

Yolo를 이용한

차량 번호판 인식 시스템 구현

2팀 (권보라, 박나은, 이나겸, 정성우)



CONTENTS

01 프로젝트 개요

02 프로젝트 과정

03 프로젝트 결과

04 프로젝트 소감

CONTENTS

01 프로젝트 개요

권보라



- 데이터 라벨링 작업
- yolov5, yolov7 학습
- Flask 웹

박나은



- 데이터 라벨링 작업
- yolov5, yolov7 학습
- Flask 웹

이나겸



- 데이터 라벨링 작업
- yolov5, yolov7 학습
- OCR

정성우



- 데이터 라벨링 작업
- yolov5, yolov7 학습
- OCR

01 프로젝트 개요

도난차량추적이나 아파트, 백화점 등 건물에 사용되는 차량 번호 인식 시스템을 구현하였다.
차량 번호판 인식에서 중요한 것은 번호판의 위치를 얼마나 **빠르게** 인식하는지,
차량 번호판의 글자를 얼마나 **정확하게** 인식하는 지가 중요하다.
실시간 영상에서 차량 번호 인식을 빠르고 정확하게 하기 위해서는
스틸 이미지에서 **차량 번호판 인식률을 높이는 게** 중요할 것 같아서 목표로 하고 프로젝트를 진행하였다.

프로젝트 주제 선정



- 국내 차량번호 인식 시스템은 인식률이 높은 편이지만 극한의 상황에서 제대로 인식되지 않는 문제로 계속해서 연구되고 있음
- Object Detection과 OCR 인식이 잘 되는 모델을 만드는 것을 목표로 웹을 활용한 고도화 진행

활용 장비, 개발 환경



- Microsoft Azure
- 데이터 : 보배드림 (www.bobaedream.co.kr/)
- VS Code
- YOLOv5, YOLOv7
- Easy OCR, Tesseract-OCR
- Azure Cognitive Services

기능 구현



- 차와 번호판의 위치를 인식할 수 있도록 라벨링 작업 후 yolo 학습
- 번호판의 위치를 감지하면, 번호판 이미지만 crop해서 OCR로 인식
- OCR로 추출된 글자로 웹에서 등록된 차주와 차의 번호판 출력

02 프로젝트 과정

[illegible]

02 프로젝트 과정

01

데이터 샘플링

.....

- 웹 크롤링을 통해 수집
- Azure 라벨링 작업
- coco 형식의 포맷을 yolo 형식으로 변경
- train, valid로 split

02

모델 학습

.....

- yolo5와 yolo7
- 모델과 하이퍼 파라미터 변경하며 학습진행
- 버전, 모델, 하이퍼 파라미터 값 결과 비교

03

OCR 인식

.....

- tesseract OCR, easy OCR, MCS OCR을 비교하여 좀 더 번호판 인식이 잘 되는 모델 선정

04

Flask 웹 구축

.....

- 번호판 이미지에서 OCR로 추출한 값을 DB에서 조회하여 Flask 웹에 출력 작업

02 프로젝트 과정

데이터 수집, 라벨링, 샘플링

웹 크롤링을 통해 차량 이미지 데이터 수집



사진

보바드림 촬영팀이 직접 촬영한 사진입니다.



```
for h in href_list:
    # 파일이름
    last_url = h.find('a')['href']
    driver.get(url_path+last_url)
    html = driver.page_source
    soup = bs(html, 'html.parser')

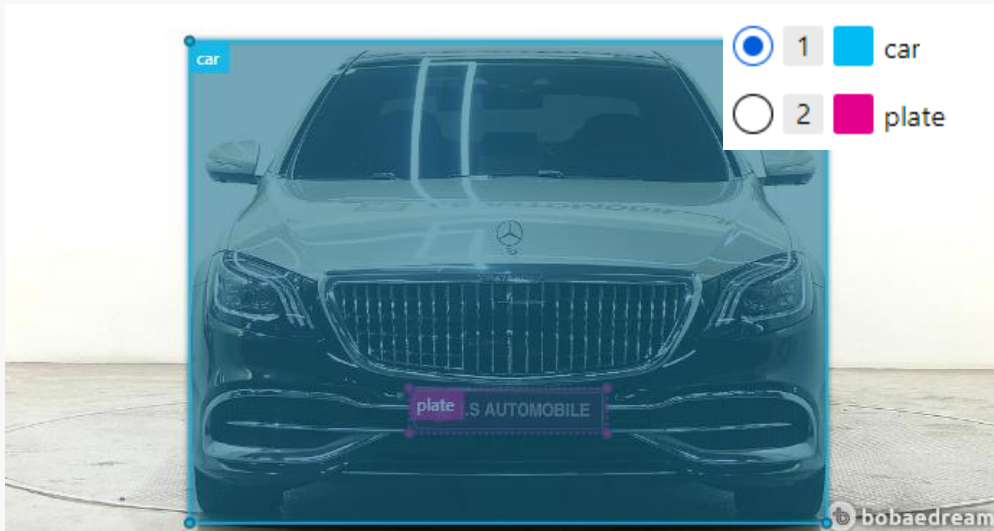
    image_list = soup.findAll('a', {"data-fancybox": "gallery"})
    for e, i in enumerate(image_list, start=1):
        # 세번째사진까지만
        if e == 4:
            break
        img_url = i['href']
        file_name = str(img_url).split('/')[-1]
        urllib.request.urlretrieve('https:'+img_url, f"./car_image/{file_n}")
```

- 게시판 페이지가 몇 페이지까지 있는 지 확인하고, 이미지를 저장하지 못 하는 상황을 염두하여 전체 페이지를 범위로 잡고 크롤링 진행
- 파일 이름은 a링크의 text를 가져와서 저장
- 게시물마다 3번째사진까지 크롤링으로 수집할 수 있게 함

02 프로젝트 과정

데이터 수집, 라벨링, 샘플링

Azure 라벨링 작업



1번 라벨 : car / 2번 라벨 : plate
▶ yolo 형식으로 바꿀 때,
0번, 1번 라벨로 변경

```
{  
  "id": 511,  
  "width": 1120.0,  
  "height": 840.0,  
  "file_name": "UI/07-15-2022_064051.UTC/L500/아우디_RS_Q8_4.0  
  "coco_url": "AmlDatastore://workspaceblobstore/UI/07-15-202  
  "absolute_url": "https://team01027405504241.blob.core.window  
  "date_captured": "2022-07-15T18:50:24.4069053Z"  
}
```

```
"annotations": [  
  {  
    "id": 1,  
    "category_id": 2,  
    "image_id": 1,  
    "area": 0.014,  
    "bbox": [  
      0.2707657442748092,  
      0.7154580152671756,  
      0.1900763358778626,  
      0.07480916030534357
```


02 프로젝트 과정

데이터 수집, 라벨링, 샘플링

coco 형식의 포맷을 yolo 형식으로 변경

```
def convert_bbox_coco2yolo(img_width, img_height, bbox):  
    x_tl, y_tl, w, h = bbox  
    dw = 1.0 / img_width  
    dh = 1.0 / img_height  
    x_center = x_tl + w / 2.0  
    y_center = y_tl + h / 2.0  
    x = x_center * dw  
    y = y_center * dh  
    w = w * dw  
    h = h * dh  
  
    return [x, y, w, h]
```

```
"bbox": [  
    0.41195913461538464,  
    0.6599080267558528,  
    0.19188963210702342,  
    0.053511705685618804
```



```
1 0.197615 0.469273 0.157967 0.073579  
0 0.506026 0.504390 0.977888 0.719064
```

- Azure에서 내보내기 한 coco 형식의 정규화 된 bbox 값이 있는 json 파일을 yolo 형식으로 변환하여 txt 파일로 저장

yolo 학습을 위한 train, valid로 파일 split

train

images

- img_1.png
- img_10.png
- img_100.png
- img_1000.png
- img_1001.png
- img_1002.png
- img_1004.png
- img_1005.png
- img_1006.png

valid

images

- img_1003.png
- img_101.png
- img_1025.png
- img_103.png
- img_1030.png
- img_1031.png
- img_1034.png
- img_1037.png
- img_1053.png



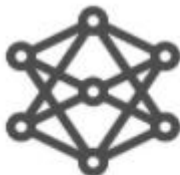
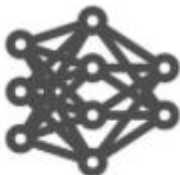
- cv2에서 한글로 된 파일 이름을 읽을 수 없기 때문에 이미지와 라벨 이름이 같은 경우 img_{num}으로 파일 저장
- sklearn의 train_test_split을 사용하여 image와 label 파일을 리스트로 불러와서 split

02 프로젝트 과정

yolov5, yolov7 detect 학습

yolov5

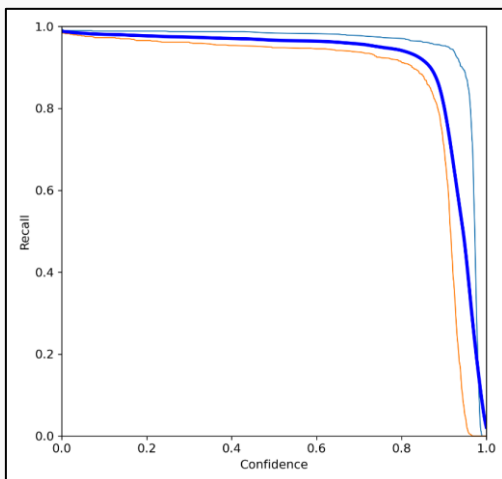
- YOLO는 단일 신경망 구조이기 때문에 구성이 단순하며 빠르다.
- SSD는 하나의 이미지를 여러 개의 이미지 데이터로 나눠서 분석하지만 YOLO는 주변 정보까지 학습하여 이미지 전체를 처리하기 때문에 background error가 적다.
- YOLO는 훈련 단계에서 보지 못한 새로운 이미지에 대해서도 검출 정확도가 높다.
- YOLO는 SOTA 객체 검출 모델에 비해 정확도(mAP)가 다소 떨어진다.
- yolo v5는 s, m, l, x의 4가지 버전이 있음 s 는 가장 가벼운 모델, x 는 가장 무거운 모델
- s 성능이 낮지만 FPS가 가장 높고, x가 성능이 제일 높지만 FPS는 가장 낮다.

			
Small YOLOv5s	Medium YOLOv5m	Large YOLOv5l	XLarge YOLOv5x
14 MB _{FP16} 2.2 ms _{V100} 36.8 mAP _{COCO}	41 MB _{FP16} 2.9 ms _{V100} 44.5 mAP _{COCO}	90 MB _{FP16} 3.8 ms _{V100} 48.1 mAP _{COCO}	168 MB _{FP16} 6.0 ms _{V100} 50.1 mAP _{COCO}

02 프로젝트 과정

yolov5, yolov7 detect 학습

yolov5s



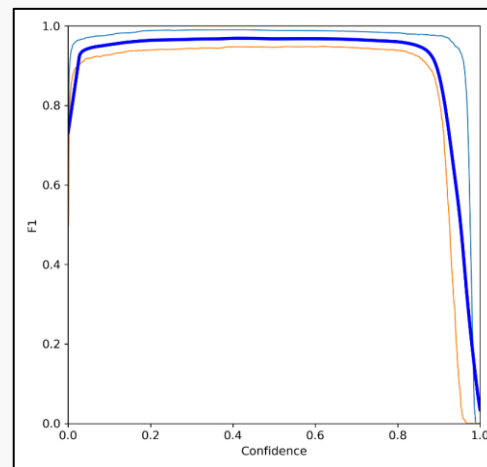
car
plate
all classes 0.99 at 0.000

car
plate
all classes 0.98 at 0.000

SGD

car
plate
all classes 0.98 at 0.000

adamW



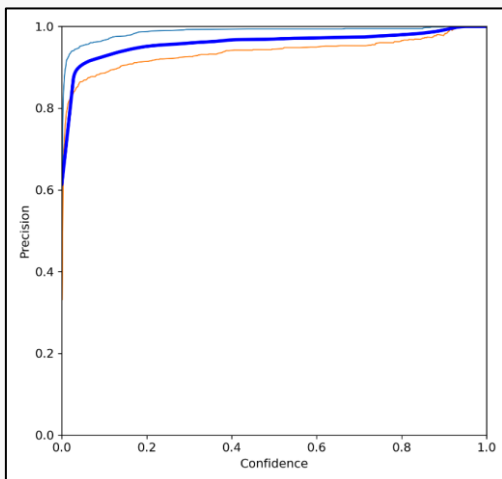
car
plate
all classes 0.97 at 0.411

car
plate
all classes 0.97 at 0.542

SGD

car
plate
all classes 0.96 at 0.527

adamW



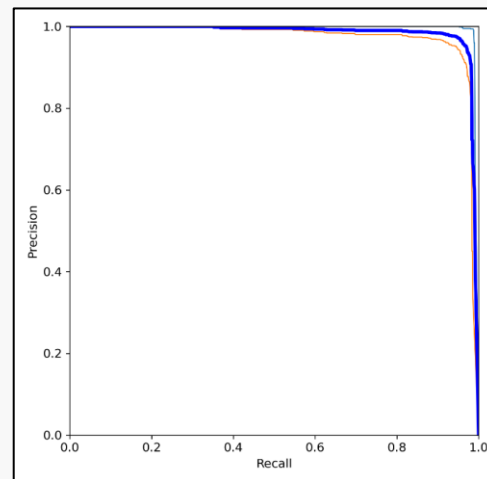
car
plate
all classes 1.00 at 0.950

car
plate
all classes 1.00 at 0.996

SGD

car
plate
all classes 1.00 at 0.981

adamW



car 0.994
plate 0.974
all classes 0.984 mAP@0.5

car 0.992
plate 0.956
all classes 0.974 mAP@0.5

SGD

car 0.992
plate 0.957
all classes 0.974 mAP@0.5

adamW

02 프로젝트 과정

yolov5, yolov7 detect 학습

yolov5s

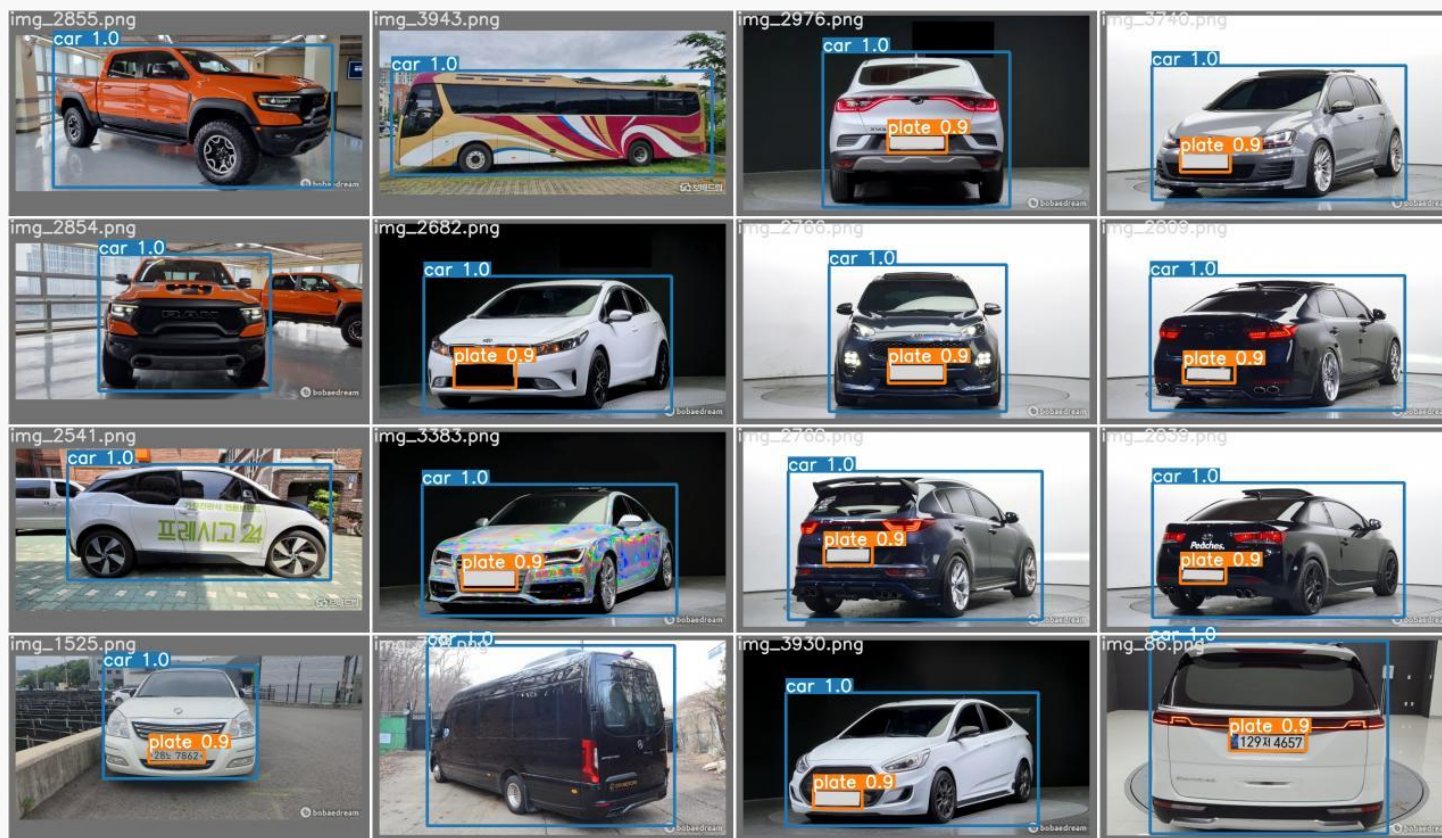


02 프로젝트 과정

yolov5, yolov7 detect 학습

yolov7

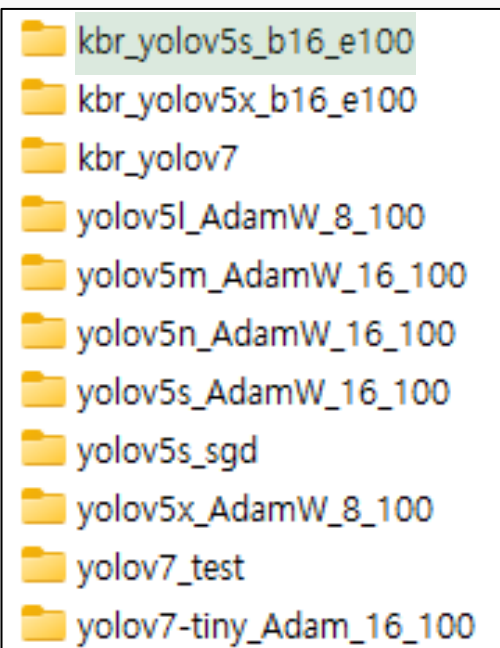
- test_batch_pred를 확인했을 때 차와 번호판 예측에 대해서 오차없이 정확하게 예측하는 것을 확인



02 프로젝트 과정

yolov5, yolov7 detect 학습

웹에 사용될 모델



- Model : yolov5s
- Epochs : 100
- Batch_size : 16
- Optimizer : SGD
- Label_smoothing : 0.0

02 프로젝트 과정

OCR 종류에 따른 인식 결과

OCR 인식을 위해 학습된 모델과 이미지 호출

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path='yolov5n_best.pt', force_reload=True)
model = torch.hub.load('../yolov5-master', 'custom', path='yolov5n_best.pt', source='local', force_reload=True)
```

- torch.hub.load로 미리 학습된 모델을 호출 (release 기준 torch 1.1.0버전부터 pytorch hub 사용가능)

```
img = Image.open('../dataset/test/img_753950_1.jpg') # PIL image
img = img.convert("L")
img = img.filter(ImageFilter.GaussianBlur(radius =1))

results = model(img, size=640)

df = results.pandas().xyxy[0]
...
```

		xmin	ymin	xmax	ymax	confidence	class	name
0	116.761169	208.202805	1044.926025	767.356384	0.976881	0	car	
1	359.405090	608.750916	566.712891	660.803223	0.876195	1	plate	
2	1051.030151	484.713440	1102.585327	516.971680	0.750280	1	plate	
...								

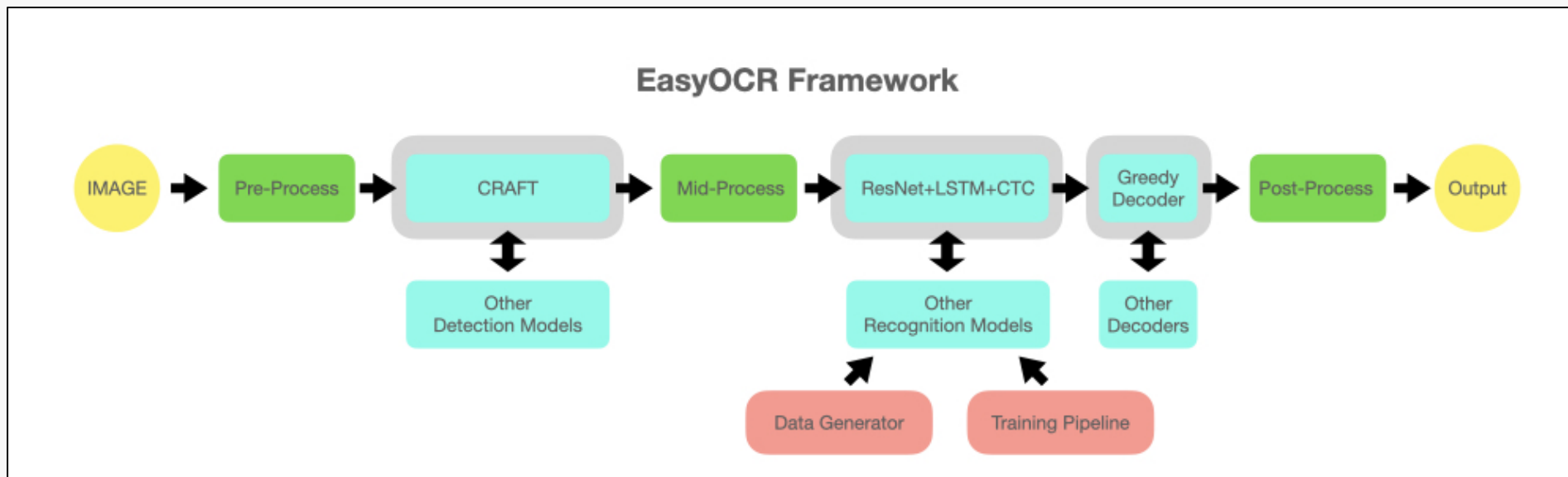
- Easy OCR을 사용할 때 cv2보다 PIL이 한글 인식이 더 잘된다는 것을 확인하여 OCR 프로젝트를 PIL로 진행
- Convert("L")을 사용해서 grayscale로 변경하고, GaussianBlur를 사용했을 때 기존 OCR 인식률보다 최소 3% 최대 6%까지 확률이 높아지는 것을 확인
- Model에 이미지를 넣어주고나서 result를 출력하면 좌표값, yolo 인식률, class, class name이 출력된다.

02 프로젝트 과정

OCR 종류에 따른 인식 결과

Easy OCR

- EasyOCR은 한국어, 일본어, 중국어 등을 포함한 80여개 이상의 언어를 사용해서 이미지 속 문자를 판독하여 텍스트로 출력
- 모든 딥러닝 부분은 PyTorch를 기반으로 만들어졌다.
- Detection 부분과 Recognition 부분으로 나누어져 있는데
Text Detection은 Clova AI의 CRAFT 알고리즘을 사용하고 Recognition 모델은 CRNN을 사용한다.
특징 추출을 위해 ResNet 모델을 사용, 시퀀스 레이블링을 위해 LSTM 모델을 사용하고 decode로 CTC 모델을 사용한다.



02 프로젝트 과정

OCR 종류에 따른 인식 결과

Easy OCR

```
def easy_ocr (path) :  
    reader = easyocr.Reader(['ko', 'en'], gpu=False)  
    result = reader.readtext(path)  
    read_result = result[0][1]  
    read_confid = int(round(result[0][2], 2) * 100)  
    bbox_value = result[0][0]
```

```
===== Crop Image OCR Read - Easy =====  
Easy OCR 결과      : 278노 2644  
Easy OCR 확률      : 85%  
bbox 좌표 확인     : [[20, 14], [160, 14], [160, 44], [20, 44]]  
=====
```

```
Using CPU. Note: This module is much faster with a GPU.  
[[[[20, 14], [160, 14], [160, 44], [20, 44]], '278노 2644', 0.8497372267446599]]
```



- Easyocr.Reader 한 이미지에서 여러 개의 문자 인식 가능
- 기본적으로 gpu를 사용하지만 gpu=False로 사용하지 않음
- Easy OCR은 OCR 인식 후 검출된 텍스트의 바운딩 박스 좌표, 검출된 텍스트 결과, 검출된 텍스트의 정확도를 확인할 수 있다.

02 프로젝트 과정

OCR 종류에 따른 인식 결과

Tesseract OCR

```
def tess_ocr (path) :  
    image = Image.open(path)  
    string_text = pytesseract.image_to_string(image, lang='kor',  
                                              config='--psm 4 --oem 3')  
    data_text = pytesseract.image_to_data(image, lang='kor', \  
                                          config='--psm 4 --oem 3')  
    boxes_text = pytesseract.image_to_boxes(image, lang='kor', \  
                                             config='--psm 4 --oem 3')
```

- `tesseract.image_to_string`
: OCR 처리된 수정되지 않은 출력을 문자열로 반환
- `Tesseract.image_to_data`
: 상자 경계, 신뢰도 및 기타 정보가 포함된 결과 반환
- `tesseract.image_to_boxes`
: 인식된 문자와 해당 상자 경계를 포함하는 결과 반환

OCR Engine modes(-oem):

- 0 - Legacy engine only.
- 1 - Neural nets LSTM engine only.
- 2 - Legacy + LSTM engines.
- 3 - Default, based on what is available.

Page segmentation modes(-psm):

- 0 - Orientation and script detection (OSD) only.
- 1 - Automatic page segmentation with OSD.
- 2 - Automatic page segmentation, but no OSD, or OCR.
- 3 - Fully automatic page segmentation, but no OSD. (Default)
- 4 - Assume a single column of text of variable sizes. **가변적인 크기의 1라인 텍스트**
- 5 - Assume a single uniform block of vertically aligned text.
- 6 - Assume a single uniform block of text. **텍스트의 균일한 단일 블록**
- 7 - Treat the image as a single text line.
- 8 - Treat the image as a single word.
- 9 - Treat the image as a single word in a circle.
- 10 - Treat the image as a single character. **특정 순서없이 가능한 많은 텍스트 찾기**
- 11 - Sparse text. Find as much text as possible in no particular order.
- 12 - Sparse text with OSD.
- 13 - Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific.

02 프로젝트 과정

OCR 종류에 따른 인식 결과

Tesseract OCR

Image_to_string, image_to_data 결과



```
=== Crop Image OCR Read - Tesseract ===  
Tess OCR 결과      :
```

level	page_num	block_num	par_num	line_num	word_num
1	1	0	0	0	0
2	1	1	0	0	55
3	1	1	1	0	55
4	1	1	1	1	55
5	1	1	1	1	55



```
=== Crop Image OCR Read - Tesseract ===  
Tess OCR 결과      : [278노2644]
```

2	1	2	0	0	0	16	0	150	37	-1
3	1	2	1	0	0	16	0	150	37	-1
4	1	2	1	1	0	16	0	150	37	-1
5	1	2	1	1	1	16	0	150	37	90.218643
5	1	2	1	1	2	80	0	19	42	0.000000
5	1	2	1	1	3	98	0	68	37	72.908066
2	1	3	0	0	0	15	4	1	35	-1

[278
노
2644]

Image_to_boxes 결과

- grayscale + gaussianblur
azure와 easy ocr과 달리 인식 못함

```
Tess OCR boxes 결과  
~ 55 54 103 55 0
```

- grayscale + edge
azure와 easy ocr에서는 인식률이
좋지 않지만 tesseract에서의
인식률은 좋은 편

```
Tess OCR boxes 결과  
~ 167 41 177 42 0  
~ 167 3 169 38 0  
~ 170 32 177 34 0  
[ 16 6 18 34 0  
2 25 8 40 42 0  
7 42 9 56 31 0  
8 54 5 166 41 0  
노 80 0 99 42 0  
2 98 5 115 42 0  
6 114 5 128 42 0  
4 127 5 141 42 0  
4 140 5 154 42 0  
] 153 5 166 42 0  
~ 15 3 16 38 0
```

02 프로젝트 과정

OCR 종류에 따른 인식 결과

Microsoft Cognitive Service Computer Vision 사용 (OCR)

- subscription_key와 endpoint에 값을 넣어주면 MCS의 ComputerVisionClient에 접속된다.
- 이미지는 이미지 URL을 넣어주거나 local 이미지를 stream image로 변경해서 넣어줘야 한다.
- 결과값으로 인식된 text와 bounding_box가 출력되는데, 필요한 text만 반환받는다.

```
def azure_ocr(path):
    subscription_key = "8d06cd8651fd43f2bf392c71478a6d67"
    endpoint = "https://platestr.cognitiveservices.azure.com/"
    computervision_client = ComputerVisionClient(endpoint, CognitiveServicesCredentials(subscription_key))

    img = open(path, "rb")
    read_response = computervision_client.read_in_stream(img, language="ko", raw=True)
    read_operation_location = read_response.headers["Operation-Location"]
    operation_id = read_operation_location.split("/")[-1]

    while True:
        read_result = computervision_client.get_read_result(operation_id)
        if read_result.status not in ['notStarted', 'running']:
            break
        time.sleep(1)

    if read_result.status == OperationStatusCodes.succeeded:
        for text_result in read_result.analyze_result.read_results:
            for line in text_result.lines:
                result = line.text
                # print(line.bounding_box)
    return result
```

02 프로젝트 과정

OCR 종류에 따른 인식 결과

OCR 인식 비교

crop된 일반이미지



grayscale + edge



grayscale + blur



Azure

```
==== Crop Image OCR Read - Azure ====
Azure OCR 결과      : 278노 2644
=====
```

- Azure에서는 Bad Request로 OCR 인식을 할 수 없음

```
==== Crop Image OCR Read - Azure ====
Azure OCR 결과      : 278노 2644
=====
```

Easy
OCR

```
==== Crop Image OCR Read - Easy ====
Easy OCR 결과       : 278노 2644
Easy OCR 확률       : 71%
=====
```

```
==== Crop Image OCR Read - Easy ====
Easy OCR 결과       : 고g노 2@예4
Easy OCR 확률       : 1%
=====
```

```
==== Crop Image OCR Read - Easy ====
Easy OCR 결과       : 278노 2644
Easy OCR 확률       : 85%
=====
```

Tesseract
OCR

```
=== Crop Image OCR Read - Tesseract ===
Tess OCR 결과       :
=====
```

```
=== Crop Image OCR Read - Tesseract ===
Tess OCR 결과       : [278노2644]
=====
```

```
=== Crop Image OCR Read - Tesseract ===
Tess OCR 결과       :
=====
```

02 프로젝트 과정


Flask 웹 + Database 구축

웹 화면 - Detection

Detection 화면에서는 이미지 파일을 업로드하고 car, plate를 조회한 후 차량번호를 추출해서 해당 차주가 있다면 이름을 출력하고, 없다면 '없음' 혹은 차량번호 인식 결과가 이상할 경우에는 '인식 오류'를 출력하도록 하였다.

파일 선택 선택된 파일 없음

업로드



이름	차량번호
인식 오류	GGHY1091

02 프로젝트 과정

Flask 웹 + Database 구축

웹 화면 – 사용자 관리

사용자 관리 화면에서는 현재 등록된 사용자 목록을 보여주고 각각 삭제할 수 있으며, 신규 사용자 등록 버튼으로 이름과 차량번호를 입력하면 등록이 가능하도록 만들었다.

이름	차량번호	삭제
권보라	23주1353	삭제
이나겸	08두4400	삭제
박나은	15저0101	삭제
정성우	175너2094	삭제
강호용	23소8305	삭제
김보라	16도4829	삭제
이지은	44호5454	삭제

신규 사용자 등록

이름

차량번호

등록

02 프로젝트 과정

Flask 웹 + Database 구축

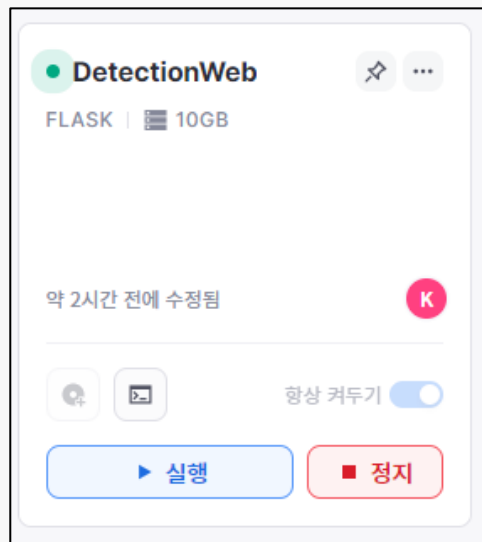
웹 구축 플랫폼 – goorm IDE

goorm IDE

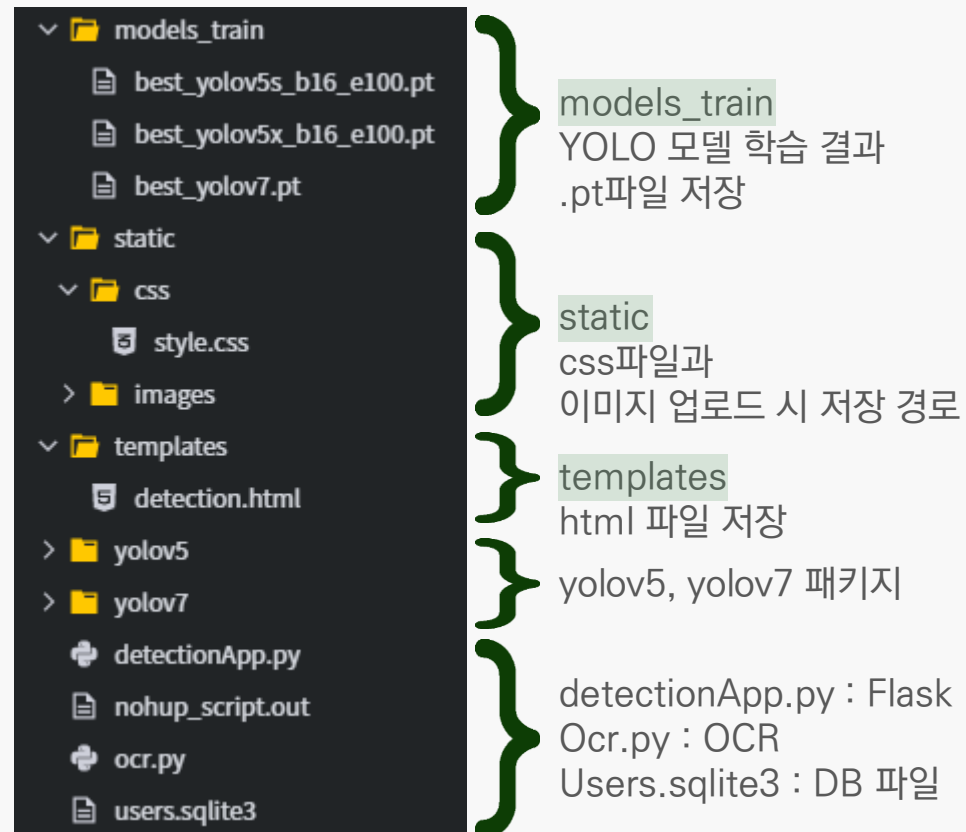
- 무료로 컨테이너를 사용해 웹을 올릴 수 있는 개발자 플랫폼

goormide

- FLASK 컨테이너를 띄운 모습



작업 디렉토리 구조



02 프로젝트 과정

Flask 웹 + Database 구축

Database 연동 – SQLite3

SQLite3 데이터베이스 사용

```
root@goorm:/workspace/DetectionWeb# sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open users_.sqlite3
sqlite> .exit
root@goorm:/workspace/DetectionWeb# sqlite3 users_.sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> CREATE TABLE users (
...> id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
...> name TEXT NOT NULL,
...> carNumber TEXT NOT NULL
...> );
sqlite> .tables
users
sqlite> █
```

- sqlite3 명령어로 sqlite CLI 진입하여 .open 파일명 으로 .sqlite3 파일을 생성해준 후 CLI를 빠져나왔다.
- 다시 sqlite3 명령어에 파일명을 주고 CLI로 진입하면 해당 DB를 사용할 수 있게 된다.
- CREATE TABLE로 id, name, carNumber를 컬럼으로 가지는 테이블을 생성해주었다.

detectionApp.py에서 연결

```
@app.route("/userList/")
def userList(innerContent=""):
    conn = sqlite3.connect('users.sqlite3')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM users')
    users = cursor.fetchall()
    conn.close()
```

- sqlite3.connect() 함수로 DB 파일을 열어준 다음 cursor를 설정해주고 execute() 함수로 쿼리 실행해주었다.
- 결과가 하나일 때는 fetchone(), 여러 개일 때는 fetchall()을 사용하였고 위 예시의 경우 users에 리스트 형태로 담겨서 HTML로 만들어주었다.

02 프로젝트 과정

Flask 웹 + Database 구축

Flask 웹 구축

detectionApp.py

```
# best.pt 파일 PATH 설정
pt_file_path = "./models_train/best_yolov5x_b16_e100.pt"
# pt_file_path = "./models_train/best_yolov7.pt"
model = torch.hub.load('./yolov5', 'custom', path=pt_file_path, source='local')

app.run(host='0.0.0.0', port=80, debug=True)
```

- torch.hub.load()에서 yolov5를 불러오고 .pt file 경로를 지정해주면 모델이 로드된다.

```
# Run model
results = model(img, size=640)

# Updates results.imgs with boxes and labels
results.render()

for img in results.imgs:
    img_base64 = Image.fromarray(img)
    img_base64.save(img_src)
```



- model(img, size=640) 로 업로드 된 이미지를 model에 넣어주고 결과를 render() 해주면 Yolo가 실행된 결과 box들이 이미지에 적용된다.
- box가 적용된 이미지를 저장하여 웹에서 볼 수 있도록 한다.

02 프로젝트 과정

Flask 웹 + Database 구축

Flask 웹 구축

detectionApp.py

```
# Crop image by boxes
crops = results.crop(save=False)
# conf = (crop[0]['conf'].item() * 100)
# to save car numbers and names
car_dict = {}
```

```
# loop every cropped image
for num, crop in enumerate(crops) :
    # if it's a plate over 50% probability
    if 'plate' in crop['label'] and crop['conf'].item() * 100 > 50 :
        image = crop['im']
        # Set cropped image and apply augmentation - grayscale, gaussianblur
        im = Image.fromarray(image)
        im = im.convert("L")
        im = im.filter(ImageFilter.GaussianBlur(radius =1))

        # Set plate image save path to FILENAME_pl2.png
        plate_src = os.path.join(app.config['UPLOAD_FOLDER'], file.filename.split('.')[0] +
                                im.save(plate_src, 'png'))

        # call solution() for ocr, and save result into car_dict
        name, carNumber = solution(plate_src)
        car_dict[carNumber] = name

# call detection.html with car_dict
return template(render_template('detection.html', filename=img_src, results=car_dict))
```

- results.crop(save=False) 이미지를 box별로 잘라낸다. save True를 주면 자동으로 폴더가 생성되고 저장되지만 plate만 따로 지정된 폴더에 저장하기위해 save 안한다.
- Car_dict 에는 인식한 차 번호와 차주 이름을 담아준다.
- Crop 결과 'label' 값에 'plate'가 있고, plate를 인식한 정확도가 50을 넘는 경우에만 OCR을 인식을 진행합니다.
- Crop된 이미지에 Grayscale, GaussianBlur를 적용했을 때 OCR 인식률이 개선되는 것을 확인하여 적용했습니다.
- plate는 각각 '파일명_pl숫자.png'로 저장해준다.
- solution() 함수에 plate 이미지를 넘겨 OCR을 돌리고 얻은 결과를 car_dict에 저장해준 후 detection.html에서 보여줄 수 있도록 넘겨주었다.

02 프로젝트 과정

Flask 웹 + Database 구축

Flask 웹 구축

detectionApp.py

```
# OCR
def solution(filename):
    ocr_result = re.sub(r'^\w\s', '', ocr.azure_ocr(filename))
    ocr_result = ocr_result.replace(' ', '')

    if re.match("(\\d{2,3}\\D\\d{4})", ocr_result) == None:
        name = '인식 오류'
    else:
        conn = sqlite3.connect('users.sqlite3')
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM users WHERE carNumber=?', (ocr_result, ))
        user = cursor.fetchone()
        conn.close()
        if user != None:
            name = user[1]
        else:
            name = "없음"

    return name, ocr_result
```

- solution()에서는 ocr.py의 azure_ocr을 호출하여 ocr 결과를 돌려준다.
- 번호판의 모서리나 나사 부분 등을 특수문자로 인식하는 경우가 있어 문자와 숫자 빼고는 삭제하도록 처리하고 공백도 삭제해주었다.
- 정규식을 사용해 결과가 차량번호 형식에 맞지 않을 경우 DB를 조회하지 않고 차주 이름 대신 '인식 오류'를 출력하도록 했다.
- 결과가 차량번호 형식에 맞을 때만 users.sqlite3에 연결하여 SQL Query로 차 번호를 조회, 결과가 나올 경우 차주 이름을, 결과가 없을 경우 '없음'을 반환하였다.

02 프로젝트 과정

Flask 웹 + Database 구축

Flask 웹 구축

주차 요금 계산 기능 간단 구현

```
#이미지를 upload 할 때 마다 실행
price=0 #주차요금은 기본적으로 0원
name, carNumber = solution(plate_src) #ocr로 차번호를 가져와서 db에 있는 이름을 가져온다
car_dict[carNumber] = (name,price) #dictionary 형태로 차번호:{이름,주차요금}
if name!="미등록" and name!="인식오류":# 등록된 차량에 한해서 주차요금 계산
    conn = sqlite3.connect('users.sqlite3') #db에 접근
    cursor = conn.cursor()
    cursor.execute('UPDATE users SET timeout =? WHERE name =? and timein!=?', (time(),name,NULL) ) #입차시간이 있으면 출차시간에 등록
    cursor.execute('UPDATE users SET timein =? WHERE name =? and timein!=?', (time(),name,NULL) ) # 입차시간이 없으면 입차시간에 등록

    cursor.execute('SELECT * FROM users WHERE name=?', (name, ))# db에 등록된 정보 가져옴
    results = cursor.fetchall()

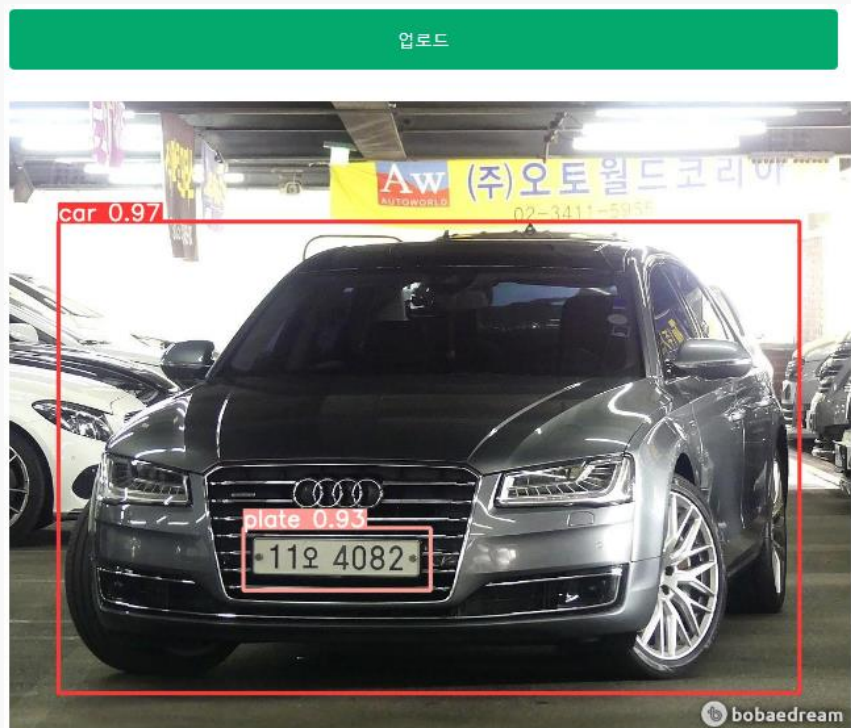
    cursor.execute('UPDATE users SET (cal,timein,timeout) =(?,?,?) WHERE name =? and timein!=? and timeout!=?',
        (results[0][4]-results[0][3],0,0,name,0,0)) #입차,출차시간이 있으면 출차시간-입차시간 계산해 주차시간에 입력하고 입차,출차시간 초기화
    cursor.execute('SELECT * FROM users WHERE name=?', (name, )) #db에 등록된 정보 가져옴
    fresults = cursor.fetchall()
    if fresults[0][5]!=0: #주차시간이 0이 아니면 초당100원 반올림
        price=round(100*fresults[0][5])
        car_dict[carNumber] = (name,price) #dictionary 형태로 주차요금 제공
        cursor.execute('UPDATE users SET cal =? WHERE name =? ', (0,name)) #주차시간 초기화
    conn.commit() #db 수정 종료
    conn.close()
```


02 프로젝트 과정

Flask 웹 + Database 구축

Flask 웹 구축

detectionApp.py



입차

이름	차량번호	주차요금
정성우	11오4082	0

출차

이름	차량번호	주차요금
정성우	11오4082	8021

```
[(4, '정성우', '11오4082', 1659006431.5850098, 0, 0)] #입차시간 저장
[(4, '정성우', '11오4082', 1659006431.5850098, 1659006472.6714787, 0)] #출차시간 저장
[(4, '정성우', '11오4082', 0, 0, 41.08646893501282)] #출차시간-입차시간 계산후 초기화
[(4, '정성우', '11오4082', 0, 0, 0)] #주차시간 초기화
```

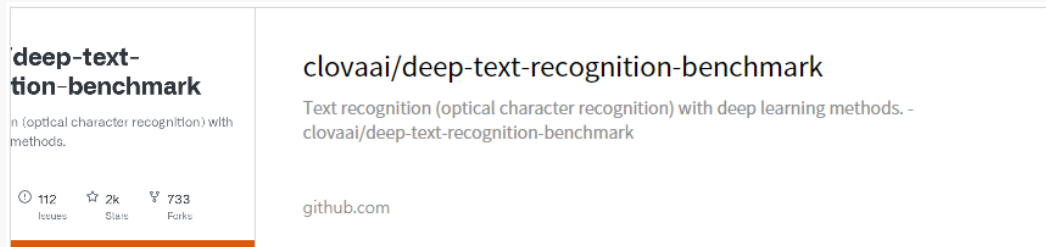
03 프로젝트 결과

goorm IDE에서 결과를 보여드리겠습니다 😊

03 프로젝트 결과

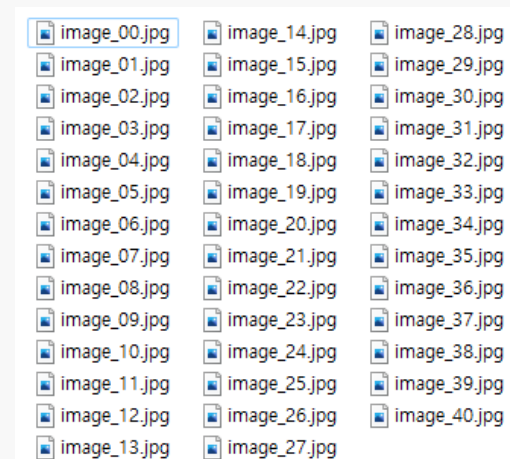
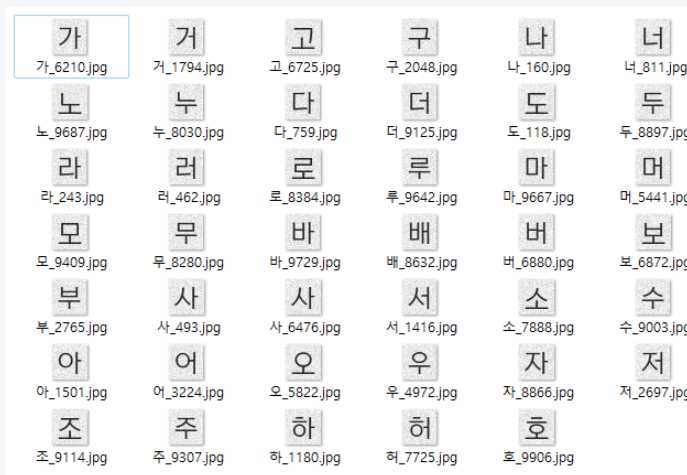
보완 및 아쉬운 점

- Easy OCR에서 제공하는 API를 통해 OCR 기능을 이용할 때 사용되는 기본 신경망 모델이 아니라 사용자가 직접 학습시키고자 하는 데이터를 준비해 학습하고, 원하는 성능의 모델을 만들어서 사용할 수 있다.
- 학습 데이터를 .lmdb 파일로 변환하는 과정에서 오류가 나서 더 이상 진행할 수 없었으나 이미 학습을 진행해 본 사용자들의 후기로는 학습 결과가 기존 모델보다 월등히 좋아지거나 하진 않는다고 한다.

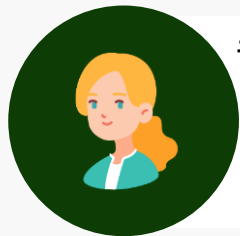


자동차 번호판의 한글	
차량 용도	한글
비사업용	가·나·다·라·마 거·너·더·러·머·버·서·어·저
	고·노·도·로·모·보·소·오·조 구·누·두·루·무·부·수·우·주 (총 32가지)
사업용	택시·버스 바·사·아·자
	택배 차량 배
	대여 차량 (렌터카) 하·허·호

출처 : [국토교통부](#)

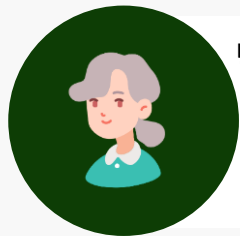


04 프로젝트 소감



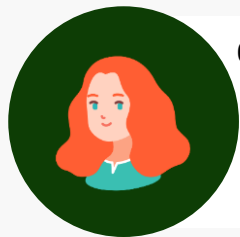
권보라

프로젝트 전에는 YOLO를 한 번도 돌려본 적이 없어서 결과 보는 방법이나 설정 방법도 잘 몰랐는데 내 손으로 직접 돌려보고 결과를 받아보니 재미있었고 다른 곳에도 적용해보고 싶다는 생각이 들었다. 이번에도 웹 구축을 맡게 되었는데 Flask에 익숙해진 것 같아 뿌듯하기도 했다.



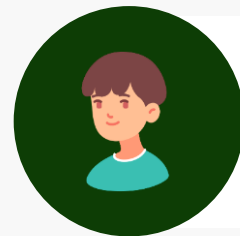
박나은

Object Detection 내용이 어려워서 프로젝트와 공부를 같이 병행하면서 진행했었다. 이론학습으로 배우는 것보다 직접 해보면서 다양한 에러가 발생했을 때에 어떻게 하면 해결할 수 있는지도 배우게 되어 여러모로 좋은 경험이었다. 남들이 한 것만 보다가 팀원들과 직접 만들어서 결과를 보게 되니 재밌었다.



이나겸

Easy OCR과 Tesseract OCR 인식률을 높이려고 해봤지만 Microsoft Cognitive Service의 OCR 보다 인식률이 더 좋지 못했다. 이걸 해결할 수 있는 방법으로는 Easy OCR이나 Tesseract OCR을 학습시키거나 CNN을 사용하는 방법으로 인식률을 더 높여보고 싶다,



정성우

Obejct detection으로 차량과 번호판을 인식하고 인식한 번호판을 ocr을 이용해 읽고 db를 만들어서 주차요금까지 계산해서 웹으로 제공하는 다양한 과정을 거치면서 많은것을 배웠고 재미있었습니다.

THANK YOU

경청해주셔서 감사합니다.