 1)
        x = self.pool(torch.relu(self.conv1(x)))
        x = torch.relu(self.conv2(x))
        x = self.gap(x).squeeze(2)
```

4. 주요 결과물 – 모델 학습

- 성공 / 실패 샷 영상을 통째로 모델에 입력
- 성공 / 실패 영상을 각각 100개로 분류해서 학습 시작







4. 주요 결과물 – 문제점

문제점

- 영상 초반 1 ~ 2초 동안은 큐질을 준비하는 동작 자세
- 딥러닝 모델들을 처음 3초만 보고 3쿠션의 성공 여부를 예측하는 구조
- 당구대 테이블이나 조명 등 무의미한 정보까지 학습을 해버림
- 데이터 영상들이 학습에 영향을 주지 않음.
- 결국 예측한 확률이 50 : 50으로 나와서 판단에 도움이 되지 않음

개선점

- 공의 속도를 계산하여 가장 속도가 빠른 타격 순간을 기준으로 데이터를 정렬하고 불필요한 앞뒤 구간을 잘라냄
- 기존의 단순 좌표 x, y 2개 특징만으로 부족하다고 판단해 x, y, dx, dy, speed, angle의 6개 채널로 확장하여 공의 물리적 특성을 반영하기로 결정

4. 주요 결과물 – 모델 재선정 및 전처리

csv 추출

- 6개의 채널 값을 추출하기 위해 성공 / 실패 샷 영상을 각각 프레임으로 추출해 시간순으로 정렬.
- 프레임 이미지를 yolo 모델을 사용해 공의 좌표값을 추출.
- 프레임별로 공의 움직임을 통해 나머지 dx, dy, speed, angle 값을 추출해 csv로 저장
- 이때 YOLO가 공을 놓치거나 좌표가 흔들리는 것을 방지하기 위해 보간과 스무딩으로 보정 작업
- 영상마다 해상도가 다를 수 있고, 좌표값이 너무 크면 학습에 문제가 있을 수 있어서 해상도 정규화

선형 보간(Linear Interpolation):YOLO 인식 실패 구간을 앞뒤 좌표 기반으로 복원.
이동 평균(Rolling Smoothing):좌표 떨림 현상을 완화하여 부드러운 궤적 생성.

4. 주요 결과물 – 전처리 과정

- 1. 노이즈 제거 및 초구 자동 식별

흰공 or 노란공 데이터만 남기고 빨간공은 버림

최대 속도를 비교해 움직임이 더 큰 공을 초구로 판단

- 2. 결측치 보간 및 스무딩

YOLO가 공을 놓쳐서 빈 구간을 채우고, 바운딩 박스가 떨리는 현상을 보정

- 3. x, y좌표를 미분하여 속도, 가속도, 각도 정보 생성
- 4. 시퀀스 정렬 및 길이 고정

영상의 시작점을 맞추고 길이를 통일

만약 데이터의 길이가 60프레임보다 짧으면 0으로 채워 길이를 맞춤

공의 속도가 최대가 되는 시점을 찾아서 그 앞 5프레임부터 자르기

- 5. 공간 정규화

픽셀 좌표를 0~1 사이 소수점으로 변환

4. 주요 결과물 – 전처리 과정

```python
# 시작 타격 공 찾기
def get_features(df):
    white = df[df['cls'] == 1].copy()
    yellow = df[df['cls'] == 2].copy()

    def calc_max_speed(ball_df):
        if len(ball_df) < 5: return 0
        ball_df = ball_df.sort_values('frame')
        dx = ball_df['x'].diff().fillna(0)
        dy = ball_df['y'].diff().fillna(0)
        return np.sqrt(dx**2 + dy**2).max()

    if calc_max_speed(white) >= calc_max_speed(yellow):
        target = white
    else:
        target = yellow

    if len(target) < 10: return None
```

```python
# 2. 전처리 (스무딩 & 보간)
target = target.sort_values('conf', ascending=False).drop_duplicates('frame').sort_values('frame')
target['x'] = target['x'].rolling(window=5, min_periods=1).mean()
target['y'] = target['y'].rolling(window=5, min_periods=1).mean()

min_f, max_f = target['frame'].min(), target['frame'].max()
full_idx = range(int(min_f), int(max_f) + 1)
target = target.set_index('frame').reindex(full_idx)
target[['x', 'y']] = target[['x', 'y']].interpolate(method='linear')
target = target.dropna().reset_index()
```

```python
# 3. 특징 계산 (6가지)
x_norm = target['x'].values / TABLE_W
y_norm = target['y'].values / TABLE_H

dx = np.diff(x_norm, prepend=x_norm[0])
dy = np.diff(y_norm, prepend=y_norm[0])
speed = np.sqrt(dx**2 + dy**2)
angle = np.arctan2(dy, dx) / np.pi # -1 ~ 1 사이로 정규화

# 4. 타격 시점 기준 정렬
peak_idx = np.argmax(speed)
if speed[peak_idx] < 0.005: return None # 너무 느리면 패스

start_idx = max(0, peak_idx - 5)
end_idx = start_idx + SEQ_LENGTH
```

A	B	C	D	E	F
x	y	dx	dy	speed	angle
0.449479	0.283962	0.017095	0.086464	0.040835	0.381719
0.474706	0.28826	0.025227	0.0583	0.063827	0.385436
0.501732	0.297917	0.027026	0.09656	0.100253	0.383285
0.556177	0.4372	0.045445	0.099294	0.1092	0.383372
0.602499	0.56796	0.04682	0.10165	0.110798	0.382709
0.648155	0.67154	0.045657	0.103679	0.113693	0.385388
0.696082	0.774366	0.046927	0.102826	0.113028	0.383777
0.742596	0.87774	0.046514	0.103375	0.113357	0.38543
0.788805	0.980549	0.046209	0.102908	0.112715	0.385544
0.834136	1.083807	0.045332	0.103254	0.112862	0.387173
0.8796	1.174012	0.045464	0.09121	0.101913	0.35261
0.92 781	1.244884	0.04218	0.070122	0.082249	0.320696
0.961085	1.294704	0.039304	0.05207	0.063854	0.288169
0.997482	1.324627	0.036398	0.029823	0.047118	0.219021
1.031902	1.334431	0.03444	0.00981	0.03581	0.069526
1.063711	1.33427	0.031809	-0.00017	0.031809	-0.00185
1.096729	1.334178	0.033018	-0.00007	0.033018	-0.00102
1.127848	1.33435	0.032119	-2.74E-05	0.032119	-0.00027
1.161259	1.334727	0.03341	0.00037	0.03341	0.0011
1.192961	1.33579	0.032703	0.001062	0.03272	0.01064
1.225491	1.338809	0.032531	0.003019	0.033449	0.045054
1.258173	1.337574	0.032682	-0.00123	0.032789	-0.00663
1.291075	1.336114	0.032577	-0.00154	0.032591	-0.00527
1.323285	1.335658	0.032535	-4.58E-05	0.032535	-0.00067
1.354593	1.33778	0.031308	0.00228	0.031309	0.00282
1.374549	1.334013	0.019957	-0.00377	0.020309	-0.05936
1.38295	1.327949	0.007901	-0.00567	0.010261	-0.22517
1.378393	1.317538	-0.00396	-0.00881	0.01068	-0.82953
1.36275	1.30497	-0.01564	-0.01256	0.020064	-0.78459
1.338751	1.283043	-0.025	-0.01792	0.030809	-0.62519
1.310905	1.273408	-0.02585	-0.01364	0.030211	-0.92676
1.289012	1.257913	-0.02489	-0.01549	0.032818	-0.82284
1.262037	1.242685	-0.02397	-0.01523	0.028405	-0.61963
1.239964	1.227544	-0.02305	-0.01514	0.02758	-0.91512
1.218784	1.213685	-0.0222	-0.01486	0.026714	-0.81235
1.196578	1.19822	-0.0212	-0.01446	0.025668	-0.80945
1.173161	1.183697	-0.02042	-0.01452	0.025057	-0.9032
1.153723	1.16924	-0.01944	-0.01446	0.024224	-0.79646
1.137121	1.155089	-0.01815	-0.01416	0.023378	-0.79981
1.119209	1.141212	-0.01791	-0.01387	0.022555	-0.79026
1.10 928	1.127642	0.01726	0.01357	0.021974	0.790 3

4. 주요 결과물 – 히트맵

DY와 ANGLE : 각도가 arctan(dy/dx)로 계산되기 때문에 dy가 변하면 angle도 변함

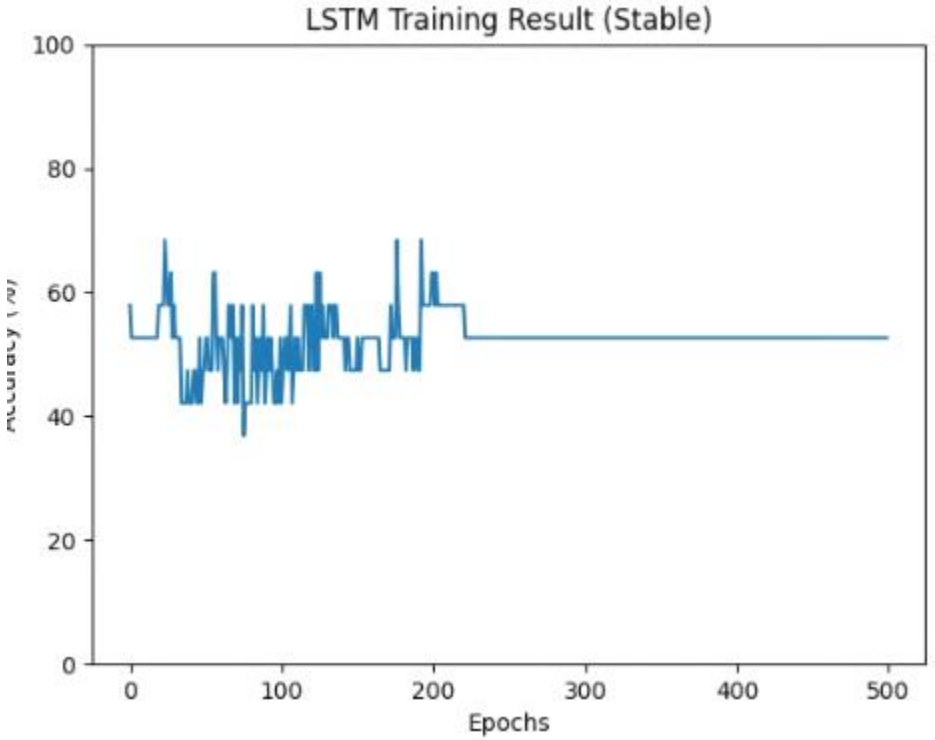
4. 주요 결과물 – 모델 재학습

```python
class makeLSTM(nn.Module):
    def __init__(self):
        super().__init__()
        # 과적합 방지
        self.lstm = nn.LSTM(6, 64, 2, batch_first=True, dropout=0.2)
        self.relu = nn.ReLU()
        self.fc = nn.Linear(64, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.relu(out[:, -1, :])
        return self.sigmoid(self.fc(out))
```

```python
def train():
    dataset = BilliardCSVDataset(DATA_FOLDER)
    if len(dataset) == 0:
        return

    train_size = int(0.8 * len(dataset))
    train_set, val_set = random_split(dataset, [train_size, len(dataset)-train_siz

    train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True)
    val_loader = DataLoader(val_set, batch_size=BATCH_SIZE)

    # 모델 선택
    if MODEL_TYPE == "LSTM":
        model = makeLSTM().to(DEVICE)
        save_name = "best_model_lstm.pth"
    else:
        model = RichCNN().to(DEVICE)
        save_name = "best_model_cnn.pth"

    optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
    criterion = nn.BCELoss()

    print(f"[{MODEL_TYPE}] 학습 시작")
    print(f"LR={LEARNING_RATE}, Batch={BATCH_SIZE}, Seed=42")

    history = []
    best_acc = 0.0
```

```python
for epoch in range(EPOCHS):
    model.train()
    train_loss = 0

    for x, y in train_loader:
        x, y = x.to(DEVICE), y.to(DEVICE)
        optimizer.zero_grad()

        output = model(x).squeeze()
        loss = criterion(output, y)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
        train_loss += loss.item()

    # 검증
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for x, y in val_loader:
            x, y = x.to(DEVICE), y.to(DEVICE)
            pred = (model(x).squeeze() > 0.5).float()
            correct += (pred == y).sum().item()
            total += y.size(0)

    acc = 100 * correct / total
    history.append(acc)

    if acc >= best_acc:
        best_acc = acc
        torch.save(model.state_dict(), save_name)

    if (epoch+1)%10 == 0:
        print(f"Epoch {epoch+1}: Loss {train_loss/len(train_loader):.4f} | Acc {acc:.2f}% (Best: {best_acc:.2f}%)")
```

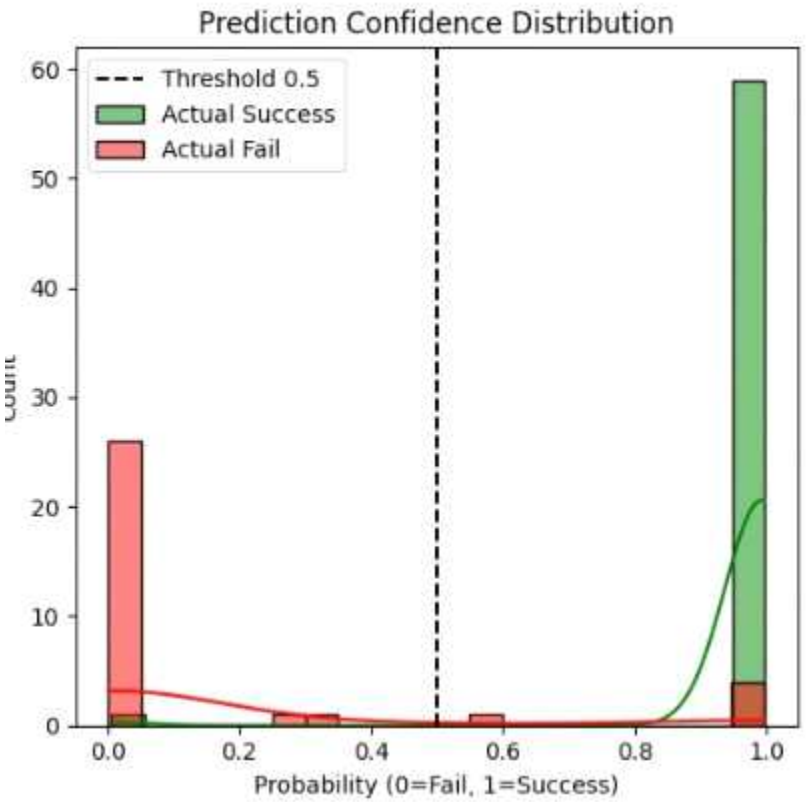
4. 주요 결과물 – LSTM 학습 결과




```
Epoch 170: Loss 0.3225 | Acc 42.11% (Best: 68.42%)
Epoch 180: Loss 0.1394 | Acc 47.37% (Best: 68.42%)
Epoch 190: Loss 0.0856 | Acc 47.37% (Best: 68.42%)
Epoch 200: Loss 0.1627 | Acc 42.11% (Best: 68.42%)
Epoch 210: Loss 0.0517 | Acc 42.11% (Best: 68.42%)
Epoch 220: Loss 0.0877 | Acc 42.11% (Best: 68.42%)
Epoch 230: Loss 0.0828 | Acc 42.11% (Best: 68.42%)
Epoch 240: Loss 0.0627 | Acc 47.37% (Best: 68.42%)
Epoch 250: Loss 0.1228 | Acc 47.37% (Best: 68.42%)
Epoch 260: Loss 0.0704 | Acc 47.37% (Best: 68.42%)
Epoch 270: Loss 0.0108 | Acc 47.37% (Best: 68.42%)
Epoch 280: Loss 0.0681 | Acc 52.63% (Best: 68.42%)
Epoch 290: Loss 0.0087 | Acc 47.37% (Best: 68.42%)
Epoch 300: Loss 0.0058 | Acc 47.37% (Best: 68.42%)
Epoch 310: Loss 0.0049 | Acc 47.37% (Best: 68.42%)
Epoch 320: Loss 0.0047 | Acc 47.37% (Best: 68.42%)
Epoch 330: Loss 0.0039 | Acc 47.37% (Best: 68.42%)
Epoch 340: Loss 0.0039 | Acc 47.37% (Best: 68.42%)
Epoch 350: Loss 0.0032 | Acc 42.11% (Best: 68.42%)
Epoch 360: Loss 0.0029 | Acc 47.37% (Best: 68.42%)
Epoch 370: Loss 0.0026 | Acc 47.37% (Best: 68.42%)
Epoch 380: Loss 0.2345 | Acc 42.11% (Best: 68.42%)
Epoch 390: Loss 0.0027 | Acc 47.37% (Best: 68.42%)
Epoch 400: Loss 0.0025 | Acc 42.11% (Best: 68.42%)
Epoch 410: Loss 0.0044 | Acc 47.37% (Best: 68.42%)
Epoch 420: Loss 0.2556 | Acc 57.89% (Best: 68.42%)
Epoch 430: Loss 0.0145 | Acc 52.63% (Best: 68.42%)
Epoch 440: Loss 0.0122 | Acc 68.42% (Best: 73.68%)
Epoch 450: Loss 0.0042 | Acc 57.89% (Best: 73.68%)
Epoch 460: Loss 0.0031 | Acc 47.37% (Best: 73.68%)
Epoch 470: Loss 0.0543 | Acc 63.16% (Best: 73.68%)
Epoch 480: Loss 0.0024 | Acc 57.89% (Best: 73.68%)
Epoch 490: Loss 0.0021 | Acc 68.42% (Best: 73.68%)
Epoch 500: Loss 0.0019 | Acc 63.16% (Best: 73.68%)
```

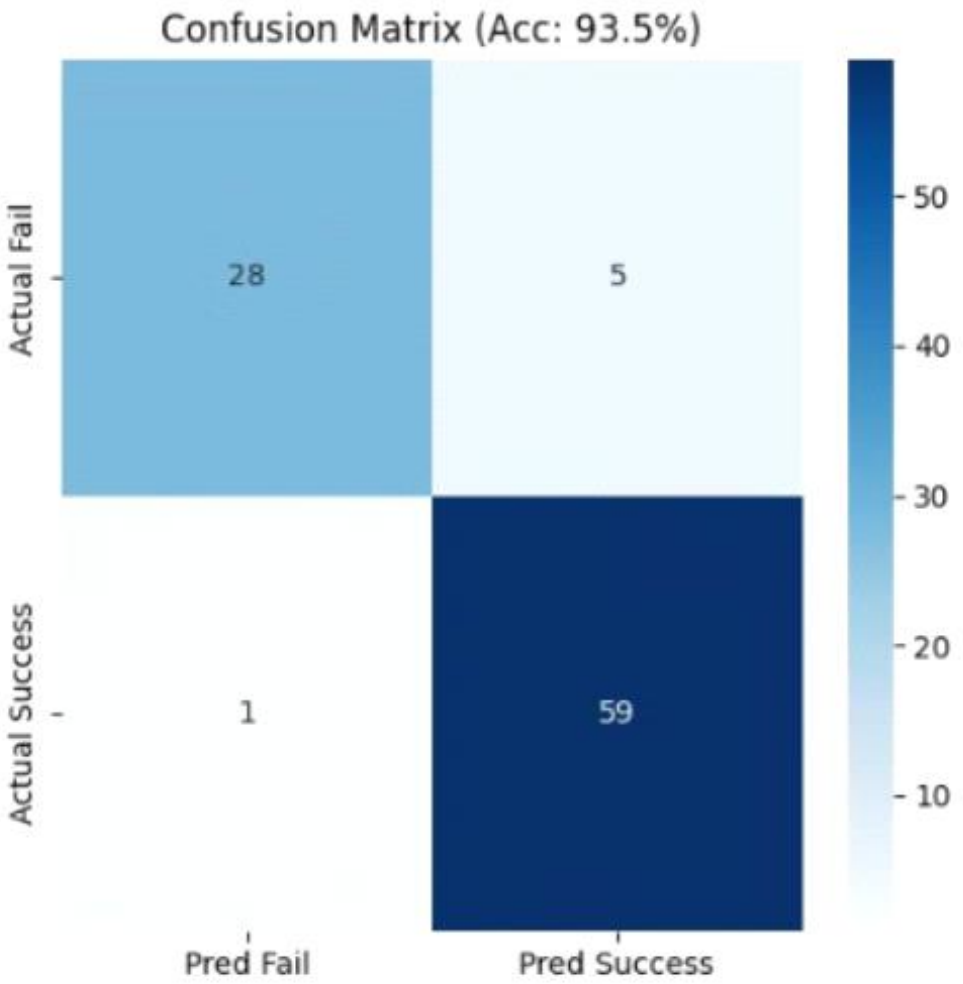
4. 주요 결과물 – 예측 확신도 분포

- 모델의 예측 확률 분포 분석 결과
- 성공 케이스는 1.0쪽으로,
- 실패 케이스는 0.0쪽으로 분포
- 모델이 애매하게 판단하는 경우가 거의 없음을 확인할 수 있음

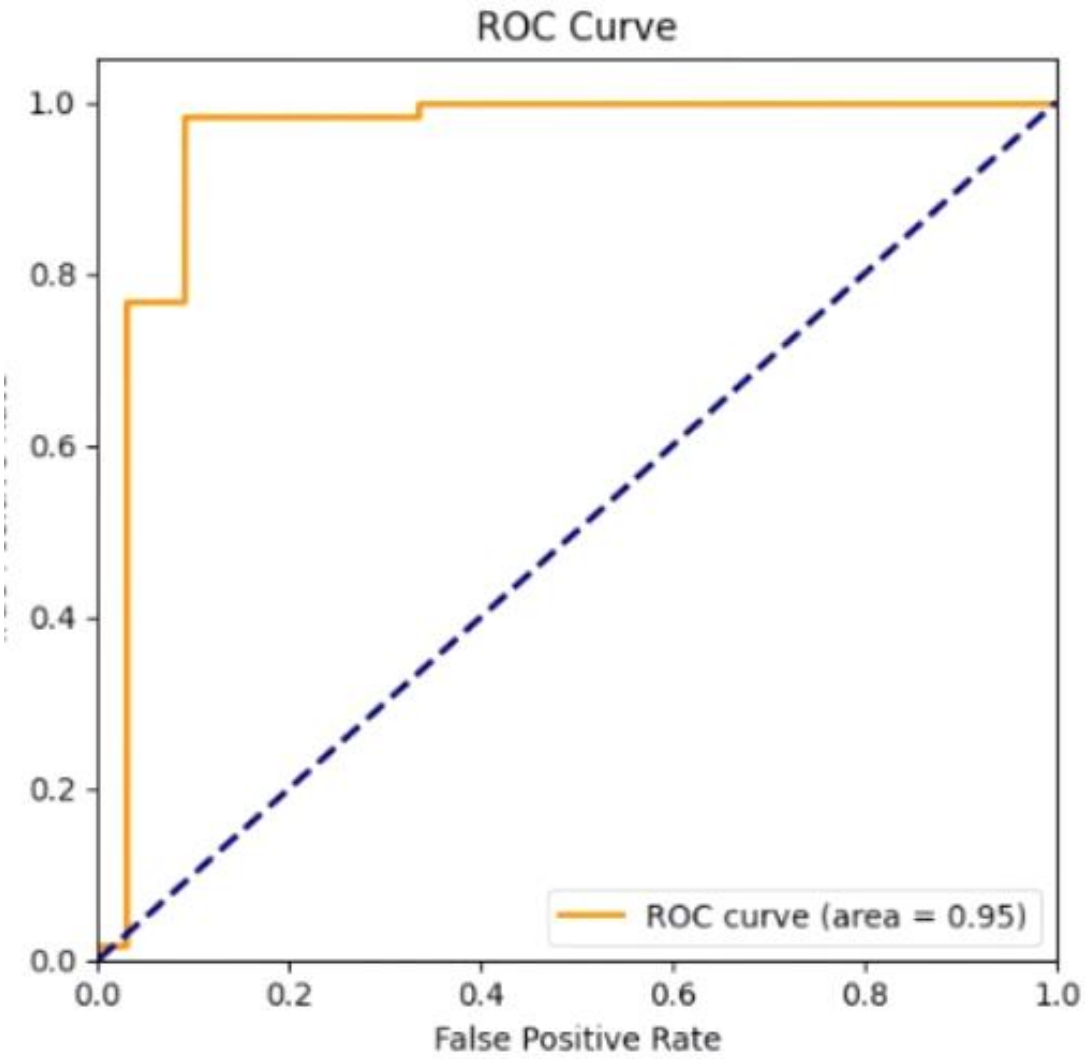


4. 주요 결과물 – 혼동 행렬

- 실제 60건 중 59건 맞추어 성공에 대한 재현율이 높음을 확인 가능
- 성공한 샷을 실패로 판정하거나 실패한 샷을 성공으로 판정한 경우가 적음을 확인 가능



4. 주요 결과물 – ROC Curve



- ROC Curve: 모델의 판정 임계값을 변화시켰을 때, 진짜를 진짜라고 맞춘 비율(TPR)과 가짜를 진짜라고 잘못 맞춘 비율(FPR) 사이를 나타낸 곡선

4. 번외 – 궤적 생성

```python
for i, frame in enumerate(frame_list):
    temp_frame = frame.copy()

    if i in points_dict: draw_pts.append(points_dict[i])

    cv2.putText(temp_frame, f"Cue Ball: {ball_color}",
                (50, 130), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2)

    if len(draw_pts) > 1:
        pts = np.array(draw_pts, np.int32).reshape((-1, 1, 2))
        cv2.polylines(temp_frame, [pts], False, color_res, 3)

    if result_text == "FAIL" and len(ghost_pts) > 1:
        g_pts = np.array(ghost_pts, np.int32).reshape((-1, 1, 2))
        cv2.polylines(temp_frame, [g_pts], False, (0, 255, 127), 2, cv2.LINE_AA)
        end_pt = ghost_pts[-1]
        cv2.putText(temp_frame, "AI Guide", (end_pt[0]-50, end_pt[1]-20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 127), 2)

    frame_rgb = cv2.cvtColor(temp_frame, cv2.COLOR_BGR2RGB)
    output_frames.append(frame_rgb)
```

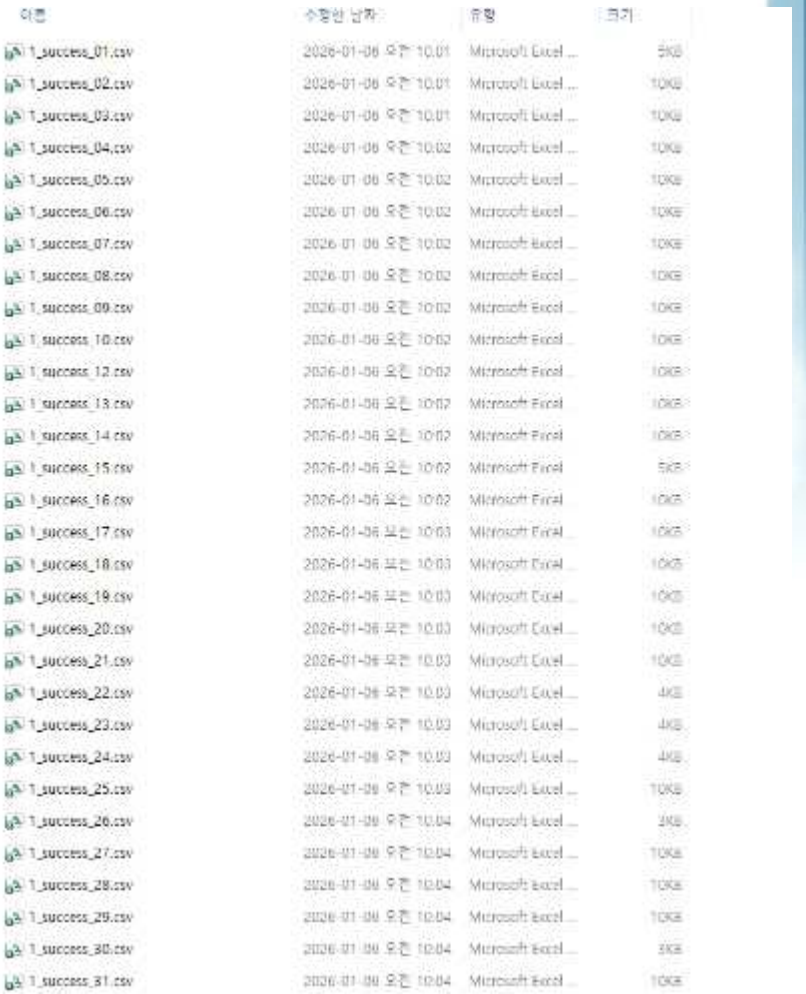
- 영상을 첨부하면 한 프레임씩 재생하면서 YOLO가 찾은 공의 좌표를 보정하고 프레임번호:(x, y) 형태로 리스트로 저장
- 프레임 번호를 확인하면서 draw_pts 리스트에 좌표를 하나씩 저장하고, 이 리스트에는 [(x1,y1),(x2,y2)...] 처럼 과거의 좌표들이 쌓임
- 이 좌표들의 모든 점을 이어 그려서 궤적을 생성

YOLO에 영상을 바로 넣으면 탐지가 불확실하기 때문에 이미지 사용 방식을 고름

5. LLM 구현 (RAG + OLLAMA)

```
# [설정] 사용할 Ollama 모델 이름
OLLAMA_MODEL = "gemma3:4b"
```

- 기존에 있던 성공샷의 궤적 csv 데이터를 학습.
- 실패 시 그 영상의 공 위치와 최대한 비슷한 궤적을 찾고 선을 그려서 정답 라인 생성
- 실패했던 샷의 각도와 속도를 분석해 피드백 해주는 LLM 생성
- RAG 검색을 위한 외부 벡터 저장소를 db로 저장.
- 샷의 정보가 담긴 db를 바탕으로 ollama가 코치해주는 RAG + OLLAMA 구조
- 공 위치(좌표) → KNN 거리 계산 → 유사 궤적 데이터 검색 → 데이터 해석 후 프롬프트 주입 → LLM



4. 주요 결과물 – 프로토타이핑(화면)

AI 3-Cushion Referee & Coach

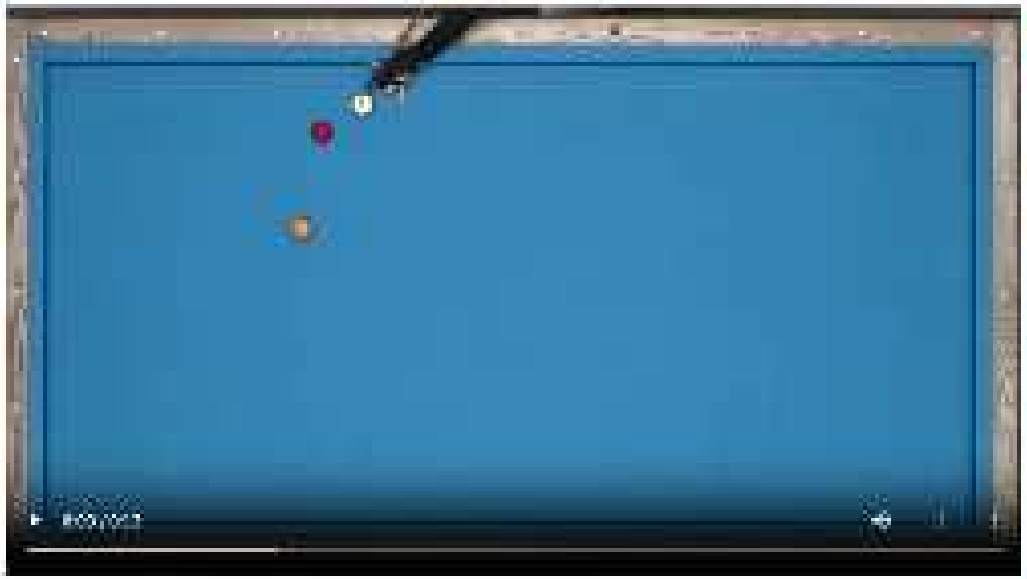
원본 영상

판독 결과

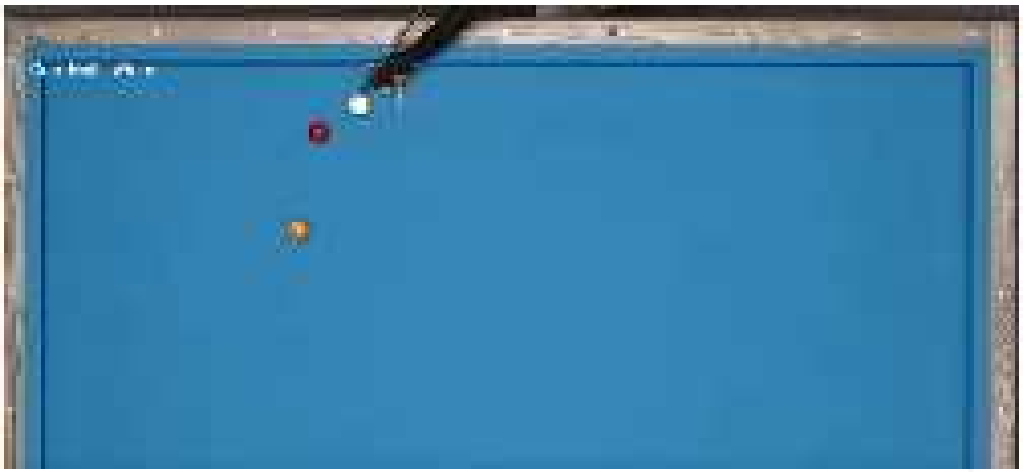
4. 주요 결과물 – 프로토타이핑(화면)

AI 3-Cushion Referee & Coach

원본 영상



판독 결과



4. 주요 결과물 – 프로토타이핑(화면)

판독 결과

판독 시작

✕ 판독 결과: 실패 (30.4%)

※AI 코치 피드백 (Ollama):

"샷 각도가 너무 얇게 쳤어. 신입각을 5도 정도 더 벌려서 볼이 테이블 반으로 더 잘 들어오게 해야 해!"

5. 보완사항 및 개선점

데이터셋 부족

- 실시간으로 돌리기 위해서는 당구공의 이미지와 샷 영상 수가 더 필요하다고 판단됨.
- 적은 데이터로 인해 학습이 만족스럽지 않음.
- 성공샷의 데이터가 부족해 RAG로 LLM을 생성했을 때 만족스러운 궤적이 나오지 않음.

진단 범위 확장

- 현재는 3구만 판정 중, 4구일때 득점 여부
- 녹화 영상 업로드 방식을 넘어 당구장 CCTV와 연동하여 즉시 판독하는 실시간 파이프라인 구현

시뮬레이션 기능 확장

- 자동 점수판 기능

6. 소감 및 후기

프로젝트 성과

1. 여러 모델을 사용해보면서 프로젝트 상황에 따른 적절한 모델을 고르는 법을 배울 수 있었습니다.
2. YOLO와 LSTM 모델을 섞어서 사용하며 판독 정확도를 올릴 수 있었습니다.
3. 추가 데이터를 확보해서 성능을 올린다면 실제로 사용할 것 같습니다.

아쉬웠던 점

1. 당구장의 실제 CCTV로 연동해서 하고싶었으나, 그럴 수 있는 환경이 없어서 아쉬웠습니다.
2. 기간이 일주일이라 데이터 라벨링, 수집이 적어 아쉬웠습니다.
3. RAG 기반으로 LLM을 구현하려다보니 실패했을 때 피드백을 해주는 성공샷의 데이터가 너무 부족해서 원하는 궤적이 생성되지 않았습니다.