백운고등학교 3학년 과학 융합

연구 논문

객체 인식 기술을 활용한

사용자 물건 탐색 시스템 개발

연구팀 이름	EnCoder

연구자	구분	성명
	연구자1	홍준화
	연구자2	현강우

백운고등학교
Paekun High School

객체 인식 기술을 활용한 사용자 물건 탐색 시스템 개발

홍준화 · 현강우

〈 목차 〉

초록

본 연구에서는 YOLO 9000 객체 인식 기술을 활용하여 사용자가 최근 이동된 5개의 물건의 위치를 확인할 수 있는 시스템을 개발하였다. 본 연구의 결과물인 Fything(Find Your Things) 시스템을 출시한다면 많은 사람의 건망증으로 인한 시간 소모를 감축시킬 수 있을 것으로 기대된다.

주요용어 : 객체 인식, YOLO 9000, 차영상 기법, base64, 서버, 앱 인벤터

백운고등학교 Paekun High School

Ⅰ. 서론

1. 연구 동기 및 목적(30122 현강우)

"일에 치여 바쁜 현대인들에게 건망증은 더 이상 노인의 병이 아니다."

바쁜 일상생활을 하는 도중 우리는 종종 중요한 서류를 어디에 놓아두었는지, 방금 있던 TV 리모컨이 어디로 갔는지 잊어버리곤 한다. 이렇듯 현대 사회를 바쁘게 살아가는 우리에게 자주 까먹는 습관은 떼놓기 어렵다. 가끔씩 위치를 잊어버린 물건이 어디에 있는지 바로 찾아주는 시스템이 있다면 좋을 것 같다는 생각을 해 본 적이 있을 것이다. 하지만 이러한 문제를 해결하기 위해 물건 하나하나에 위치추적 장치를 부착하는 것은 비용적인 측면에서 상당히 비효율적일 수 있다. 따라서 본 연구에서는 물건 각각에 장치를 부착해야 하는 시스템이 아닌 객체 인식 기술을 활용한 물건 탐색 시스템을 제시하고자 한다. 본 연구에서 고안한 시스템에서는 방이나 사무실 같은 특정 공간에 카메라를 설치해 두고, 카메라에서 움직임이 인식되면 움직임이 멈추었을 때 객체 인식을 통해 사물의 위치를 검출하여 사진을 서버에 저장한다. 이에 따라 사용자는 스마트폰 어플리케이션을 통해 간편히 서버에 저장된 물체의 위치를 실시간으로 확인할 수 있다. 이러한 시스템을 통해 사용자가 실시간으로 자신이 물건을 둔 위치를 확인할 수 있다면 건망증으로 인한 시간을 효과적으로 감축시킬 수 있을 것으로 기대된다.

2. 구현 방법

(1) 영상 촬영 및 객체 인식(30223 홍준화, 30122 현강우)

본 연구에서는 노트북의 웹캠을 이용해 영상을 촬영하고 OpenCV[1]로 차영상을 검출하여 움직임을 감지할 것이다. 또한 YOLO[2] 9000을 활용해 객체 인식을 구현할 것이다.

(2) 서버 및 이미지 송수신(30223 홍준화)

서버는 Python의 Flask 모듈을 통해 구현될 것이며 이미지 송수신은 base64 암·복호화를 통해 텍스트로 이루어질 것이다.

(3) 사용자 어플리케이션 개발(30122 현강우)

사용자 어플리케이션은 MIT의 앱인벤터를 활용해 개발될 것이다.

Ⅱ. YOLO 9000을 활용한 객체 인식

1. 객체 인식 기술(30223 홍준화, 30122 현강우)

(1) 객체 인식 기술의 종류(30223 홍준화)

딥러닝을 활용한 객체 인식 알고리즘은 크게 이단계 방식(Two-Stage Methods)과 단일 단계 방식(Single-Stage Methods)으로 분류된다. 이단계 방식 객체 인식 알고리

---

1) OpenCV: Open Source Computer Vision, 실시간 컴퓨터 영상 처리를 목적으로 한 라이브러리

2) YOLO: You Only Look Once, 딥러닝을 통해 객체 인식을 수행하는 알고리즘

즘은 객체를 포함할 가능성이 높은 영역들을 선택적으로 탐색하는 알고리즘으로, FasterR-CNN, R_FCN, and FPN-FRCN 등의 알고리즘이 이 방식을 사용한다. 단일 단계 방식 객체 인식 알고리즘은 원본 이미지를 여러 개의 영역으로 나누고 각 영역 내에서 객체의 개수를 예측하여 객체를 검출하는 알고리즘으로, YOLO, SSD 등의 알고리즘이 이 방식을 사용한다. 이단계 방식 알고리즘은 정확도는 높지만 속도가 비교적 느리다. 반면 단일 단계 방식 알고리즘은 정확도가 비교적 낮고 속도가 빠르다. 본 연구에서는 단일 단계 방식 알고리즘 중 비교적 정확도가 높은 YOLO를 채택하였다.

[그림1]객체 인식 알고리즘의 속도 및 정확도 비교

(3) YOLO(30223 홍준화)

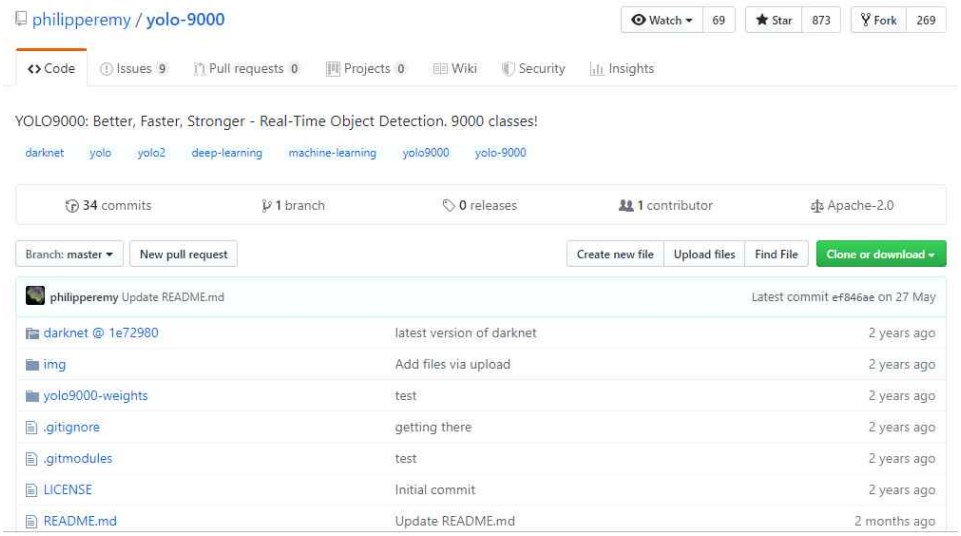
YOLO는 You Only Look Once의 약자로 짧은 순간에 사물들을 포착할 수 있다는 뜻을 내포하고 있는 객체 인식 프로그램이다. YOLO 연구(REDMON, Joseph, et al. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p.1)에 따르면 YOLO는 입력된 이미지를 448x448 크기로 나누어 이미지 내에서 합성곱 신경망[3]을 실행하고, 객체 존재 확률값을 계산하여 인식 결과를 출력한다. 본 연구에서는 9000개의 데이터가 학습되어 있는 YOLO 9000을 활용하여 객체 인식을 진행할 것이다.

3) 합성곱 신경망(CNN, Convolutional Neural Network): 입력 데이터에 필터를 적용하여 특징을 찾아 작은 데이터로 변환하는 신경망, 영상 처리에서 주로 사용된다.

2. 객체 인식 구현(30223 홍준화, 30122 현강우)

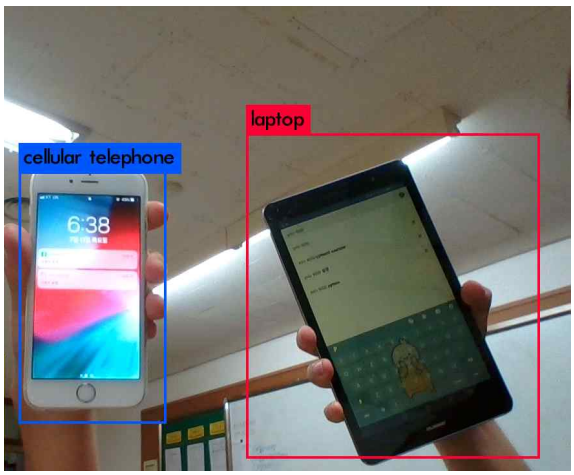
(1) YOLO 9000 다운로드 및 실행(30122 현강우)

YOLO 9000은 philippremy의 Github[4]에서 다운로드 할 수 있다.



[그림2]YOLO 9000 Github 저장소

yolo-9000/darknet 폴더에서 './darknet detector test cfg/combine9k.data cfg/yolo9000.cfg ../yolo9000-weights/yolo9000.weights 변환할 이미지' 를 입력하면 yolo 9000이 실행되면서 prediction.png로 결과를 출력해준다. CPU로 YOLO 9000을 실행시켰을 때 약 7~10초의 시간이 소요되는 것을 확인할 수 있었다.

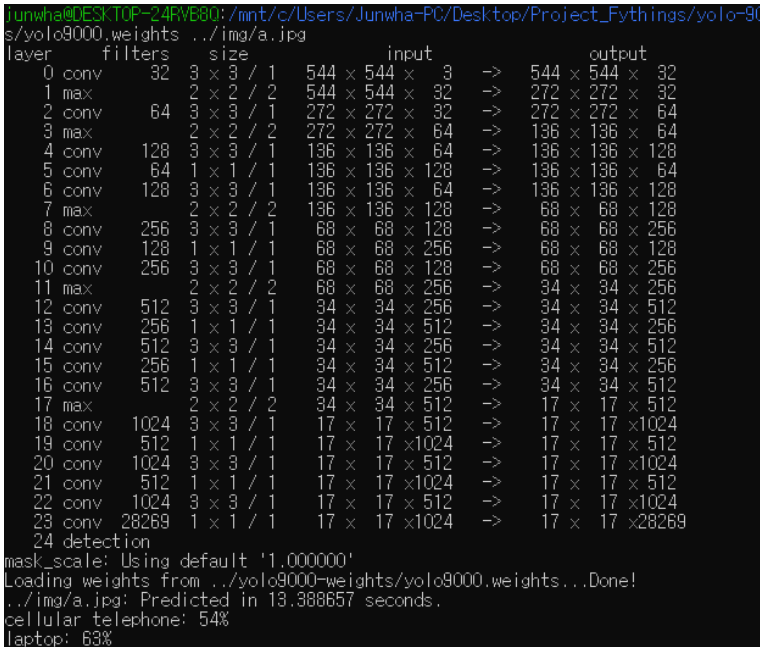


[그림3]아이폰과 태블릿을 감지한 predictions.png

[그림3]과 같이 아이폰을 cellular telephone으로, 태블릿을 laptop으로 인식한

4) https://github.com/philipperemy/yolo-9000

것을 확인할 수 있었다.



[그림4]콘솔 창의 출력 결과

콘솔 창에서는 YOLO 9000이 객체를 탐색하는 과정이 나타난다. [그림4]에서 이미지를 544x544로 나누고 23번을 더 쪼개어 24번째에 객체를 인식한 것을 확인할 수 있었다. cellular telephone은 54%의 신뢰도, laptop은 63%의 신뢰도로 인식되었다.

(2) 파이썬에서의 객체 인식 실행(30223 홍준화)

본 연구에서 YOLO 9000의 역할은 웹캠에서 찍은 사진에서 객체를 인식하여 표시한 사진을 출력하는 것이다. 이에 따라 파이썬 코드를 웹캠에서 찍은 사진이 위치한 C 드라이브 하위 폴더에서 다섯 개의 이미지를 읽어와 각각 변환하도록 구현하였다.

```
#콘솔 조작 모듈
import os
#시간 모듈
import time

#bash 명령을 통해 우분투를 실행시킨다
os.system('bash')

#darknet 폴더로 이동한다
os.system('cd/mnt/c/Users/JUNWHA/Desktop/Project_Fythings/yolo-9000/darknet')
```

```
#num변수를 1부터 5(6-1)까지 증가시키며 반복
for num in range (1,6):
   #콘솔 명령을 통해 YOLO 9000을 실행시키고 N번째 이미지를 인식시킨다
   os.system('./darknet  detector  test  cfg/combine9k.data  cfg/yolo9000.cfg
../yolo9000-weights/yolo9000.weights /mnt/c/a'+str (num)+'.jpg')
   #이미지가 인식되는 동안 대기한다(10초)
   time.sleep(10)
```

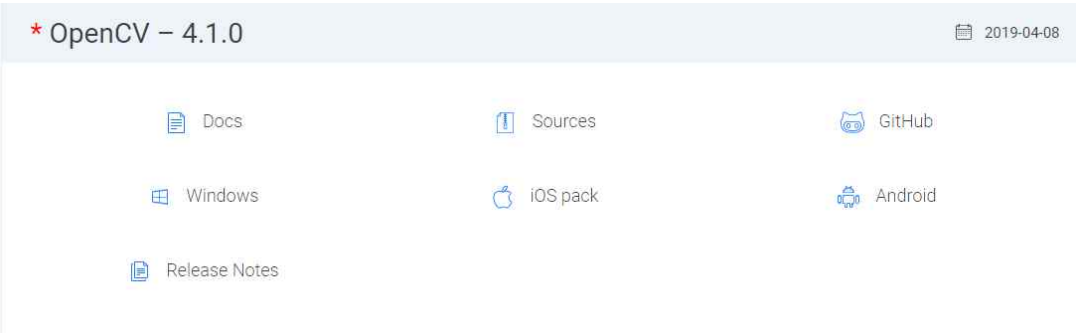
[코드1]YOLO 9000을 실행시키는 파이썬 코드

Ⅲ. 시스템 개발 과정

1. 차영상 기법을 활용한 물건 이동 탐지(30223 홍준화)

(1) OpenCV

OpenCV는 Intel에서 개발한 영상 처리 라이브러리로 이진화, 노이즈 제거, 외곽선 검출, 패턴 인식, 기계학습, 이미지 변환 등의 컴퓨터 비전과 영상 처리에 유용한 알고리즘들을 함수로 제공한다. 본 연구에서는 물건 이동 탐지를 위해 Python3에서 OpenCV 4.1.0 버전을 사용할 것이다.

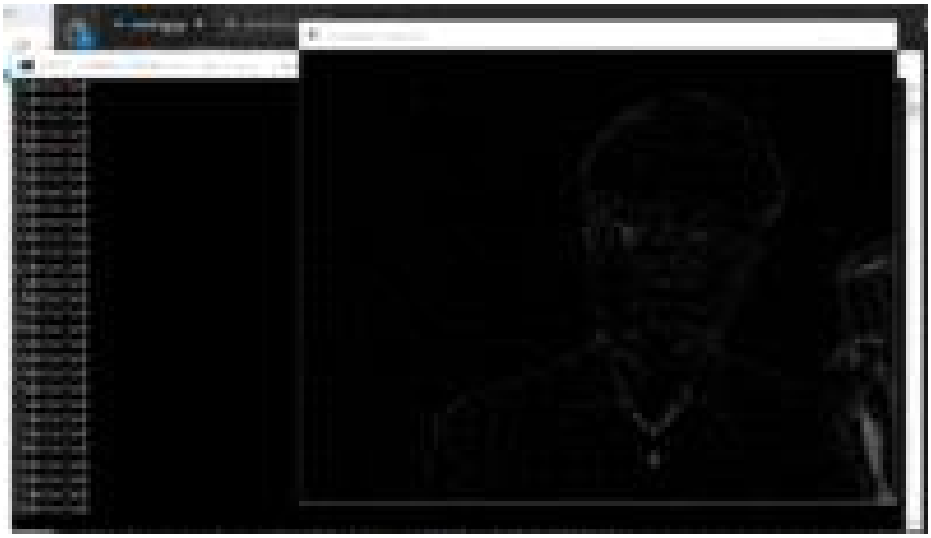


[그림5]OpenCV 4.1.0

(2) 물건 이동 탐지

웹캠에서 물건이 이동하는 것을 탐지하기 위해 차영상 기법을 사용하였다. 차영상 기법은 영상의 픽셀 데이터의 차이를 얻어내는 기법이다.

차영상을 얻어내기 위해 사진을 연속적으로 3회 촬영하여 흑백으로 변환하고, OpenCV의 absdiff 함수를 사용해 첫 번째 사진과 두 번째 사진, 두 번째 사진과 세 번째 사진의 픽셀 데이터 차이를 구하도록 하였다. 또한 이 과정을 통해 얻어진 두 픽셀 데이터에 AND 연산을 수행하여 하나의 사진으로 출력되도록 하였다.



[그림6]AND 연산을 수행한 이미지

사진을 흑백으로 변환시키면 픽셀 데이터가 명도에 따라 연속적으로 나타나게 되는데, 흑백으로 변환시킨 두 사진의 차를 구한 사진의 픽셀 데이터가 양수인 부분이 물건이 이동한 위치이다. 이때, 세 개의 사진의 차를 구해 두 값을 AND 연산을 시키면 오차 범위가 줄어들게 된다.

```
#OpenCV 모듈
import cv2
#시간 모듈
import time
#저장 횟수 전역 변수
n = 1
#flag(저장 코드 실행 여부 결정)
flag = True
#두 사진의 차를 구하는 함수
def diffImg (t0 , t1 , t2 ):
    #t0과 t1의 사진을 서로 뺀다
    d1 = cv2.absdiff(t2,t1)
    #t1과 t0을 서로 뺀다.
    d2 = cv2.absdiff(t1,t0)
    #d1과 d2에 대해 and 연산을 수행하여 리턴한다.
    return cv2.bitwise_and(d1, d2)
#웹캠 촬영
video = cv2.VideoCapture(0 )
#창 이름 설정
winName ="Movement Detector"
cv2.namedWindow(winName)
#흑백 변환
t_minus = cv2.cvtColor(video.read()[1], cv2.COLOR_RGB2GRAY)
```

```
t = cv2.cvtColor(video.read()[1], cv2.COLOR_RGB2GRAY)
t_plus = cv2.cvtColor(video.read()[1], cv2.COLOR_RGB2GRAY)
#감지 횟수 전역 변수
detectedNum =0

while True :
    #세 영상의 차의 AND를 구한다
    diff = diffImg(t_minus, t, t_plus)
    sum1 =0
    sum0 =0
    #각각의 픽셀 데이터를 검사해 0보다 크면 1로 설정한다
    for i in diff:
        for j in i:
            if j >0 :
                j =1
                sum1 +=1
            else:
                sum0 +=1
    print ('sum0: '+str (sum0))
    print ('sum1: '+str (sum1))
```

[코드2]세 영상의 차를 구해 AND 연산을 수행하는 코드

세 영상의 차를 구해 AND 연산을 수행하여 양수인 부분을 1로, 음수인 부분을 0으로 설정해 출력하도록 [코드2]와 같이 구현하였다. 실험적으로 측정해본 결과 물체가 움직이면 값이 1인 픽셀 데이터의 개수가 110,000개 이상이 되었으므로, 역치를 110,000개로 설정하여 물체를 감지하도록 구현하였다.

```
        #1의 개수가 11만보다 크면 움직임이 감지된 것으로 판단
        if sum1 >=110000 :
                print (str (detectedNum)+'detected')
                detectedNum += 1
                #flag를 True로 변경
                flag =True
        #11만보다 작을 때(움직임 감지X)
        else :
                #감지가 되었었다면 물체의 이동이 멈춘 것으로 판단
                if flag:
                        #이미지 캡쳐 후 저장
                        cv2.imwrite('C:/a'+n +'.jpg',video.read()[1 ])
                        if n ==5 :
                                n =0
                        n +=1
                        #flag를 False로 변경
                        flag =False
```

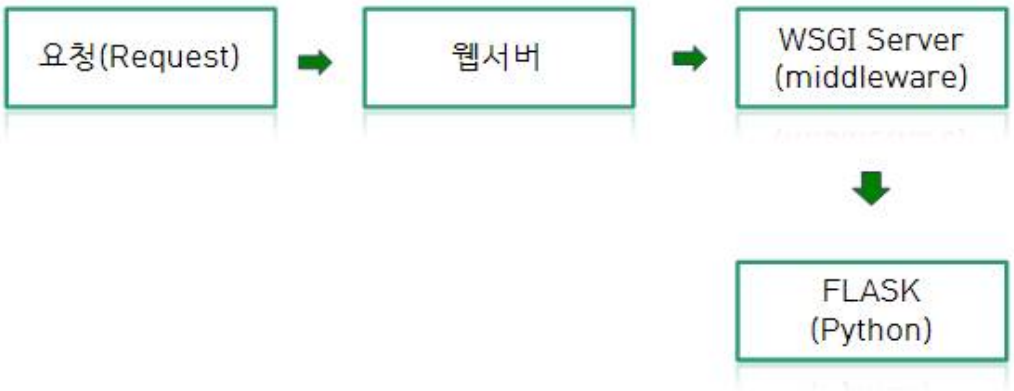
[코드3]값이 1인 픽셀 데이터의 개수가 110,000개 이상이면 이미지 저장

[코드3]에서는 값이 1인 픽셀 데이터의 개수가 110,000개 이상이면 flag가 True가 되고, 값이 1인 픽셀 데이터의 개수가 110,000 개 미만이 되었을 때 N 번째 사진을 C 드라이브의 하위 폴더에 저장한 후 flag를 False로 바꾼다. 즉, 물체의 이동이 감지되면 물체가 이동을 멈추었을 때의 순간을 저장한다.

2. Flask 서버 개발(30223 홍준화)

(1) Flask

Flask는 파이썬에서 서버를 구축하기 위해 사용되는 마이크로 웹 프레임워크로, WSGI 마이크로 프레임워크로 불리기도 한다. WSGI(Web Server Gateway Interface)는 파이썬 스크립트가 웹 서버와 통신하기 위한 중간 단계로 서버의 관점에서는 클라이언트처럼, 클라이언트의 관점에서는 서버처럼 행동한다. 따라서 Flask는 파이썬에서 자유롭게 기능을 구현하면서도 서버의 형태를 유지할 수 있다는 장점이 있는 프레임워크이다.



[그림7]Flask 서버의 통신 구조

(2) 서버 구축

Flask에서는 [코드4]와 같이 서버 경로마다 경로 접속 시 실행될 함수를 설정하여 서버를 구축할 수 있다. app.run() 함수의 매개 변수에서 host는 바인딩할[5] ip주소이고, port는 개방할 포트이다.

```
#Flask 모듈
from flask import Flask
```

5) 서버 소켓(출력부)과 ip주소를 연결하는 작업

```
#앱 객체
app = Flask(__name__)
#루트 주소
@app.route ('/')
def main ():
	return 'server is initiated'
#바인딩 호스트 및 개방 포트 설정
if __name__=='__main__':
	app.run(host ='0.0.0.0', port =5000 )
```

[코드4]서버 구축

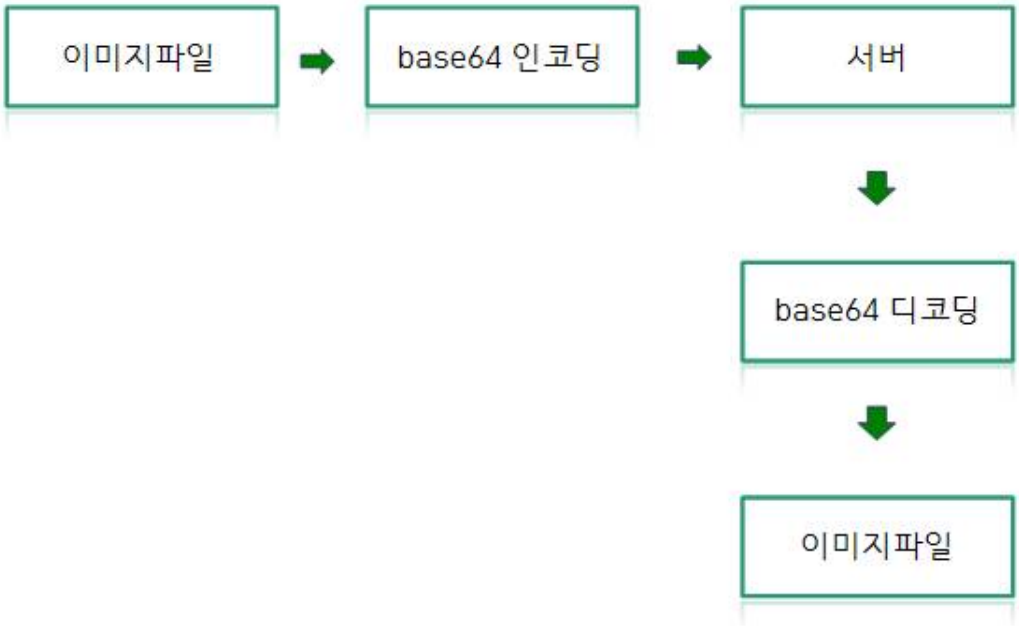
[코드4]에서는 0.0.0.0에 서버 소켓을 바인딩하고 5000번 포트를 열었으므로 http://localhost:5000/, http://컴퓨터의 IP 주소:5000/[6], http://127.0.0.1:5000/ 등의 주소로 접속할 수 있다.

[그림8]서버에 접속한 모습

(3) 이미지 통신 설계

컴퓨터에서 이미지는 파일 헤더, 정보 헤더, 색상 테이블, 픽셀 데이터를 바이너리 형태로 저장하고 있다. 서버 통신에서 전달되는 정보들은 대부분 json[7] 자료구조에 담겨 텍스트의 형태로 전달되는데 그 이유는 텍스트가 가장 처리되기 쉬운 자료형이기 때문이다. 따라서 본 연구에서는 바이너리로 이루어진 이미지 데이터를 송신 전 base64로 인코딩하여 텍스트 데이터로 만들고, 수신 시 base64로 디코딩하여 바이너리 데이터로 변환하는 로직을 채택하였다.

6) 컴퓨터의 IP주소는 명령프롬포트에서 ipconfig 명령어를 통해 확인할 수 있다.
7) JSON, JavaScript Object Notation, 키-값 쌍으로 이루어진 자료구조



[그림9]base64 인코딩/디코딩을 활용한 이미지 송수신 로직

(3) 이미지 통신 송신부 구현

본 연구의 송신부는 [코드1]의 YOLO 실행 프로그램이므로 [코드1]에 이미지를 base64로 인코딩하는 부분을 추가하여야 한다.

```
#Request 모듈
import requests
#base64 모듈
import base64

----[코드]중략----

    #IP 설정
    IP ='http://172.30.1.60'
    #포트 설정
    PORT ='5000'
    #IP와 포트를 합한 upload 경로 구성
    URL = IP +':'+PORT +'/upload'
    #이미지를 열어 base64로 인코딩 후 이미지 번호와 함께 POST Request로 전송
    requests.post(URL,data ={'number':num,
'file':base64.b64encode(open('/mnt/c/Users/JUNWHA/Desktop/Project_Fythings/yo
lo-9000/darknet/predictions.png','rb').read())})
```

[코드5]이미지를 base64로 인코딩 후 서버에 전송하는 코드

[코드5]에서는 파이썬의 base64 모듈의 b64encode() 함수를 활용해 이미지의 바이너리 데이터를 암호화 된 텍스트 데이터로 바꾸고 requests 모듈을 통해 서버의 /upload 경로에 POST Request로 이미지를 전달한다. POST Request는 GET Request와 달리 데이터를 주소에 포함하지 않아 로컬 PC에서의 데이터 탈취를 막을 수 있다.

(4) 이미지 통신 수신부(서버) 구현

본 연구에서 base64로 인코딩 된 이미지 데이터를 수신하는 곳은 서버이다. 따라서 서버 구현 코드인 [코드4]에 이미지를 디코딩하여 서버에 저장하는 기능과 앱 인벤터에 이미지에 접근 가능한 이미지 주소를 제공하는 기능을 구현하여야 한다.

```
#Flask, 파일 전송, Request 모듈
from flask import Flask, send_file, requests
#base64 모듈
import base64

----코드 중략----

#각 번호의 이미지 주소에 이미지 전달
@app.route ('/image1')
def returnImage1 ():
	return send_file('C:/b1.png')
@app.route ('/image2')
def returnImage2 ():
	return send_file('C:/b2.png')
@app.route ('/image3')
def returnImage3 ():
	return send_file('C:/b3.png')
@app.route ('/image4')
def returnImage4 ():
	return send_file('C:/b4.png')
@app.route ('/image5')
def returnImage5 ():
	return send_file('C:/b5.png')

#POST Request로 받은 이미지 처리
@app.route ('/upload',methods =['POST'])
def uploadImage ():
if request.method =='POST':
	#이미지
	encodedFile = request.form['file']
	#번호
	number = request.form['number']
```

```
        #base64로 디코딩 후b번호.png로 저장
        with open ('C:/b'+number +'.png', 'wb') as f:
                f.write(base64.decodebytes(encodedFile.encode()))
                f.close()
                return 'succeed'

----코드 후략----
```

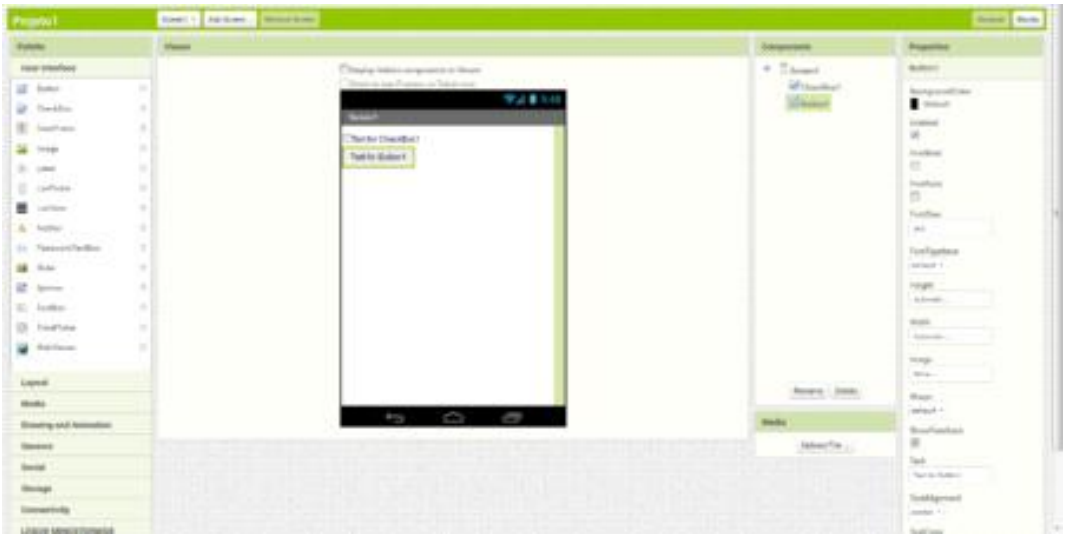
[코드6]이미지 통신 수신부의 기능이 구현된 서버

[코드6]에서는 사용자가 이미지 주소에 접속할 때 Flask의 send_file() 함수로 로컬의 이미지 파일을 전송한다. 또한 사용자가 POST Request로 인코딩 된 이미지 파일을 전송하면 base64 모듈의 decodeBytes()함수를 사용해 이미지 파일을 바이너리로 변환하여 로컬에 저장한다.

3. 앱 인벤터를 활용한 사용자 어플리케이션 개발(30122 현강우)

(1) 앱 인벤터

앱 인벤터(App Inventor for Android)는 구글이 개발한 오픈소스 웹 애플리케이션으로, 현재는 매사추세츠 공과대학교(MIT)에 의해 관리되고 있다.



[그림10]MIT APP INVENTOR2 접속 화면

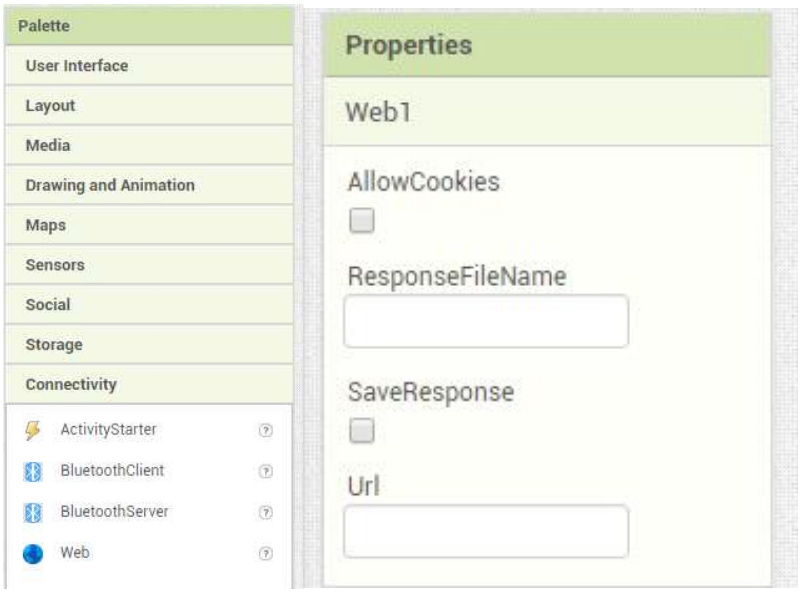
안드로이드 개발을 위해서는 JAVA 언어 또는 코틀린 언어를 사용하여야 하지만 이러한 언어는 매우 복잡하고 사용하기가 어려워 학생들이나 비전공자들이 접근하기에 어려움이 있다. 앱 인벤터는 스마트폰 하드웨어 기능, 웹 통신, 미디어 출력 등의 기능을 블록코딩을 통해 접근할 수 있게 개발된 프로그램이다. 따라서 앱 인벤터를 활용하면 학생들이나 비전공자들이 복잡한 언어들을 배우지 않아도 손쉽게 스

마트폰 앱을 개발할 수 있다.

본 연구에서는 웹 통신을 통해 사용자가 서버에 이미지를 요청하여 수신하는 간단한 기능만을 구현하여야 하므로, 접근성이 뛰어난 앱 인벤터를 활용하는 것으로 결정하였다.

(2) 웹 통신

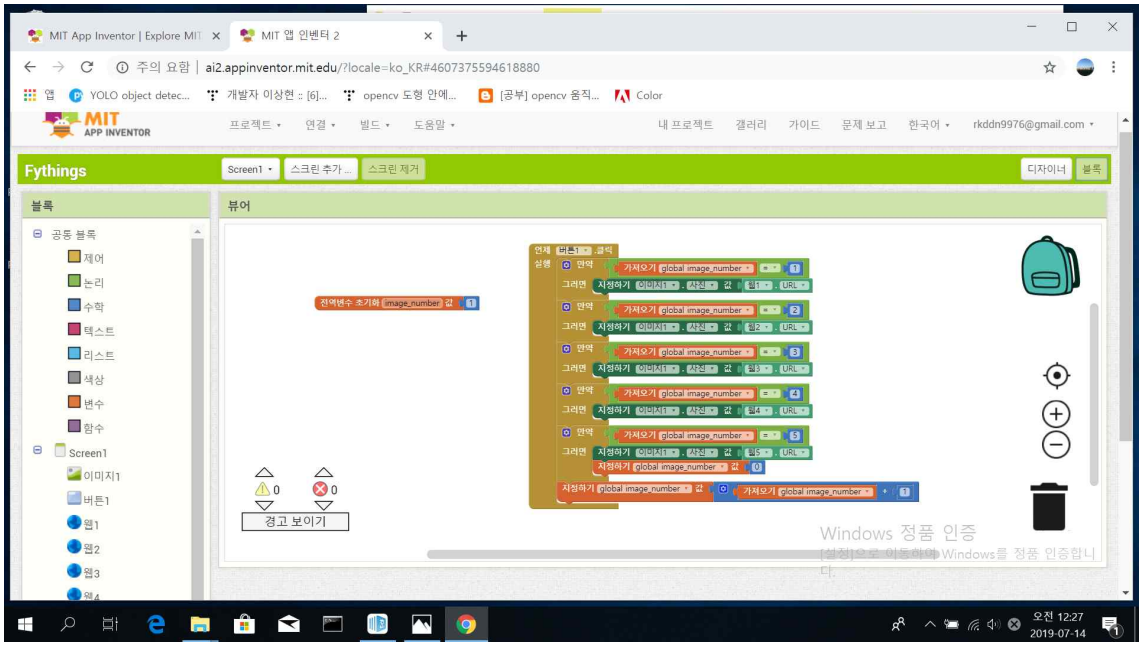
본 연구의 사용자 어플리케이션은 Flask 서버에서 제공하는 이미지 주소 5개를 받아와 사용자에게 순차적으로 보여주어야 한다. 이를 위해서는 앱 인벤터의 웹 통신 기능에 대한 이해가 필요하다. 앱 인벤터에서는 Palette의 Connectivity에서 웹 통신 기능을 제공한다. Web을 끌어와 Properties를 보면 Url을 입력할 수 있는데, 이곳에 GET Request를 요청할 URL을 입력하면 서버로부터 데이터를 받아올 수 있다.



[그림11]Palette의 Connectivity(좌)와 Properties의 Url(우)

(3) 순차적 이미지 출력

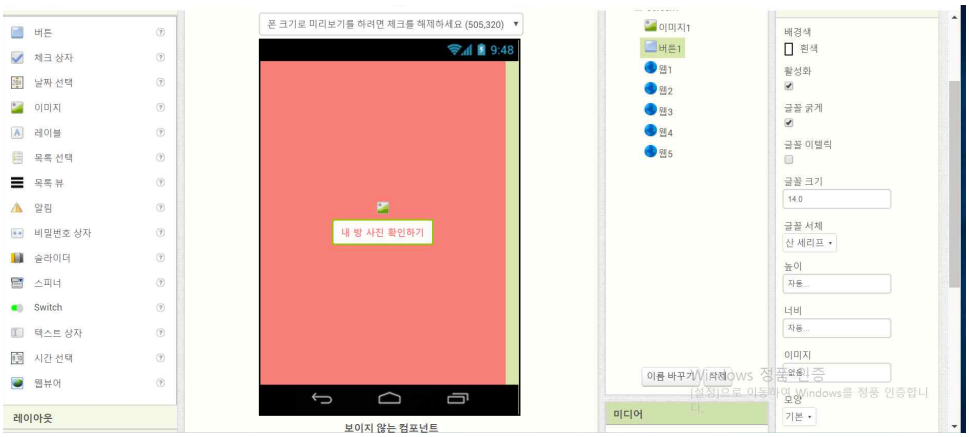
사용자에게 5장의 순차적 이미지를 출력해주는 로직으로 전역 변수의 초기값을 1로 설정하고, 전역 변수의 값에 해당하는 이미지를 출력한 후 전역 변수를 증가시키는 로직을 설계하였다. 또한 5번째 이미지를 출력할 때는 전역 변수를 0으로 설정해 전역 변수를 증가시켰을 때 그 값이 1이 되도록 하였다. 분기문으로는 만약~/그러면~을 사용하였고, 전역 변수의 이름은 image_number로 설정하였다



[그림12]전역 변수에 따른 이미지 출력

(4) UX/UI 디자인

본 연구에서는 사용자가 명확히 기능을 파악하고 사용할 수 있도록 어플리케이션을 사용자 경험(UX, User Experience)과 사용자 인터페이스(UI, User Interface)를 고려하여 디자인하였다. 버튼과 이미지를 가져와 앱 화면 중앙에 배치하고, 버튼의 이름을 '내 방 사진 확인하기'로 설정하여 사용자가 버튼의 기능을 예측하도록 한다. 사용자는 버튼을 반복하여 누르는 경험을 통해 5장의 사진이 순차적으로 제시된다는 것을 확인하여 어플리케이션의 기능을 예측할 수 있을 것이다.



[그림13]디자인 화면

사용자가 시각적 피로를 느끼지 않고 어플리케이션을 사용할 수 있도록 어플리케이션 기본 색상을 #F27C76으로 설정하였으며, Adobe Color CC를 사용해

적절한 색 조합을 사용하도록 하였다.

[그림14]앱 아이콘

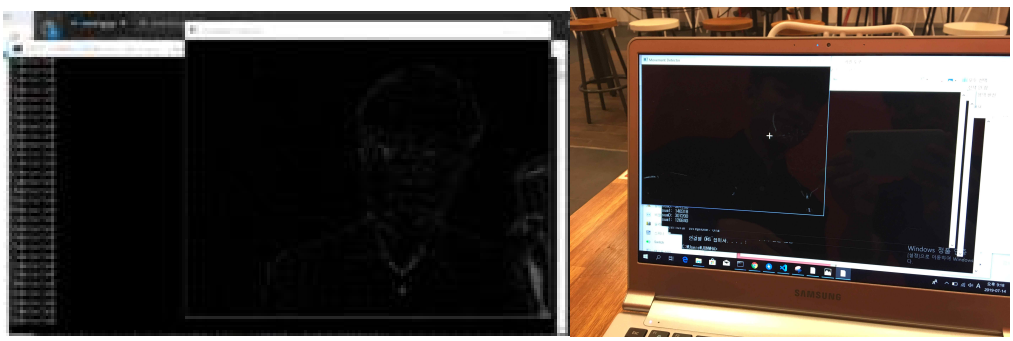
또한 앱 아이콘과 로고를 돋보기에서 오브젝트를 탐색하는 모양으로 만들어 사용자가 앱 아이콘만으로도 기능을 예측하고 범주화할 수 있도록 설계하였다. 또한 사용자가 어플리케이션의 이름만으로 어플리케이션의 기능을 예측할 수 있도록 전체 프로그램의 이름과 어플리케이션의 이름을 'Find Your Things'를 축약한 'Fythings'로 명명하였다.

Ⅳ. 구현 결과

1. 결과 화면(30223 홍준화, 30122 현강우)

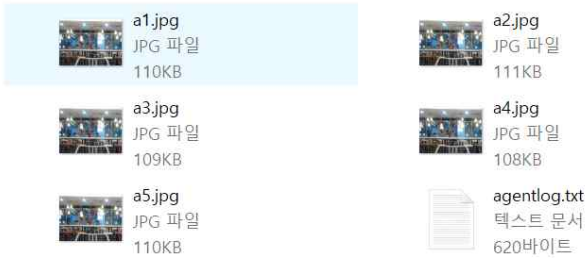
(1)부터 (4)까지 순서대로 실행하였다.

(1) 사물 움직임 인식 화면(30223 홍준화)



[그림15]사물이 움직이는 모습 캡쳐(좌)와 외부 촬영(우)

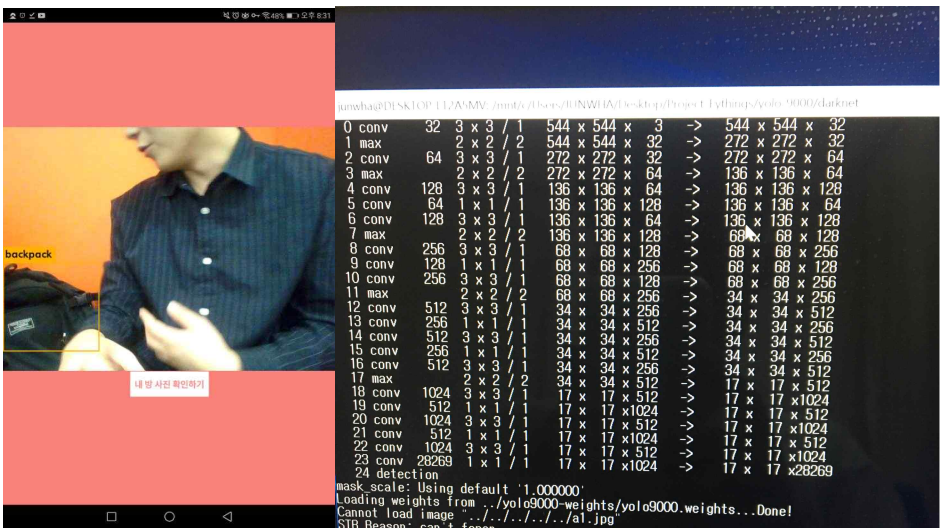
사물의 움직임이 인식되면 움직임이 끝난 후 캡쳐가 이루어진다. 캡쳐된 사진이 저장된 것을 [그림]과 같이 C 드라이브의 하위 폴더에서 확인할 수 있다.



[그림16]C 드라이브 하위 폴더에 a1~a5.jpg가 저장된 모습

(2) 객체 인식 화면(30223 홍준화, 30122 현강우)

C 드라이브 하위 폴더에서 a1~a5.jpg를 읽어와 YOLO 9000으로 변환하여 서버에 b1~b5.jpg를 업로드 한다.



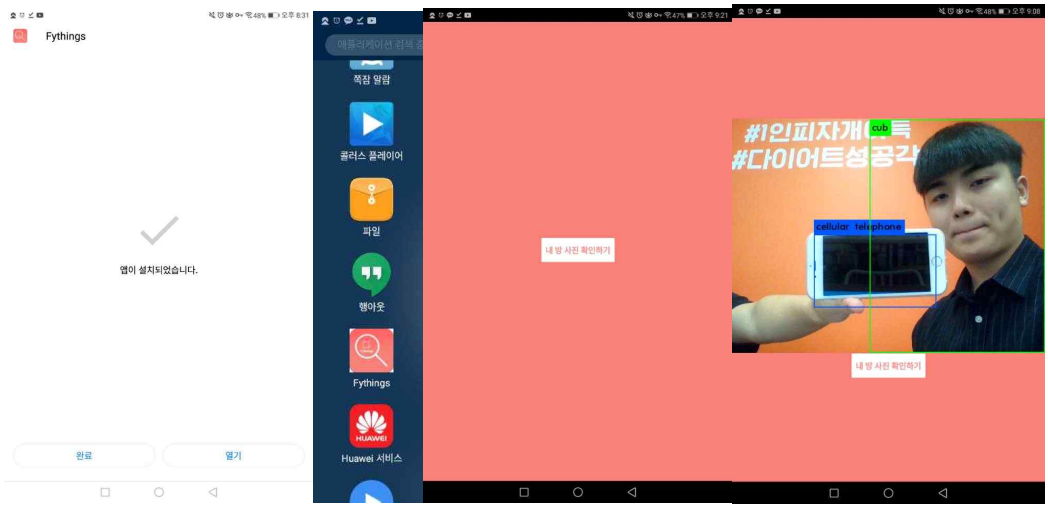
[그림17]가방에 대한 객체 인식이 이루`어진 사진(좌)와
콘솔에서의 객체 인식 과정(우)

(3) 어플리케이션 화면(30122 현강우)

앱 인벤터2 사이트에서 개발한 어플리케이션을 QR 코드[8]로 받아 설치한다.

8) 어플리케이션의 QR코드는 2시간 동안만 유효하므로 현재는 해당 QR 코드로 어플리케이션을 다운로드 할 수 없다.

[그림18]어플리케이션 QR 코드



[그림19]어플리케이션 설치 후 모습(좌)와 어플리케이션 접속 후 모습(우)

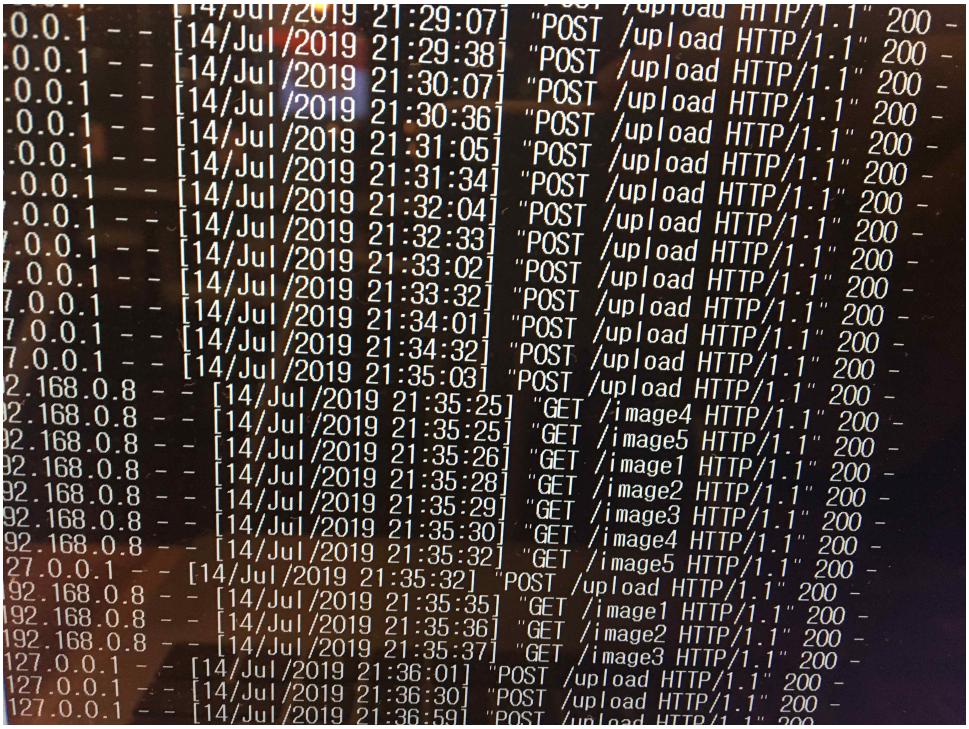
[그림19]와 같이 어플리케이션에 접속하여 '내 방 사진 확인하기' 버튼을 클릭하면 서버로부터 사진을 읽어온다.

(4) 서버 화면(30223 홍준화)

객체 인식 프로그램으로부터 받은 b1~b5가 서버 로컬에 저장된 것을 [그림20]을 통해 확인할 수 있다.

[그림20]서버 로컬 폴더

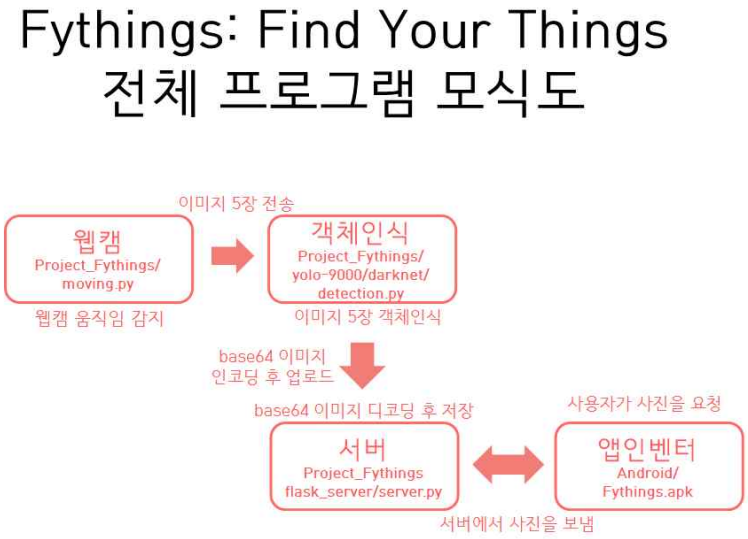
또한 [그림21]의 서버 로그를 통해 GET Request와 POST Request가 이루어진 것을 확인할 수 있다. GET Request는 사용자 어플리케이션으로 이미지를 송신한 기록, POST Request는 객체 인식 프로그램으로부터 이미지를 수신한 기록이다.



[그림21]서버 로그

2. 전체 프로그램 모식도(30223 홍준화)

Fythings(Find Your Things)의 전체 프로그램 모식도는 [그림]과 같다.



[그림22]전체 프로그램 모식도

Ⅴ. 결론 및 제언

1. 결론 및 기대 효과(30223 홍준화, 30122 현강우)

본 연구의 성취 결과는 다음과 같다.

가) 특정 공간에서의 움직임을 포착하고 객체 인식을 통해 사물을 검출하여 사용자 어플리케이션의 요청에 따라 검출된 사물의 위치를 보여주는 시스템을 설계/개발하였다.

나) 물체의 움직임을 OpenCV를 사용해 감지하였고, YOLO-9000을 활용해 객체 인식을 하였다. 이에 따라 물건의 개수에 따른 추가적인 비용 없이 효율적인 물건 탐색 시스템을 구축하였다.

다) MIT 앱 인벤터를 이용하여 사용자 어플리케이션을 구현하였고, 이를 통해 사용자가 간편하게 앱을 설치/실행하고 물체의 위치를 찾아낼 수 있도록 하였다.

따라서 본 연구의 성취 결과인 'Fythngs' 시스템을 상용화한다면 웹캠과 스마트폰을 가진 사람들에게 건망증으로 인한 시간 소모를 효과적으로 감축할 수 있는 서비스를 제공할 수 있을 것이다. 또한 현재 5회로 설정된 횟수를 사용자가 직접 설정할 수 있게 한다면 사용자가 비교적 과거에 잃어버린 물건 역시 찾을 수 있게 될 것이다.

2. 향후 연구 계획(30223 홍준화, 30122 현강우)

본 연구진은 고등학생의 신분으로 짧은 기간 내에 연구를 수행하면서 시간적 한계와 기술적 한계를 동시에 겪으며 크게 성장할 수 있었다. 본 연구진은 대학에 진학하였을 때 아래와 같이 본 연구에 대해 추가적인 연구를 수행하고 MicroSoft Store와 Google Play Store에 'Fythings'를 정식 출시할 계획이다.

1. 이미지의 5회 저장 횟수를 사용자가 선택할 수 있도록 N회로 변경
2. 더욱 뛰어난 정확도와 속도를 가진 객체 인식 알고리즘 개발 및 활용
3. 개인정보보호를 위한 물건 위치 영역 외 영역 블라인드
4. JAVA 또는 코틀린을 사용한 정식 어플리케이션 개발 및 플레이스토어 등록
5. 객체 인식 시스템 MS 스토어 등록
6. 추가적인 연산을 통한 움직임 탐지 기술 향상
7. 도메인 구매 및 AWS 서버 구매를 통한 정식 서버 구축

참고 문헌

REDMON, Joseph, et al. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 779-788.

LIN, Tsung-Yi, et al. Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision. 2017. p. 2980-2988.

미아 스타인. 2019. 파이썬 자료구조와 알고리즘.

이세우. 2019. 파이썬으로 만드는 OpenCV 프로젝트.

“딥러닝을 활용한 객체 탐지 알고리즘 이해하기”. accessed Dec.21.2018. https://blogs.sas.com/content/saskorea/2018/12/21/%EB%94%A5%EB%9F%AC%EB%8B%9D%EC%9D%84-%ED%99%9C%EC%9A%A9%ED%95%9C-%EA%B0%9D%EC%B2%B4-%ED%83%90%EC%A7%80-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0/

정종면, 김호영, 송시온. (2014). 차영상 기법을 이용한 이동 객체 탐지 및 계수 시스템. 한국컴퓨터정보학회 학술발표논문집 , 22(1), 251-252.

오명관, 한군희, 최동진, 전병민. (2004). 차영상을 이용한 이동 객체 추적. 한국콘텐츠학회 종합학술대회 논문집, 2(1), 396-400.

이아현, 박수민, 홍정수. (2018). YOLO를 이용한 야생동물 접근 및 피해 예방 시스템 개발. 한국정보과학회 학술발표논문집, (), 1897-1899.