종합설계 프로젝트 수행 보고서

프로젝트명	딥러닝과 CCTV를 이용한 응급상황 인식 시스템
팀번호	S1-2
문서제목	수행계획서( ) 2차발표 중간보고서( ) 3차발표 중간보고서( ) 최종결과보고서(O)

2020.11.25

팀원 : 박세준 (팀장)
정범진

지도교수 : 이 정준 교수

문서 수정 내역

작성일	대표작성자	버전(Revision)	수정내용	
2020.01.21	박세준(팀장)	1.0	수행계획서	최초작성
2020.03.02	박세준(팀장)	2.0	2차발표자료	설계서추가
2020.04.29	박세준(팀장)	3.0	3차발표자료	시험결과추가
2020.11.25.	박세준(팀장)	4.0	최종결과보고서	시험결과 수정

문서 구성

진행단계	프로젝트 계획서 발표	중간발표1 (2월)	중간발표2 (4월)	학기말발표 (6월)	최종발표 (10월)
기본양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식
포함되는 내용	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~3) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~4) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~5) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~7) 참고자료	Ⅰ Ⅱ Ⅲ

이 문서는 한국산업기술대학교 컴퓨터공학부의
“종합설계” 교과목에서 프로젝트 “딥러닝과 CCTV를 이용한
응급상황 인식 시스템” 을 수행하는
(S1-2, 박세준, 정범진)들이 작성한 것으로 사용하기 위해서는
팀원들의 허락이 필요합니다.

목 차

Ⅰ. 서론

1. 작품선정 배경 및 필요성 ……
2. 기존 연구/기술동향 분석 ……
3. 개발 목표 ……
4. 팀 역할 분담 ……
5. 개발 일정 ……
6. 개발 환경 ……

Ⅱ. 본론

1. 개발 내용 ……
2. 문제 및 해결방안 ……
3. 시험시나리오 ……
4. 상세 설계 ……
5. Prototype 구현 ……
6. 시험/ 테스트 결과 ……
7. Coding & DEMO ……

Ⅲ. 결론

1. 연구 결과 ……
2. 작품제작 소요재료 목록 ……

참고자료 ……

Ⅰ. 서 론

1. 작품선정 배경 및 필요성

① 작품 선정 배경
- 2015년 기준으로, 공공기관이 공개 된 장소에 설치된 CCTV수는 총 739,232대
- 이는 5년 전, 2010년과 비교했을 때 2배 이상 급증
- 급격히 늘어난 CCTV의 개수에 비해 이를 관리하는 인력의 부족 및 즉각적인 대처 불가능

② 작품 선정의 필요성
- 인적이 드문 곳을 지날 경우, 갑자기 지병으로 쓰러지거나 강도를 만나서 상해를 입는 등의 경우에도 CCTV 수에 비해 관리 인원이 적다 보니, 현재 CCTV는 사후의 증거자료 수집용의 목적이 가장 큼
- 사건 사고가 발생 시, 기존의 시스템은 이를 저지할 방법이 없음
- 이에 대해 합리적인 해결책 제시의 필요성 증가

2. 기존 연구/동향 분석

- 한국기술교육대학교 산학 협력단 - 딥러닝 기반 CCTV용 얼굴&차량 인식 제안서
- 한국전자통신연구원 - 딥러닝 기반 다중 CCTV 영상 내 사람 탐지 및 이동경로 추적 기술
- Image Classification to Support Emergency Situation Awareness
- Transfer Learning (한정된 DataSet으로 딥러닝하는 방법)

3. 개발 목표

- Image Classifier를 이용하여 사람 인지 학습
- Machine Learning을 기반으로 Situation Awareness 학습
- 이미지 분석 및 응급상황에 대하여 알림을 제공하는 소프트웨어 개발

4. 팀 역할 분담

- 박세준 (팀장) : Using Tensorflow CNN Deap Learning Model to Image Classification
- 정범진 : Sub Image Classification , Web Server Coding

5. 개발 일정

항 목	12월	1월	2월	3월	4월	5월	6월	7월	8월	9월	10월
계획 수립 및 실습 환경 구축											
졸업연구 설계											
프로토타입 구현											
프로토타입 오류 해결 및 유지보수											
졸업연구 구현											
졸업연구 테스트											
최종 마무리 작업											
눈문 작성											

6. 개발 환경

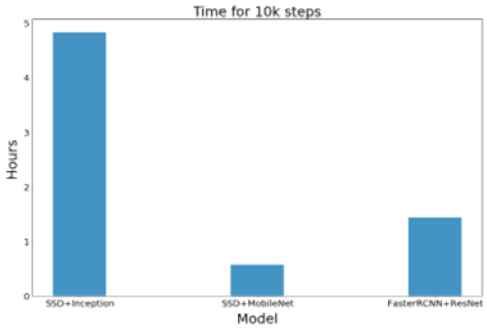
- 언어 : Python, Tensorflow, Javascript
- 학습 서버 : Google Colab, AWS
- DataSet : 일반 길거리 CCTV 비디오 약 950개,
  절도, 폭행, 총기난사, 강도 등 범죄 행위 CCTV 비디오 각각
  50~100개
- Object Detection Model API 사용
- Transfer Learning 이용하여 학습 속도 증가, 예측율 증가

Ⅱ. 본 론

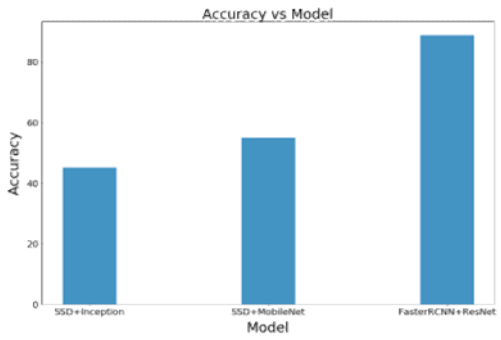
1. 개발 내용

- TensorFlow에서 Object Detection API를 사용하여 입력된 이미지에서 사람을 인식 및 지역화하는 과정 구현한다. 그리고 지역화 된 레이어의 변화값과 상황 인식 알고리즘을 종합하여 대상자가 위급상황인지 아닌지 판단 후 사용자에게 알린다. 사용자는 웹 서비스로 해당 CCTV영상과 위급상황 알림을 열람할 수 있다.

- 사람을 인식하는 부분은 TensorFlow Object Detection API Model중 FasterRCNN inception v2 coco를 사용하도록 한다. 이는 다른 모델(SSD inception)에 비해 월등한 학습 속도를 가지고 있으며, 사물 인식 정확도 또한 가장 높기 때문이다.



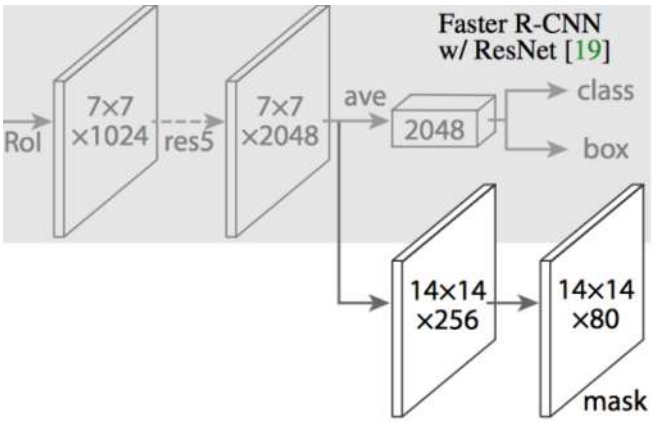
▲ API Model간의 학습속도 비교



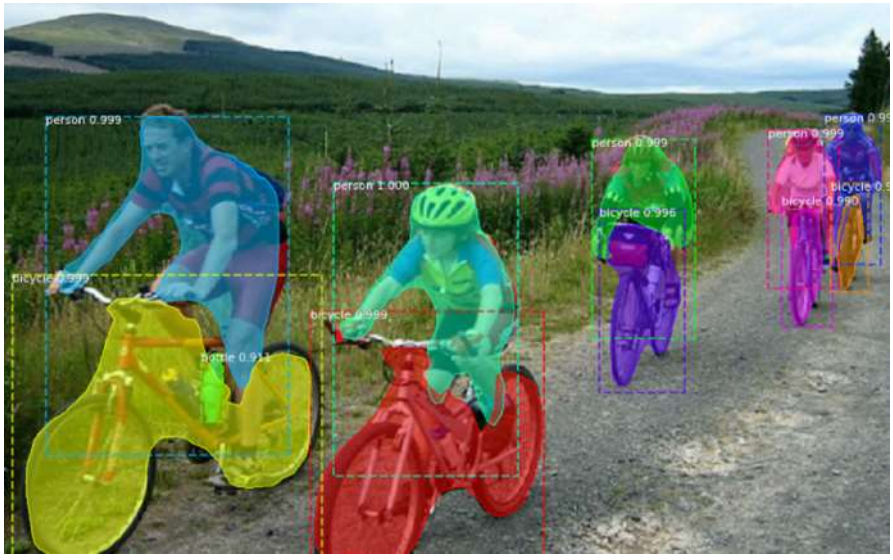
▲ API Model간의 정확도 비교

- 인식되는 사람 객체의 행동이나 자세를 분석하기 위하여 FasterRCNN 모델에서 segment 인식을 추가한 maskRCNN을 사용한다. maskRCNN은 FasterRCNN에서 각 픽셀이 오브젝트에 해당하는 것인지 아닌지를 마스킹하는 네트워크(CNN)를 추가한 것이다. 이를 바이너리 마스크라고 하고 2-dimensional Numpy array로 구성되며 마스크 배열에는 0 아니면

1이 들어가게 된다. Segment까지 잡아주는 모델을 사용한 이유는 후에 객체의 행동이나 자세를 인식할 때 해당 객체의 Segment가 지표가 되어줄 수 있기 때문이다. Segment에 따른 인식 결과를 라벨링하고 MLP(multi layer perceptron)으로 학습시키도록 한다.



▲ MaskRCNN의 원리



▲ 객체 인식 및 지역화의 예제

- 사람 객체의 움직임을 weight값으로 잡아 인식되고 있는 객체와 응급상황 판단이 오판인지 아닌지 사용자가 알기 쉽도록 로그 형식으로 보여준다.

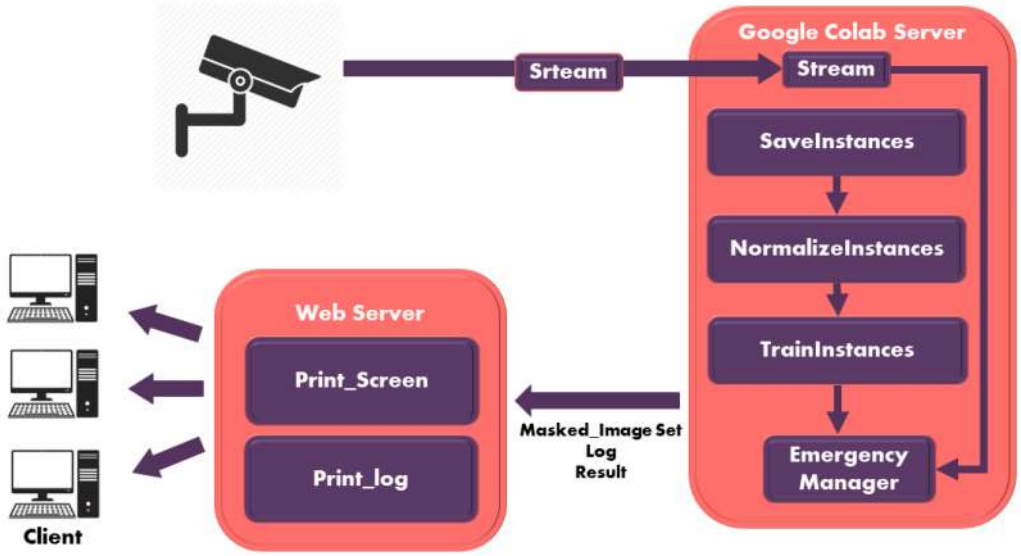
2. 문제 및 해결방안

- TensorFlow에서 오브젝트를 학습시키려면 고성능의 GPU가 필요한데 이는 구글 Colab 서버를 이용하여 모델을 학습시키고 객체 분류에 사용한다. 또한 구글 Colab은 세션이 유지가 되지 않기 때문에 구글드라이브와 연동하거나 로컬 세션에 연결한다.

- 실시간 영상을 처리속도가 따라가지 못해서 성능문제가 발생할 수 있는데 이는 Object Detection 모델을 개발 환경에 맞춰 최적화된 모델을 찾고 또한 정확도를 다소 낮추더라도 입력되는 이미지의 화소를 줄이도록 한다.

- 딥러닝 모델이 사람을 지역화하고 응급상황을 임의로 판단하기 때문에 오작동이 생길 수 있다. 만약 사람이 아닌 부분을 사람으로 지역화 하거나 사람모양의 포스터, 판넬 등을 계속해서 사람으로 인식한다면 weight값을 추가하여 사용자에게 로그를 제공해준다. 응급상황이 아님에도 불구하고 오판으로 응급상황이라고 알리거나 응급상황임에도 알리지 아니하면 MLP 모델의 학습 자료를 보강하여 정확성을 높인다.

3. 시험시나리오



- CCTV가 동작

> Google Colab에서 이를 로컬카메라로 받아 응급상황 인식 모델에 전달

> 응급상황 인식 모델은 해당 이미지에서 사람 객체만을 찾아 지역화/마스킹 해준 뒤 사전에 학습된 상황인식 모델을 불러와 응급상황 여부를 판단

> 응급상황 모델은 응급상황 결과값과 CCTV의 이미지, 응급상황이 발생한 곳의 위치와 로그를 웹 서버로 전송

> 웹 서버에서는 이미지와 크롭 이미지, 응급상황 여부와 로그를 출력. 응급상황 발생 시 사용자가 파악하기 쉽게 현재 이미지에 연출 적용

4. 상세 설계

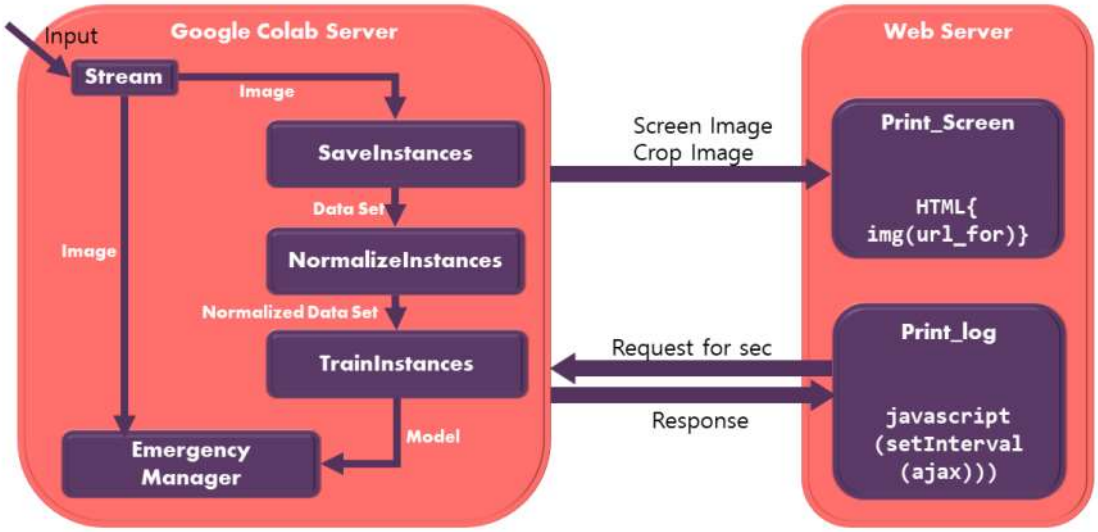
환경	Google Colab
모듈	SaveInstances
변수	1. file_names : 경로상의 모든 이미지파일 리스트 2. new_height, new width : 새로 만들어질 이미지의 shape값 3. index : 저장될 파일 명의 인덱스 값
설명	- 경로상에 있는 이미지들의 사람 객체에 대한 Segment값을 얻고 마스크된 64x64 이미지들로 저장해준다. - 해당 마스크의 width 값과 height 값을 비교하고 더 큰 값에 이미지의 원본 형식을 맞춘다. 크기가 재설정된 이미지는 후에 학습시키기 위해 64x64 크기로 정규화되며 jpg 포맷으로 저장한다.

환경	Google Colab
모듈	NormalizeInstances
변수	1. XS_data : 경로상의 Segement 이미지들 2. YS_data : Segment 이미지들에 대한 라벨 3. x_train : XS_data가 셔플 된 학습시키기 위한 Dataset 4. x_test : XS_data가 셔플 된 테스트하기 위한 Dataset 5. y_train : x_train에 상응하는 라벨값 6. y_test : x_test에 상응하는 라벨값
설명	- 경로상에 있는 Stand Segment 이미지들을 모두 불러와서 Grayscale과 Not 논리곱을 적용한 뒤 x 리스트에 넣는다. - 그에 맞는 라벨값을 y 리스트에 넣는다. - 마찬가지로 경로상에 있는 Lay Segment 이미지들을 모두 불러와서 Grayscale과 Not 논리곱을 적용한 뒤 x 리스트에 넣고 y도 마찬가지로 넣어준다. - 균등한 학습을 위해 위의 데이터들을 같은 시드값으로 섞어준다. - 해당 데이터셋을 학습데이터, 시험데이터로 분류한다.

환경	Google Colab
모듈	TrainInstances
변수	1. model : 학습에 사용될 모델을 정의한다. 2. loss : 학습에 대한 loss를 반환한다. 3. acc : 학습에 대한 정확도를 반환한다.
설명	- 학습에 사용될 모델 레이어와 옵션을 변수 model에 정의한다. 필자는 256,128개의 노드를 가지는 히든 레이어 두 개와 Dropout(0.7), 그리고 input(4096), output(2)로 정의했고 optimizer와 loss function에 대해선 Adam, binary cross entropy를 적용하였다. - NormalizeInstances 모듈에서 정의한 데이터 셋을 Flattening하여 shape(64,64)에서 shape(4096)으로 만들어준다. - flattening 된 train set 이미지로 해당 모델을 학습시킨다. 실시간으로 loss와 acc를 계산한다. - 미리 분리해놓은 test set 이미지로 loss와 acc를 계산한다.

환경	Google Colab
모듈	EmergencyManager
변수	1. weight_frame : instance들의 segment가 얼마나 가중되었는지 나타내주는 numpy array 2. frame_acc : 현재 인식된 Instance가 가중치 array인 weight_frame과 얼마나 일치하는지 나타내는 변수 3. label_index : 현재 인식된 Instance가 TrainInstances 모듈에서 학습시킨 모델에서 어느 쪽에 적용되는지 나타내는 변수 4. isEmergency : 모든 상황을 종합하여 현재의 instance가 응급상황인지 아닌지 판별하는 것을 나타내는 변수
설명	- 모든 사람 객체를 인식한 뒤 각 instance의 segment만큼 weight_frame에 가중시킨다. - 현재 instance가 weight_frame에 얼마나 일치하는지 계산한다 - 현재 instance를 model에 적용시켰을 때 무슨 라벨이 붙는지 label_index에 넣어준다. - label_index가 0이고 weight_frame과의 일치율인 frame_acc가 90%가 넘어가면 응급상황 isEmergency를 True로 반환한다.

환경	Web Flask	
변수	응급상황 판단 여부 사항 오판 주의 경고 사항	영상 프레임 이미지 응급 인식된 사람의 Crop 이미지
설명	- Javascript 안에서 1초마다 한번씩 Colab 서버로 Request를 요청 - Request : 응급상황 여부 결과, 오판 주의 경고사항 - 현재 프레임 내의 사람의 수 출력 - 응급상황일 시 사람의 수 출력을 중지하고 알림 문구 출력 - 같은 자리에서 한 사람이 오래 있지만 응급상황이라고 판단되지 않는 경우 확인이 필요함으로 주의 문구 출력	- Colab 서버에서 현재 영상 이미지 프레임은 yield 객체로 계속해서 지정 - 현재 영상의 이미지는 HTML의 이미지 source로 지정하여 출력 - HTML javascript 내에서 응급상황 여부 확인 Request를 1초마다 요청 - 응급상황일 시, Colab에서 해당 객체의 Crop 이미지를 yield로 지정하도록 설정 - Crop 이미지는 Request의 결과 값이 응급상황일 때만 Colab 서버의 yield 객체를 출력



▲ 시스템 상세 구성도

● Training Server

> Google Colab을 이용한 서버 구축

> Colab의 클라우드 형식의 GPU를 사용하여 해당 모델의 Training 사용

> 모델 빌드 후 Raw Image를 받아 이미지 전처리 및 응급상황 역시 Colab의 GPU사용

● **Web**

* Flask web container 사용 : Google Colaboratory 서버 내에서, flask를 사용하여 현재 영상 프레임, 응급상황 여부, 응급 객체 프레임, Log를 HTML로 보내어 flask 서버가 라우팅으로 받을 수 있게끔 연결

* HTML : HTML & javascript 사용

- 현재 영상 프레임 : 프레임 객체를 Colab 서버에서 해당 변수를 연결

- 응급상황 변수 : 모델 결과 값에서 Integer 자료형으로 반환받아 만일 응급상황이라는 변수를 받게 되면 응급상황 String을 연결

- 응급 객체 프레임 : 응급 상황 시, 모델 결과 값에서 응급 객체만이 포커싱된 프레임을 반환 받아 연결

- Log : 인식된 사람의 수, 현재 응급상황임을 알리는 문구, 오판 가능성이 보이는 프레임의 지속 등 String 형식으로 Colab 서버와 연결

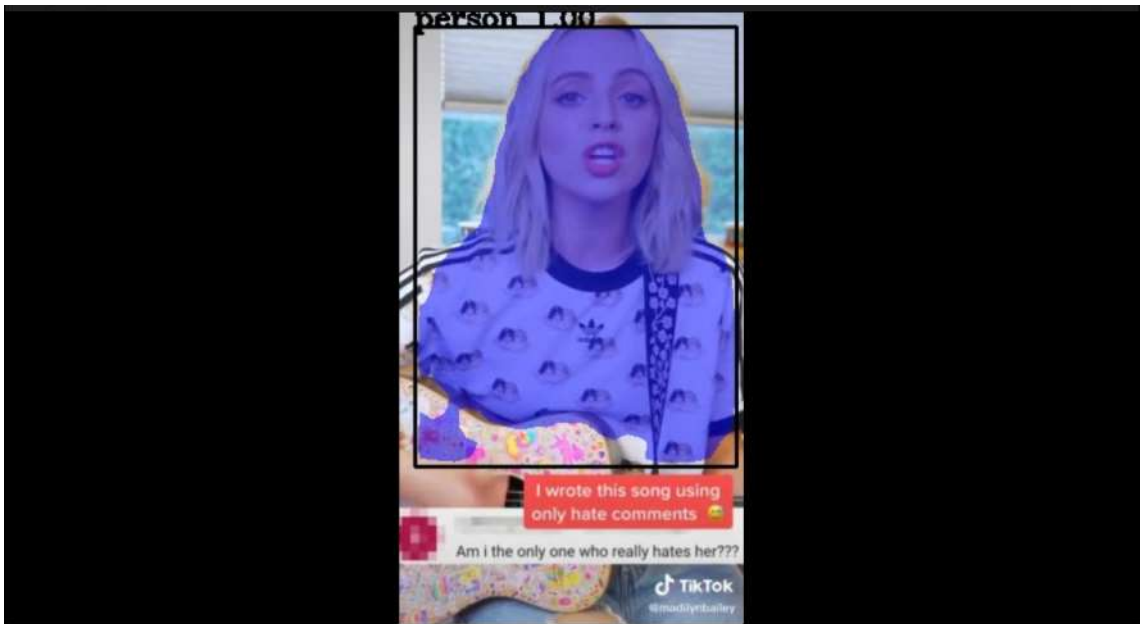
5. Prototype 구현

● Main Function

사용된 모델 : faster Mask RCNN coco

: Faster RCNN에서 Masking을 하는 레이어가 하나 추가된 형태로, Object Detection 뿐만 아니라 Instance Segmentation 까지 다루는 Two shot detector.

모델 input에 영상이 입력되면 아래 사진에서와 같이 사람을 인식하고 인식된 레이아웃에 Object name(오브젝트 이름)과 accuracy(정확도)가 라벨로 입력된 것을 확인할 수 있다.



▲ 임의의 영상에 모델이 적용된 예

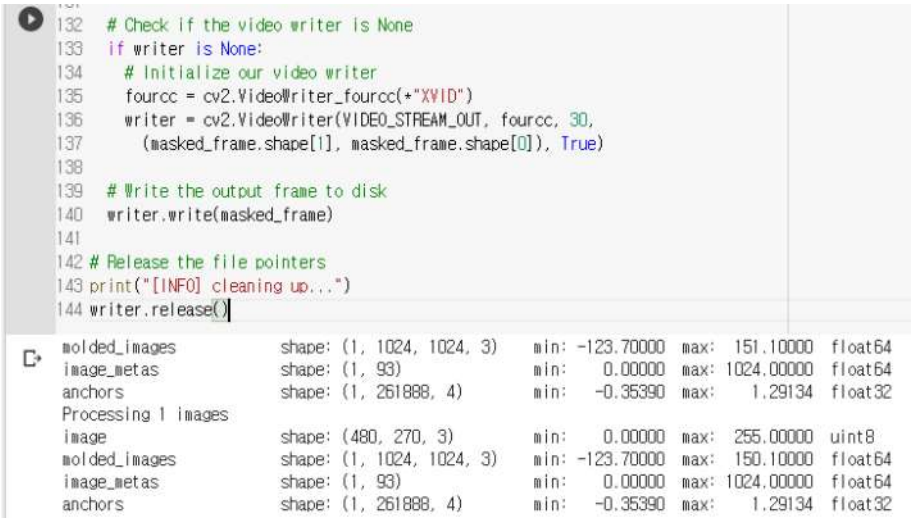
▲ CCTV영상에 모델이 적용된 예

같은 모델로 현재 소유하고 있는 데이터 셋 중에 임의의 실제 CCTV영상을 Input에 입력하였을 때 모델이 적용된 예시 화면이다.

적용된 Faster Mask RCNN coco 모델에 classification에서 'person', 'handbag', 'car', 'train' 등 약 80여개의 오브젝트가 training 된 상태이며, 위 사진에서는 두 명의 사람과 한 사람이 메고 있는 핸드백까지 인식하였다.

▲ 오브젝트를 잘못 인식하여 Accuracy (정확도)가
상대적으로 낮은 예시 화면

위 사진과 같이 오브젝트에 대하여 인식이 오판이 날 경우에 정확성이 떨어진 예시를 보여주고 있다. 이와 같이 오판이 날 경우, 직접 라벨링을 지정해주어 오판율을 줄일 수 있으며 이는 웹 서비스 내에서 실행 할 수 있다.

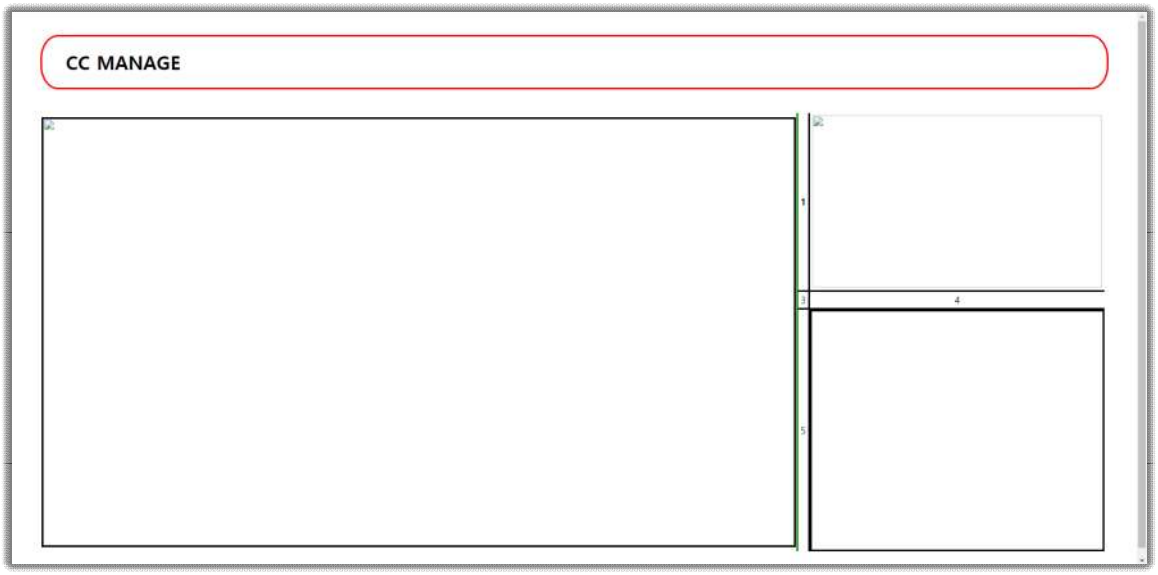


▲ Google Colab과 Gdrive를 연동한 사물 인식 모델 적용

기존 tensorflow로 모델의 사물 인식을 적용시켰을 때에는 2~30초에 1프레임씩 처리하여 실시간 영상에 적용시킴에 무리가 있었지만 Colab에 tensorflow GPU 버전을 설치하여 적용시켰더니 1초에 약 2~3프레임씩 처리하여 실시간 영상의 사물 인식에 무리가 없음을 보여줬다.

현재는 OpenCV 라이브러리를 이용하여 컴퓨터에 연결된 카메라에서 받아온 영상을 즉시 Colab으로 받아서 실시간으로 사물 인식을 적용시키는 단계에 있으며 사물 인식 성능에 따라 받아온 영상을 전처리해주는 작업이 요망된다.

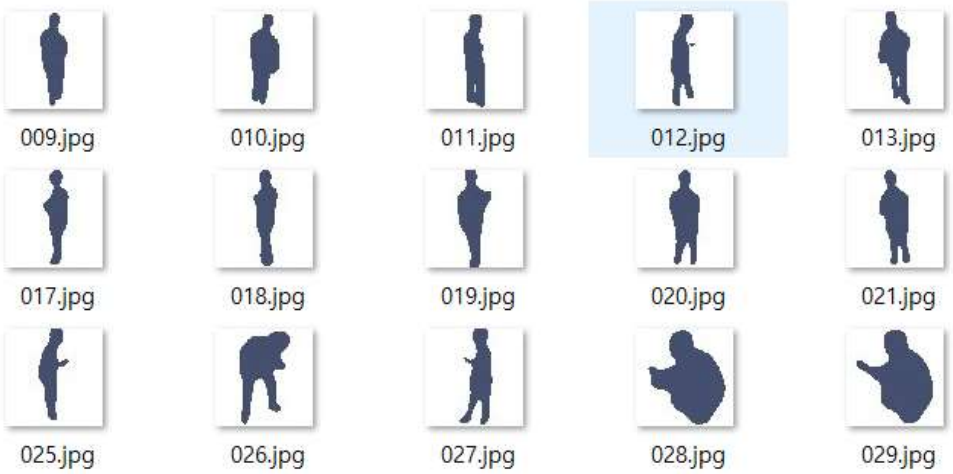
* HTML UI



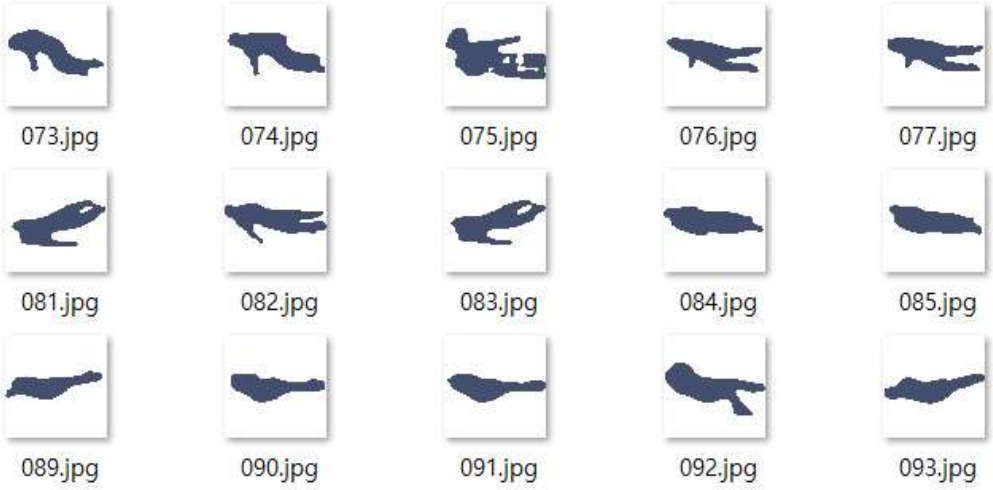
- 현재 영상의 프레임을 크게 띄워논 후, 오른쪽 상단은 응급 객체 프레임을, 오른쪽 하단은 로그를 받는 창으로 설정
- CSS 삽입 예정

6. 시험 / 테스팅 결과

응급상황인식을 하는데 있어 먼저 해당 객체의 행동을 인식하기 위해 사람으로 분류된 객체들의 Segment를 사용한다. MaskRCNN에서는 별도의 API를 제공하지 않음으로 내부 파일인 visualize_cv2.py에 직접 접근하여 받아들이는 이미지들의 Segment를 파일로 저장하는 모듈을 새롭게 만들어준다. 다만 Multi Layer Perceptron의 Input Layer에 넣기 위해선 평활화 작업이 필요하므로 정규화를 시켜준다. (Segment를 파일로 받는 모듈을 만들고 그 곳에 사용된 이미지들은 전부 직접 촬영하였으나 CCTV마다 시야각이 모두 다르므로 위치가 고정된 CCTV에서 이미지를 받아오는 것이 가장 좋다.)

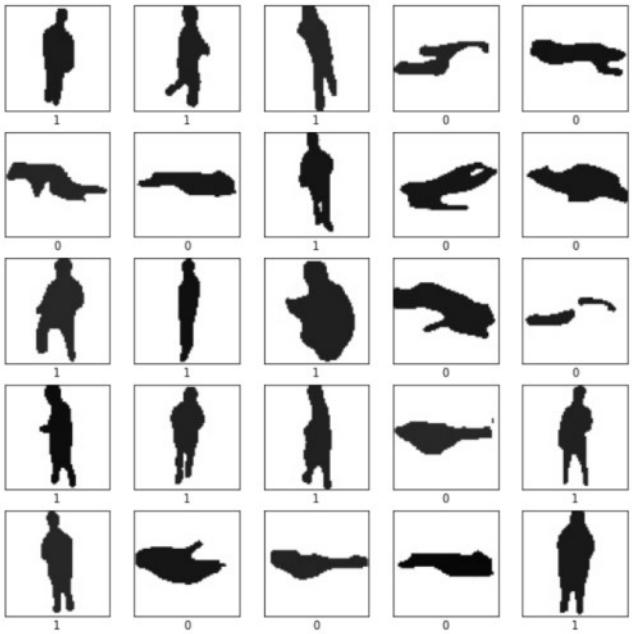


▲ 평범한 사람에 대한 Segment Images (64x64)



▲ 쓰러진 사람에 대한 Segment Images (64x64)

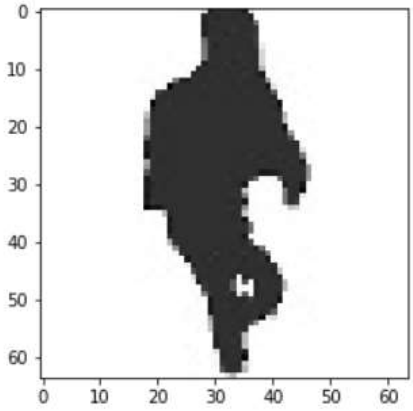
위의 Dataset들을 학습시켜줄 모듈 또한 만들어준다. 학습방법은 앞서 서술하였듯이 MLP를 사용하였고 Optimizer로는 Adam Optimizer를 적용시켰으며 결과 라벨이 두 가지(0 or 1)이므로 loss function은 binary_crossentropy를 사용하였다. 자세한 코드 내용은 CODING & DEMO에서 서술한다.



▲ Grayscale로 변환 후 균등한 학습을 위해 섞은 뒤 라벨을 적용시킨 모습

▲ Label 1 : Normal // Label 0 : Emergency

```
Epoch 200/200
160/160 [==============================] - 0s 253us/step - loss: 0.5777 - accuracy: 0.9625
30/30 [==============================] - 0s 1ms/step
Loss: 1.0960467022869125e-07, Acc: 1.0
(1, 4096)
1
<function matplotlib.pyplot.show>
```



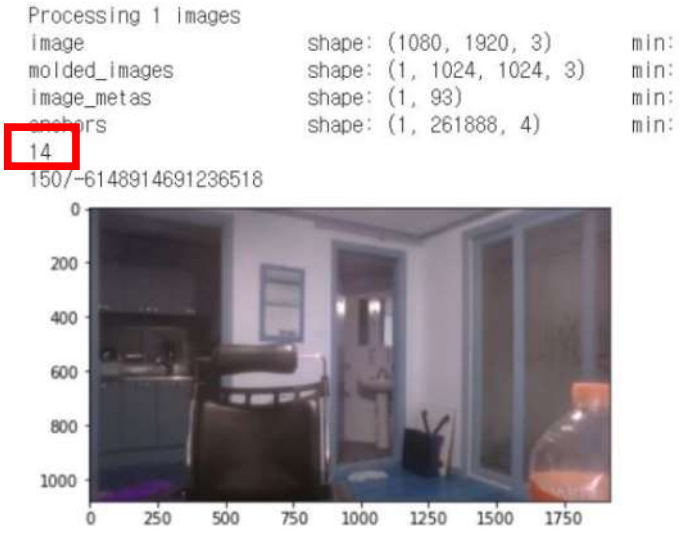
▲ Train Dataset에 대한 학습이 완료된 후 Test Dataset에 대한 Accuracy : 100%

▲ 학습된 모델에 태국의 한 cctv 영상을 적용시킨 결과 응급상황이 아닌 것으로 인식됨

단순히 위의 결과만으로 응급상황을 판단하기에는 MLP 모델의 오판 확률이 있으므로 응급상황 조건에 하나를 더 추가한다. 사람으로 인식되는 Segment 좌표의 weight 값을 추가하는 넘파이 배열을 만들어줘서 사람 객체들의 행동 양상을 인식할 수 있게 한다. 즉 한 지역에 반복적으로 인식되는 객체가 있다면 weight값이 상승할 것이고 해당 좌표의 weight값, 그리고 해당 좌표 객체의 Segment를 모델에 적용시켜 받은 결과값을 종합적으로 분석하여 응급상황인지 아닌지를 판단한다. 이에 대한 코드도 뒷장에서 자세히 다루도록 한다.

▲ 객체가 움직이지 않으면 계속해서 상승하는 Weight값 (np.max(Weight))

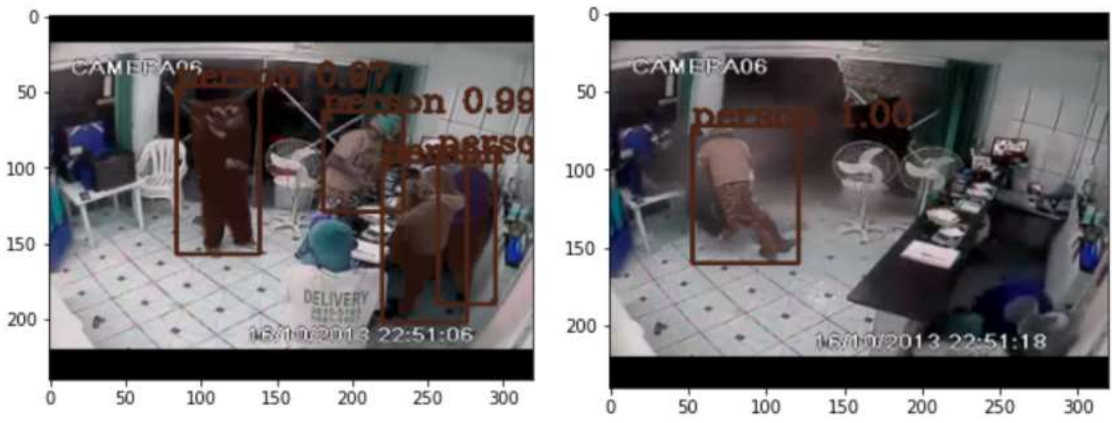
▲ frame acc란 MAXIMUM_WEIGHT에 도달한 지역이 현재 세그먼트와 얼마나 일치하는 정도의 퍼센테이지이다. (위의 경우 97% 일치)



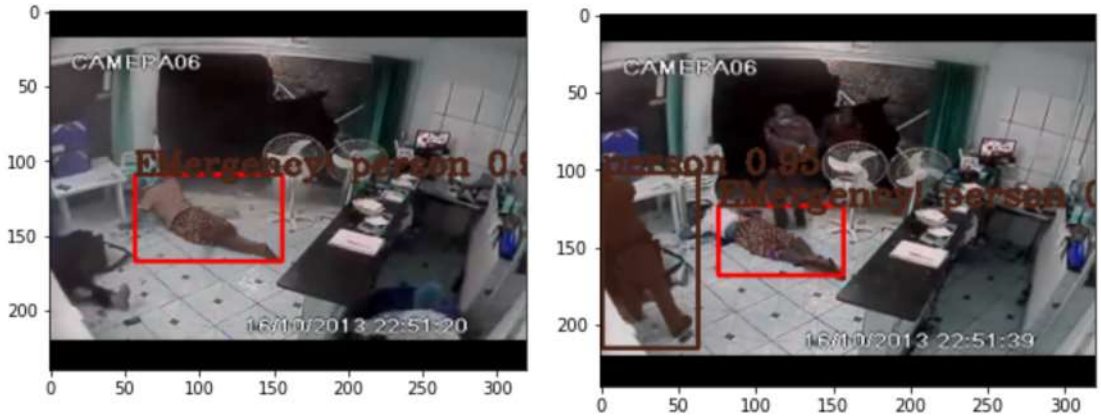
▲ 객체가 움직이거나 사라지면 Weight값은 서서히 떨어진다

※ 실제 CCTV 영상에 적용하여 테스팅

다음 이미지들은 2013년도 강도가 경찰의 총격에 맞는 실제 사건 영상에 위의 학습 모델을 적용시킨 것이다.



▲ 돈을 챙기는 강도(좌)와 경찰의 총격을 맞고 쓰러지는 중인 강도(우)



▲ 총에 맞고 쓰러진 강도를 응급상황으로 잘 인식하고 있다.
(붉은 테두리와 Emergency 라벨)

320x240의 저해상도 CCTV영상에서도 아주 잘 잡아내는 모습을 볼 수 있었다. CCTV의 화질이 개선되거나 고정된 CCTV 위치에서 받아들여 오는 데이터로 학습시킨다면 더 좋은 퍼포먼스를 보일 것으로 기대된다.

7. CODING & DEMO

● Main Function

```
165     return image, layer_frame
166
167 # Load a random image from the images folder
168 file_names = next(os.walk(TEST_IMAGE_DIR))[2]
169
170 for i, f_name in enumerate(file_names):
171   image = skimage.io.imread(os.path.join(TEST_IMAGE_DIR, f_name))
172   results = model.detect([image], verbose=1)
173   r = results[0]
174   print(f_name)
175   tt, ss = save_instances(image, r['rois'], r['masks'], r['class_ids'],
176                           class_names, r['scores'])
177   plt.imshow(tt)
178   plt.show()
```

1. 학습 데이터 저장 모듈

상황 인식 모델을 만들기 위해서는 이미지들에서 사람 영역의 Segment를 따와 정규화 처리를 해줄 필요가 있다. 위는 Image Directory에서 모든 이미지들을 불러 들여 객체 분류를 한 뒤 사람에 해당하는 부분의 Segment를 크롭하고 정규화 해 주는 모듈이다.

```
def save_instances(image, boxes, masks, ids, names, scores):
    """
        take the image and results and apply the mask, box, and Label
    """
    os.chdir("/content/drive/My Drive/Colab Notebooks/MRCNN_pure")
    n_instances = boxes.shape[0]

    if not n_instances:
        print('NO INSTANCES TO DISPLAY')
    else:
        assert boxes.shape[0] == masks.shape[-1] == ids.shape[0]

    for i in range(n_instances):
        if not np.any(boxes[i]):
            continue
        if not (ids[i] == 1) :
            continue
        print('test27')
        image = cv2.UMat(image)
        image = cv2.UMat.get(image)
        height = image.shape[0]
        width = image.shape[1]
```

```
    mask_list = []
    y1, x1, y2, x2 = boxes[i]
    label = names[ids[i]]
    color = class_dict[label]
    score = scores[i] if scores is not None else None
    caption = '{} {:.2f}'.format(label, score) if score else label
    mask = masks[:, :, i]
    mask_list.append(mask)

    global layer
    global layer_frame

    layer = 255 * np.ones((height,width,3), dtype=np.uint8)
    layer = apply_mask(layer, mask, color, alpha=0.8)
    layer = layer[y1:y2, x1:x2, :]
    layer_height = layer.shape[0]
    layer_width = layer.shape[1]
    if(layer_height >= layer_width) :
      scale_percent = 64 / layer_height
      new_height = int(layer_height * scale_percent)
      new_width = int(layer_width * scale_percent)
      layer = cv2.resize(layer, dsize=(new_width, new_height), interpolation=cv2.INTER_CUBIC)
      layer_frame = 255 * np.ones((64,64,3), dtype=np.uint8)
      #layer_frame[0:new_width, 32-(new_height//2):32+(new_height//2),:] = layer
      rows, cols, channels = layer.shape
      layer_frame[0:rows, 32-(cols//2):32-(cols//2) + cols] = layer
    else :
      scale_percent = 64 / layer_width
      new_height = int(layer_height * scale_percent)
      new_width = int(layer_width * scale_percent)
      layer = cv2.resize(layer, dsize=(new_width, new_height), interpolation=cv2.INTER_CUBIC)
      layer_frame = 255 * np.ones((64,64,3), dtype=np.uint8)
      #layer_frame[32-(new_width//2):32+(new_width//2), 0:new_height,:] = layer
       rows, cols, channels = layer.shape
       layer_frame[0:rows, 32-(cols//2):32-(cols//2) + cols] = layer
     else :
       scale_percent = 64 / layer_width
       new_height = int(layer_height * scale_percent)
       new_width = int(layer_width * scale_percent)
       layer = cv2.resize(layer, dsize=(new_width, new_height), interpolation=cv2.INTER_CUBIC)
       layer_frame = 255 * np.ones((64,64,3), dtype=np.uint8)
       #layer_frame[32-(new_width//2):32+(new_width//2), 0:new_height,:] = layer
       rows, cols, channels = layer.shape
       layer_frame[32-(rows//2):32-(rows//2) + rows, 0:cols] = layer

     global index
     image_name = 'stand' + str(index) + '.jpg'
     index = index + 1
     cv2.imwrite(os.path.join(ROOT_DIR , image_name), layer_frame)

     image = apply_mask(image, mask, color)
     image = cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
     image = cv2.putText(
         image, caption, (x1, y1), cv2.FONT_HERSHEY_COMPLEX, 0.7, color, 2
     )

 return image, layer_frame
```

▲ 사람 객체로 인식된 Segment를 정규화해주고 저장해주는 save_instances

2. 학습 데이터 정규화 모듈

다음은 이렇게 정규화 시킨 이미지들을 불러들여 MLP 방식으로 해당 이미지들을 학습시킬 수 있도록 Train Dataset과 Test Dataset을 구성하고 편향적이지 않은 학습 모델을 위하여 데이터 셋을 섞어주는 모듈이다.

```
import glob

X_data = []
files = glob.glob ("/content/drive/My Drive/Colab Notebooks/MRCNN_pure/MLP_images/lay/*.jpg")
IMG_SIZE = 64

for myFile in files:
    print(myFile)
    image = cv2.imread (myFile, cv2.IMREAD_GRAYSCALE)
    image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
    image = ~image
    X_data.append (image)
```

▲ 이런 방식으로 X 리스트에 데이터를 불러오고 Y 데이터에는 라벨링을 한다

```
s = np.arange(XS_data.shape[0])
np.random.shuffle(s)

XS_data = XS_data[s]
YS_data = YS_data[s]
```

▲ 편향되지 않은 학습을 위한 데이터 셋 분포

```
125 x_train = XS_data[0:160]
126 x_test  = XS_data[160:190]
127 y_train = YS_data[0:160]
128 y_test  = YS_data[160:190]
```

▲ Train Dataset과 Test Dataset을 분리

3. 모델 트레이닝 모듈

위의 데이터로 MLP 모델을 학습시키는 모듈이다. MLP input layer로 들어갈 수 있도록 데이터 셋을 Flattening 해주며 y_Dataset에 대해선 원 핫 인코딩으로 인덱스를 만들어준다. 모델은 두 개의 히든 레이어와 input layer, output layer로 구성된다. 각각의 히든 레이어에서는 overfitting을 방지하기 위해 dropout을 0.7로 설정했다. 또한 모델 컴파일에선 두 개의 객체를 분류하기 위해 적절한 loss function인 binary crossentropy를 채택했고 optimizer는 Adam을 사용한다.

```
x_train = x_train.reshape(160, 4096)
x_test = x_test.reshape(30, 4096)
num_classes = 2
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(256, activation='relu', input_dim=4096))
model.add(Dropout(0.7))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.7))
model.add(Dense(2, activation='sigmoid'))
model.compile(optimizer=Adam(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model, iterating on the data in batches of 32 samples
model.fit(x_train, y_train, epochs=200, batch_size=30)

loss, acc = model.evaluate(x_test, y_test)
print('Loss: {}, Acc: {}'.format(loss,acc))


predict_test = test_image

predict_test = predict_test.reshape(1, 4096)
print(predict_test.shape)

result = model.predict(predict_test)
print(np.argmax(result))

plt.imshow(backup, cmap=plt.cm.binary)
plt.show
```

▲ 모델의 구성과 학습

4. 응급상황 판단 모듈

```
def display_instances(image, boxes, masks, ids, names, scores):
    """
        take the image and results and apply the mask, box, and Label
    """
    os.chdir("/content/drive/My Drive/Colab Notebooks/MRCNN_pure")
    n_instances = boxes.shape[0]
    isEmergency = 0
    emergencyImage = image
```

기본적인 골격은 학습 데이터 저장 모듈과 매우 흡사하다. save_instances가 Segment를 정규화하여 저장하는 역할을 했다면 display_instances는 Segment를 읽어 객체의 행동 양상을 따르는 Weight값과 비교하고, 해당 Segment가 쓰러진 상황인지 아닌지 미리 학습시킨 모델에 적용시켜 알아낸다. 이렇게 종합한 정보가 응급상황이라고 판단되면 isEmergency 값에 True를 반환한다.

```
148         current_frame = np.zeros((height,width), dtype=np.uint8)
149         current_frame = apply_mask2(current_frame, mask, num = MAXIMUM_WEIGHT)
150         current_frame[current_frame==0] = 255
151
152         pixel_count = np.count_nonzero(current_frame==20)
153         print("Current frame pixels : ", pixel_count)
154         same_pixel = np.count_nonzero(current_frame==weight_frame)
155         print("Same pixels: ", same_pixel)
156         frame_acc = same_pixel / pixel_count
157         print("Frame acc: ", frame_acc)
```

▲ 현재 Segment와 Weighted Segment가 얼마나 일치하는지 분석

```
183         backup = layer_frame
184         layer_frame = cv2.cvtColor(layer_frame, cv2.COLOR_BGR2GRAY)
185         layer_frame = ~layer_frame
186
187         predict_test = layer_frame
188         predict_test = predict_test.reshape(1, 4096)
189
190         result = mlp_model.predict(predict_test)
191         label_index = np.argmax(result)
```

▲ 현재 Segment가 쓰러진 상황인지 아닌지 model에 적용 후 판단

```
if frame_acc >= 0.9 and label_index != 1:
  isEmergency = 1

if(label_index != 1) :
  caption = 'EMergency! {} {:.2f}'.format(label, score) if score else label
  image = cv2.rectangle(image, (x1, y1), (x2, y2), red, 2)
  if isEmergency == 1:
    emergencyImage = image[y1:y2, x1:x2, :]
else :
  image = cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
image = cv2.putText(
    image, caption, (x1, y1), cv2.FONT_HERSHEY_COMPLEX, 0.7, color, 2
)
```

▲ 모델 예상 결과가 0이면 쓰러져있다고 판단, 프레임을 붉게 만들어준다.

▲ 쓰러져있음과 동시에 weight값이 높다면 응급상황으로 인식하고 해당 객체의 이미지와 응급상황이라는 상태를 반환한다.

```
print(np.max(weight_frame))
weight_frame[weight_frame>=1] -= 1

return image, emergencyImage, isEmergency
```

▲ 객체가 움직인다면 Weight값은 서서히 줄어야하기 때문에 Weight 감소 후 변수 반환

● **Web**

```
<script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>
<script src="http://code.jquery.com/jquery-migrate-1.2.1.min.js"></script>
```

- ajax를 이용해야 하기 때문에 jquery 사용

```
<img id="scr_whole" style="border-style:solid;" width="1270px" height="720px" src="{{ url_for('video_feed') }}">
```

▲ 현재 영상 프레임

- 이미지 URL를 라우팅 한 주소로 설정

```
<img width="490px" height="290px" src="{{ url_for('isEmg') }}">
```

▲ 응급 객체 프레임

- 이미지 URL를 라우팅 한 주소로 설정
- 응급 상황일 때 Python에서 해당 응급 객체 프레임을 넘겨주기로 설정

```
<script type="text/javascript" language="javascript">
  $(document).ready(function(){
    setInterval(function(){
      $.ajax({
        type : "GET",
        url : "/for_log",
        dataType : "text",
        error : function(){
          alert("failed");
        },
        success : function(data){
          document.getElementById("scr_whole").style.borderColor = "blue"
          if(data=='emergency'){
            var text = "[System] 현재 응급상황입니다! Crop 이미지를 확인해주십시오.";
            var br = document.createElement("br");
            document.getElementById("log_scr").prepend(br);
            document.getElementById("log_scr").prepend(text);
            document.getElementById("scr_whole").style.borderColor = "red"
          }
        }
      });
    }, 1000);
  });
</script>
```

▲ 로그 및 스크린 테두리

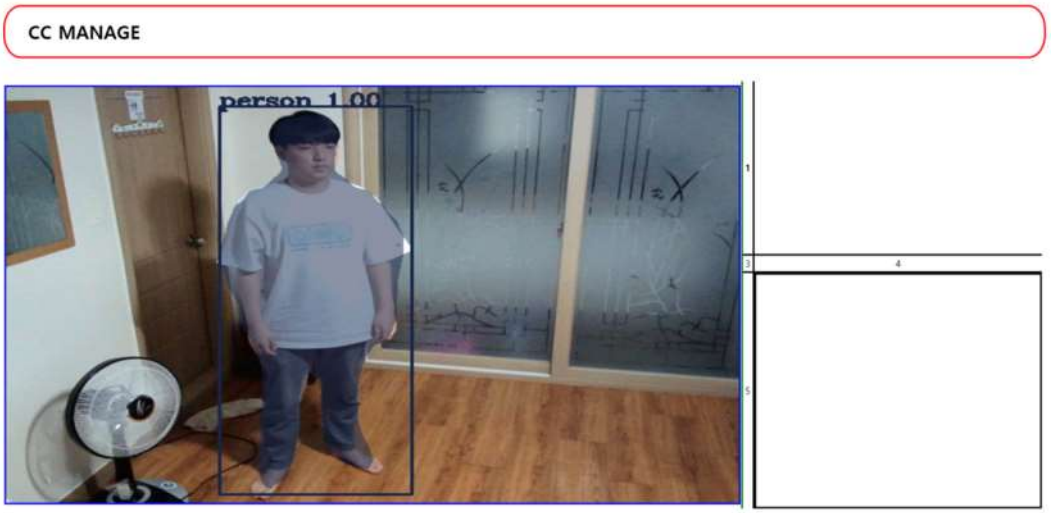
- 1초마다 응급상황 여부를 체크하는 스크립트
- Python에서 모델의 응급상황 여부 판단 결과 값이 응급상황이라고 출력이 된다면 현재 응급상황이라는 알림 문구 출력
- 새로운 로그 문구는 아래 방향이 아닌 윗방향으로 출력되게 설정 (편의성)
- 사람 수, 오판 가능성이 보이는 프레임의 지속 등 다른 로그도 추가 예정
- 응급상황 시 스크린의 테두리를 빨간색으로 지정하도록 설정

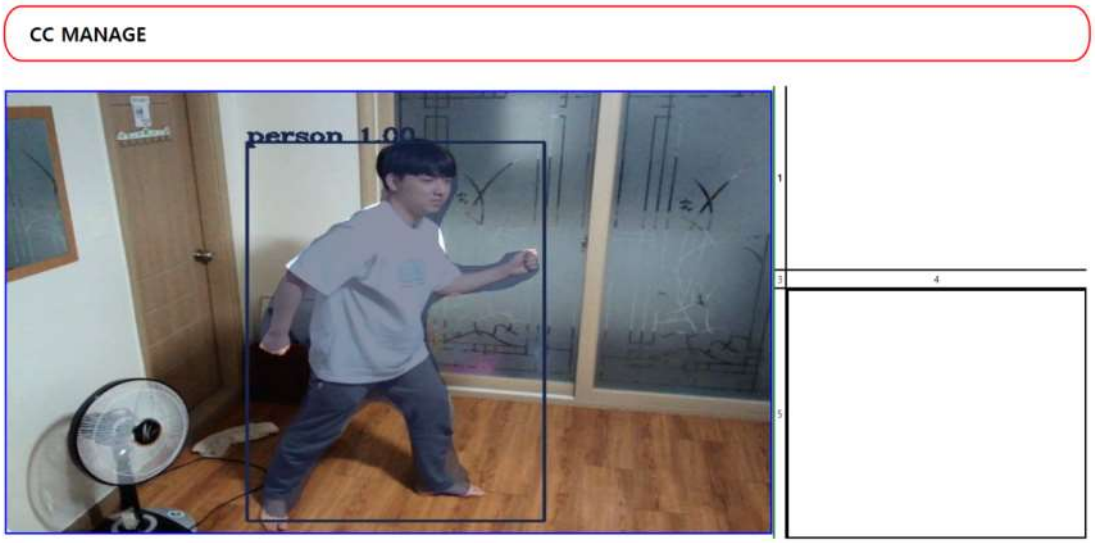
● DEMO

데모는 다음과 같은 사용처를 고려하여 두 가지의 환경에서 진행했다.
- 2M 높이의 방 안 : 주택 방범용, 건물 복도용, 매장 내 CCTV
- 3층 높이의 건물 밖 : 길거리 방범용 CCTV

1. 2M 높이의 방 안

1) 응급상황이 아닐 때





CC MANAGE

2) 응급상황일 때

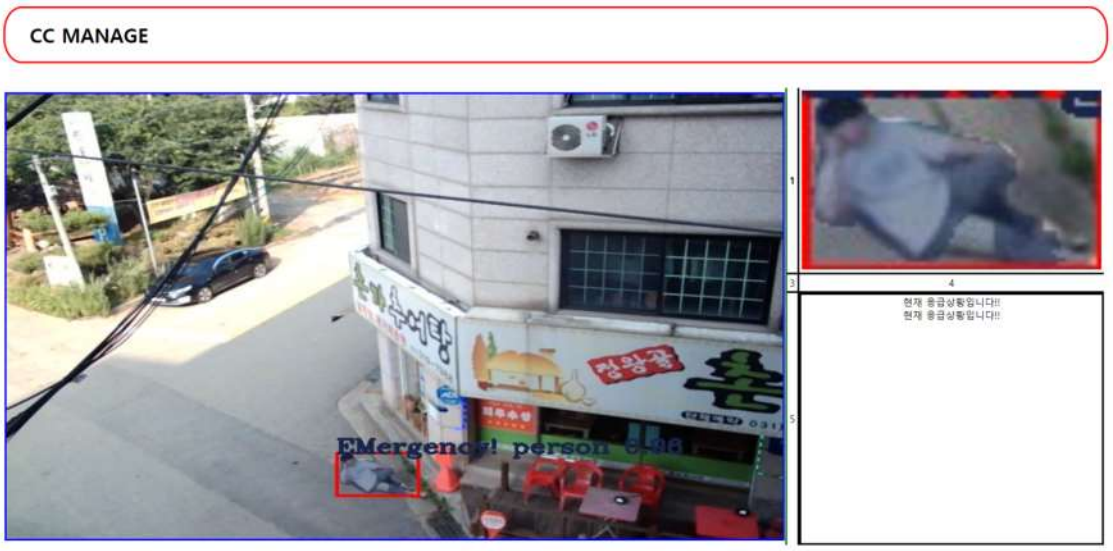
CC MANAGE

CC MANAGE

2. 3층 높이의 건물 밖

1) 응급상황이 아닐 때

2) 응급상황일 때



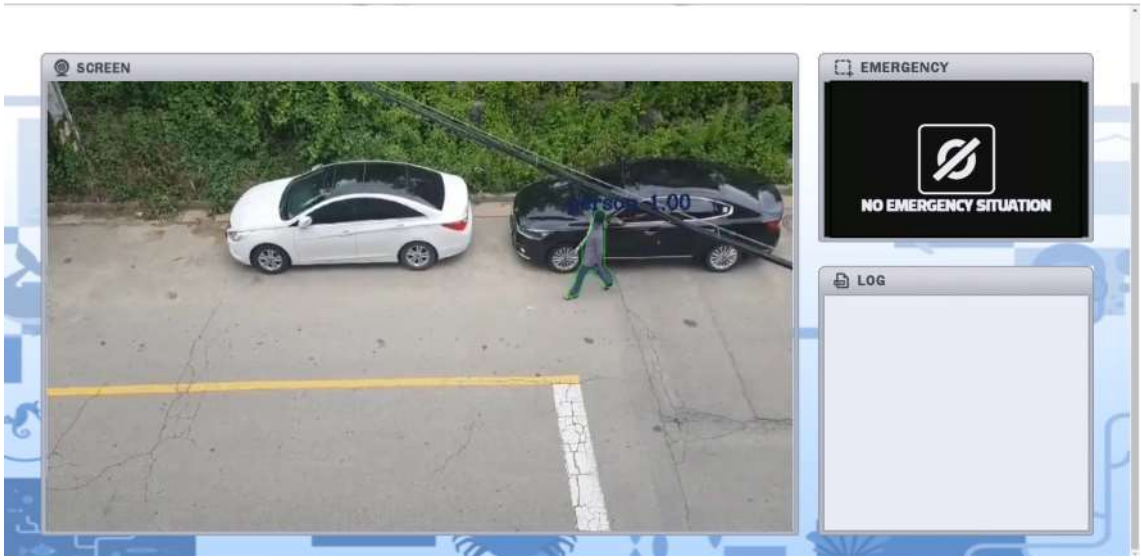
Ⅲ. 결론

1. 연구 결과

1) 최종 UI

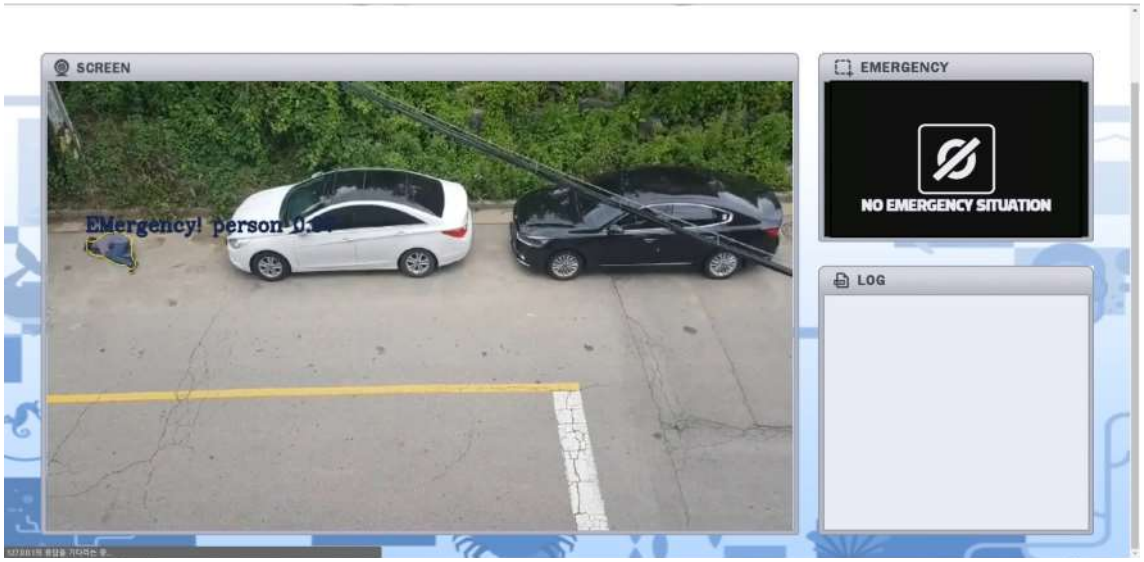
2) 결과 화면

2-1) 응급상황이 아닌 경우



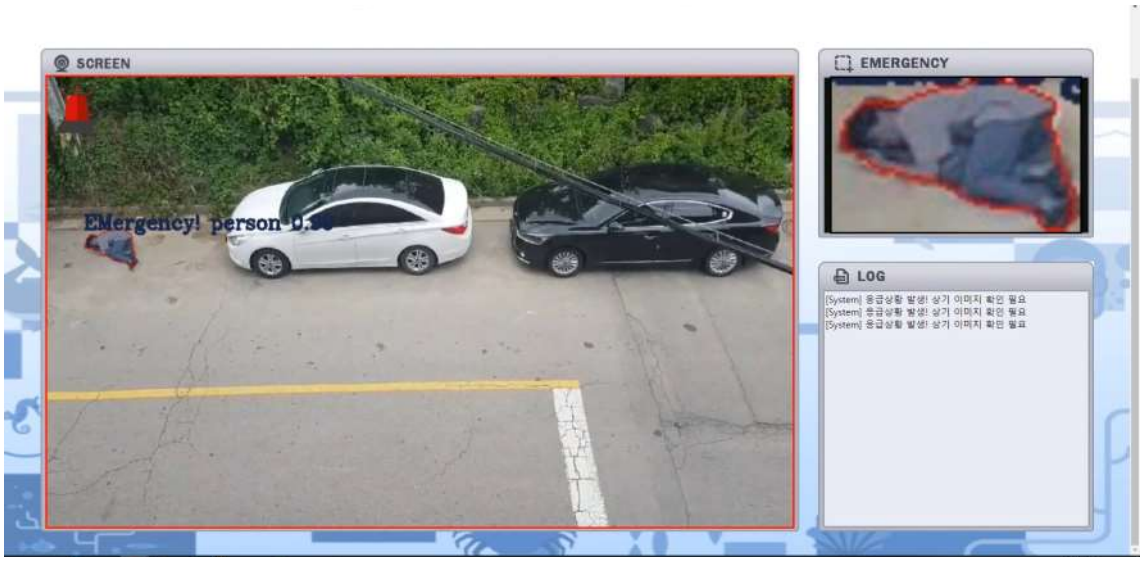
▲ 사람의 레이아웃을 초록색으로 나타내는 화면

2-2) 주의해야할 사람이 있을 경우



▲ 주의해야 할 사람이 있는 경우 황색 테두리로 나타내는 화면

2-3) 응급상황이라고 판단 될 경우



▲ 응급상황이라 판단하여 빨간색 테두리로 나타내는 화면

3) 최종 결론

▶ 다양한 자세를 학습시킬 수도 있었지만 사람이 쓰러졌다는 응급상황과 그 외의 상황만 학습시켜 판단하고자 함

▶ 각 상황별로 녹색, 황색, 적색으로 나타내어 MLP 모델에 의해 쓰려졌다고 인식되는 객체는 곧바로 황색으로 표시하고, 그러한 객체들 중 가중치가 높은 객체를 적색으로 표시하도록 하였다.

▶ 정상, 주의, 응급이라는 상황을 구별해내는데 성공하였으며 객체 인식 모델을 사용하여 상황 인식을 구현할 수 있다는 결과를 도출,

▶ 오판율은 CCTV의 위치와 각도에 따라 변화되었다. 기술이 사용될 CCTV에서 학습 데이터를 받아온다면 응급 상황 인식률이 향상될 것이라고 예상한다. 하지만 단순히 객체의 형태에 따라 응급상황을 판단하기 때문에 자세한 상황을 판단하지 못한다는 단점을 가지고 있다.

▶ 더 자세한 상황을 인식시키고 싶다면 단순한 사람의 객체 하나가 아닌 팔, 다리, 머리 등을 분할하여 인식시키는 것이 선행되어야 할 필요성이 보인다. 또한 MLP의 은닉층 수를 늘리거나 조금 더 깊이 있는 모델이 필요할 것으로 보인다.

2. 작품 제작 소요 재료 목록

이름	사진	비고
웹캠 QCAM-M30		1,600만 화소 지원

※참고 문헌

[1] S.Y. Lim, J.D. Huh, “Technology Trends of Context Aware Computing Applications”, Electronics and Telecommunications Trends, vol 19, 31-40, 2004
[2] YoungIm Cho, Sungsoon jang, “Implementation of Intelligent Speech Recognition System according to CCTV Emergency Information”, Journal of Korean Institute of Intelligent systems, 415-420, 2009
[3] Kaiming He, Geogia Gkioxari Piotr Dollar, Ross Girshick, “Mask R-CNN”, Facebook AI Research(FAIR), Computer Vision and Pattern Recognition (cs.CV), 2018
[4] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, “Faster R-CNN: Toward Real-Time Object Detection with Region Proposal, Networks”, Advances in Neural, 2015 Information Processing Systems 28(NIPS 2015), 2016