광진구
GWANGJIN-GU

YOLO 및 ResNet를 이용한 **실시간 CCTV 감시 범죄 발생 예방 솔루션** 개발 제안

2023년 광진구 빅데이터 분석 공모전 출품

세상에 좋은 폭행은 없다

KT AIVLE School 곽우석
KT AIVLE School 이민혁
KT AIVLE School 장재형

목차

01. 프로젝트 개요

가. 시스템 개요 1

02. 프로젝트 설계

가. 개발 환경 2

나. Why YOLO 3

다. 설계 방향 4

라. 시스템 시나리오 5

03. 프로젝트 구현

가. 데이터셋 수집 6

나. 데이터셋 분류 7

다. 데이터셋 학습 9

라. 프로그램 구현 11

04. 결론

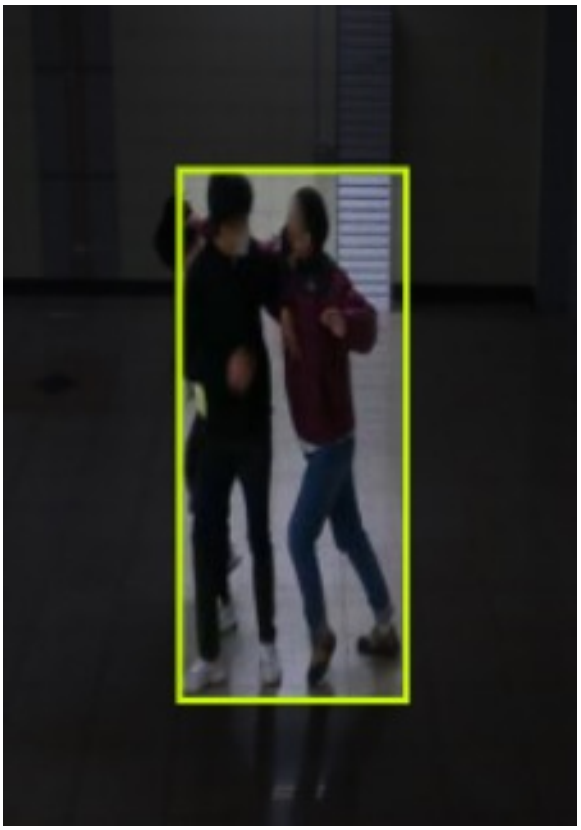
가. 시연 영상 13

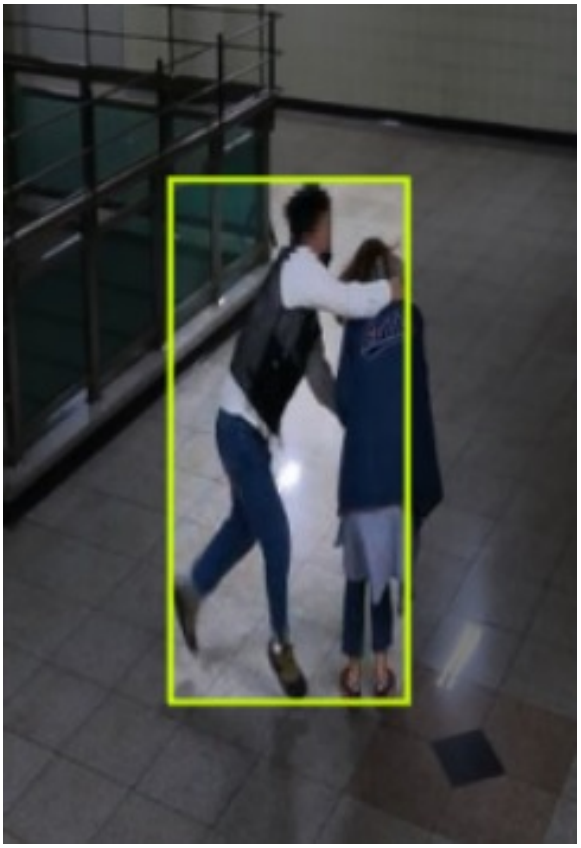
나. 성능 지표 14

다. 기대효과 15

프로젝트 개요

시스템 개요





확보한 폭행 관련 샘플 데이터를
AI 모델에게 반복적으로 학습시키고
치안 관계기관과의 유기적 연결을 통해
이상 패턴이 감지되면 빠르게 확인할 수 있도록 하여
범죄 현장에 대한 빠른 대응은 물론
CCTV 감시인력의 육체적 피로도를 줄일 수 있음.

프로젝트 설계

개발 환경

라이브러리 (Tensorflow)

텐서플로는 주로 이미지 인식이나 반복 신경망 구성, 기계 번역, 필기 숫자 판별 등을 위한 각종 신경망 학습에 사용됨. 특히 대규모 예측 모델 구성에 뛰어나 테스트부터 실제 서비스까지 사실상 거의 모든 딥러닝 프로젝트에서 범용적으로 활용할 수 있음
이 밖에도 추상화 수준이 높아 텐서플로를 활용하면 개발자는 알고리즘의 세세한 구현보다 전체적인 논리 자체에 더 집중할 수 있게 되며 확장성 또한 뛰어남.

인공지능 (YOLO)

Yolov5는 모델 탐지의 정확도가 높고, 초당 최대 140 프레임으로, 연산속도가 매우 빠름
: Yolov5 네트워크는 1. 높은 탐지 정확도 2. 가벼움 3. 빠른 탐지속도
의 장점을 가지고 있기 때문에 기민한 대처를 요구하는 범죄 대응에 장점을 가짐

서버 (Google Colab)

구글 코랩(Google Colab)은 클라우드 기반의 플랫폼으로,
머신 러닝과 데이터 분석을 위한 무료 Jupyter 노트북 환경을 제공함.
본 모델에서는 학습에 Colab Pro에서 지원하는 Tesla T4 Tensor Core GPU를 이용하였음.

프로젝트 설계

Why YOLO

특징은 실시간으로 객체를 탐지 할 수 있는 것이다. YOLO가 유명해진 이유는 높은 성능은 아니더라도 준수한 성능으로 실시간으로 Object Detection이 가능했기 때문이다. 기존의 Faster R-CNN보다 6배 빠른 성능을 보여준다.

Darknet 모델이 탑재되어 디텍팅에 용이하여 실시간으로 관리자가 확인할 수 있다.

Detext.py 파일과 train.py 등 파일 등을 수정하여 설계자가 원하는 방향으로 변경이 가능하다.

프로젝트 설계

설계 방향

Yolo의 detect.py를 수정하여 디텍팅 된 클래스 중 설계자가 원하는 클래스가 디텍팅 된 사진을 따로 저장하여 2차로 ResNet을 이용하여 최종 저장.
사고를 당한 사람이 빠르게 사건 처리가 용이하도록 설계.

학습된 모델에 실시간 CCTV 영상을 입력했을 때 기존에 학습되어 있던 범죄 장면과 00% 이상 유사하다고 판단하면, 해당 장면과 감지된 CCTV 정보를 치안 관계부서에 즉각적으로 전달하도록 하여 빠르게 범죄를 예방할 수 있도록 함

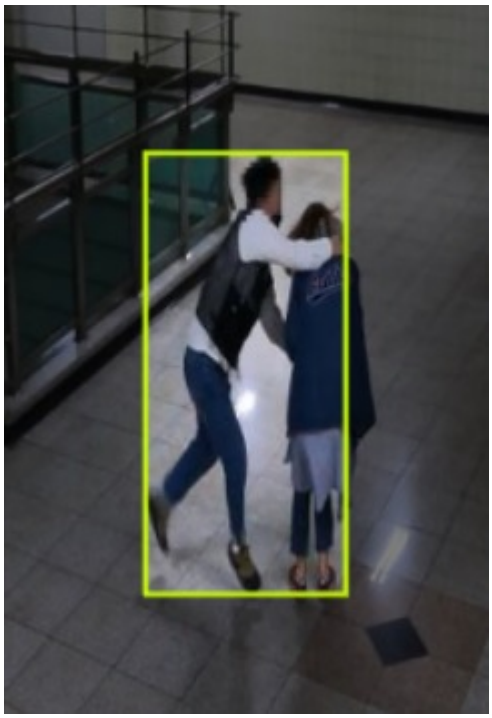
또한 즉각적으로 반응을 하지 못했을 경우 자동으로 저장된 이미지 파일을 본 뒤 범죄를 예방 할 수 있도록 함

프로젝트 설계

시스템 시나리오

솔루션 내장 CCTV

촬영됨



상시 영상 전송

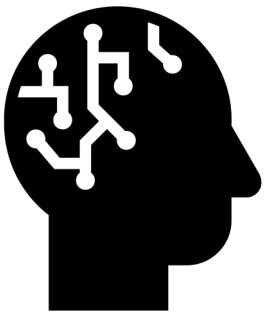
서버 PC

범죄 현장

영상 입력

탐지 결과 출력

치안서비스 제공



비정상 행동 감지 사실 전송

CCTV 정보 및 인식된 사진

솔루션 AI 모델

치안 관계부서

프로젝트 구현

데이터셋 수집

Train 데이터

AI hub 지하철 역사 내 CCTV이상행동 영상의 폭행 이미지 및 라벨 사용

-출처 https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100

What we did

1. 폭행 사진의 라벨을 수정하여 보았다.
싸우는 사람의 전체적인 이미지와 부분적인 이미지를 비교해 보았을 때 폭행 이미지를 라벨링 하는 것보다 행위에 집중을 하는 것 이 분류 정확도에 더 좋은 결과를 보였다.

2. ResNet 모델을 이용하려고 하였으나 실시간 영상학습과 디텍팅 부분에서 아쉬운 결과를 보임
-> Resnet을 2차 가공에 활용하여 더 정확한 모델링에 사용

프로젝트 구현

데이터셋 분류

```
"frames": [
    {
        "number": 4322,
        "image": "frame_4322.jpg",
        "annotations": [
            {
                "label": {
                    "x": 1549,
                    "y": 317,
                    "width": 175,
                    "height": 598
                },
                "category": {
                    "code": "assault",
                    "attributes": []
                }
            },
```

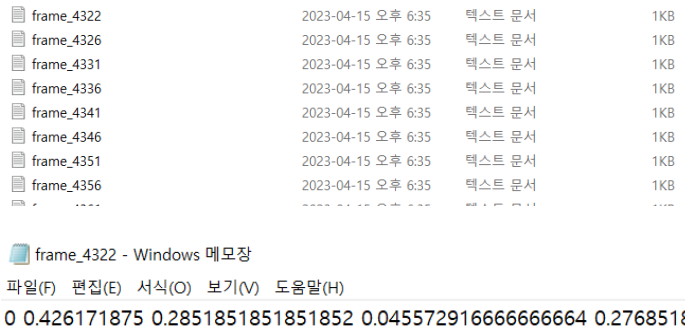
Json 파일 형태

```python
import json

# JSON 파일 경로
json_file = "/content/annotation_2253015.json"

# 변환된 TXT 파일 경로
txt_file = "/content/txt_data10/"

# YOLO에서 사용하는 클래스 이름
class_names = ["폭행"]

# JSON 파일 열기
with open(json_file, "r") as f:
    labels = json.load(f)


for label in labels['frames']:
  k = label['image'].split('.')[0]
  with open(txt_file + f'{k}.txt','w') as f:
    #img_path = label["image_path"]
    img_width = 3840
    img_height = 2160
    img_labels = label['annotations'][0]['label']
    #print(img_labels)
      # YOLO 형식으로 변환하여 TXT 파일에 저장
    # for img_label in img_labels:
    class_id = class_names.index('폭행')
    #print(img_labels)
    x, y, w, h = img_labels["x"], img_labels["y"], img_labels["width"], img_labels["height"]
    x_center = x + w / 2
    y_center = y + h / 2
    x_center /= img_width
    y_center /= img_height
    w /= img_width
    h /= img_height
    f.write(f"{class_id} {x_center} {y_center} {w} {h}\n")
```

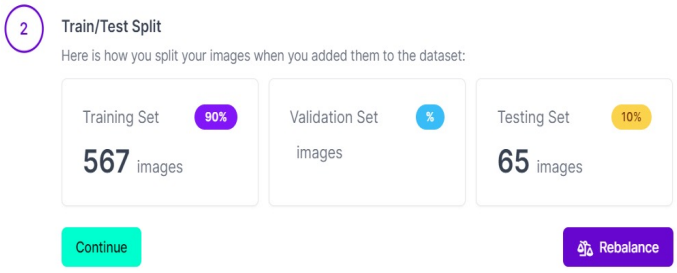
Json 파일을 txt파일로 나누는 과정

프로젝트 구현

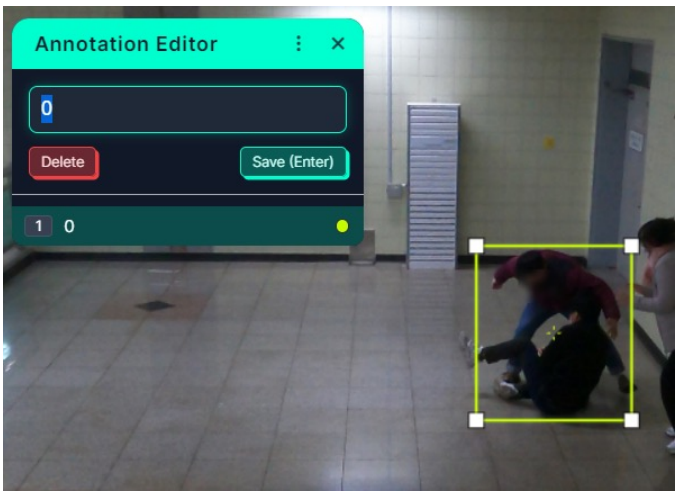
데이터셋 분류



txt 파일로 변환 시켜준 뒤 이미지와 함께 roboflow에 업로드



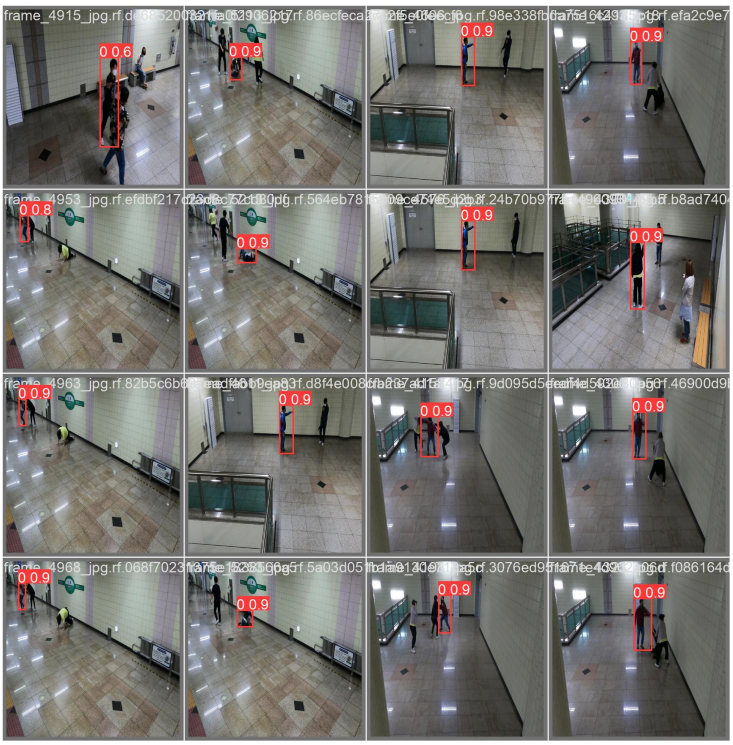
train : test 비율 9:1 로 설정



라벨링이 잘 안된 부분은 확인한 뒤 직접 x,y좌표를 설정하였고 불필요한 라벨 또는 이미지 파일은 삭제 후 다시 업로드 함

프로젝트 구현

데이터셋 학습



Roboflow에 저장해 놓은 train, test 데이터를 불러와 yolov5에 활용.

Detect.py 를 수정하여 영상에서 원하는 클래스가 디텍션 되었을 때 사진을 저장.(참조 https://bong-sik.tistory.com/30) -> 사진은 저장하여 모델이 재학습하는 식으로 모델의 성능을 향상 + 사건이 일어났을 때를 기록하여 이후 사건 처리에 용이하게 활용.

```
!python train.py --img 620 --batch 12 --epochs 30 --data /content/action-recognition2-10/data.yaml --weights yolov5m.pt --name picture_detection
```

```
Validating runs/train/picture_detection2/weights/best.pt...
Fusing layers...
Model summary: 212 layers, 20852934 parameters, 0 gradients, 47.9 GFLOPs
               Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:01<00:00,  1.53it/s]
                 all         65         65          1      0.985      0.987      0.873
```

- 모델학습 파라미터 설정 및 결과
- Epochs:30,batch: 12 만으로 mAP50-95가 0.873을 보여줌
- 실시간 영상을 잡아야 하기 때문에 mAP50 보다 mAP50-95에 중점을 뒀다.

프로젝트 구현

데이터셋 학습

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 224, 224, 3)]     0

 tf.__operators__.getitem_4   (None, 224, 224, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_4 (TFOpLambd  (None, 224, 224, 3)      0
 a)

 resnet50 (Functional)       (None, 2048)              23587712

 flatten_4 (Flatten)         (None, 2048)              0

 dense_4 (Dense)             (None, 2)                 4098

=================================================================
Total params: 23,591,810
Trainable params: 4,098
Non-trainable params: 23,587,712
_________________________________________________________________
```

ResNet: 잔차 네트워크 모델. 잔차 네트워크란 심층 신경망에서 공통적으로 발생하는 문제인 소멸 문제를 해결하기 위해 설계. 핵심은 전체 매핑이 아닌 입력과 출력간의 잔여 매핑을 학습하도록 허용하는 것.

우리는 pretained 모델을 불러와 transfer learning을 하여 2차 분류에 활용. (이미지 참고) 이후 미세조정을 추가 학습하려 하였지만 모델이 충분히 학습된 것으로 보였고 과적합 우려를 고려하여 진행하지 않음.

프로젝트 구현

데이터셋 구현

```
with open('/content/' + 'train.txt','w') as f:
    f.write('₩n'.join(train_image_list) + '₩n')

with open('/content/' + 'validation.txt','w') as f:
    f.write('₩n'.join(validation_image_list)+'₩n')
```

txt파일에 학습에 필요한 roboflow에서 가져온 데이터명을 저장.

```
import yaml

with open('/content/action-recognition2-10/data.yaml','r') as f:
    data = yaml.load(f, Loader=yaml.FullLoader)
print(data)
print(type(data))
```

```
# train Key값에는 train.txt 파일을 val key 값에는 validation.txt를

data['train'] = '/content/train.txt'
data['val'] = '/content/validation.txt'

with open('/content/action-recognition2-10/data.yaml','w') as f:
    yaml.dump(data,f)
```

Yaml 파일 train과 val 부분에 학습용으로 만든 txt 파일을 dump 해준다.

```
# Save results (image with detections)
if save_fight:
    if dataset.mode == 'image':
        cv2.imwrite(save_path, im0)
    else:  # 'video' or 'stream'
        if vid_path[i] != save_path:  # new video
            vid_path[i] = save_path
            if isinstance(vid_writer[i], cv2.VideoWriter):
                vid_writer[i].release()  # release previous video writer
            if vid_cap:  # video
                fps = vid_cap.get(cv2.CAP_PROP_FPS)
                w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
            else:  # stream
                fps, w, h = 30, im0.shape[1], im0.shape[0]
            #save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4 suffix on results videos
            #vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
        #vid_writer[i].write(im0)
        cv2.imwrite(save_path, im0)
```

결과를 보여주는 코드. 이미지를 저장하도록 수정

```
save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
#(save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir
(save_dir / 'labels').mkdir(parents=True, exist_ok=True)
(save_dir / 'images').mkdir(parents=True, exist_ok=True)
```

이미지를 저장하는 폴더 설정.

```
save_path = str(save_dir / 'images' / p.stem) + f'_{frame}' + '.jpg'
txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
```

이미지 경로 저장.

```
for *xyxy, conf, cls in reversed(det):
    #if save_txt:  # Write to file
    if cls == 0:
        save_fight = True
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        line = (1, *xywh)
        #line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
        with open(txt_path + '.txt', 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '₩n')
```

이미지를 저장할 class를 설정.

```
if save_img or save_crop or view_img:  # Add bbox to image
    c = int(cls)  # integer class
    label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
    #annotator.box_label(xyxy, label, color=colors(c, True))
if save_crop:
    save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)
```

저장될 사진에는 annotation을 제외.

프로젝트 구현

데이터셋 구현

```
# 라벨1
with open('/content/annotation_2249159.json', 'r') as f:
  label_data = json.load(f)
y =[]
for i in range(len(label_data['frames'])):
  y_la = label_data['frames'][i]['annotations'][0]['category']['code']
  #print(y_la)
  y.append(y_la)
```

```
#라벨2
with open('/content/annotation_2082551.json', 'r') as f:
  label_data = json.load(f)
for i in range(len(label_data['frames'])):
  y_la = label_data['frames'][i]['annotations'][0]['category']['code']
  #print(y_la)
  y.append(y_la)
```

JSON 파일에서 ResNet을 학습하기 위해 타겟 값을 추출 하는 과정

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, MaxPooling2D
from tensorflow.keras import datasets
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras import Input
from tensorflow.keras.layers import Dropout, BatchNormalization
from sklearn.model_selection import train_test_split


base_model = ResNet50(include_top=False, pooling = 'avg' , input_shape = (224,224 ,3), weights = 'imagenet')
base_model.trainable = False


# 모델 layer 설계
inputs = Input(shape=(224,224,3))
x = tf.keras.layers.experimental.preprocessing.Resizing(32, 32)(inputs)
x = tf.keras.applications.resnet50.preprocess_input(inputs)
x = base_model(x, training = False)
x = Flatten()(x)
outputs = Dense(2, activation = 'softmax')(x)
model_res = tf.keras.Model(inputs, outputs)
```

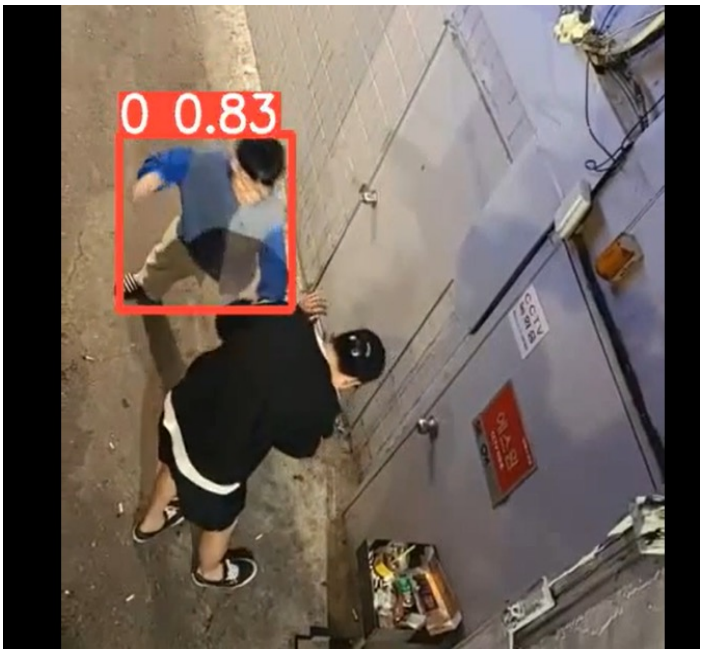
ResNet transfer learning 과정 및 학습 과정

```
# 모델 fitting
model_res.fit(x_train, y_train,validation_data=(x_val,y_val), epochs = 20, batch_size= 32, callbacks=[early])
```

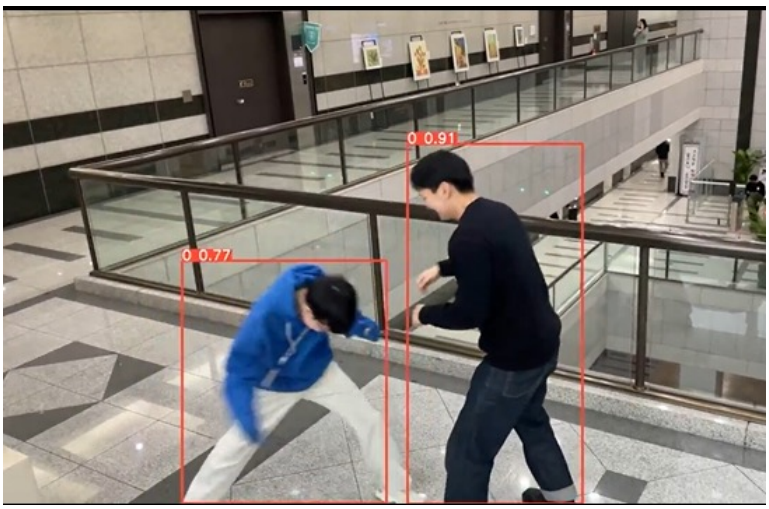
Epochs: 데이터 반복횟수
Batch_size : 데이터를 n만큼 나눠서 학습에 활용
Callbacks : 데이터가 충분히 학습되었다 판단되면 학습을 멈춤

결론

시연 영상



https://blog.naver.com/wkroraosi/223092768330

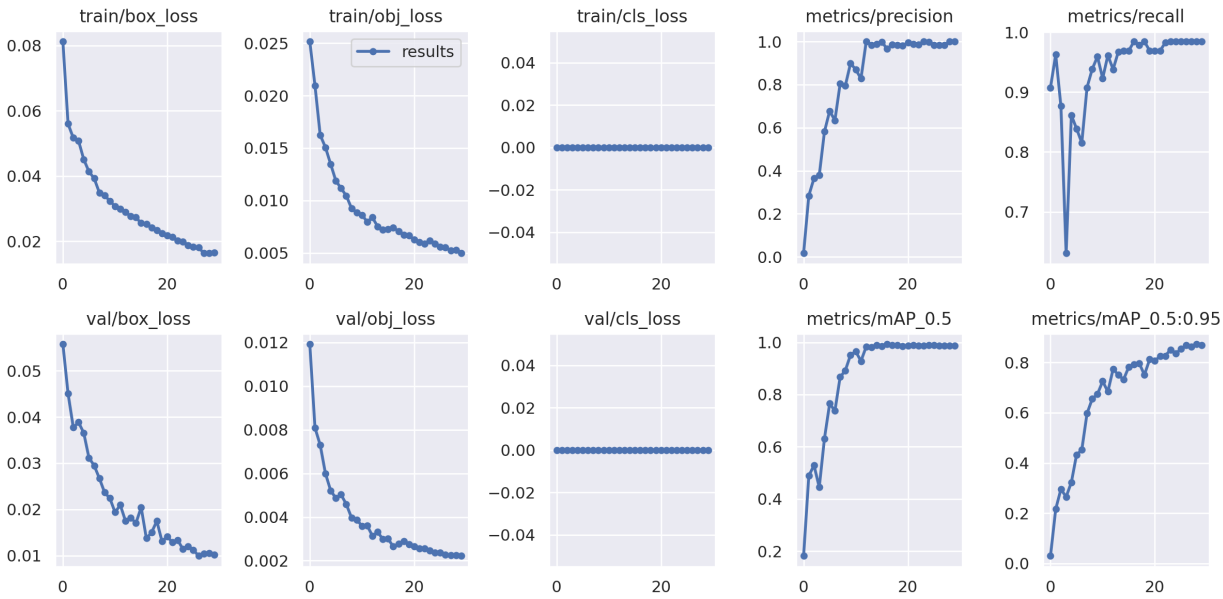


https://blog.naver.com/wkroraosi/223092774519

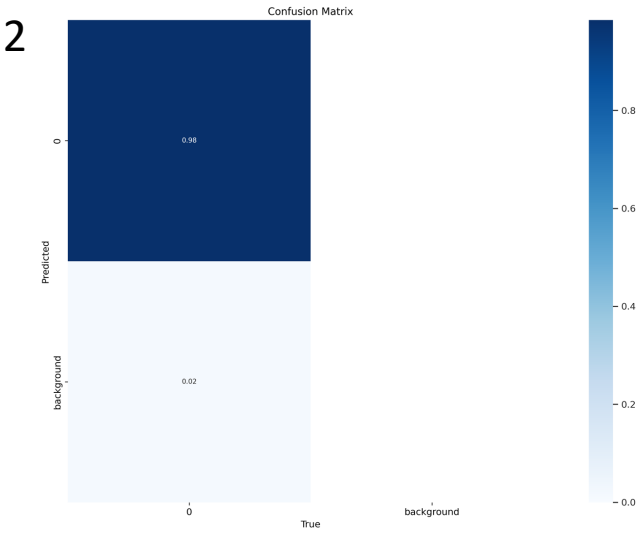
결론

성능 지표

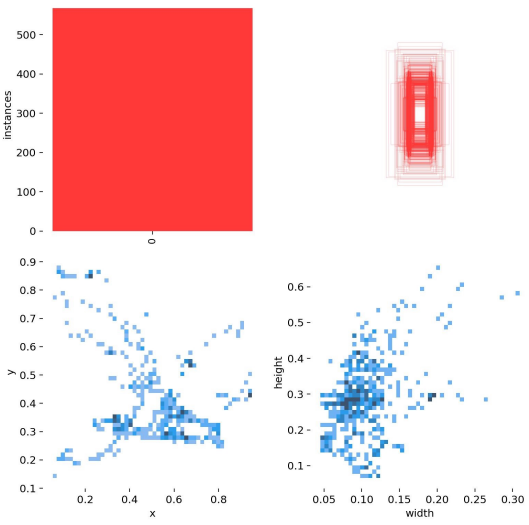
1



2



3



(1) epoch반복에 따른 loss값과 precision/recall, mAP50값을 보여줌.
(2)은 confusion metrics 테스트 데이터 결과를 시각화
(3)은 detecting 된 객체의 분포를 시각화.

결론

기대효과

기대효과 1

폭행을 하는 장면이 포착되면 서버 PC에 이미지 및 시간, 장소를 저장하여 CCTV를 하나 하나 다 뒤져야 하는 시간을 줄여 피해자들의 신고를 신속히 처리한다.

기대효과 2

한적한 골목에서는 납치를 당하여도 목격자가 없어서 신고 조차 할 수 없는 상황이 올 수도 있는데 그때 서버 PC에 자동으로 저장된 사진을 보고 범인 검거율을 높일 수 있다.

기대효과 3

최종적으로 모델의 정확도가 매우 높아 모든 범죄를 검거 가능하다면 신고 건수보다 이미지가 저장된 폴더가 더 많을 경우 불의의 사고로 신고를 하지 못했다 판단하여 피해자가 부당한 2차 피해를 받을 일을 줄일 수 있다.

기대효과 4

CCTV에 촬영된 다른 범죄 장면도 저장하여 자동 학습을 시켜 모델의 다양성 및 정확성을 늘릴 수 있고
비용측면에서도 다른 Detection 모델들 보다 저렴하다.