2023

자동차 번호판 인식

강형근, 울지바야르

프로젝트 개요 |
Summary of Project

제목	자동차 번호판 인식 모델		
목표	중고차 사이트에서 자동차 이미지를 크롤링하여 해당 이미지의 번호판을 인식하기		
기록자	울지바야르	프로젝트 책임자	박규호
기록일자	2023.10.26	프로젝트 책임자 승인일	

실행환경:

Python	3.11.4
OpenCV	4.8.1
pytesseract	0.3.10
OS	windows 11
IDE	vscode(1.83.1)

학습데이터: 다나와 중고차 사이트 1~10 페이지 이미지 50 개

Base_model: tesseract

1-1. 크롤링 단계

다나와 중고차 사이트 1~10 페이지까지의 이미지를 크롤링을 하였다. 저장은 img 폴더내의 파일명은 페이지번호+이미지 순서로 저장을 하는 코드를 작성을 하였다. 그러나 이미지를 저장하는 과정에서 다음 페이지로 넘어가도 파일명이 1 페이지로 저장되는 문제가 있었는데 그 원인은 코드에서 page += 1 이라는 변수를 선언을 안했기 때문이었고 해당 변수를 코드에 추가한 결과 제대로 저장이 되었다. 또한, 해당 사이트는 크롤링을 막기 위해 4 번 이미지의 속성값을 공백으로 두어서 속성값을 순서대로 클릭하는 for 문을 돌렸을 때 오류가 발생하였다. 따라서 이를 막기 위해 공백일 경우 넘어가고 다음 순서의 속성값을 받아오는 코드를 새로 작성하여 해결을 했다.

1-2. 모델 선택

번호판 인식 모델은 가장 대중적인 모델인 tessearct 를 사용했고 사용법은 https://github.com/kairess/license_plate_recognition 해당 github 을 참고했다. 크롤링한 모든 이미지 데이터를 모델로 인식하기 전에 먼저 이미지 한 장씩 넣으며 결과를 확인해봤다.

1-3. 모델 실행 결과

먼저 인식이 잘되는지 확인하기 위해 무작위 이미지 한개를 진행했다. 그러나 번호판 인식이 제대로 안되는 문제가 발생하였다. 원인은 크롤링한 이미지의 화질이 안 좋고 자동차의 구도와 각도가 제각각이기 때문이다. 따라서 크롤링한 자동차 이미지의 번호판 모두를 인식하는 것은 무리라고 생각하여 목표를 수정하였다. 구글에서 번호판이 잘 나오고 화질이 좋은 이미지 15 장을 선별하여 이들의 번호판을 인식하는 것으로 바꾸었다. 다음은 tesseract 를 사용하여 크롤링한 이미지 중 한 개를 인식해본 과정이다.

1-4-1. 원본 이미지 예시 샘플

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pytesseract
plt.style.use('dark_background')

img_ori = cv2.imread('11-1.jpg')

# 이미지 크기를 확대할 배율을 설정합니다.
scale_factor = 2  #2배 확대합니다.

# 이미지 크기를 조절합니다.
img_resized = cv2.resize(img_ori, None, fx=scale_factor, fy=scale_factor, interpolation=cv2.INTER_LINEAR)

height, width, channel = img_resized.shape


gray = cv2.cvtColor(img_resized, cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(12, 10))
plt.imshow(gray, cmap='gray')
```

✓ 0.2s Pyth

<그림 1-4-1 이미지 데이터를 BGR 채널에서 GRAY 채널로 변경>

<그림 1-4-2> 크롤링한 이미지 50 장 중 테스트용 원본 이미지>

위 이미지는 다나와 중고차 판매 사이트에서 크롤링한 사진 50 장 중 하나이다. 전체적으로 이미지가

작게 저장이 되어 이미지를 불러올 때 2 배 확대를 하였다.

1-4-3. 이미지를 0 과 255 로 이진화 처리

```
structuringElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))

imgTopHat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT, structuringElement)
imgBlackHat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, structuringElement)

imgGrayscalePlusTopHat = cv2.add(gray, imgTopHat)
gray = cv2.subtract(imgGrayscalePlusTopHat, imgBlackHat)
```

0.0s

```
#GaussianBlur: 노이즈를 줄이기 위해서
#adaptiveThreshold: 이미지 구별 쉽게 0 or 255 로 (검정 or 흰색으로)

img_blurred = cv2.GaussianBlur(gray, ksize=(5, 5), sigmaX=0)

img_thresh = cv2.adaptiveThreshold(
    img_blurred,
    maxValue=255.0,
    adaptiveMethod=cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    thresholdType=cv2.THRESH_BINARY_INV,
    blockSize=19,
    C=9
)

plt.figure(figsize=(12, 10))
plt.imshow(img_thresh, cmap='gray')
```

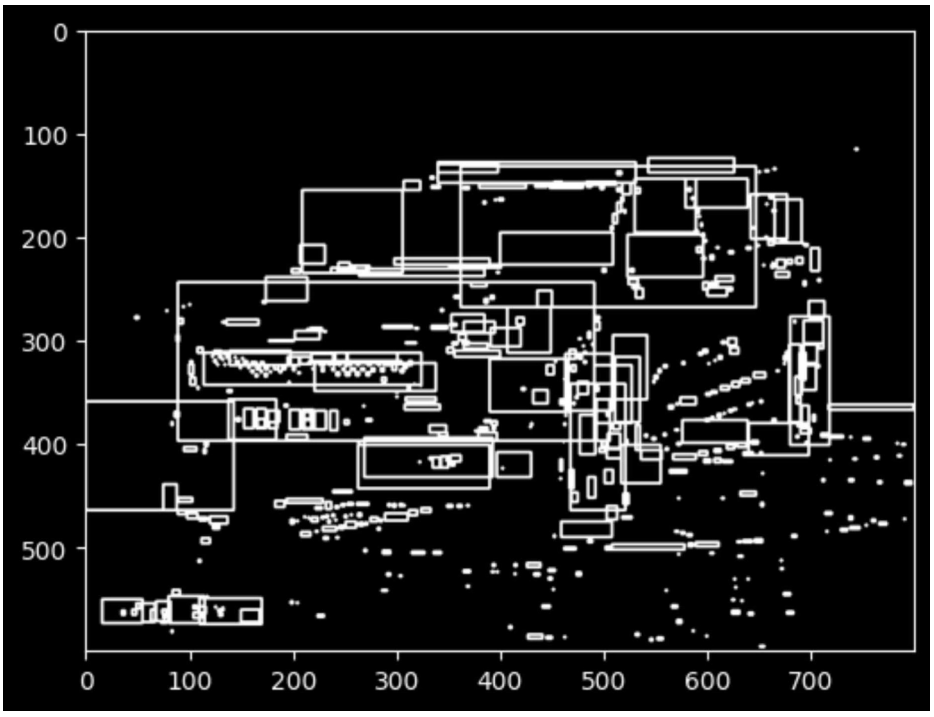
0.1s

<그림 1-4-3. 이진화된 자동차 이미지>

1-4-4 이미지 데이터의 객체 윤곽선을 찾고 표시한다.

```
contours, _ = cv2.findContours(
    img_thresh,
    mode=cv2.RETR_LIST,
    method=cv2.CHAIN_APPROX_SIMPLE
)

temp_result = np.zeros((height, width, channel), dtype=np.uint8)

cv2.drawContours(temp_result, contours=contours, contourIdx=-1, color=(255, 255, 255))

temp_result = np.zeros((height, width, channel), dtype=np.uint8)

contours_dict = []

for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    cv2.rectangle(temp_result, pt1=(x, y), pt2=(x+w, y+h), color=(255, 255, 255), thickness=2)

    # insert to dict
    contours_dict.append({
        'contour': contour,
        'x': x,
        'y': y,
        'w': w,
        'h': h,
        'cx': x + (w / 2),
        'cy': y + (h / 2)
    })
plt.figure(figsize=(6,5))
plt.imshow(temp_result, cmap='gray')
```

✓ 0.1s Pytho



<그림 1-4-4. 이미지의 각 요소들의 윤곽선>

1-4-5. 윤곽선 중에서 번호판이 될 수 있는 후보 윤곽선 찾기

```
MIN_AREA = 40   # 번호판 윤곽선 최소 범위 지정
MIN_WIDTH, MIN_HEIGHT = 1, 8         # 최소 너비 높이 범위 지정
MIN_RATIO, MAX_RATIO = 0.25, 1.0     # 최소 비율 범위 지정

possible_contours = []  # possible_contours에 저장

cnt = 0
for d in contours_dict:
    area = d['w'] * d['h']
    ratio = d['w'] / d['h']


  # 위에 설정한 범위의 조건을 비교, 맞추면서 다시한번 possible_contours에 저장해준다.
  # 각 윤곽선의 idx값을 매겨놓고, 나중에 조건에 맞는 윤곽선들의 idx만 따로 빼낼 것이다.  d['idx'] = cnt
    if area > MIN_AREA \
    and d['w'] > MIN_WIDTH and d['h'] > MIN_HEIGHT \
    and MIN_RATIO < ratio < MAX_RATIO:
        d['idx'] = cnt
        cnt += 1
        possible_contours.append(d)

# visualize possible contours
# possible contours의 정렬방식을 보고 번호판 후보들을 추려낸다. 번호판은 어느정도 규칙적으로 일렬로 나타난다. 손차적,각도,배열모양
temp_result = np.zeros((height, width, channel), dtype=np.uint8)

for d in possible_contours:
#     cv2.drawContours(temp_result, d['contour'], -1, (255, 255, 255))
    cv2.rectangle(temp_result, pt1=(d['x'], d['y']), pt2=(d['x']+d['w'], d['y']+d['h']), color=(255, 255, 255), thickness
plt.figure(figsize=(6,5))
plt.imshow(temp_result, cmap='gray')
```

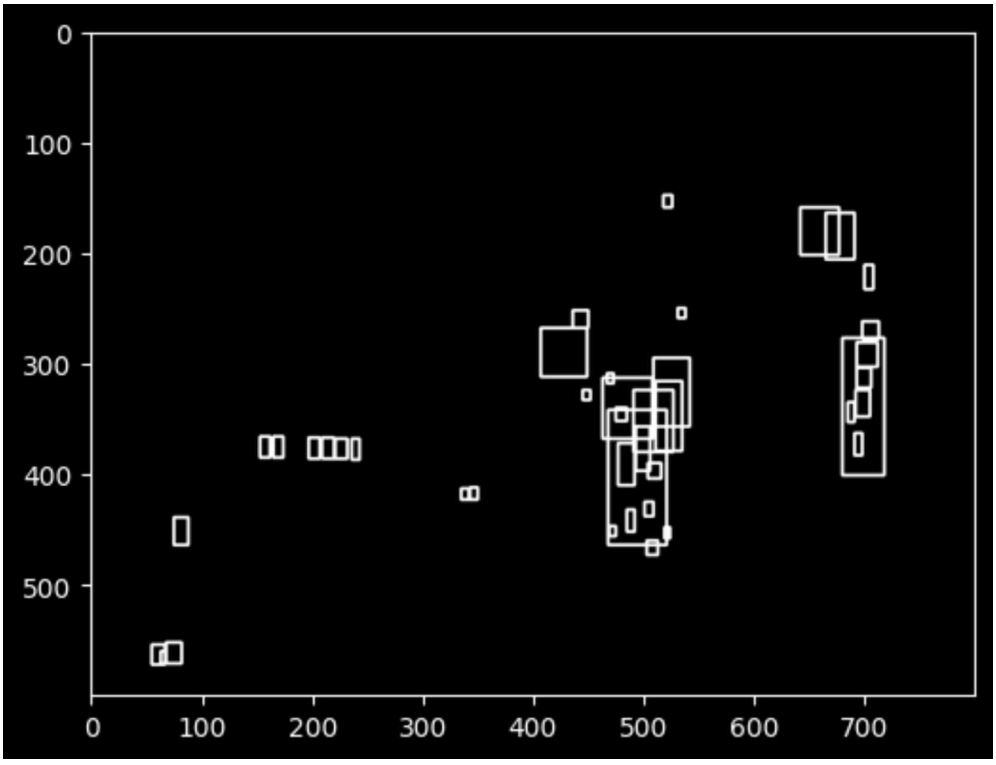
0.1s Python

MIN_AREA= 40 : 번호판 윤곽선 최소 범위 지정

MIN_WIDTH, MIN_HEIGHT= 1, 8 : 최소 너비 높이 범위 지정

MIN_RATIO, MAX_RATIO= 0.25, 1.0 : 최소 비율 범위 지정

번호판 인식이 안될 경우 위 매개변수들을 조정하면 된다. 그러나 모든 이미지를 수동으로 조정할 수 없기 때문에 다른 모델을 사용하거나 데이터를 수정해야한다고 느꼈다. 그 과정은 이 다음 단계인 2 번 항목에서 다루겠다.



<그림 1-4-5 윤곽선들 중에서 번호판 후보 윤곽선>

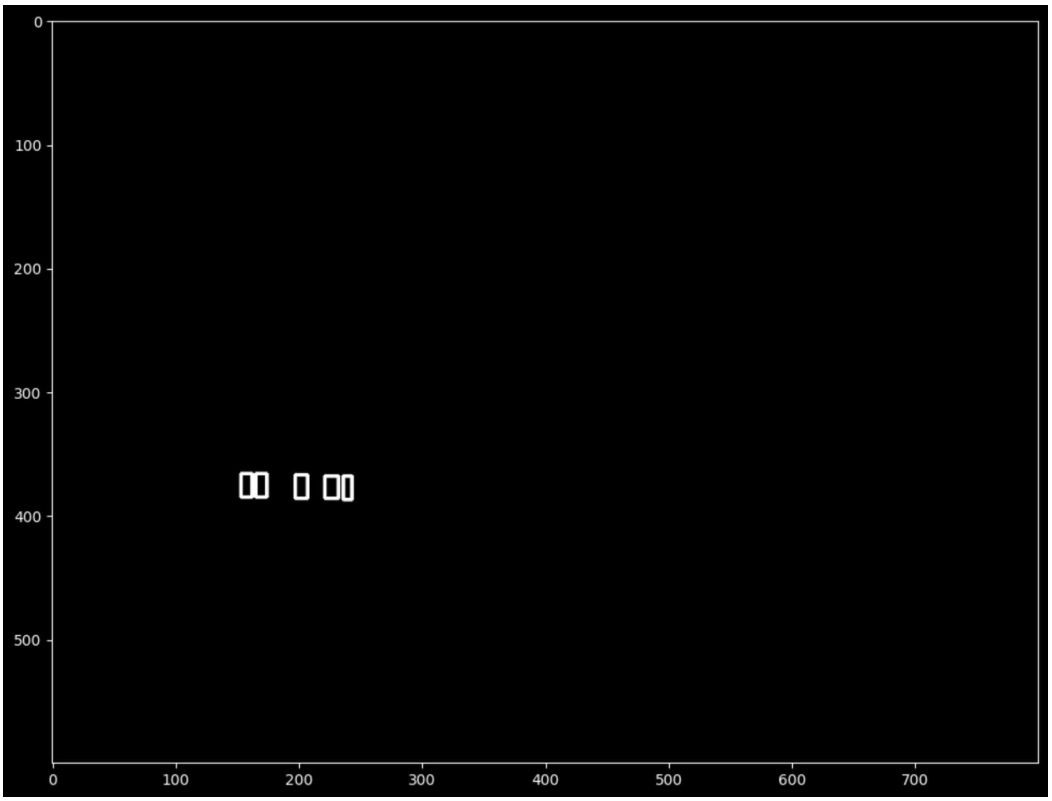
1-4-6. 후보 윤곽선 중에서 실제 번호판 윤곽선 찾기

MAX_DIAG_MULTIPLYER= 5 : 대각선길이

MAX_ANGLE_DIFF= 7.0 : 1 번째 contour 와 2 번째 contour 의 각도

원래는 다른 윤곽선도 후보 번호판으로 인식했는데 이 값을 12.0 에서 7.0 으로 수정했더니 번호판 윤곽선을 잘 찾게 되었다.

MAX_AREA_DIFF= 0.5 : 면적의 차이
MAX_WIDTH_DIFF= 0.8 : 너비 차이
MAX_HEIGHT_DIFF= 0.2 : 높이 차이
MIN_N_MATCHED= 3 : 위에 조건들이 3 개이상 충족해야 번호판이다



< 그림 1-4-6 후보 윤곽선 중에서 실제 번호판 윤곽선>

1-4-7. 번호판에 대한 정보

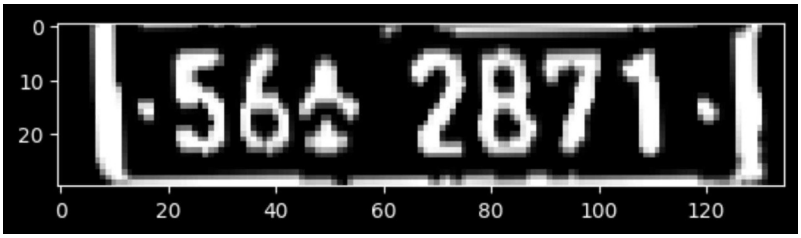
PLATE_WIDTH_PADDING= 1.5# 1.3

PLATE_HEIGHT_PADDING= 1.6# 1.5

MIN_PLATE_RATIO= 3

MAX_PLATE_RATIO= 10

번호판이 잘리거나 불필요한 이미지가 포함되었다면 이 값을을 수정하면 된다. 적정한 값은 사진의 구도와 크기에 따라 달라지기 때문에 변수를 수정하는 방법은 좋은 방법이 아니다.



<그림 1-4-7 번호판 정보를 토대로 찾은 번호판 이미지>

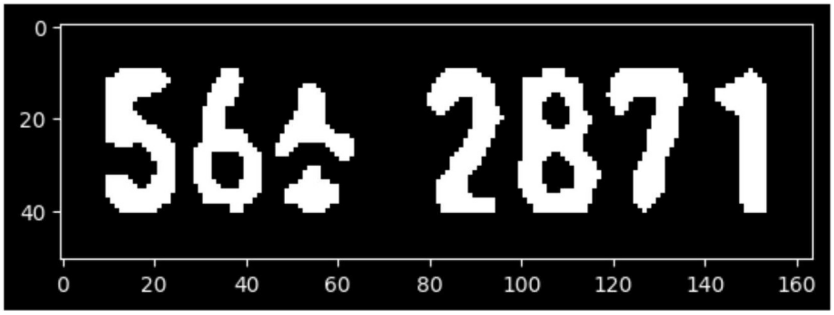
1-4-8. 번호판 이미지에서 인식된 문자를 추출

```python
longest_idx, longest_text = -1, 0
plate_chars = []

for i, plate_img in enumerate(plate_imgs):
    plate_img = cv2.resize(plate_img, None, fx=1.6, fy=1.6, interpolation=cv2.INTER_LINEAR)

    _, plate_img = cv2.threshold(plate_img, thresh=0.0, maxval=255.0, type=cv2.THRESH_BINARY | cv2.THRESH_OTSU)

    # find contours again (same as above)
    contours, _ = cv2.findContours(plate_img, mode=cv2.RETR_LIST, method=cv2.CHAIN_APPROX_SIMPLE)


    plate_min_x, plate_min_y = plate_img.shape[1], plate_img.shape[0]
    plate_max_x, plate_max_y = 0, 0

    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)

        area = w * h
        ratio = w / h

        if area > MIN_AREA \
        and w > MIN_WIDTH and h > MIN_HEIGHT \
        and MIN_RATIO < ratio < MAX_RATIO:
            if x < plate_min_x:
                plate_min_x = x
            if y < plate_min_y:
                plate_min_y = y
            if x + w > plate_max_x:
                plate_max_x = x + w
            if y + h > plate_max_y:
                plate_max_y = y + h

    img_result = plate_img[plate_min_y:plate_max_y, plate_min_x:plate_max_x]

    img_result = cv2.GaussianBlur(img_result, ksize=(3, 3), sigmaX=0)
    _, img_result = cv2.threshold(img_result, thresh=0.0, maxval=255.0, type=cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    img_result = cv2.copyMakeBorder(img_result, top=10, bottom=10, left=10, right=10, borderType=cv2.BORDER_CONSTANT, value=(0,0,0))

    chars = pytesseract.image_to_string(img_result, lang='kor', config='--psm 7 --oem 0')

    result_chars = ''
    has_digit = False
    for c in chars:
        if ord('가') <= ord(c) <= ord('힣') or c.isdigit():
            if c.isdigit():
                has_digit = True
            result_chars += c

    print(f'인식된 문자:', result_chars)
    plate_chars.append(result_chars)

    if has_digit and len(result_chars) > longest_text:
        longest_idx = i

    plt.subplot(len(plate_imgs), 1, i+1)
    plt.imshow(img_result, cmap='gray')
```

✓ 0.1s Python

이진화된 이미지에서 문자 윤곽을 찾고 번호판의 경계와 위치 정보를 사용하여 tesseract 모델을 사용하여 이미지에서 텍스트를 추출하고 한글 문자와 숫자를 선택하여 plate_chars 리스트에 저장한다.

인식된 문자: 56충2871



<그림 1-4-8. 번호판 인식 결과>

그림 1-4-8의 해당 결과를 보면 숫자는 잘 인식을 하는데 한글 문자는 인식이 안되는 것을 볼 수 있다. 다나와 중고차 사이트에서 크롤링한 이미지의 화질은 거의 대부분 동일하기에 다른 이미지도 비슷한 양상을 보였다. 때문에 이미지 데이터를 바꾸거나 다른 모델을 사용할 필요성을 느꼈다.

실험제목	easyocr 모델을 이용한 자동차 번호판 인식		
실험목적	구글에서 선별한 퀄리티 좋은 이미지 데이터를 이용하여 모델의 성능을 테스트한다.		
기록자	울지바야르	책임자	박규호
기록일자	2023.10.27	책임자 승인일	

개발환경:

Python	3.11.4
OpenCV	4.8.1
easyocr	1.7,1
OS	windows 11
IDE	vscode(1.83.1)

학습데이터: 구글 검색을 통해 화질이 좋은 이미지 15장을 선별했다.

Base_model: easyocr

2-1. easyocr 모델을 이용하여 자동차 번호판 인식하는 코드

```python
import os
import cv2
import easyocr
import re
import json
from datetime import datetime
def adjust_contrast(image, factor=1.3):
    # 이미지 대비 조정
    return cv2.convertScaleAbs(image, alpha=factor, beta=0)
def process_images_in_folder(folder_path, output_json_path):
    # 지정한 폴더에서 모든 파일 목록을 가져옴
    image_files = [f for f in os.listdir(folder_path) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp'))]
    # EasyOCR 리더 초기화
    reader = easyocr.Reader(['ko'])
    results_list = []  # 결과를 저장할 리스트
    for image_file in image_files:
        # 이미지 파일 경로 생성
        image_path = os.path.join(folder_path, image_file)
        # 이미지 불러오기
        img = cv2.imread(image_path)
        # 이미지 대비 조정
        img = adjust_contrast(img)
        # 이미지를 그레이스케일로 변환
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # 번호판 인식
        results = reader.readtext(gray)
        # 숫자와 한글만 포함된 번호판 패턴 정의
        pattern = re.compile("[가-힣0-9]+")
        # 신뢰도에 따라 정렬된 결과 중 가장 높은 신뢰도의 결과만 출력
        results.sort(key=lambda x: x[2], reverse=True)
        for (bbox, text, prob) in results:
            # 정규식에 맞는 문자열만 출력
            filtered_text = "".join(pattern.findall(text))
            if filtered_text.isdigit():
                print(f"인식이 올바르지 않습니다. ({image_file}): {filtered_text}")
            else:
                print(f"인식된 번호판 ({image_file}): {filtered_text}")
                # JSON으로 저장할 정보 구성
                result_info = {
                    "이미지명": image_file,
                    "이미지형식": image_file.split('.')[-1],
                    "X축 위치": int(bbox[0][0]),  # bbox[0]에서 X 좌표 추출
                    "Y축 위치": int(bbox[0][1])   # bbox[0]에서 Y 좌표 추출
                }
                results_list.append(result_info)
                break
        else:
            print(f"인식되지 못했습니다. ({image_file})")
    # JSON 파일로 저장
    dataset_info = {
        "데이터셋이름": "인식된 번호판 데이터셋",
        "데이터셋설명": "번호판 이미지와 해당 번호판 번호를 포함한 데이터셋",
        "데이터셋출처": "다나와 중고차 (https://auto.danawa.com/usedcar/?Work=list&Tab=list), 구글 이미지 ",
        "작성자정보": "강형근, 울지바야르",
        "생성일자": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "이미지정보": results_list
    }
```

```
    with open(output_json_path, 'w', encoding='utf-8') as json_file:
        json.dump(dataset_info, json_file, ensure_ascii=False, indent=4)

if __name__ == "__main__":
    # 이미지 파일이 있는 폴더 경로 입력
    input_folder_path = "./auto1.3"  # 실제 폴더 경로로 대체해주세요
    # 결과를 저장할 JSON 파일 경로
    output_json_path = "recognized_plates.json"
    process_images_in_folder(input_folder_path, output_json_path)
```

✓ 3.7s

```
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
인식된 번호판 (1-4.jpeg): 355구2906
인식된 번호판 (1-5.jpeg): 123가4568
인식된 번호판 (1.jpeg): 154러7070
인식된 번호판 (15.jpg): 123가4568
인식된 번호판 (19.jpeg): 절96사2268
인식된 번호판 (2.jpeg): 381마8947
인식된 번호판 (3.jpeg): 50노6821
인식된 번호판 (4.jpeg): 60라7213
인식된 번호판 (6.jpeg): 45가6492
```

<그림 2.1 지방 택시 번호판>

19 번을 제외한 나머지 번호판은 모두 인식이 잘 되었다. 19 번 번호판은 위 이미지를 인식한 결과인데 지방 택시의 경우 앞자리에 전북과 같이 세로 형태로 지역 정보를 나타내었고 다른 번호판과 다른 형태이기 때문에 인식이 잘 안된 것 같다. 이를 해결하려면 윤곽선을 찾고 후보 번호판의 윤곽선을 고를 때 번호판의 패턴을 하나 더 추가하면 될 것 같다.

2.2 결론

위 코드는 easyocr 모델을 사용했고 1-4 번에서 진행한 tesseract 모델과는 달리 코드를 함수화하였다. 그 이유는 json 파일로 저장하는 코드, 폴더 내의 모든 이미지를 for 문을 사용하여 모델에 인식하는 코드 등을 간결히 보여주기 위함이다. 자동차 번호판을 인식한 결과 변수값을 1.3 으로 했을 때 인식률이 가장 높았다. 그러나 수많은 이미지 데이터를 맞춤형으로 매개변수를 조정하여 인식이 잘 되게끔 하는 방법은 현실적으로 의미가 없고 인공지능의 장점을 살리지 못하는 방법이기 때문에 이번 프로젝트를 통해 각도와 거리, 화질 등이 달라도 어떻게 이미지를 전처리하면 모델의 정확도가 높아질지에 대해서 더 고민을 할 필요가 있다고 느꼈다.