이미지 기반 실시간 화재 감지 및 구조대상자 파악 AI 시스템 구현

인공지능창의설계 프로젝트
김기영, 박혜민, 배병찬, 여예진

요 약

YOLO v4를 이용하여 추출한 AI 모델로 화재를 감지하고 구조대상자를 파악하는 시스템을 구현했다. Roboflow를 통해 화재 감지를 위한 학습 이미지 데이터셋 약 1700여 장을 수집하여 학습을 진행했다. 일상생활에서 흔히 접할 수 있는 화재부터 재해로부터 일어나는 화재 상황까지 다양한 경우의 이미지를 선별하여 데이터에 다양성을 부여했다. 구조대상자 파악의 경우 실시간으로 사람을 파악하고 감지된 사람의 수를 알려준다. 이 시스템은 실시간으로 화재를 감지하여 사후 신속히 대처 가능하게 하며 구조 대상자 감지 및 인원수 파악을 통해 구조에 편의성을 제공한다.

목차

Ⅰ. 서론

화재 발생 시 빠른 대처를 하는 것은 매우 중요하다. 신속한 대응이 이루어지기 위한 요소들은 소화전을 이용한 화재 사전 제압과 소방차의 화재 현장 도착 시간 등이 있지만 화재 발생 사실을 빨리 인지하는 것이 화재 시 골든 타임을 지킬 가능성을 높인다. 현재 일반적으로 쓰이는 열 탐지 기반 화재 탐지 시스템은 온도 탐지 센서에 의존하기 때문에 실내 면적이나 천장의 높이 등 센서가 설치된 장소의 영향을 받아 탐지 시간이 지연에 영향을 미친다. 이러한 화재 탐지 시스템의 취약점을 해결하기 위해서 기존의 센서를 이용했던 방식을 실시간 영상으로 불을 탐지하는 방식으로 변경했다. 열 탐지 기반의 화재 인식에서 벗어나 CCTV와 같은 영상 감지 장치에 이미지 기반 화재 감지 시스템을 겸용한다면 화재를 사전에 빠르게 탐지할 것이다. 또한 화재 발생 영역 주변의 인원수를 구해 구조대상자를 파악하는 기능을 추가하여 신속한 구조가 가능하도록 한다.

이와 같이 새롭게 구현하고자 하는 시스템은 기존의 화재 감지 시스템과 달리 신속한 초기 발견에 의의를 두므로 작은 객체 탐지에 능한 YOLO v4를 사용하였다. 최근 딥러닝을 이용한 객체 인식 방법이 많이 사용되고 있으나 정확한 학습 이미지 데이터 확보가 객체 인식 모델의 성능을 결정한다. 이를 염두에 두며 다양한 환경에서 얻어진 화재 관련 이미지를 학습시키기 위해 Roboflow에서 불꽃(fire)과 연기(smoking) 데이터셋을 수집하여 선별하였다. 이를 바탕으로 실시간 탐지에 적합한 YOLO v4 darknet을 이용해 화재와 관련된 객체들을 탐지하는 모델과 사람 및 인원수를 감지하는 모델을 구현하였다.

Ⅱ. 화재 및 사람 탐지 관련 사용 기술

1. YOLO (You Only Look Once)

YOLO는 하나의 CNN(convolution neural network)을 이용하여 이미지 전체를 여러 장으로 분할하지 않고 한 번만 보는 실시간 객체 탐지 모델이다. 이미지의 픽셀로부터 bounding box의 위치, 클래스 확률을 구하는 과정을 하나의 회귀 문제로 재정의한 것이므로 이미지를 분할 학습하던 YOLO 이전 기존의 R-CNN과 같은 객체 탐지 모델에 비해 훨씬 빠른 추론 속도와 정확성을 갖는다. 또한, YOLO는 이미지 전체를 처리하기 때문에 클래스의 모양에 대한 정보 뿐만 아니라 주변 정보까지 학습하여 background error를 낮춘다. YOLO를 화재 감지 시스템에 적용했을 때, 불 객체가 없는 배경에서 노이즈가 발생하면 이를 불로 인식하는 오류를 막을 수 있는 것이다. 이러한 특징은 지정된 클래스에 해당하는 어떠한 객체가 감지되는 시간보다 하나도 감지되지 않는 시간이 긴 화재 감지 모델에 적합하다.

2. YOLO알고리즘 - YOLOv4

본 프로젝트에서 YOLO 알고리즘 YOLOv1~v5 중 PANet을 사용하여 다양한 크기의 객체 추출이 가능한 YOLOv4를 사용했다. YOLO 4는 작은 물체 인식에는 높은 성능을 보이지만, 학습 속도 지연 문제가 존재한다. 이를 해결하기 위해서 CSPDarkNet을 백본으로 사용한다. 실제로 YOLOv4는 MS COCO 데이터셋 기준 43.5%의 AP(Average Precision)와 약 70%의 FPS의 성능을 보이며, YOLOv3에 비해 각각 약 10%, 12% 정도 향상되었다.

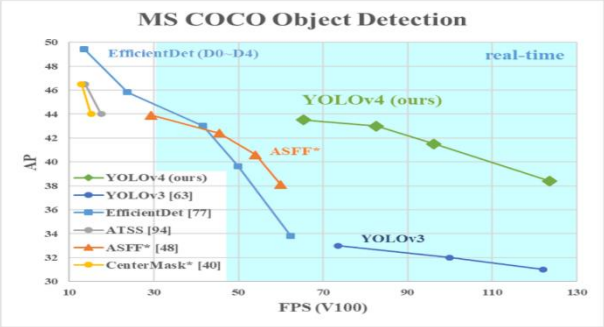


Figure 1. YOLOv4 및 최신 객체탐지 모델 간 성능 비교

Ⅲ. 구현 과정

1. 화재 감지 모델 구현

1) 화재 이미지 데이터 선정 및 라벨링(labeling)

화재 감지 모델은 실내 뿐만 아니라 주로 야외에서 사용되기 때문에 해당 모델 기능을 갖는 장치가 손상되거나 이물질이 포함되는 등 객체의 인식을 방해하는 요소에 많이 노출되어 있다. 이를 고려하여 이상치나 노이즈에도 제대로 분류할 수 있는 roboust 특징을 기준으로 데이터셋을 선정했다. 아래 Figure 2.와 같이 정상 이미지와 손상된 이미지를 데이터로 선정하여 이미지 품질이 불안정한 경우에도 객체를 정확하게 탐지할 수 있도록 훈련시켰다.

Figure 2-1. 정상이미지

Figure 2-2. 손상된 이미지

화재 여부를 판단하기 위해 감지해야 할 객체를 불꽃(fire)과 연기(smoking)로 설정했다. Roboflow에서 fire과 smoke 두 클래스로 구분되어 있는 'Fire and Smoke Training Image Dataset'을 다운로드 받아 약 3500개의 데이터를 얻었다. 다양한 배경에서 발생된 연기와 불꽃 이미지를 5:5의 적절한 비율로 설정하고, 학습 과정 시에 사용될 GPU 성능을 고려하여 최종 1731개의 이미지를 훈련 데이터로 선정하였다.

'fire = 0', 'smoke = 1'과 같이 불꽃의 클래스 넘버를 0, 연기의 클래스 넘버를 1로 한 후에 이미지 각각을 클래스와 좌표로 이루어진 텍스트 파일로 라벨링하여 데이터 전처리를 하였다.

Figure 3-1. 이미지파일 (.jpg파일)

```
1 0.32211538461538464
0.3918269230769231
0.6370192307692307
0.33413461538461536
0 0.6105769230769231
0.5480769230769231
0.19110576923076922 0.203125
```

Figure 3-2. 텍스트파일 (.txt파일)

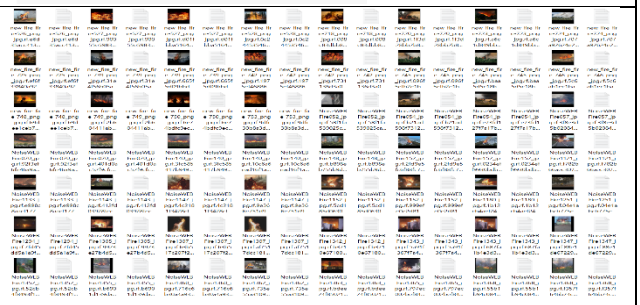
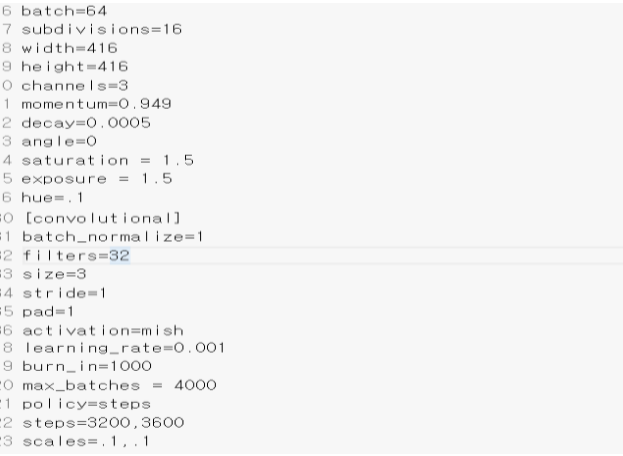
2) 화재 감지 인공지능 모델 구현과 테스트

구글에서 지원하는 colab 학습환경에서 YOLOv4 모델 구현을 진행했다. 사전훈련 가중치 파일 yolov4.conv.137에 전이학습을 하고 yolov4-custom.cfg 파일을 수정하여 커스텀 트레이닝을 진행했다. 빠른 학습 속도를 위해 GPU와 CUDNN을 설치하여 사용하고, training data와 test data 비율은 90:10으로 설정했다.

a) 커스텀 데이터 업로드

obj.data, obj.names, obj.zip, process.py, yolov4-custom.cfg파일을 아래와 같이 구현할 모델에 맞게 수정한 후, 구글 드라이브의 yolov4 폴더에 업로드하여 colab에 마운트 했다.

파일명	용도	수정결과
obj.data	클래스 수, 데이터 파일 경로, obj.names 파일 경로, 백업파일 경로 training폴더에 학습 시 생성되는 가중치 파일을 저장함.	1 classes = 2 2 train = data/train.txt 3 valid = data/test.txt 4 names = data/obj.names 5 backup = /mydrive/yolov4/training

obj.names	클래스 좌표에 따른 클래스명 (0: fire, 1: smoke)	1 fire 2 smoke
obj.zip	라벨링된 데이터셋	
process.py	train.txt, test.txt 파일 생성 후 경로와 파일이름으로 내용 삽입하는 함수	
yolov4-custom.cfg	- max_batches, steps, filters : batch size, image size, steps, iteration 등 training에 대한 정보 클래스 수에 맞게 설정 - learning rate : 기본 값 0.001로 설정	

b) Darknet git reponsitory를 yolov4 폴더에 clone

yolov4폴더 내에 darknet폴더를 생성 후 makefile에서 OPENCV, GPU, CUDNN, CUDNN_HALF, LIBSO를 설치했다.

```
[ ]  !git clone https://github.com/AlexeyAB/darknet
```

```
[ ]  %cd darknet/
     !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
     !sed -i 's/GPU=0/GPU=1/' Makefile
     !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
     !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
     !sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

c) 전이 학습을 위한 사전 훈련된 yolov4 weight파일 다운로드

전이 학습은 어떠한 영역에서 학습한 모델을 다른 영역에 사용함으로써 적은 데이터로 추가 학습할 수 있게 한다. 사전에 역전파를 반복하여 조정된 각 노드의 가중치가 기록된 파일을 다운로드하여 해당 모델의 적은 데이터 수를 보완하였다.

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

d) Training

라벨링 된 데이터 파일, cfg파일, 사전 훈련 가중치 파일로 학습을 진행하였는데 이때 avg loss 값과 mAP값의 변화에 주목하며 훈련 과정을 모니터링했다.

```
[14] !./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show -map
```

training 단계에서 학습 시간이 매우 길거나 학습이 중단되는 문제가 발생했다. 이 문제를 비롯하여 최적의 학습 속도를 구하고 테스트 손실을 줄이기 위해 이미지 데이터 수와 학습률 (learning rate) 하이퍼 파라미터에 변화를 두며 여러 번의 학습을 시도하였다.

(i) 첫 번째 시도

- 이미지 데이터 수: 약 3500개

많은 데이터 수에 의해 학습 시간이 지연되며 GPU할당이 종료되는 문제가 발생했다.

```
 608 x 608
 try to allocate additional workspace_size = 189.86 MB
 CUDA allocate done!
Loaded: 0.000052 seconds
^C
```

(ii) 두 번째 시도

- 이미지 데이터 수 : 1731개
- learning rate : 0.001
- 학습 시간 : 약 2시간 20분

obj 파일에 있는 이미지에 데이터 확장 기술이 과하게 적용된 부분을 발견했다. 정확도를 높이기위해 원래의 학습 데이터에 반전, 회전 등의 변환을 가하여 데이터 양을 늘릴 수 있지만, 과하게 적용되면 특정 훈련 데이터만 학습하는 과적합(Overfitting)이 발생할 우려가 있다. 이를 방지하기 위해 변형된 데이터를 삭제함으로써 학습 데이터의 수를 축소하였다. 이때 기존의 fire과 smoke의 이미지 비율은 유지하며 제거하였다.

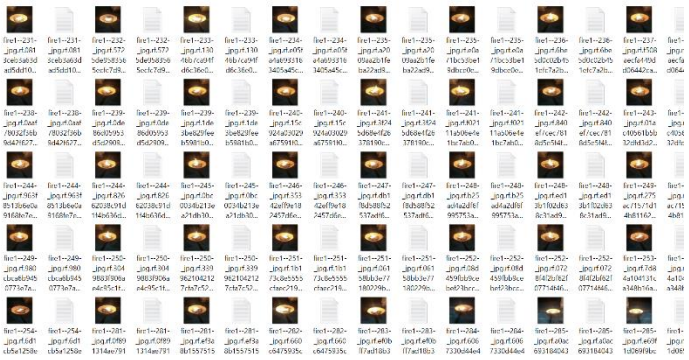


Figure 4. obj 파일에 적용된 Data Augmentation

또한 학습 반복에 따른 avg loss와 mAP에 주목했을 때, 아래 표1과 같이 avg loss의 감소 속도가 느림을 발견했고 그 원인이 낮은 earning late에 있다고 판단했다.

iteration	avg loss	mAP@0.5(%)
10000	5.230429	43.53
20000	3.506853	62.73
2810	3.030481	70.04
3700	2.839432	70.07
40000	2.557897	70.08

표1. iteration에 따른 avg loss, mAP : 최종 avg loss=2.5, mAP=70.08

(iii) 최종 모델

- 이미지 데이터 수 : 1731개
- learning rate : 0.003
- 학습 시간 : 약 1시간 40분

(class_id = 0, name = fire, ap = 75.50% / class_id = 1, name = smoke, ap = 63.95%)

데이터의 수는 그대로 유지하고, yolov4-custom.cfg의 learning rate를 0.003으로 변경했다.

iteration	avg loss	mAP@0.5(%)
10000	4.230984	41.18
20000	3.743452	59.34
2810	2.865643	68.12
3900	1.997565	69.67
40000	2.154356	69.72

표2. iteration에 따른 avg loss, mAP : 최종 avg loss=2.1, mAP=69.72

표2를 보면 두 번째 시도에 비해 avg loss가 약 0.4035 감소하고 mAP값은 거의 유지한 것으로 확인할 수 있다. fire 클래스의 ap는 75.50%이지만 smoke 클래스의 ap는 63.95%로 화재 감지 정확도에 비해 연기 감지 정확도가 떨어졌다.

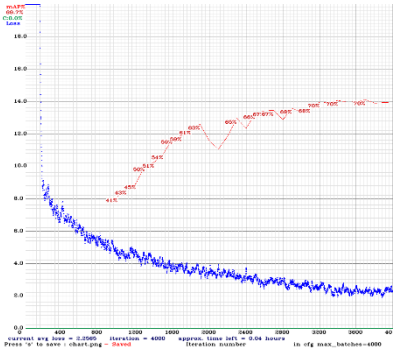
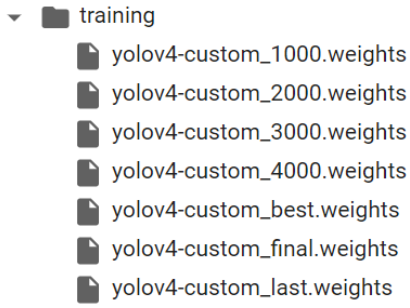


Figure 5. chart.png

e) Test

학습을 수행하는 동안 training 폴더 아래에 1000 iteration 마다 가중치 값을 저장하는 가중치 파일이 아래와 같이 생성되었다.



라벨링 된 데이터 파일, cfg파일, 가장 정확도가 높은 가중치 파일으로 테스트를 진행한다.

3) 구현 모델 결과

구현 모델의 훈련 결과는 아래와 같다.

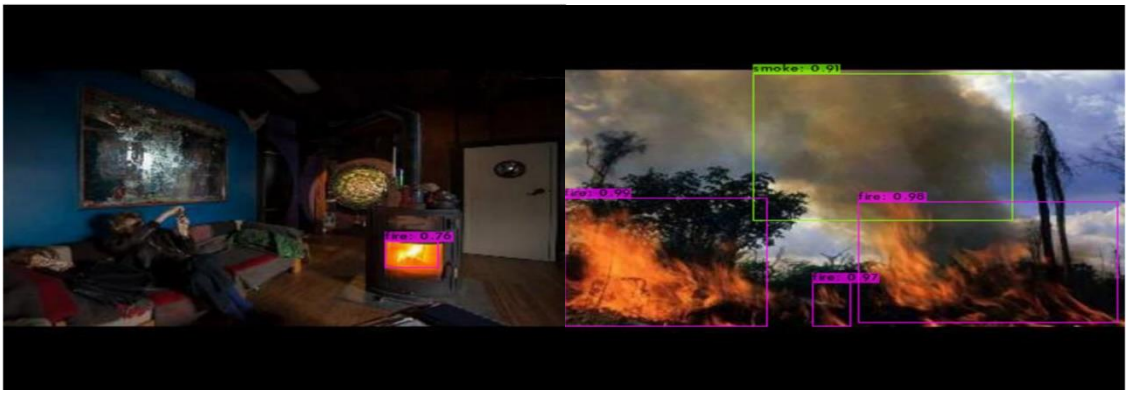


Figure 6. 모델 훈련 결과

a) 오류 1 : smoke 객체 감지

fire 객체 감지의 정확도는 높은 편이지만, Figure 7 그림과 같이 smoke 객체의 일부를 감지하지 못하는 경우가 발생했다. 그러나 화재 발생 판단 기준은 불꽃이나 연기 객체의 유무이기 때문에 두 객체 중 하나라도 감지된다면 화재 발생 가능성이 있

Figure 8. 클래스 반대로 분류되는 경우 우

다고 판단하는 방향으로 모델을 활용하기로 했다.

b) 오류 2 : 클래스의 분류

Figure 8 그림과 같이 클래스가 반대로 분류되는 오류가 발생했다. 오류의 원인을 초기에는 정확도의 문제라고 판단했으나, 파일의 클래스 넘버 지정에서 문제가 있었다. 초반에 obj.zip파일에 라벨링 된 txt파일에서 "fire"클래스의 번호를 0, "smoke"클래스의 번호를 1로 설정했는데 obj.names 파일 안에 fire과 smoke의 위치를 바꾸어서 클래스 넘버를 반대로 지정했다. 클래스 넘버를 다시 정정함으로써 오류를 해결하였다.

4) 보완점

a) smoke 클래스의 ap가 낮게 도출되는데, 이는 데이터셋에 smoke 이미지의 비율을 늘려 정확도를 높일 수 있다. 다만 GPU 사용 제한과 학습 시간의 지연으로 smoke 데이터를 늘리기에 한계가 있어 실제로 데이터를 늘리지 못했다.

b) avg loss 값이 여전히 높다. 기존의 training data와 test data의 비율에서 test data 비율을 늘려 모델의 성능을 여러 번 검증한다면 avg loss값을 줄일 수 있다.

c) 최종 모델의 avg loss가 감소하다 3900 iteration을 기준으로 약간 증가한 것을 확인할 수 있었다. early stopping을 통해 정규화를 한다면 더 낮은 손실 값을 얻을 수 있다.

2. 구조대상자 및 인원수 파악 모델 구현

yolov4폴더 내에 darknet 폴더 생성 후 makefile에서 OPENCV, GPU, CUDNN, CUDNN_HALF, LIBSO를 설치하여 darknet git reponsitory를 yolov4 폴더에 clone 했다.

```
[ ]  !git clone https://github.com/AlexeyAB/darknet
```

```
[ ]  %cd darknet/
     !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
     !sed -i 's/GPU=0/GPU=1/' Makefile
     !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
     !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
     !sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

전이 학습을 위한 사전 훈련된 yolov4 weight 파일을 다운로드하기 위해 yolov4 가중치 파일을 가져왔다. 이 파일은 공유 google 드라이브에서 80개의 클래스(객체)를 감지하도록 미리 훈련하여 조정된 것이다.

```
[ ] # get bthe scaled yolov4 weights file that is pre-trained to detect 80 classes (objects) from shared google drive
    !wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cook
```

객체 감지를 수행하기 위해 다크넷 모듈을 import 하고, ms coco 데이터를 YOLOv4 아키텍처 네트워크에 로드했다. 이미지에서 감지를 실행하는 darknet helper function을 만들어 객체 감지 모델을 불러왔다. 불러온 객체 감지 모델은 80개의 객체를 감지하도록 훈련됐기 때문에 사람만 감지하도록 조정했다. 감지한 모델 중 label에 person이 감지되었을 때에만 바운딩 박스 처리하고 감지한 label을 표시하도록 코드를 수정했다. 또한 구조대상자 인원을 파악하기 위해 사람을 감지했을 때 몇 명인지 세는 기능(counting)을 아래와 같이 코드 작성하여 추가했다.

```
# loop through detections and draw them on transparent overlay image
for label, confidence, bbox in detections:
  if label == "person":
    left, top, right, bottom = bbox2points(bbox)
    left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int(right * width_ratio), int(bottom * height_ratio)
    bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bottom), class_colors[label], 2)
    bbox_array = cv2.putText(bbox_array, "{} [{:.2f}]".format(label, float(confidence)),
                     (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                       class_colors[label], 2)
    count.append([bbox])
cv2.putText(image1, f"COUNT : {len(count)}",
            (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 5,
            (255,0,0), 2)
label_html = f"COUNT : {len(count)}"
person_count=len(count)
count=[]
```

3. 모델 결합

아래 Figure 9과 같이 사람 인원수를 파악한 후 불꽃 및 연기를 감지하도록 두 모델을 병합하였다. 구조대상자 객체와 불꽃 및 연기 객체의 카운팅 수를 기준으로 객체 감지 결과를 출력하여 직관적으로 파악할 수 있도록 코드를 작성했다.

Figure 9-1. 사람 인원수 우선 파악


```
if(fire_count >=1) or (smoke_count >=1):
  print("fire detection")
  print("person:",person_count)
else:
  print("fire is not detect")
```

```
fire detection
person: 2
```

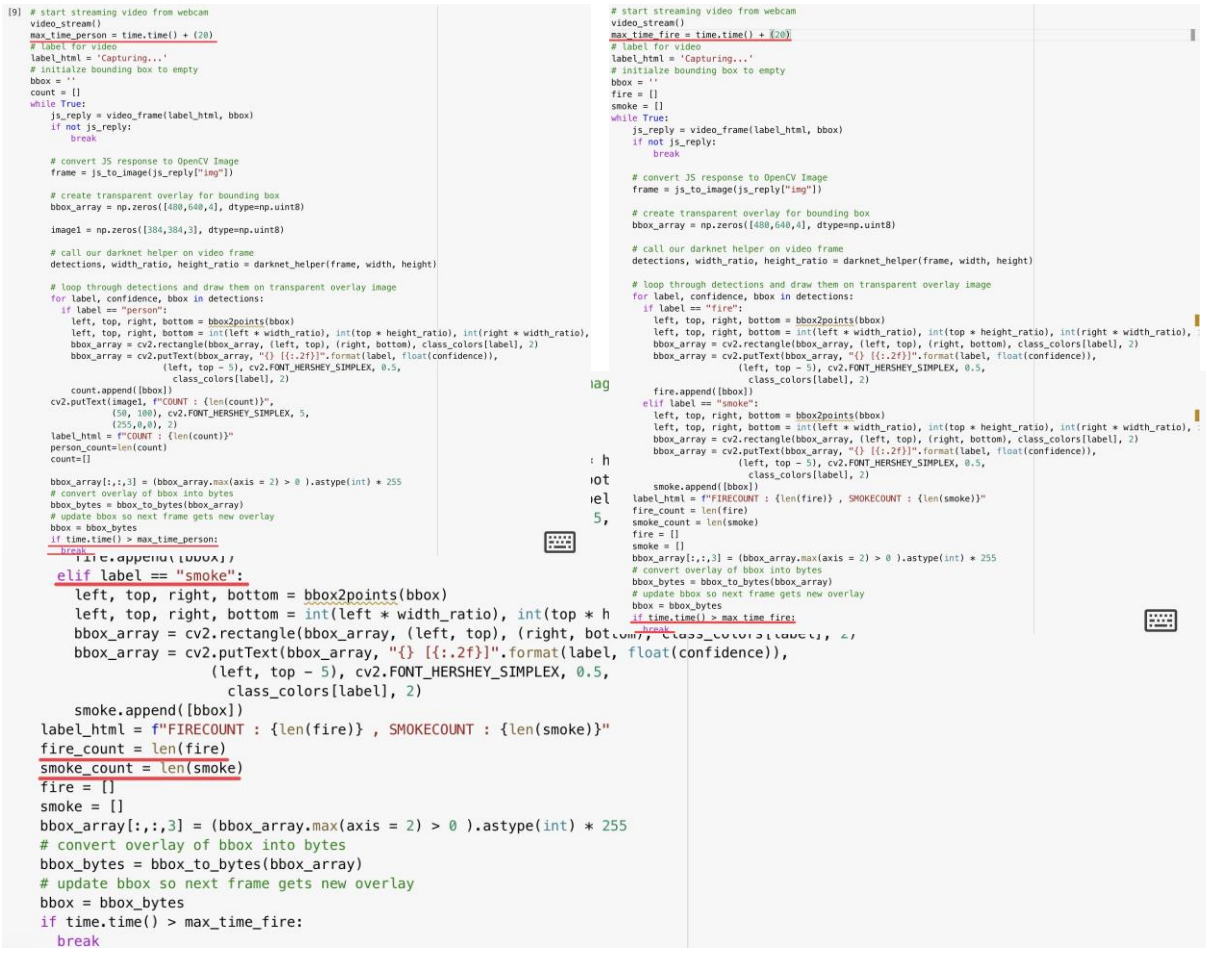
Figure 9-2. 불꽃 및 연기 파악과 결과 출력

1) 불꽃 및 연기 객체 수 카운팅

Figure 9-1의 결과 출력을 위해 사람 객체 뿐만 아니라 불꽃과 연기 감지 수도 카운팅 처리하도록 아래 코드를 작성했다. 바운딩 박스의 수를 카운팅 기준으로 설정하여 각 객체의 개수를 fire_count와 smoke_count에 저장한 후 fire_count 또는 smoke_count가 1 이상이면 화재 감지, 미만이면 화재가 감지되지 않은 것으로 나눴다.

2) 두 모델 간 연결

사람 객체를 세는 모델의 코드에서 무한루프가 발생하기 때문에 불꽃과 연기를 감지하는 모델로 연결되기 위해 break를 사용했다. break 작동의 기준은 바운딩 박스의 시간 제한으로 설정하였다. Time 모듈을 활용해 프로그램 시작 때의 시간에 20초를 더해 변수를 만들고, 바운딩 박스를 만드는 함수 마지막에 조건문을 사용하여 현재 시간이 "시작 때의 시간 + 20초"보다 클 경우에 break를 실행하여 다음 모델로 넘어가도록 했다.



```
# start streaming video from webcam
video_stream()
max_time_person = time.time() + (20)
# label for video
label_html = 'Capturing...'
# initialze bounding box to empty
bbox = ''
count = []
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # convert JS response to OpenCV Image
    frame = js_to_image(js_reply["img"])

    # create transparent overlay for bounding box
    bbox_array = np.zeros([480,640,4], dtype=np.uint8)

    image1 = np.zeros([384,384,3], dtype=np.uint8)

    # call our darknet helper on video frame
    detections, width_ratio, height_ratio = darknet_helper(frame, width, height)

    # loop through detections and draw them on transparent overlay image
    for label, confidence, bbox in detections:
      if label == "person":
        left, top, right, bottom = bbox2points(bbox)
        left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int(right * width_ratio),
        bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bottom), class_colors[label], 2)
        bbox_array = cv2.putText(bbox_array, "{} [{:.2f}]".format(label, float(confidence)),
                         (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                         class_colors[label], 2)
        count.append([bbox])
    cv2.putText(image1, f"COUNT : {len(count)}",
                (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 5,
                (255,0,0), 2)
    label_html = f"COUNT : {len(count)}"
    person_count=len(count)
    count=[]

    bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype(int) * 255
    # convert overlay of bbox into bytes
    bbox_bytes = bbox_to_bytes(bbox_array)
    # update bbox so next frame gets new overlay
    bbox = bbox_bytes
    if time.time() > max_time_person:
      break
    elif label == "smoke":
      left, top, right, bottom = bbox2points(bbox)
      left, top, right, bottom = int(left * width_ratio), int(top * h
      bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bot
      bbox_array = cv2.putText(bbox_array, "{} [{:.2f}]".format(label, float(confidence)),
                       (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                         class_colors[label], 2)
      smoke.append([bbox])
  label_html = f"FIRECOUNT : {len(fire)} , SMOKECOUNT : {len(smoke)}"
  fire_count = len(fire)
  smoke_count = len(smoke)
  fire = []
  smoke = []
  bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype(int) * 255
  # convert overlay of bbox into bytes
  bbox_bytes = bbox_to_bytes(bbox_array)
  # update bbox so next frame gets new overlay
  bbox = bbox_bytes
  if time.time() > max_time_fire:
    break
```

IV. 결론

카메라, 웹캠 등을 통해 얻은 영상을 Yolov4 알고리즘을 이용해 사람 수를 탐지하고, 불꽃 및 연기를 탐지하여 결과창을 출력하는 시스템을 구현하였다. 그러나 구현 모델의 불, 연기 객체 탐지 유무로 화재 발생을 판단하는 데에는 생일 초나 모닥불, 가스레인지 불 등 예외가 존재했다. 불이 다양한 분야에서 사용되기 때문에 불꽃과 연기가 감지되었다고 화재가 발생한 것이 아니다. 이 문제점은 인공지능과 인간의 협업으로 해결 가능하다. 모델이 불과 연기 객체를 탐지했을 때 불 감지 알람이 가게 하여 사람이 화재 여부를 판단하도록 시스템 설계를 변경했다.

이미지 기반 실시간 화재 감지 및 구조대상자 파악 AI 시스템을 활용한다면, 기존의 열 탐지 기반의 화재 탐지 시스템보다 탐지 속도가 빠르고, 정확도도 높으며, 구조의 편의성을 제공하기에 빠른 화재 발생 대처가 가능하다. 또한 이 시스템을 CCTV 뿐만 아니라 IoT에 적용하여 가정 내 생활 화재 진압에 신속함을 더할 수 있다. CCTV와 같이 일상생활 곳곳에 쓰이는 시각적 영상 촬영 장치에 ai 구현 모델이 만연하게 적용되어 화재 피해 최소화의 초석이 될 것으로 기대된다.