

제 2023-24 호

상장

동상(3위)

과학과 3학년 3반
성명: 이지호

위 학생은 교내 발명 아이디어
공모전에서 창의적인 아이디어를
통해 우수한 능력을 발휘하였기
에 이 상장을 수여합니다.

2023년 04월 11일

인천진산과학고등학교장 장훈



TensorFlow와 YOLO를 활용한 CCTV 분석 스마트 보안 시스템

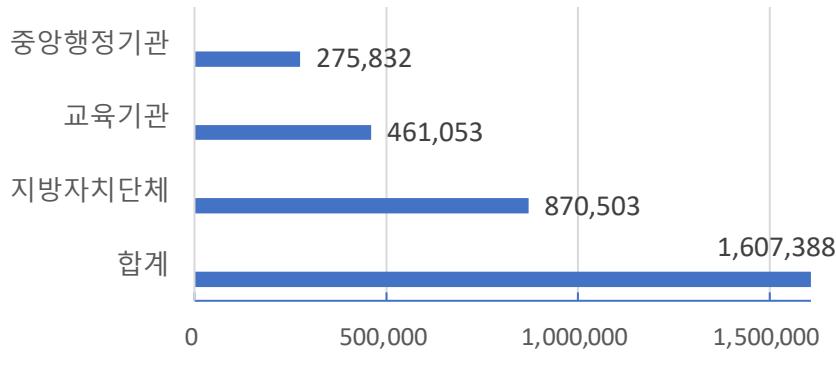
3306 이지호

1. 연구 동기

2023년 현재 대한민국에서는 신림역 4번 출구 칼부림 사건과 같은 다양한 강력 범죄가 일어나고 있다. 또한, 2023년 2월에 발생한 인천 편의점 강도살인 사건과 같은 사건들도 여럿 발생하고 있다. **다양한 일련의 사건들로 봤을 때 우리나라도 절대로 범죄의 그늘에서 안전하지 않다는 것을 알 수 있다.** 통계청 자료에 따르면, 2021년에 발생한 총 범죄 수는 1,429,826건이고, 그중 검거된 건수는 1,136,665건이다. 약 79.5%의 검거율이 나오는 것을 알 수 있는데, 나는 이 검거율을 더 끌어올릴 수는 없을지에 대한 궁금증이 생겼다. 다양한 기술적 방법론을 모색하던 중 **언급한 문제를 효율적으로 해결할 수 있는 방안을 찾기 위해서 CCTV를 선택하였다.**

현재, 한국인터넷진흥원에 따르면 2021년 말 **약 1,600만대의 CCTV가 설치되어 있을 것으로 추정하였고**, 이렇게 많은 양의 CCTV를 단지 사건 확인용으로 쓰기에는 너무 아깝다는 생각이 들었다. 우리는 흔히 'CCTV 녹화중'이라는 문구를 붙여놓는 경우도 만나볼 수 있는데, 이는 감시를 하고 있다는 것을 알리고 경각심을 주기 위함이라고 생각한다. 현재 대한민국 사회에서 CCTV는 어떠한 사건이 일어났을 때 사건의 전후 상황을 판단하는 용도로만 사용하게 된다.

나는 CCTV를 사건의 전후 상황을 파악하는 용도로만 사용한다는 사실에 집중했다. **CCTV를 직접적인 사건 신고 용도로 사용할 수 있는 방법은 없는지에 대한 의문이 들었고, 프로그래밍을 이용하면 CCTV의 영상을 불러와 실시간으로 그 영상을 처리하고 해석하는 것이 가능하다는 사실을 알았다.** 이렇게 실시간으로 CCTV 영상 처리하는 것이 가능해지면 이것을 자동으로 신고할 수 있게 하면 범죄에 더 빠른 대응을 할 수 있을 것이라고 생각했다. 그래서 마지막에 자동 신고 시스템을 만들면 CCTV를 통한 범죄의 인식부터 자동으로 신고하는 것까지 가능할 것이라고 생각한다.



▲ [Fig. 1] 2022년 말 공공기관 CCTV 설치/운영 현황

2. 선행연구 분석

딥러닝 기반 범죄자 신원 인식 시스템 설계 및 구현

위 연구는 **딥러닝을 이용해 얼굴인식 시스템을 설계해 이를 범죄자의 신원을 인식하는데 사용하였다.** 증가하는 위험요인과 지능화된 범죄에 대응하기 위해서이다.

실제 범죄 발생시 CCTV 영상을 통해 범죄자를 검거하는데, 사람의 인력을 필요로 하는 작업이므로 효율적인 작업이 아니어서 이를 효율적으로 하는 방안 중 하나였다. 위 연구에서는 입력받는 영상 데이터에서 얼굴 영역을 추출하여 추출한 얼굴 영역을 정규화하여 딥러닝을 통해 특징 벡터를 추출하고 분류하여 신원 인식 후 결과를 전송하는 기능을 설계하였다. C++과 Python2를 이용해서 연구를 진행하였다.

딥러닝 기반의 CCTV를 활용한 이상 행동 감지 시스템 설계

위 연구는 **딥러닝 모델인 3D ResNet을 활용하여 CCTV 영상에서 이상 행동을 감지하는 시스템**을 소개한다. UCF-Crime 데이터세트를 사용하여 이상 행동과 정상 행동을 구분하는 2-class 분류 문제에 3D ResNet 모델을 적용하여 실험을 수행한다. 훈련된 모델은 비디오 데이터를 입력으로 받아 행동 패턴을 추론하며, 이를 통해 인력이 아닌 딥러닝 시스템을 기반으로 한 신뢰도 높은 이상 행동 판단이 가능함을 입증한다.

향후 추론 모델의 세부 행동 패턴을 더 다양하게 분석하고 신뢰도를 향상시키는 알고리즘 연구를 계획한다. 이를 통해 적은 인력으로도 보다 효율적인 방범 시스템 운영이 가능하며, 선제적 범죄 예방 효과를 높일 것으로 기대한다.

선행연구 분석을 통한 연구 설계 착안점

본 연구의 선행연구로 두 연구를 선택한 이유가 있다. 연구 사례를 통해서 진행 아이디어를 얻고, 대략적인 가설을 선택할 수 있었기 때문이다. “**딥러닝 기반 범죄자 신원 인식 시스템**”에서는 얼굴 영역을 정규화하여 특징을 추출해 신원을 인식한다는 것이 인상 깊었다. 연구에 포함되어야 하는 내용을 크게 포즈(동작) 인식과 object(물체) 인식으로 나눌 수 있는데 물체 인식을 하는 데에 있어 충분한 아이디어를 제공할 수 있을 것으로 보인다.

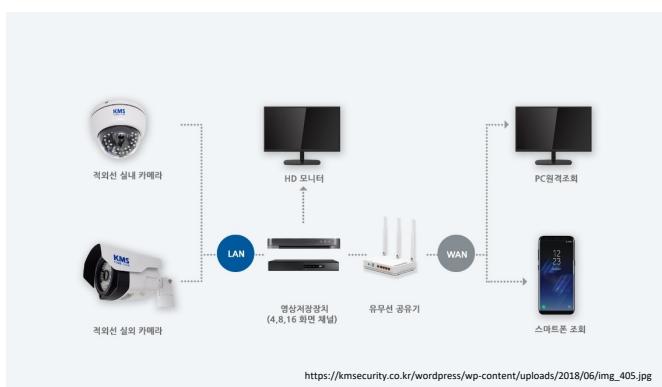
딥러닝 기반의 CCTV를 활용한 이상 행동 감지 시스템 설계이다. 앞서 포즈(동작)을 감지하는 과정이 필요하다고 언급하였는데, 첨부한 연구를 참고하면 **이상 행동을 감지하는 원리를 본 연구에서는 폭력적인 동작을 감지하는 과정으로 활용, 각색할 수 있다고** 생각하여 그 부분에 대한 아이디어를 제공할 수 있을 것 같았다.

3. 이론적 배경

CCTV

폐쇄회로 텔레비전을 말한다. 특정목적을 위하여 특정인들에게 제공되는 TV라는 뜻이다. 이러한 목적을 가짐으로 인해 CCTV는 유무선으로 밖과 연결되지 않아서 폐쇄회로라고 불리는 것이다. CCTV는 카메라와 이 카메라가 찍는 영상을 녹화해 줄 DVR로 구성된다. DVR은 영상을 녹화하는 장비로, CCTV를 구성하는 요소 가운데 가장 비싸다. 이 장비의 성능에 따라서 녹화 가능 영상의 화질이나 동시 녹화 가능 카메라 수가 다르다.

CCTV는 대표적으로 방범, 감시, 화재예방 등 안전을 위해 설치한다. 사람들이 많이 지나다니는 번화가 같은 곳과 범죄 위험지역, 그리고 폐쇄된 실내(엘리베이터, 지하철 등), 건물 내/외부에 설치하여 그 곳의 상황을 알 수 있다.



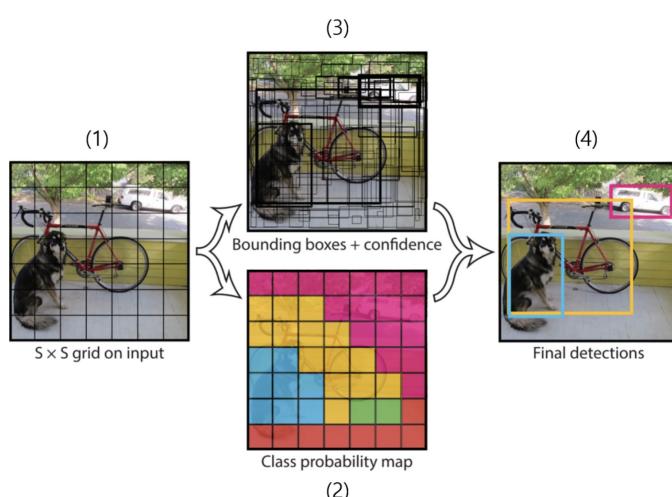
▲ [Fig. 2] IP카메라(외부 접속 가능) 시스템 구성도

YOLO

YOLO는 입력된 이미지를 일정 분할로 그리드한 다음, 신경망을 통하여 최종 감지 출력을 결정한다. You Only Look Once라는 이름 뜻에서도 알 수 있듯이, YOLO는 이미지 전체를 한 번만 본다는 특징을 가지고 있다. region proposal, feature extraction, classification, bbox regression에 대한 단계를 전부 한 단계로 통합하여 매우 빠르다는 특징을 가진다. 각 개체만을 학습하는 모델과는 다르게 YOLO는 주변 정보까지 학습하기 때문에 적은 error를 가진다. 또한, 훈련 단계에서 보지 못한 새로운 이미지에 대해 높은 검출 정확도를 가진다.

YOLO는 다음과 같은 절차로 작동한다:

1. 사진이 입력되면 이것을 정사각형으로 쪼갠다.
2. 그리드 영역에 대해서 어디에 사물이 존재하는지 Bounding box와 box에 대한 Confidence Score를 예측한다.
3. 신뢰도가 높을수록 굵게 박스를 그려 준다. 이와 동시에 2에서와 같이 어떤 사물인지에 대한 분류 작업이 동시에 진행된다.
4. 그 후 굵은 박스들만 남겨 놓는다.



▲ [Fig. 3] YOLO 메커니즘

그리드 영역 : YOLO는 이미지/영상을 분석할 때 각각의 구역으로 쪼개어 분석한다. 쪼개는 단위를 s 라고 할 때, $s \times s$ 의 평면으로 쪼갠다.

바운딩 박스 : 물체 인식을 위한 가이드라인 포인트로서 사용되고, 해당 요소에 대한 충돌 박스를 생성하는 사각형이다.

Confidence Score: Confidence Score는 다음과 같이 정의된다.

$$\text{Pr}(\text{object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

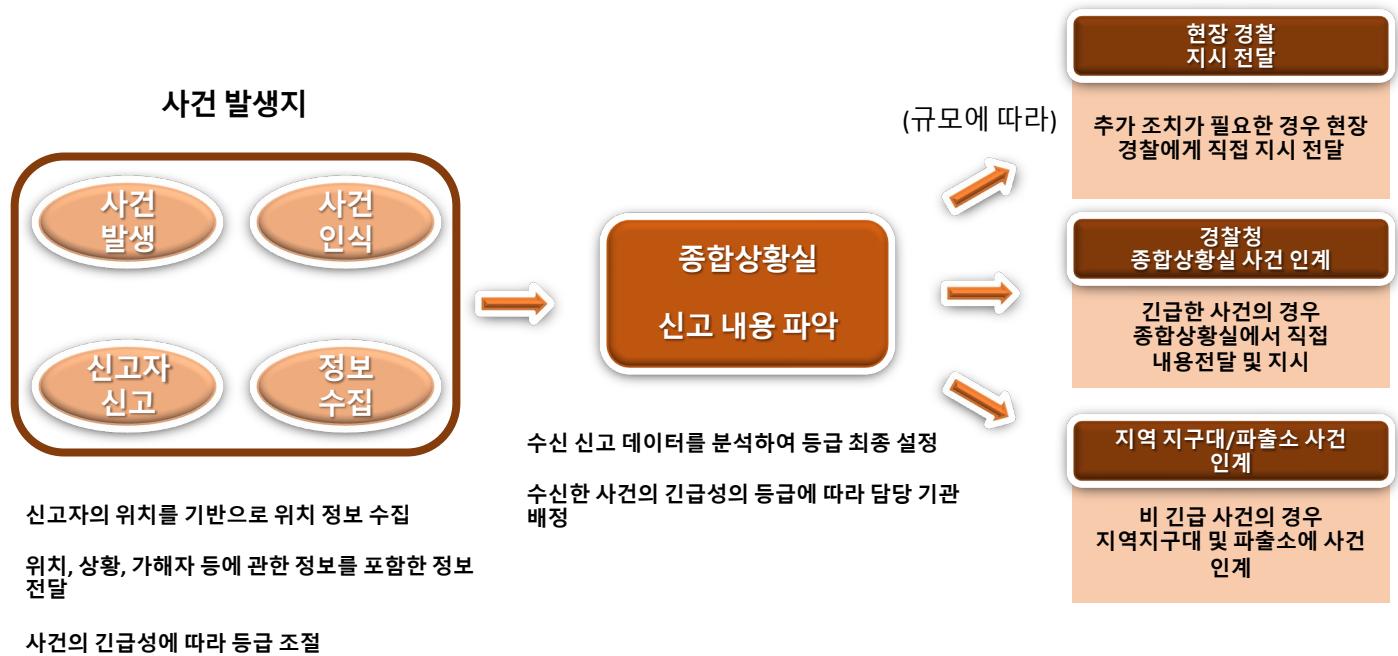
해당 Bounding box가 Object를 얼마나 많이 포함하고 있는지 정도를 말한다.

경찰 신고 메커니즘

신고자가 신고를 하면 위치 추적 시스템이 작동하여 정보를 수집한다. 이 정보는 종합상황실로 전송되며, 신고 내용은 위치, 상황, 가해자 등에 관한 정보를 포함하고 있다. 이 데이터는 사건 처리에 필요한 정보를 제공하기 위해 사용되고, 수렴된 정보는 분석되며, 긴급성에 따라 처리 등급이 조정된다.

추가 조치가 필요한 경우, 종합상황실은 현장에 있는 경찰에게 무선으로 지시를 내린다. 긴급한 사건의 경우 시·도 경찰청 112 종합상황실에서 해당 경찰서의 112 종합상황실로 내용이 전달되어 지시가 이뤄진다. 비긴급한 사건의 경우, 시·도 경찰청 상황실은 지역의 지구대 또는 파출소로 사건을 인계한 후 해당 지역 담당 경찰에서 순찰차를 지정한다.

경찰이 현장에 도착하면 초동조치가 이루어진다. 이는 현행범 체포 및 임의 동행, 현장 보존 및 증거품 수집 등과 같은 사법적 조치를 포함한다. 사건에 따라서는 전문부서로 사건이 이관되며, 추가적인 조사와 조치를 거친 후 사건이 해결된다.



4. 연구 방법 및 절차

[연구 1] Dataset을 이용한 폭행상황 분류 모델 제작

[Part. 1] Dataset 결정

폭행상황 분류 모델을 제작하기 위해서는 그 기반이 되는 Dataset이 필요하다. 특정한 작업을 위해서 관련있는 데이터를 모아놓은 것을 Dataset이라고 한다. Dataset 내의 데이터는 많을 수록 좋지만, 많은 데이터를 직접 제작하기에는 시간이 너무 많이 들기 때문에 Kaggle에 나와있는 데이터를 활용하였다. “Real Life Violence Situations Dataset”을 활용하였다. 비폭력(Nonviolence) 영상 데이터 1000개와, 폭력(Violence) 영상 데이터 1000개가 포함되어 있다.



▲ [Fig. 3] Nonviolence Video Example



▲ [Fig. 4] Violence Video Example

[Part. 2] 코드 작성

Jupyter Notebook을 이용하여 코드를 작성하였다. Kaggle 데이터셋을 이용하는 과정에서 외부 코드를 참고하였다. OpenCV, NumPy, Tensorflow, Keras 등의 라이브러리 사용하였다.

(1) 데이터 처리

```

1 IMAGE_HEIGHT , IMAGE_WIDTH = 64, 64
2
3 image_cnt = 16
4
5 Data_path = r"/Users/jiho/Desktop/personal/school/IJSHS/2023/'' 2023 일시/특기 입증자료/\
6 | | | OpenCV를 이용한 CCTV 분석 스마트 보안 시스템/violencedetection/dataset/archive/Real Life Violence Dataset"
7
8 CLASS_TYPE = ["NonViolence","Violence"]
9

```

- 학습 이미지의 해상도를 64*64로 지정한다. 또한, 한 영상에 16개의 이미지를 구한다. 인공지능은 예측하기 위해서는 숫자로 하기에 class 출력하기 위해 list로 가져오는 코드를 작성했다.

```

1 def frames_extraction(video_path):
2
3     frames_list = []
4
5     video_reader = cv2.VideoCapture(video_path)
6
7     video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
8
9     skip_frames_window = max(int(video_frames_count/image_cnt))
10
11    for frame_counter in range(image_cnt):
12
13        video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)
14
15        success, frame = video_reader.read()
16
17        if not success:
18            break
19
20        resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))
21
22        normalized_frame = resized_frame / 255
23
24        frames_list.append(normalized_frame)
25
26    video_reader.release()
27    return frames_list

```

- 분석하고자 하는 동영상에서 화면(프레임 단위)을 추출하기 위해 frames_extraction(video_path) 함수를 구현하였다.
- frames_extraction 함수는 다음과 같이 구성하였다.
 - cv2.VideoCapture(video_path)를 사용하여 동영상 파일을 읽는 video_reader 객체를 생성
 - cv2.CAP_PROP_FRAME_COUNT를 사용하여 동영상 파일의 전체 프레임 수를 가져오기
 - skip_frames_window는 동영상 파일에서 추출할 프레임 수를 결정하는 변수
 - video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)을 사용하여 현재 읽을 프레임의 위치를
 - video_reader.read()를 호출하여 현재 위치에서 다음 프레임을 읽음
 - normalized_frame 변수는 프레임의 픽셀 값을 0에서 1 사이로 정규화
 - 추출된 정규화 프레임이 저장된 frames_list를 리턴

(2) 데이터셋 생성

```

1 def create_dataset():
2
3     features = []
4     labels = []
5     video_files_paths = []
6
7     for class_index, class_name in enumerate(CLASS_TYPE):
8
9         print(f'Extracting Data of Class: {class_name}')
10
11         files_list = os.listdir(os.path.join(Data_path, class_name))
12
13         for file_name in files_list:
14
15             video_file_path = os.path.join(Data_path, class_name, file_name)
16
17             frames = frames_extraction(video_file_path)
18
19             if len(frames) == image_cnt:
20
21                 features.append(frames)
22                 labels.append(class_index)
23                 video_files_paths.append(video_file_path)
24
25         features = np.asarray(features)
26         labels = np.array(labels)
27
28
29     return features, labels, video_files_paths

```

- 분석 데이터를 생성하기 위해 `create_dataset` 함수를 다음과 같이 정의하였다.

 - 추출된 프레임데이터, 비디오 클래스 레이블, 추출된 비디오 파일 경로를 저장할 `features`, `labels`, `video_files_paths` 리스트 선언
 - 각 클래스에 대해서는 해당 클래스 폴더 내의 파일 목록을 얻기 위해 `os.listdir()`를 사용
 - `Data_path`는 데이터 폴더의 경로를 나타냄
 - `video_file_path` 변수에 비디오 파일의 전체 경로를 저장
 - `frames_extraction(video_file_path)` 함수를 호출하여 해당 비디오 파일에서 추출된 프레임 저장
 - `features` 리스트에 `len(frames) == image_cnt`에 적합한 프레임 데이터를 추가
 - `Video_file_path`를 `video_files_paths` 리스트에 추가하여 추출된 비디오 파일의 경로 기록
 - 모든 클래스와 비디오에 대한 처리가 완료되면, `features`와 `labels` 리스트를 NumPy 배열로 변환
 - `features`, `labels`, 및 `video_files_paths` 반환

```
1 features, labels, video_files_paths = create_dataset()
```

- 함수를 호출해 데이터셋의 형태로 구한다.

```

1 np.save("features.npy", features)
2 np.save("labels.npy", labels)
3 np.save("video_files_paths.npy", video_files_paths)

```

- 데이터셋을 각각 .npy 확장자로 디렉토리에 저장한다. 그 후, 이를 메모리에 등록한다.

(3) 데이터 쪼개기

- `to_categorical(labels)`를 통해 레이블을 one-hot encoding으로 변환
 - `train_test_split(features, one_hot_encoded_labels, test_size=0.1, shuffle=True, random_state=42)`: 데이터를 훈련 세트와 테스트 세트로 나눔 (train data set 90% / test data set 10%)

(4) 모델 제작 및 반환

```
1 def MobileNet_Add_LSTM_model():
2
3     model = Sequential()
4
5     model.add(Input(shape = (image_cnt, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
6
7     model.add(TimeDistributed(mobilenet))
8
9     model.add(Dropout(0.25))
10
11    model.add(TimeDistributed(Flatten()))
12
13
14    lstm_fw = LSTM(units=32)
15    lstm_bw = LSTM(units=32, go_backwards = True)
16
17    model.add(Bidirectional(lstm_fw, backward_layer = lstm_bw))
18
19    model.add(Dropout(0.25))
20
21    model.add(Dense(256,activation='relu'))
22    model.add(Dropout(0.25))
23
24    model.add(Dense(128,activation='relu'))
25    model.add(Dropout(0.25))
26
27    model.add(Dense(64,activation='relu'))
28    model.add(Dropout(0.25))
29
30    model.add(Dense(32,activation='relu'))
31    model.add(Dropout(0.25))
32
33
34    model.add(Dense(len(CLASS_TYPE),activation = 'softmax'))
35
36    model.summary()
37
38    return model
```

- 인공지능 딥러닝 모델을 MobileNet_Add_LSTM_model()로 정의하고 구현 내용은 다음과 같다
 1. model = Sequential(): 새로운 순차적인 모델을 생성
 2. model.add를 활용하여 레이어 래핑, 레이어 추가, 출력 평탄화, LSTM 레이어 추가, 출력 레이어 추가 진행
 3. lstm_fw = LSTM(units=32) 및 lstm_bw = LSTM(units=32, go_backwards=True): 순방향과 역방향의 LSTM 레이어를 생성
 - ✓ 모델은 MobileNet을 사용하여 프레임 데이터를 추출하고, Bidirectional LSTM을 사용하여 시간적 특성을 고려한 분류를 수행하는 딥러닝 모델이다. 이 모델을 컴파일하고 학습 데이터를 사용하여 학습시킨 다음, 테스트 데이터로 평가할 수 있다.

(5) 데이터셋 생성

```

1 from tensorflow.keras.callbacks import ReduceLROnPlateau
2
3 early_stopping_callback = EarlyStopping(monitor = 'val_accuracy', patience = 3, restore_best_weights = True)
4
5 reduce_lr = ReduceLROnPlateau(monitor='val_loss',
6 | | | | | factor=0.6,
7 | | | | | patience=3,
8 | | | | | min_lr=0.00005,
9 | | | | | verbose=1)
10
11 MoBiLSTM_model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ["accuracy"])
12
13 MoBiLSTM_model_history = MoBiLSTM_model.fit(x = features_train, y = labels_train, epochs = 50, batch_size = 8,
14 | | | | | shuffle = True, validation_split = 0.2, callbacks = [early_stopping_callback,reduce_lr])

```

- TensorFlow를 사용하여 생성한 딥러닝 모델(MoBiLSTM_model)을 컴파일하고 학습시키는 과정

- from tensorflow.keras.callbacks import ReduceLROnPlateau로 TensorFlow의 콜백 중 하나인 ReduceLROnPlateau 콜백을 불러옴
- early_stopping_callback로 조기 종료 콜백을 설정
- restore_best_weights=True로 설정되어 가장 좋은 성능을 가진 epochs 의 가중치를 복원
- reduce_lr로 학습률 감소 콜백을 설정
- MoBiLSTM_model.compile()로 모델을 컴파일
- MoBiLSTM_model.fit()로 모델을 학습
- 모델의 학습 과정은 MoBiLSTM_model_history에 저장되며, 이를 사용하여 학습 곡선 및 성능을 시각화 및 분석

```

Epoch 1/50
180/180 [=====] - 29s 147ms/step - loss: 0.6662 - accuracy: 0.6062 - val_loss: 0.7614 - val_accuracy: 0.5139 - lr: 0.0010
Epoch 2/50
180/180 [=====] - 25s 140ms/step - loss: 0.4761 - accuracy: 0.8021 - val_loss: 0.3610 - val_accuracy: 0.8556 - lr: 0.0010
Epoch 3/50
180/180 [=====] - 24s 135ms/step - loss: 0.4041 - accuracy: 0.8333 - val_loss: 0.3640 - val_accuracy: 0.8583 - lr: 0.0010
Epoch 4/50
180/180 [=====] - 24s 136ms/step - loss: 0.3751 - accuracy: 0.8472 - val_loss: 0.3798 - val_accuracy: 0.8583 - lr: 0.0010
Epoch 5/50
180/180 [=====] - ETA: 0s - loss: 0.3758 - accuracy: 0.8493
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.000600000284984708.
180/180 [=====] - 24s 132ms/step - loss: 0.3758 - accuracy: 0.8493 - val_loss: 0.4528 - val_accuracy: 0.7861 - lr: 0.0010
Epoch 6/50
180/180 [=====] - 25s 136ms/step - loss: 0.2934 - accuracy: 0.8889 - val_loss: 0.3477 - val_accuracy: 0.8861 - lr: 6.0000e-04
Epoch 7/50
180/180 [=====] - 25s 138ms/step - loss: 0.2531 - accuracy: 0.9111 - val_loss: 0.3194 - val_accuracy: 0.8917 - lr: 6.0000e-04
Epoch 8/50
180/180 [=====] - 24s 136ms/step - loss: 0.2321 - accuracy: 0.9187 - val_loss: 0.2198 - val_accuracy: 0.9250 - lr: 6.0000e-04
Epoch 9/50
180/180 [=====] - 25s 136ms/step - loss: 0.1984 - accuracy: 0.9382 - val_loss: 0.2316 - val_accuracy: 0.9250 - lr: 6.0000e-04
Epoch 10/50
180/180 [=====] - 25s 137ms/step - loss: 0.1840 - accuracy: 0.9403 - val_loss: 0.2726 - val_accuracy: 0.8972 - lr: 6.0000e-04
Epoch 11/50
180/180 [=====] - ETA: 0s - loss: 0.1718 - accuracy: 0.9403
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0003600000178999825.
180/180 [=====] - 24s 136ms/step - loss: 0.1718 - accuracy: 0.9403 - val_loss: 0.2974 - val_accuracy: 0.9111 - lr: 6.0000e-04

```

- 학습 결과는 다음과 같음을 알 수 있다. ReduceLROnPlateau가 정상적으로 작동해 learning rate를 확 낮춤
- 실제 학습 결과를 살펴보면 손실함수는 0.1718, 정확도는 0.9403 정도 나오는 것을 볼 수 있고, 검증결과 손실함수는 0.2974, 정확도는 0.9111 정도가 나오는 것을 확인할 수 있음

```

1 model_evaluation_history = MoBiLSTM_model.evaluate(features_test, labels_test)
2
3 7/7 [=====] - 2s 177ms/step - loss: 0.2982 - accuracy: 0.9000

```

- 다음 코드는 학습한 모델을 실제로 평가하는 과정이다.
- 정확도는 0.9000, 손실함수는 0.2982가 나오는 것을 볼 수 있다.

(6) 모델 평가

```

1 def model_predict_histroy(model_training_history, metric_name_1, metric_name_2, plot_name):
2
3     metric_value_1 = model_training_history.history[metric_name_1]
4     metric_value_2 = model_training_history.history[metric_name_2]
5
6     epochs = range(len(metric_value_1))
7
8     plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
9     plt.plot(epochs, metric_value_2, 'orange', label = metric_name_2)
10
11    plt.title(str(plot_name))
12
13    plt.legend()

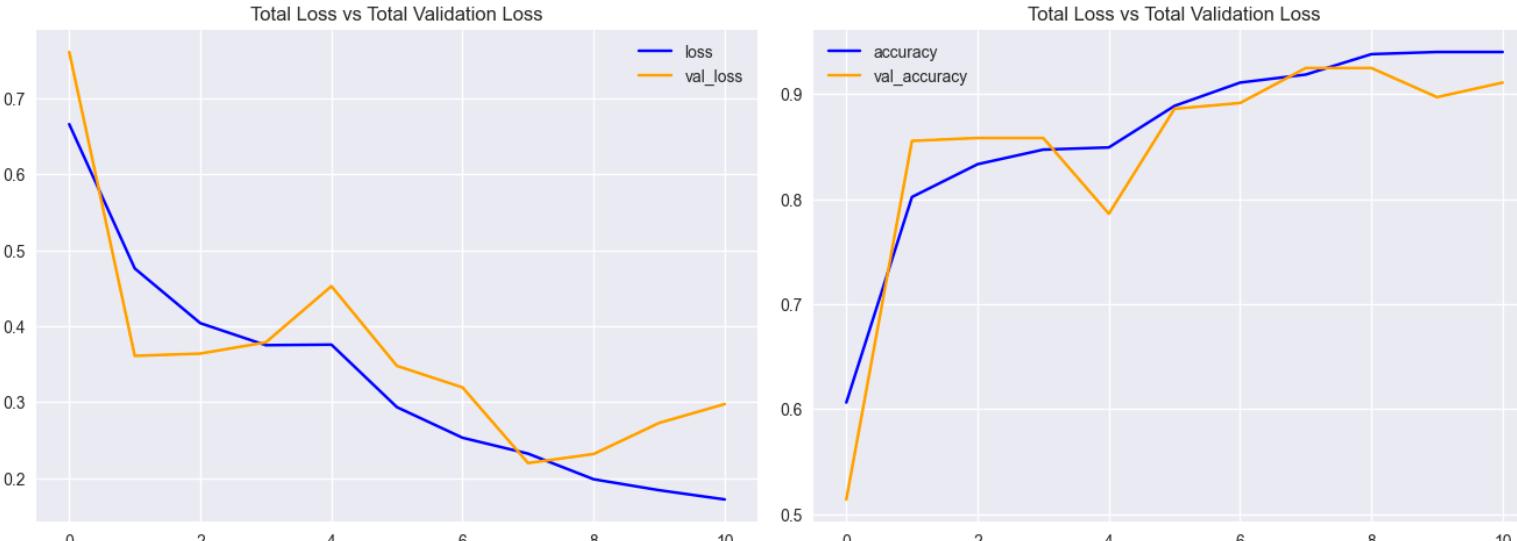
```

1. `model_training_history.history[metric_name_1]` 및 `model_training_history.history[metric_name_2]`를 사용하여 `model_training_history`에서 지정한 `metric_name_1`과 `metric_name_2`의 값을 추출
2. epoch의 수를 `epochs`라는 변수에 저장
3. `plt.plot(epochs, metric_value_1, 'blue', label=metric_name_1)`와 `plt.plot(epochs, metric_value_2, 'orange', label=metric_name_2)`를 사용하여 두 개의 메트릭 값을 시각화
4. `plt.legend()`를 호출하여 그래프에 범례(legend)를 추가한다. 범례는 어떤 선이 어떤 메트릭을 나타내는지 표현

```

1 model_predict_histroy(MobBiLSTM_model_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
1 plot_metric(MobBiLSTM_model_history, 'accuracy', 'val_accuracy', 'Total Loss vs Total Validation Loss')

```



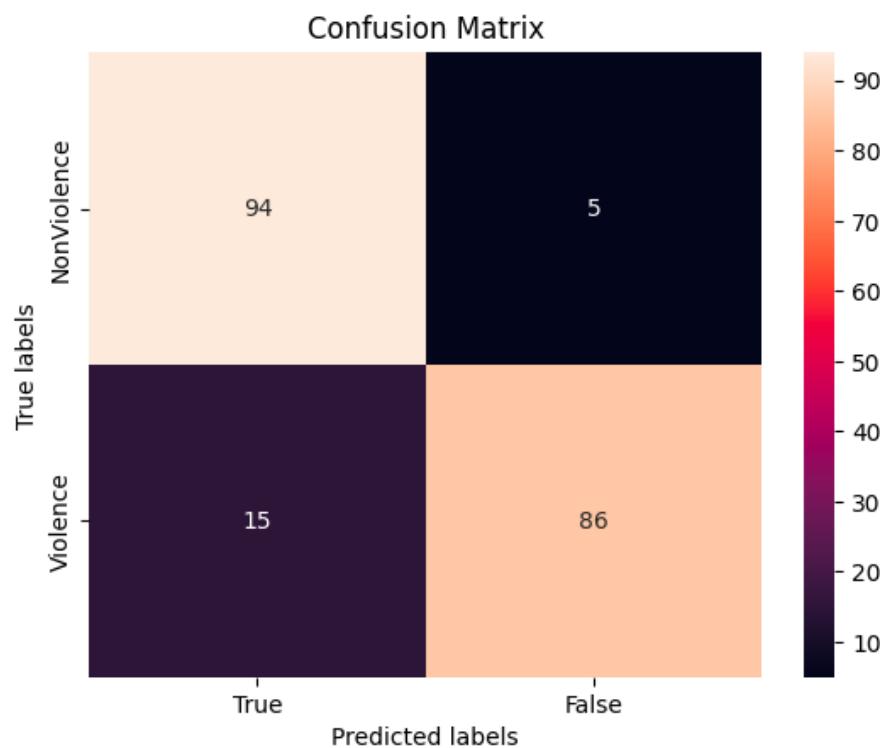
- 훈련 손실(loss)과 검증 손실(val_loss) 그리고 훈련 정확도(accuracy)와 검증 정확도(val_accuracy)와 같은 두 개의 메트릭을 추출하고 이를 시각화하는 역할을 한다. 이 함수를 통해 학습 및 검증 세트에 대한 손실 또는 정확도의 추이를 시각화할 수 있다.

(6) 추가 시각화

```

1 import seaborn as sns
2 from sklearn.metrics import confusion_matrix
3
4 ax=plt.subplot()
5 cm=confusion_matrix(labels_test_normal, labels_predict)
6 sns.heatmap(cm, annot=True, fmt='g', ax=ax);
7
8 ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
9 ax.set_title('Confusion Matrix');
10 ax.xaxis.set_ticklabels(['True', 'False']); ax.yaxis.set_ticklabels(['NonViolence', 'Violence']);

```



- 모델 성능 평가를 위해 Python에서 seaborn과 scikit-learn을 사용하여 혼동 행렬(Confusion Matrix)을 시각화 진행
- ✓ 혼동 행렬이란 분류 모델의 성능을 평가하는 데 사용되며, 실제 클래스와 예측된 클래스 간의 일치와 불일치를 표시하는 표이다.
- 1. Seaborn 및 sklearn.metrics 라이브러리 임포트
- 2. ax=plt.subplot()은 그래프의 서브플롯(subplot)을 생성
- 3. cm=confusion_matrix(labels_test_normal, labels_predict)은 confusion_matrix 함수를 사용하여 혼동 행렬을 계산하고 cm 변수에 저장
- 4. sns.heatmap(cm, annot=True, fmt='g', ax=ax)는 seaborn의 heatmap 함수를 사용하여 혼동 행렬을 heatmap으로 시각화
- 5. ax.set_xlabel('Predicted labels'); ax.set_ylabel('True labels')는 x축과 y축의 라벨(축의 이름)을 설정
- 6. ax.set_title('Confusion Matrix')는 그래프의 제목을 설정
- 7. ax.xaxis.set_ticklabels(['True', 'False']); ax.yaxis.set_ticklabels(['NonViolence', 'Violence'])는 x축과 y축의 눈금 레이블을 설정
- 이렇게 시각화된 혼동 행렬을 통해 모델이 어떤 클래스를 어떻게 예측하고 있는지를 쉽게 확인할 수 있고 혼동 행렬은 주로 이진 분류 또는 다중 클래스 분류 문제에서 모델의 성능을 평가하는 데 사용

(7) 모델 추론 및 시각화

```

1 def predict_frames(video_file_path, output_file_path, image_cnt):
2
3     video_reader = cv2.VideoCapture(video_file_path)
4
5     original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
6     original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))
7
8     video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc('m', 'p', '4', 'v'),
9                                     video_reader.get(cv2.CAP_PROP_FPS), (original_video_width, original_video_height))
10
11     frames_queue = deque(maxlen = image_cnt)
12
13     predicted_class_name = ''
14
15     while video_reader.isOpened():
16
17         ok, frame = video_reader.read()
18
19         if not ok:
20             break
21
22         resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))
23
24         normalized_frame = resized_frame / 255
25
26         frames_queue.append(normalized_frame)
27
28         if len(frames_queue) == image_cnt:
29
30             predicted_labels_probabilities = MoBiLSTM_model.predict(np.expand_dims(frames_queue, axis = 0))[0]
31
32             predicted_label = np.argmax(predicted_labels_probabilities)
33
34             predicted_class_name = CLASS_TYPE[predicted_label]
35
36             if predicted_class_name == "Violence":
37                 cv2.putText(frame, predicted_class_name, (5, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0), 12)
38             else:
39                 cv2.putText(frame, predicted_class_name, (5, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 255), 12)
40
41             video_writer.write(frame)
42
43     video_reader.release()
44     video_writer.release()

```

- 폭력 상황에 대한 예측 및 시각화를 위해 predict_frames 함수 정의

1. `video_reader = cv2.VideoCapture(video_file_path)`는 주어진 `video_file_path`를 사용하여 OpenCV의 VideoCapture 객체 생성
2. `video_writer`를 설정하여 비디오 파일을 출력하기 위한 VideoWriter 객체 생성
3. `frames_queue = deque(maxlen=image_cnt)`는 프레임을 저장하고 관리하기 위한 queue를 생성
4. `Predicted_class_name`은 현재 예측된 클래스 이름을 저장하는 변수
5. `while video_reader.isOpened():`는 비디오 파일을 열고 읽어오는 루프 시작
6. `ok, frame = video_reader.read()`는 `video_reader`를 사용하여 비디오에서 다음 프레임을 읽기
7. 프레임을 `IMAGE_HEIGHT`와 `IMAGE_WIDTH`로 크기 조정하고, 이미지 픽셀 값을 $[0, 1]$ 범위로 조정하는 normalization를 수행
8. `frames_queue.append(normalized_frame)`는 queue에 현재 프레임을 추가한다. queue의 최대 길이를 유지하면서 가장 오래된 프레임이 제거
9. `predicted_labels_probabilities`는 MoBiLSTM_model을 사용하여 현재 queue에 대한 예측 확률을 계산한 결과
10. `Predicted_label`은 가장 높은 확률을 갖는 클래스의 인덱스
11. `predicted_class_name`은 예측된 클래스
12. `video_writer.write(frame)`은 예측 결과가 추가된 프레임을 출력 비디오 파일에 기록
- `predict_frames` 비디오 파일에서 프레임을 읽어와 딥러닝 모델을 사용하여 각 프레임의 클래스를 예측하고, 예측 결과가 표시된 비디오 파일을 생성

```

1 plt.style.use("default")
2
3 def show_pred_frames(pred_video_path):
4
5     plt.figure(figsize=(20,15))
6     video_reader = cv2.VideoCapture(pred_video_path)
7
8     frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
9
10    random_range = sorted(random.sample(range(image_cnt), 12))
11
12    for counter, random_index in enumerate(random_range, 1):
13
14        plt.subplot(5, 4, counter)
15
16        video_reader.set(cv2.CAP_PROP_POS_FRAMES, random_index)
17
18        ok, frame = video_reader.read()
19
20        if not ok:
21            break
22
23        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
24
25        plt.imshow(frame);ax.figure.set_size_inches(20,20);plt.tight_layout()
26
27    video_reader.release()

```

- 예측 결과 비디오 파일에서 임의의 프레임을 선택하여 시각적으로 표시하는 기능을 수행하는 함수인 show_pred_frames 정의

 1. plt.style.use("default")는 Matplotlib에서 사용할 스타일을 default로 설정
 2. plt.figure(figsize=(20,15)) 새로운 Matplotlib 그림을 생성하고, 그림의 크기를 20 * 15로 지정
 3. video_reader = cv2.VideoCapture(pred_video_path)는 주어진 pred_video_path로 지정된 예측 결과 비디오 파일을 읽어오기 위해 OpenCV의 VideoCapture 객체를 생성
 4. frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))는 비디오의 프레임 수를 추출하여 frames_count 변수에 저장
 5. random_range = sorted(random.sample(range(image_cnt), 12))는 0부터 image_cnt-1까지의 범위에서 중복되지 않는 무작위 숫자 12개를 선택하고, 이를 정렬하여 random_range에 저장
 6. counter는 루프의 반복 횟수를 나타내고, random_index는 선택된 무작위 인덱스
 7. plt.subplot(5, 4, counter)는 Matplotlib에서 5행 4열의 그리드 형태로 여러 개의 subplot를 생성
 8. video_reader.set(cv2.CAP_PROP_POS_FRAMES, random_index)는 VideoCapture 객체의 현재 프레임 위치를 random_index로 설정하여 해당 프레임을 읽어옴
 9. ok, frame = video_reader.read()는 VideoCapture 객체를 사용하여 현재 프레임을 읽어옴
 10. frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)은 OpenCV의 BGR 색상 형식으로 읽어온 프레임을 RGB 색상 형식으로 변환
 11. plt.imshow(frame); ax.figure.set_size_inches(20, 20); plt.tight_layout()는 현재 프레임을 Matplotlib를 사용하여 시각화

```

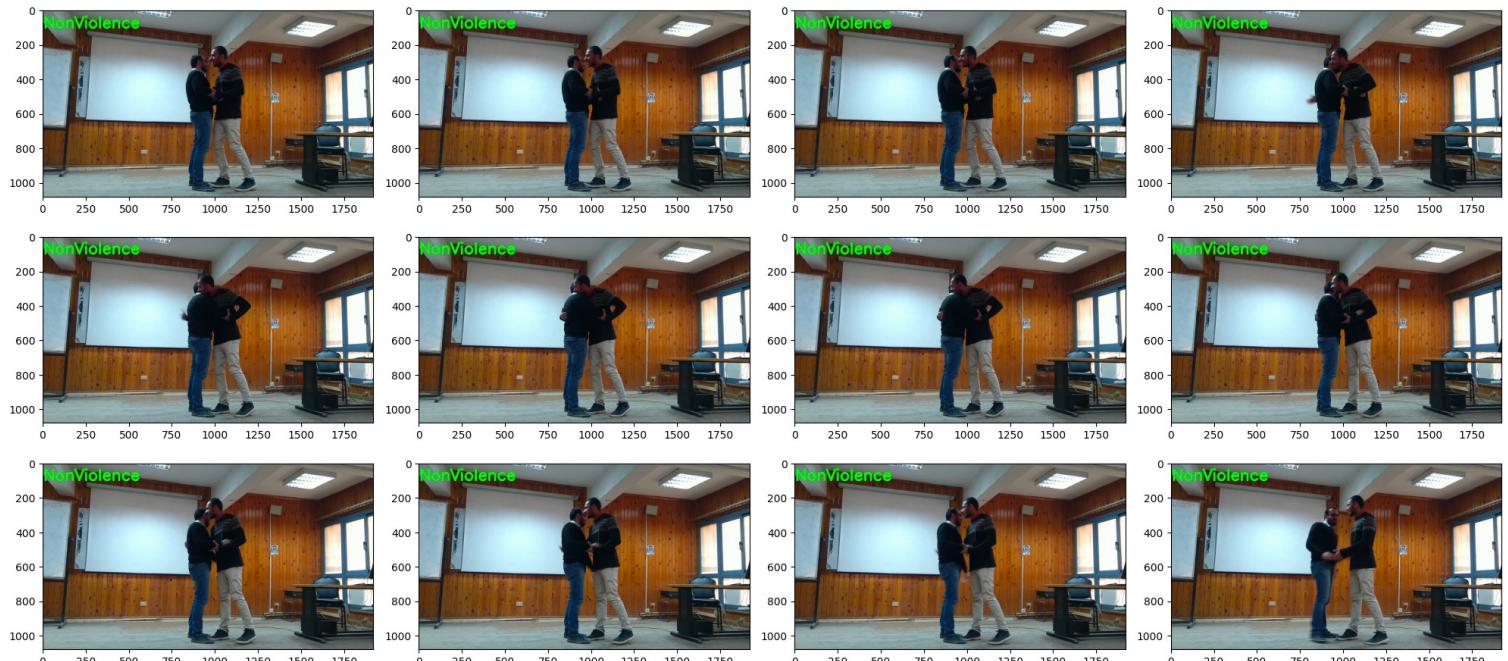
1 input_video_file_path = "/Users/jiho/Desktop/personal/school/IJSHS/2023/'' 2023 입시/특기 입증자료/\\
2                                         OpenCV를 이용한 CCTV 분석 스마트 보안 시스템/violencedetection/dataset/archive/Real Life Violence Dataset/Violence/V_374.mp4"
3
4 predict_frames(input_video_file_path, output_video_file_path, image_cnt)
5
6 show_pred_frames(output_video_file_path)

```

1. input_video_file_path로 지정된 비디오 파일을 읽음
2. 각 프레임을 크기 조정하고 normalization하고, 딥러닝 모델을 사용하여 각 프레임의 클래스를 예측
3. 예측된 클래스를 시각화하여 프레임에 추가
4. 예측 결과가 저장된 비디오 파일(output_video_file_path)에 정보 추가
5. 저장된 예측 결과 비디오 파일을 플레이어를 사용하여 시각화



- ✓ 폭력 영상을 입력하였고, 그것을 시각화한 결과는 다음과 같다. 중간에 하나의 오류가 생기는 것을 볼 수 있는데 이는 프레임을 자르는 과정에서 생기는 오류로 볼 수 있음



- ✓ 이번에는 아까와 다른 비폭력 영상을 입력하였고, 시각화한 결과는 다음과 같다. 이번엔 오류 없이 매끄럽게 진행되는 것을 볼 수 있음

(8) 비디오 예측

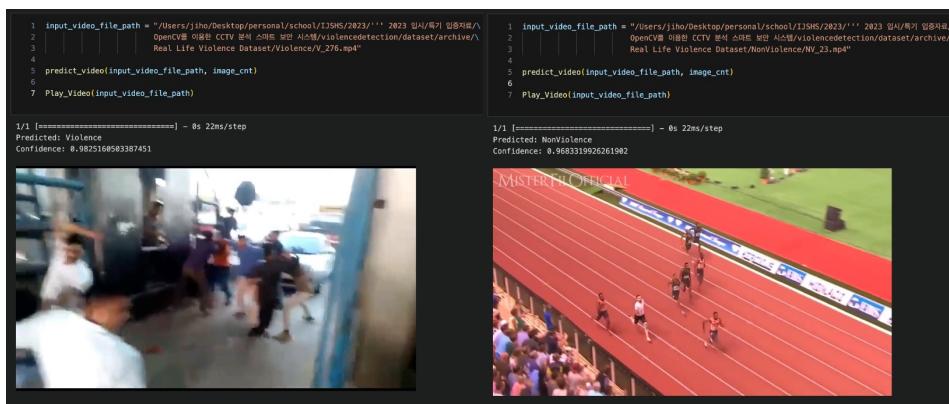
```

1 def predict_video(video_file_path, image_cnt):
2     video_reader = cv2.VideoCapture(video_file_path)
3
4     original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
5     original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))
6
7     frames_list = []
8
9     predicted_class_name = ''
10
11    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
12
13    skip_frames_window = max(int(video_frames_count / image_cnt))
14
15    for frame_counter in range(image_cnt):
16
17        video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)
18
19        success, frame = video_reader.read()
20
21        if not success:
22            break
23
24        resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))
25
26        normalized_frame = resized_frame / 255
27
28        frames_list.append(normalized_frame)
29
30    predicted_labels_probabilities = MoBiLSTM_model.predict(np.expand_dims(frames_list, axis=0))[0]
31
32    predicted_label = np.argmax(predicted_labels_probabilities)
33
34    predicted_class_name = CLASS_TYPE[predicted_label]
35
36    print(f'Predicted: {predicted_class_name}\nConfidence: {predicted_labels_probabilities[predicted_label]}')
37
38    video_reader.release()

```

- 주어진 비디오 파일로부터 일정 간격으로 추출한 프레임들을 사용하여 MoBiLSTM_model을 사용하여 폭력 여부를 예측하는 함수인 predict_video를 정의
1. predict_video(video_file_path, image_cnt)는 video_file_path로 지정된 비디오 파일에서 image_cnt 만큼의 프레임을 추출하고, 추출한 프레임들을 사용하여 비디오의 클래스를 예측
 2. video_reader = cv2.VideoCapture(video_file_path)는 OpenCV의 VideoCapture 객체를 사용하여 video_file_path로 지정된 비디오 파일 열기
 3. Original_video_width와 original_video_height 변수에는 원래 비디오의 너비와 높이가 저장
 4. frames_list = []는 추출한 프레임들을 저장하기 위한 빈 리스트인 frames_list를 생성
 5. skip_frames_window` 변수는 비디오의 총 프레임 수를 사용하여 추출할 프레임 간격을 계산
 6. video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)는 VideoCapture 객체를 사용하여 현재 프레임 위치를 설정한다. 이 위치는 추출할 프레임의 위치를 결정하는 데 사용
 7. success, frame = video_reader.read()는 VideoCapture 객체를 사용하여 현재 위치에 있는 프레임을 읽어옴
 8. Resized_frame은 읽어온 프레임을 지정된 크기로 크기 조정하고, normalized_frame은 normalize된 프레임을 나타냄
 9. Normalized_frame을 frames_list에 추가
 10. 모든 프레임을 추출하고 frames_list에 저장한 후, MoBiLSTM_model을 사용하여 frames_list의 프레임들에 대한 클래스 예측을 수행
 11. predicted_label은 예측 결과 중 가장 높은 확률을 갖는 클래스의 인덱스를 나타냄

(9) 성능 확인



- 학습에 사용하지 않은 Violence와 NonViolence 영상을 입력하였더니, 각각 다음과 같은 정확도를 얻을 수 있었다.

[연구 2] Dataset으로 YOLO 학습시키기

[Part. 1] Object Detection

우리가 YOLO를 통해 detect해야 하는 object는 현실에서 충분히 일어날 수 있는 폭력성 범죄에 사용되는 물품들이다. 대표적인 것으로 칼, 각목, 쇠파이프 등이 있으며, 총과 같은 object는 개인의 총기 소지가 불가능한 대한민국 내에서 적용하기에는 적합하지 않다고 판단하였기에 제외하였다.

CCTV, IP카메라 등 영상을 받아들여, 그 영상 안에 우리가 정한 object가 들어가 있을 경우 잠재적으로 범죄의 위험성이 있을 것이라고 판단한다. 만약 그 영상 안에 내가 정한 object가 없을 경우 범죄의 위험성이 낮다고 판단한다.

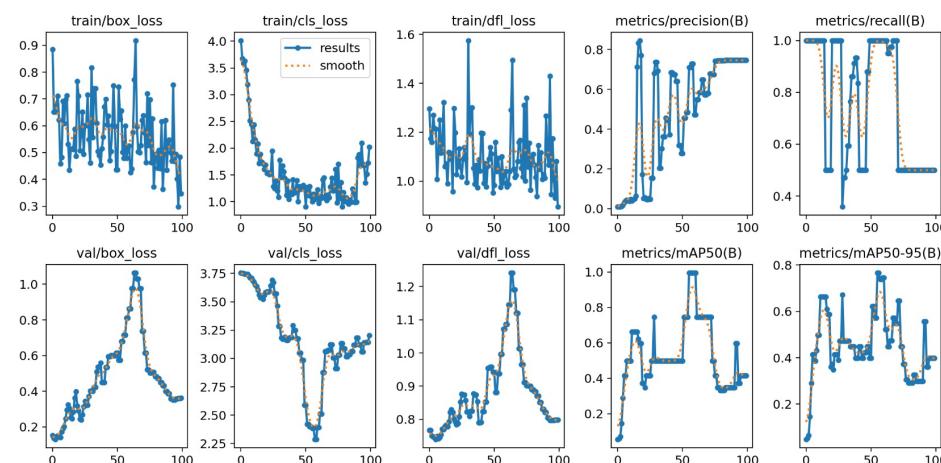
```

1 from ultralytics import YOLO
2 import torch
3
4 device = torch.device('mps:0' if torch.backends.mps.is_available() else 'cpu')
5
6 model = YOLO("yolov8n.pt")
7
8 model.train(data = r"/Users/jiho/Desktop/personal/school/IJSHS/2023/2023 입시/" 
9 특기 입증자료/OpenCV를 이용한 CCTV 분석 스마트 보안 시스템/attack object detection/" 
10 OD-WeaponDetection-master/Weapons and similar handled objects/dataset.yaml", epochs=100, imgsz = 640, device = "mps")
11

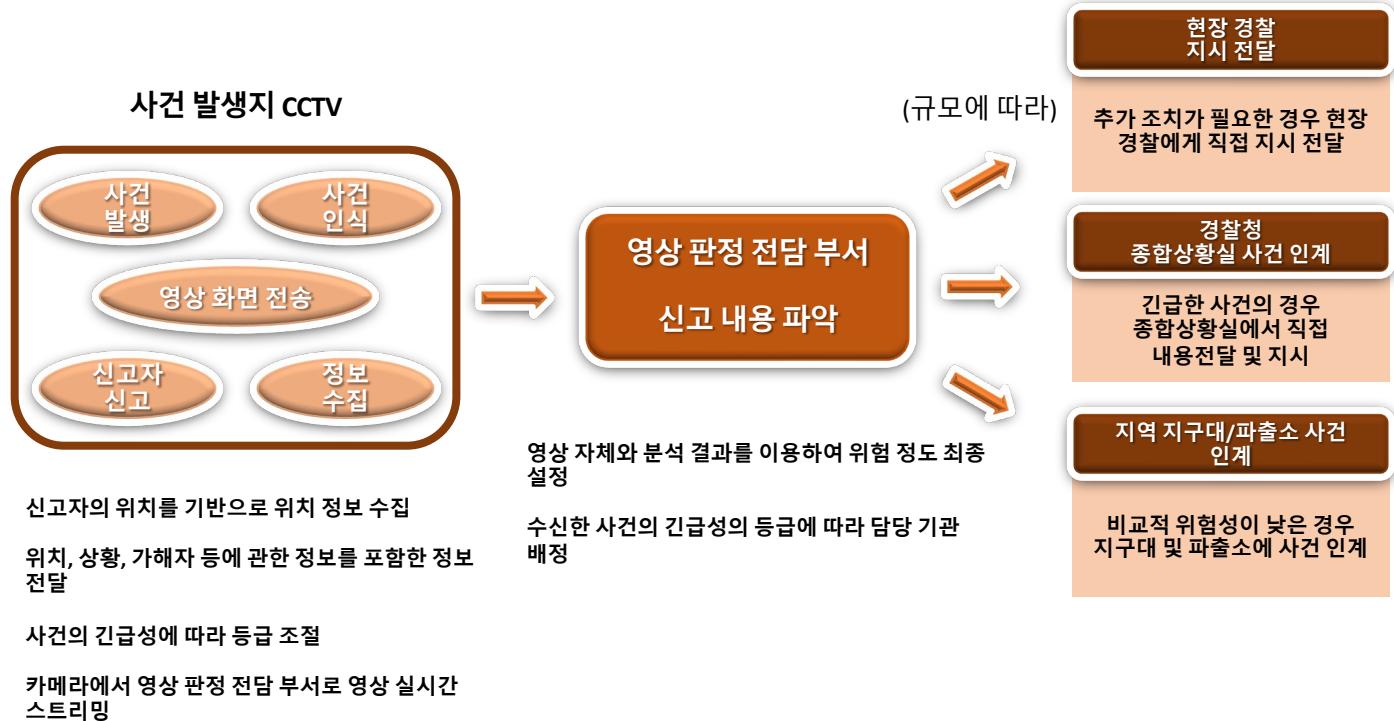
```

- Ultralytics 라이브러리에서 YOLO 모델을 불러옴

- Device = torch.device(' mps:0 ' if torch.backends.mps.is_available() else ' cpu ')로 PyTorch의 torch.device를 사용하여 딥러닝 모델을 실행할 장치를 선택
- torch.backends.mps.is_available() 함수를 사용하여 여러 디바이스 중에서 mps를 사용할 수 있는지 확인
- model = YOLO("yolov8n.pt")로 Ultralytics의 YOLO 클래스를 사용하여 YOLO 모델을 초기화
- Model.train(data, epochs=100, imgsz=640, device= " mps ")로 YOLO 모델을 훈련



[연구 4] 신고 메커니즘



앞서 이론적 배경에서 지금까지의 경찰 신고 메커니즘에 대해 서술한 바 있다. 하지만 그러한 방식이 지금 본 연구에서는 그렇게 효율성이 좋지 못하다고 느꼈다.

사람이 일일히 많은 CCTV를 모니터링하기에는 굉장히 많은 인력이 투입되어야 하는 것이 지금의 현실이고 이것을 인공지능의 역할로 충분히 대체 할 수 있고, 인력의 부담을 줄일 수 있을 것이라고 생각한다. 당연히 사람이 판단하는 것보다 인공지능이 판단하는 것이 더 즉각적일 수는 있겠지만, 한 가지 분명한 것은 인공지능의 분석도 오류가 생길 수 있다는 점은 간과하지 말아야 한다는 것이다. 그래서 자동 신고 시스템을 만들되, 인공지능 CCTV 영상 판정 전담 부서를 신설하고, 신설된 부서에서는 CCTV 알고리즘에서 자동으로 신고하는 경우를 2차 판정하는 업무를 담당하는 것으로 생각해보았다.

CCTV는 위험성이 감지된 시점과 그 영상을 계속해서 보내주는 역할을 한다. 영상을 보내주면 전담 부서에서 영상을 판정하고 만약 위험성이 존재한다는 판단을 내리면 수신된 CCTV 영상과 함께 종합상황실에 접수를 하고 그 이후 사건 진행은 종합상황실로 넘기는 것까지 인공지능 CCTV 영상 판정 전담 부서가 할 일이다.

인공지능이 내린 판단을 신뢰할 수 있는가에 대한 의문점이 충분히 생길 수 있다는 점도 인지하고 있다. 하지만, 그 문제는 앞서 오류가 생길 수 있기 때문에 전담 부서에서 영상을 판정하는 것으로 해결할 수 있다.

5. 결론 및 제언

앞선 연구 과정을 통해서 충분히 분석 가능하다는 사실을 확인할 수 있었다. 인공지능 학습 모델을 통해서 대략 90% 정도의 정확도가 나오는 것을 보아 충분히 잘 나온 모델임을 알 수 있다. YOLO 학습 후에도 흥기의 인식이 잘 되는 것을 보았기 때문에 그들을 잘 조합하기만 한다면 우리는 제대로 된 시스템을 구축할 수 있을 것이다.

연구 과정 및 결과를 기반으로 하여 시스템을 개선할 수 있는 몇 가지를 소개하려고 한다. 모델의 성능을 향상시키려면 더 많은 훈련 데이터를 수집하고 정제해야 한다. 특히 흥기 인식과 관련된 데이터를 더 다양하고 대표적으로 수집하여 모델의 일반화 능력을 향상시킬 수 있다.

개선된 모델을 다양한 테스트 데이터셋에서 평가하여 일반화 능력을 확인해야 한다. 정확도 외에도 모델의 속도, 실시간 처리 능력, 혼동 행렬 등을 고려하여 모델의 전반적인 성능을 평가해야 한다.

연구가 끝나고 이 프로젝트를 웹 플랫폼에 연계시킬 수 있을 것 같다는 생각이 들었다. 많은 사람들에게 정보를 전달하는 플랫폼은 상당히 매력적으로 다가왔고, 공공의 이익이기 때문에 전혀 나쁠 이유가 없었다. 추가 연구를 진행한 후 이를 웹 플랫폼에 연계하여 지역별 다양한 영상을 제공하고 싶다.

- [1] <https://www.kaggle.com/code/abduulrahmankhalid/real-time-violence-detection-mobilenet-bi-lstm>
- [2] 조나래. (2017) 딥러닝 기반 범죄자 신원 인식 시스템 설계 및 구현. 가천대학교
- [3] 이용주. (2022). 딥 러닝 기반의 CCTV를 활용한 이상 행동 감지 시스템 설계. 한국지식정보기술학회 논문지, 17(2), 183-191.
- [4] <https://www.boannews.com/media/view.asp?idx=107850&kind=3>
- [5] <https://www.data.go.kr/data/15075538/fileData.do>