캡스톤 프로젝트 중간발표

지도 교수님 : 박용범 교수님

팀원 : 김정민, 마진우,

프로젝트 개요

- pyTesseract를 이용한 Tesseract OCR의 한글 인식률을 높인다.

- pyTesseract는 python tesseract의 약자로 Tesseract를 파이썬으로 사용할 수 있도록 Tesseract OCR 엔진을 래핑한 라이브러리

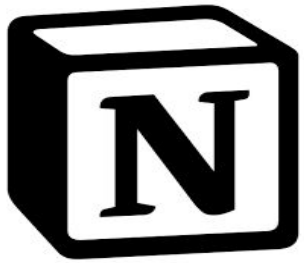
Tesseract OCR

- Optical Character Recognition
  - 광학 문자 인식 – 사진 속의 문자를 인식하는 기술
  - 이미지로부터 문자를 추출하기 위해 사용

- Tesseract OCR은 OCR의 여러 종류 중 하나로, 오픈소스 형태로 제공되며 Apache 2.0 라이센스를 따른다.

- 이미지 전처리 과정이 필요하며 한글 인식률이 낮다는 단점이 있다.

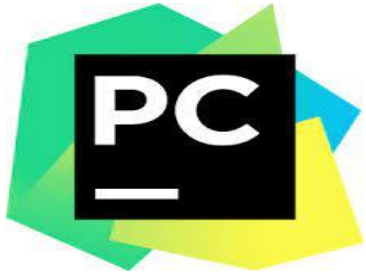
프로젝트 메니지먼트 툴



Notion

Google docs

개발환경, 버전관리



pyCharm

Anaconda

GitHub

라이브러리

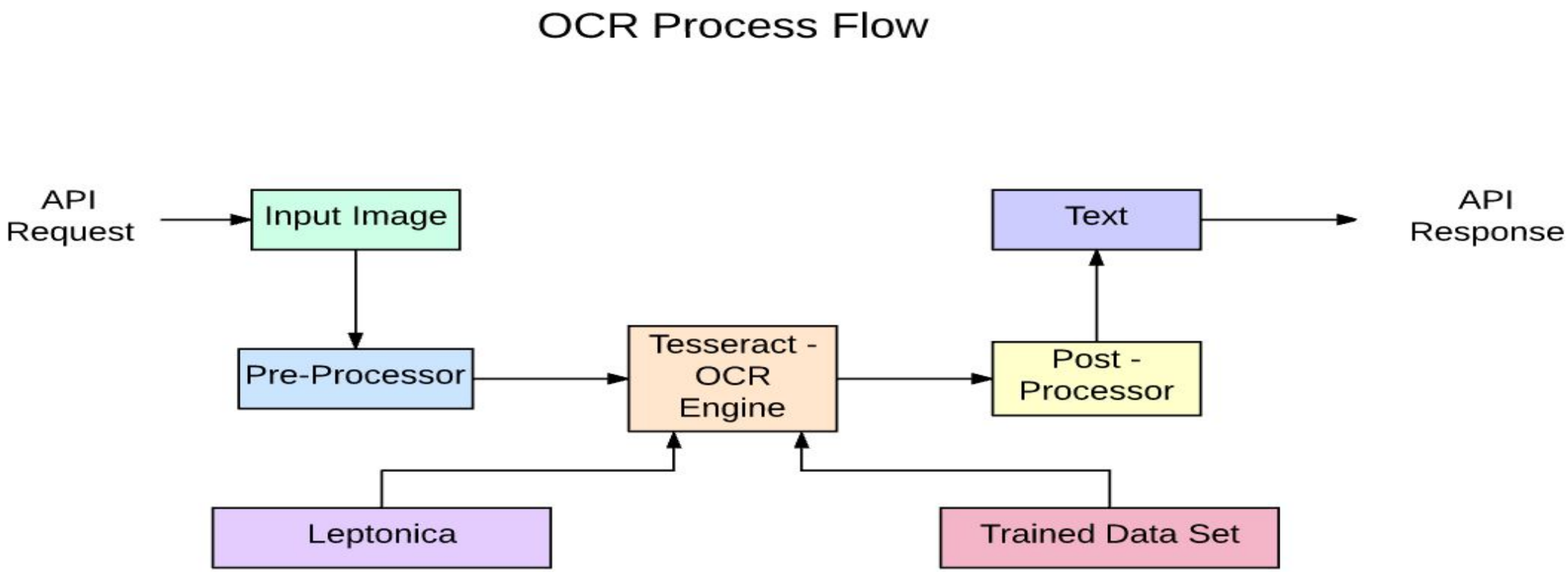
인식 과정

❖ 전처리 (Pre-processing)
- 글자들이 잘 보이도록 밝기, 색 등의 이미지 속성을 변형

❖ 문자 검출 (Text Detection)
- 이미지로부터 글자를 찾고 글자들을 bounding box로 결합
- 묶인 글자들은 인식 모델의 입력으로 들어가게 된다.

❖ 문자 인식 (Text Recognition)
- Bounding box 안의 글자를 읽는다.

OCR Process Flow



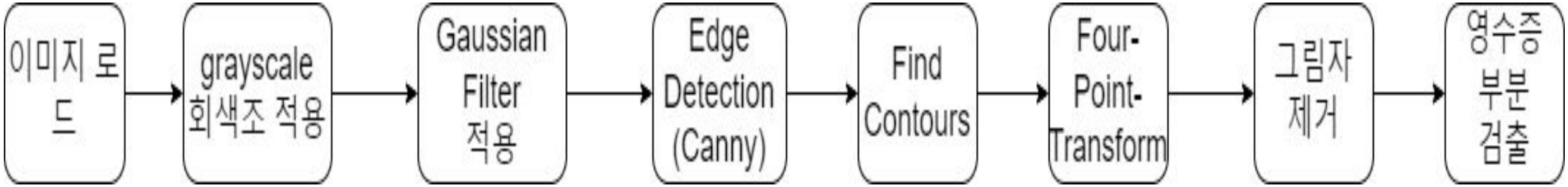
인식 개선 방법

❖ 이미지 전처리
- 이진화, 노이즈 제거, 이미지 팽창과 침식, 이미지 회전 및 기울기 보정, 투명도 조절, 테두리 제거

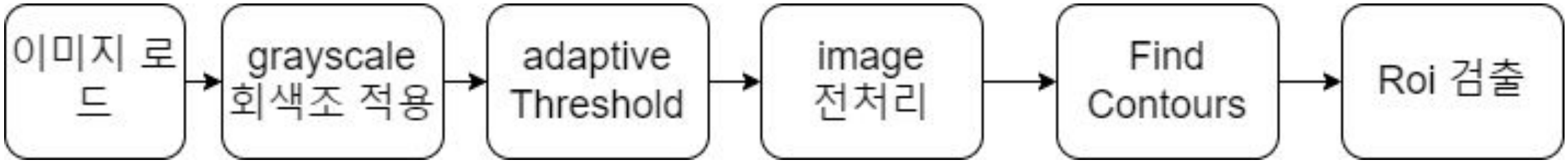
❖ 한글데이터 학습
- 한글데이터를 Tesseract에 학습시켜 인식률을 높인다.

이미지 전처리 방법

❖ 이미지 중 영수증 부분만 검출



❖ 영수증에서 글자 부분만 검출



기법 소개

❖ Grayscale

- RGB이미지랑 다르게 Grayscale 이미지는 밝기만을 표현하며 이를 0~255의 값으로 표현한다.

❖ GaussianBlur

- 컴퓨터가 이미지를 분석할 때는 흐릿한 이미지가 효율적이기 때문에 이미지를 흐리게 할 필요성이있다.
- GaussianBlur는 OpenCV에서 제공하는 필터링 함수이며 이미지를 흐리게 하는 기능을한다.

기법 소개
Contour

❖ Contour
- 동일한 색 또는 색상 강도를 가진 부분의 가장자리 경계를 연결한 선이다.
- OpenCV에서 contour을 찾으려면 배경을 검은색으로 대상은 흰색으로 변경해야한다.
- 여기서는 findContours()를 사용하여 contour을 찾는다.

이미지 전처리 방법(공통)
회색조 적용

❖ cv2.cvtColor(src, dst, cv2.BGR2GRAY)
- OpenCV에서 이미지는 BGR 형태
- 해당 이미지를 회색조로 변환시켜줌

이미지 영수증 부분만 검출
Guassian Blur 적용

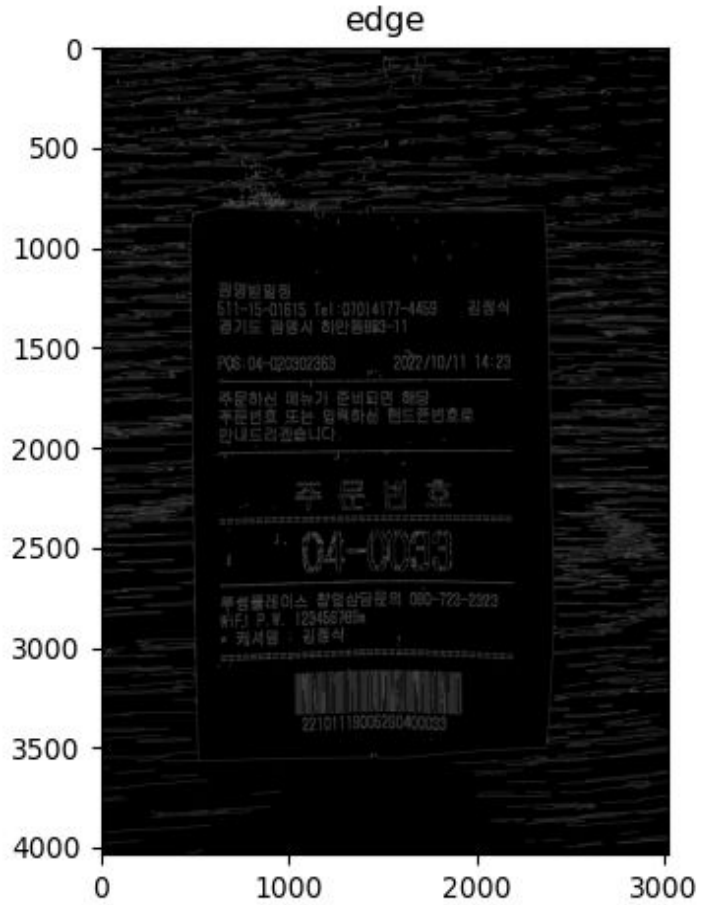
- cv2.GuassianBlur(src, kernel, 0)
  이미지를 흐릿하게 한다.

이미지 영수증 부분만 검출
Edge Detection – Canny

- edge는 객체의 외각선(객체간 혹은 배경과의 경계)을 나타내고 Canny()함수를 사용하여 이미지의 edge를 추출한다.

- 여기선 gray이미지에 blur를 사용한 이미지를 Canny()함수에 사용한다.

이미지 영수증 부분만 검출
Edge Detection – Canny



이미지 영수증 부분만 검출
Find Contour, Four-Point-Transform

- 1. Edge Detection이후 Contours 검출
- 2. Contour를 Contour영역의 넓이 순서대로 정렬
- 3. 가장 큰 Contour를 Four-point-Transform 함수를 적용시켜 영수증 ROI를 구한다.

이미지 영수증 부분만 검출
그림자 부분 제거

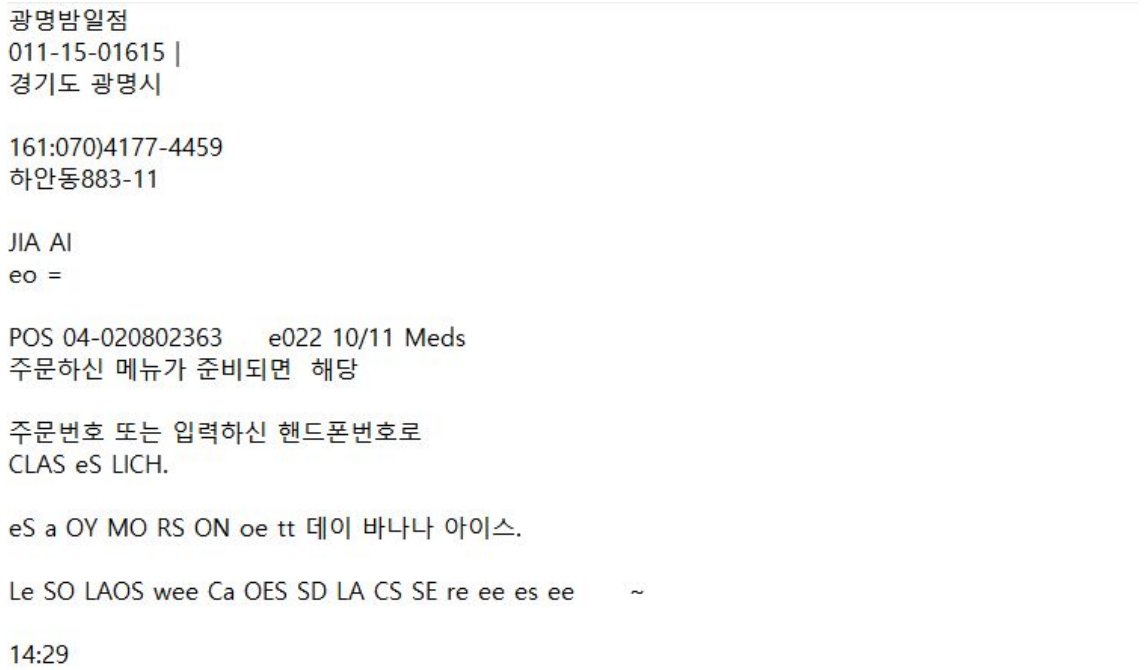
❖ 이전 과정에서 구한 영수증 ROI의 그림자 부분을 제거한다.

```
def remove_shadow(image):
    rgb_planes = cv2.split(image)

    result = []

    for plane in rgb_planes:
        dilated_img = cv2.dilate(plane, np.ones((8, 8), np.uint8))
        bg_img = cv2.medianBlur(dilated_img, 21)
        diff_img = 255 - cv2.absdiff(plane, bg_img)
        norm_img = cv2.normalize(diff_img, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8UC1)
        result.append(norm_img)
    result = cv2.merge(result)

    return result
```

이미지 영수증 부분만 검출 이후 Pytesseract 적용

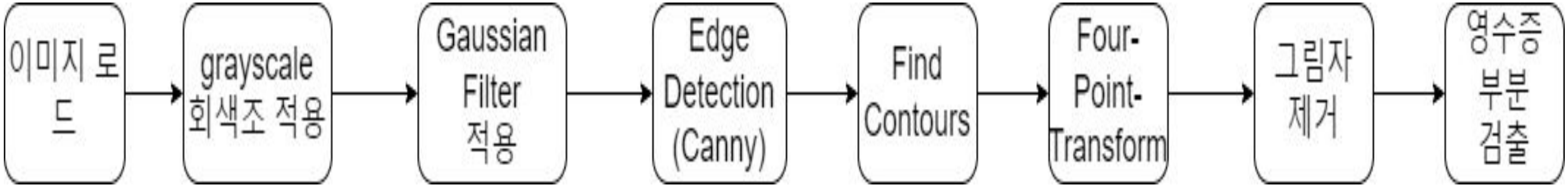
- 영수증 ROI에 Pytesseract을 적용시켜 검출한 결과

- 사진 수정 필요

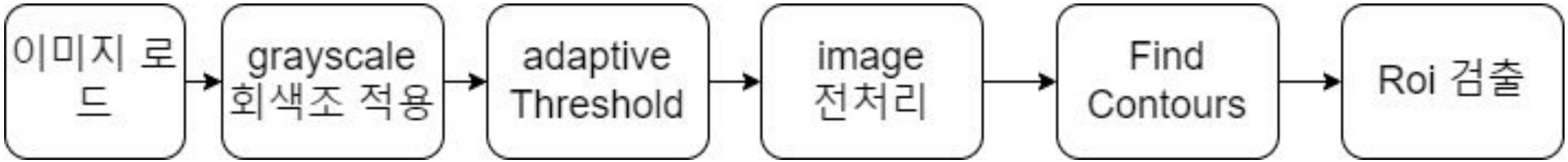


이미지 전처리 방법

❖ 이미지 중 영수증 부분만 검출



❖ 영수증에서 글자 부분만 검출



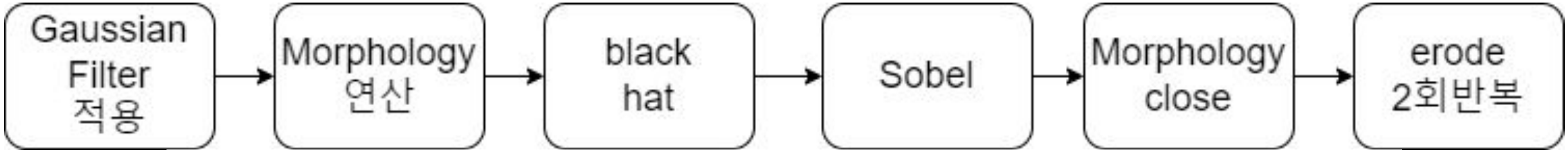
영수증에서 글자 부분 검출하기
AdaptiveThreshold

❖ AdaptiveThreshold

- 원래 Threshold 함수는 모든 화소마다 같은 임계값을 적용하지만, 해당 함수는 다른 임계값을 적용하여 이미지를 계산합니다.
- 파라미터
  - AdaptiveMethod: 적응형 임계값의 종류로 주변값에 적용하는 방법으로 해당 함수에선 3*3크기의 이웃에서 평균을 계산한 다음 C를 뺄셈한 값을 임계값을 사용합니다.
  - 임계값 종류론 THRESH_BINARY_INV를 사용했으며, 해당 임계값 종류는 값을 이진화시킨다.
  - 해당 이미지에 명암차가 큰 경우에 사용한다.

```
adap_thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 3, 12)
```

영수증에서 글자 부분 검출하기
Image 전처리




```
def morph_close(img):
    gray = copy.deepcopy(img)
    rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (30, 20))
    square_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (50, 50))

    gray = cv2.GaussianBlur(gray, (11, 11), 0)
    blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, rect_kernel)

    grad = cv2.Sobel(blackhat, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=-1)
    grad = np.absolute(grad)
    min_val, max_val = np.min(grad), np.max(grad)
    grad = (grad - min_val) / (max_val - min_val)
    grad = (grad * 255).astype('uint8')
```

영수증에서 글자 부분 검출하기
Image 전처리

❖ Gaussian Filter 적용

- 전체 이미지에서 영수증 영역을 검출할 때 과정처럼 이미지에 가우시안 블러를 적용한다.
- 목적: 이미지의 가장자리 경계를 검출을 쉽게하기 위해서 사용한다.

```
gray = copy.deepcopy(img)
rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (30, 20))
square_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (50, 50))

gray = cv2.GaussianBlur(gray, (11, 11), 0)
```

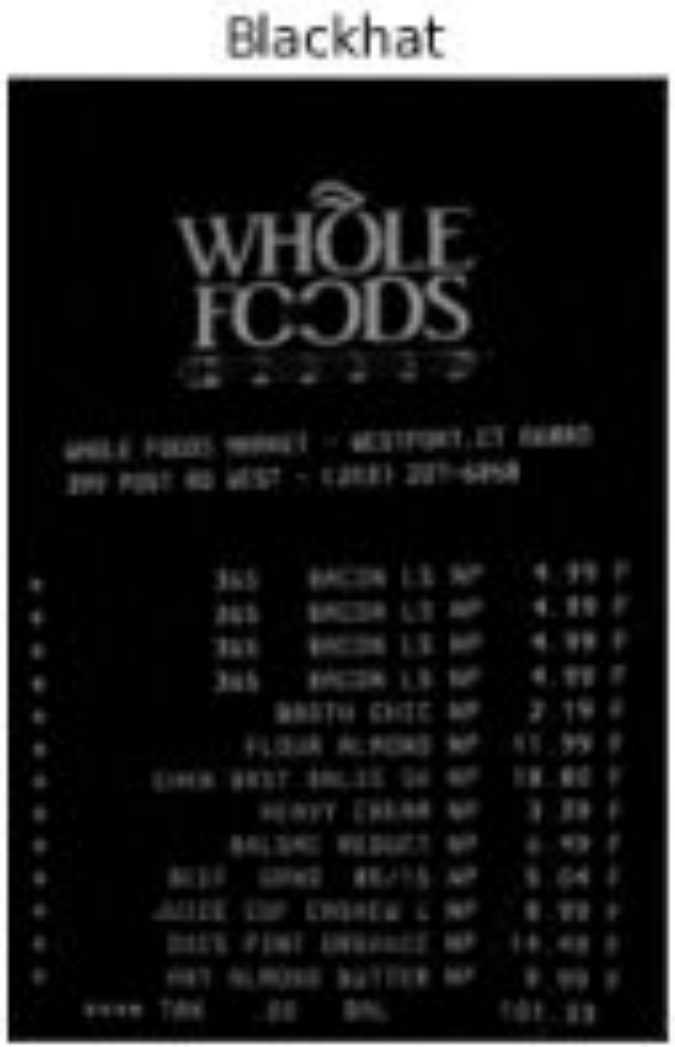
영수증에서 글자 부분 검출하기
Image 전처리 – black hat

- Morphology 연산
  - MorphologyEx에는 Erosion, Dilation, Opening, Closing 연산하는 함수등 여러 옵션이 파라미터로 저장되어있다.
  - 먼저 MORPH_BLACKHAT은 이미지에 BLACKHAT을 적용시켜준다.

```
blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, rect_kernel)
```

기법 소개

- Blackhat
  - 밝은 배경에서 어두운 영역을 드러내기 위해 사용된다.
  - 영수증의 경우 밝은 배경은 영수증의 배경, 어두운 영역은 텍스트가 된다.



영수증에서 글자 부분 검출하기
Image 전처리 – Sobel

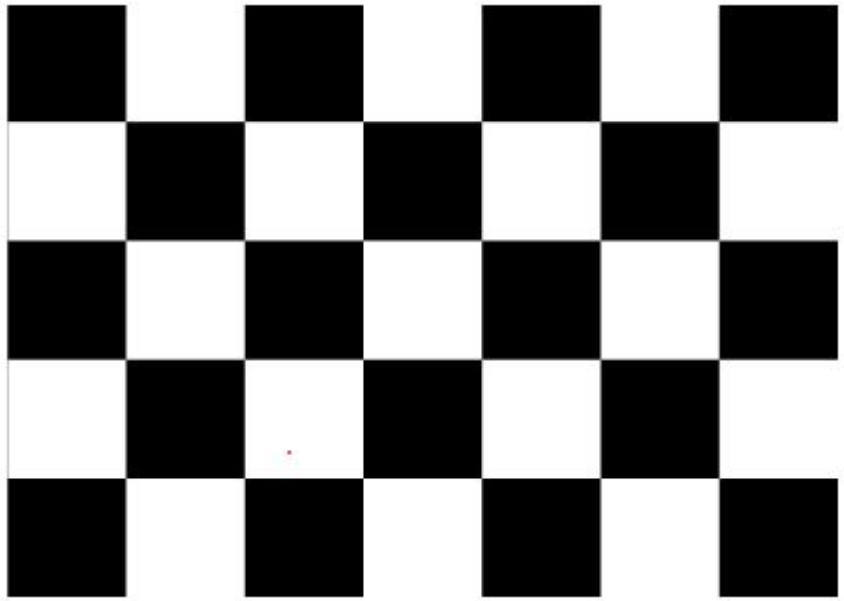
```
grad = cv2.Sobel(blackhat, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=-1)
```

-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter



영수증에서 글자 부분 검출하기
Image 전처리 – Sobel

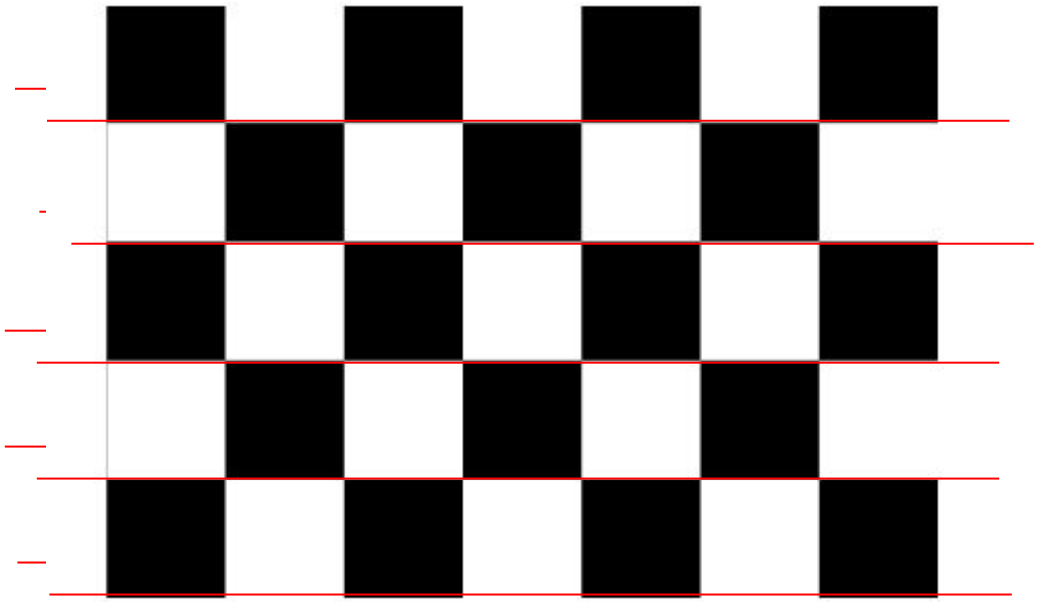
```
grad = cv2.Sobel(blackhat, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=-1)
```

-1	0	+1
-2	0	+2
-1	0	+1

x filter

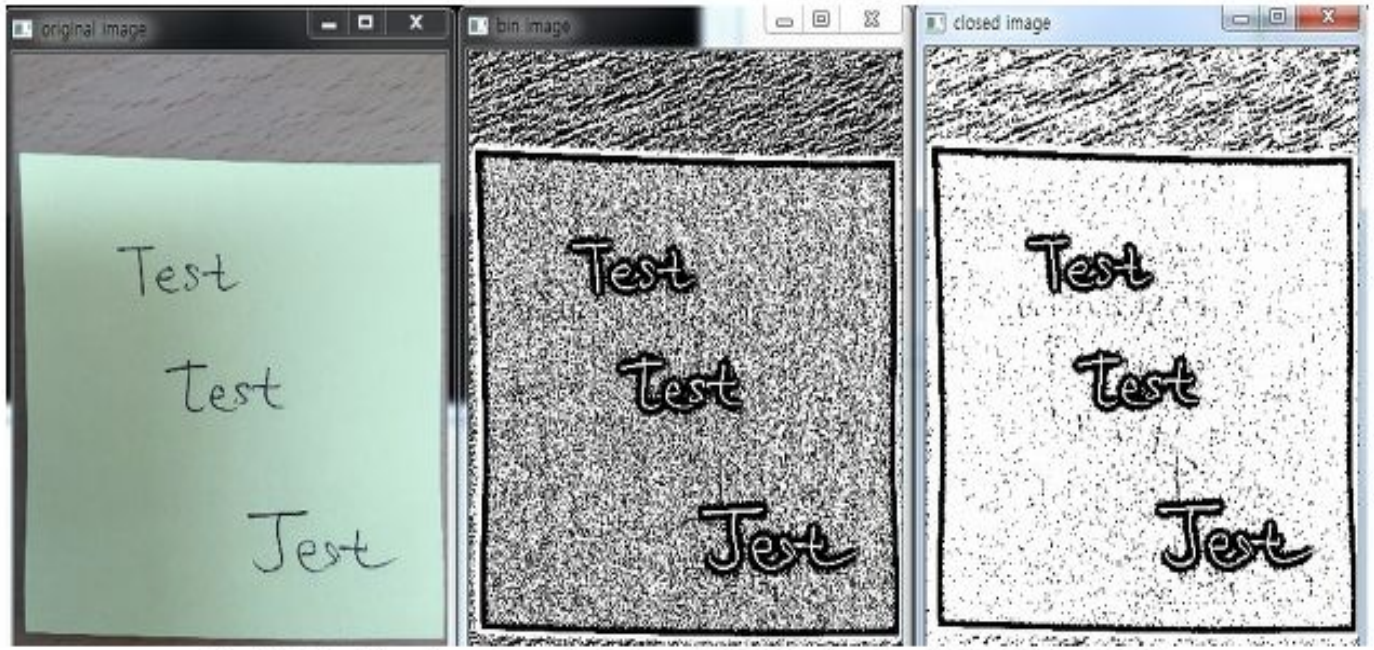
+1	+2	+1
0	0	0
-1	-2	-1

y filter



영수증에서 글자 부분 검출하기
Image 전처리 – Morphology close

❖ Morphology 연산 – Close



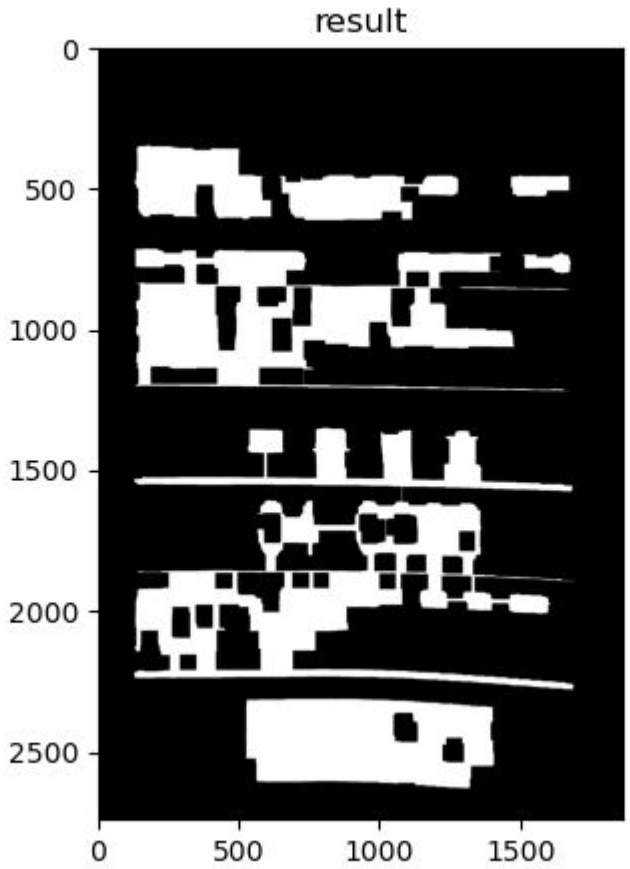
[그림 5: Adaptive thresholding 결과에 closing을 적용한 결과]

```
grad = cv2.morphologyEx(grad, cv2.MORPH_CLOSE, rect_kernel)
thresh = cv2.threshold(grad, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

square_thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, square_kernel)
square_thresh = cv2.erode(square_thresh, None, iterations=2)
```

영수증에서 글자 부분 검출하기
Image 전처리 – Morphology

❖ Morphology 연산 – Close

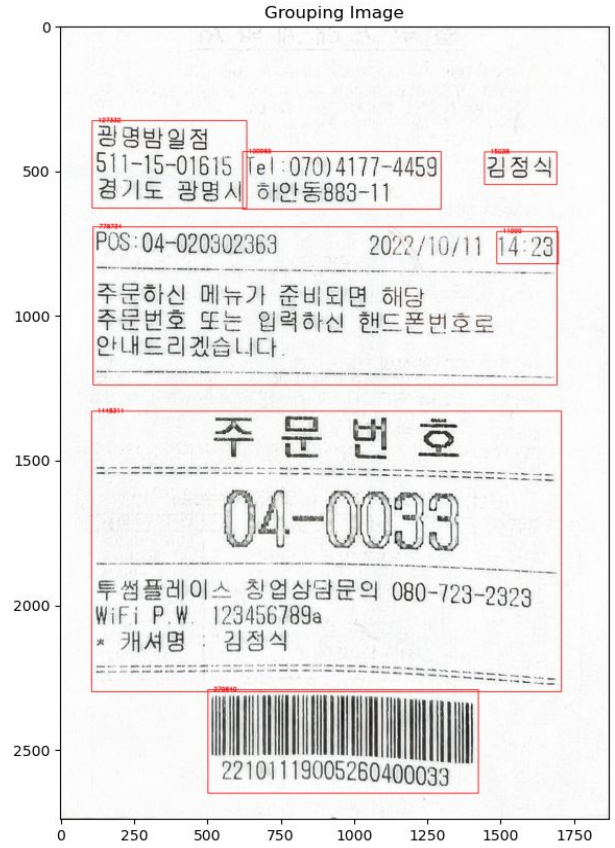
```
def morph_close(img):
    gray = copy.deepcopy(img)
    rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (30, 20))
    square_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (50, 50))

    gray = cv2.GaussianBlur(gray, (11, 11), 0)
    blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, rect_kernel)

    grad = cv2.Sobel(blackhat, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=-1)
    grad = np.absolute(grad)
    min_val, max_val = np.min(grad), np.max(grad)
    grad = (grad - min_val) / (max_val - min_val)
    grad = (grad * 255).astype('uint8')

    grad = cv2.morphologyEx(grad, cv2.MORPH_CLOSE, rect_kernel)
    thresh = cv2.threshold(grad, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

    square_thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, square_kernel)
    square_thresh = cv2.erode(square_thresh, None, iterations=2)

    plt_imshow('result', square_thresh)
    return square_thresh
```



영수증에서 글자 부분 검출하기
Find Contours

❖ 이미지 전처리 이후 글자 부분의 Contour를 직사각형 형태로 만들어, 좌표를 구한후 ROI를 구해 글자부분만 검출한다.

```
def detection_string_contours(img):
    """
        이미지 영역(영수증 부분만 전처리한 후)에서 글자 부분을 인식함
    """
    img_clone = copy.deepcopy(img)
    gray = cv2.cvtColor(img_clone, cv2.COLOR_BGR2GRAY)

    adap_thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 3, 12)

    morph_img = morph_close(adap_thresh)
    contours, hierarchy = cv2.findContours(morph_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = sort_contours(contours, method='top-to-bottom')[0]

    return contours, gray
```



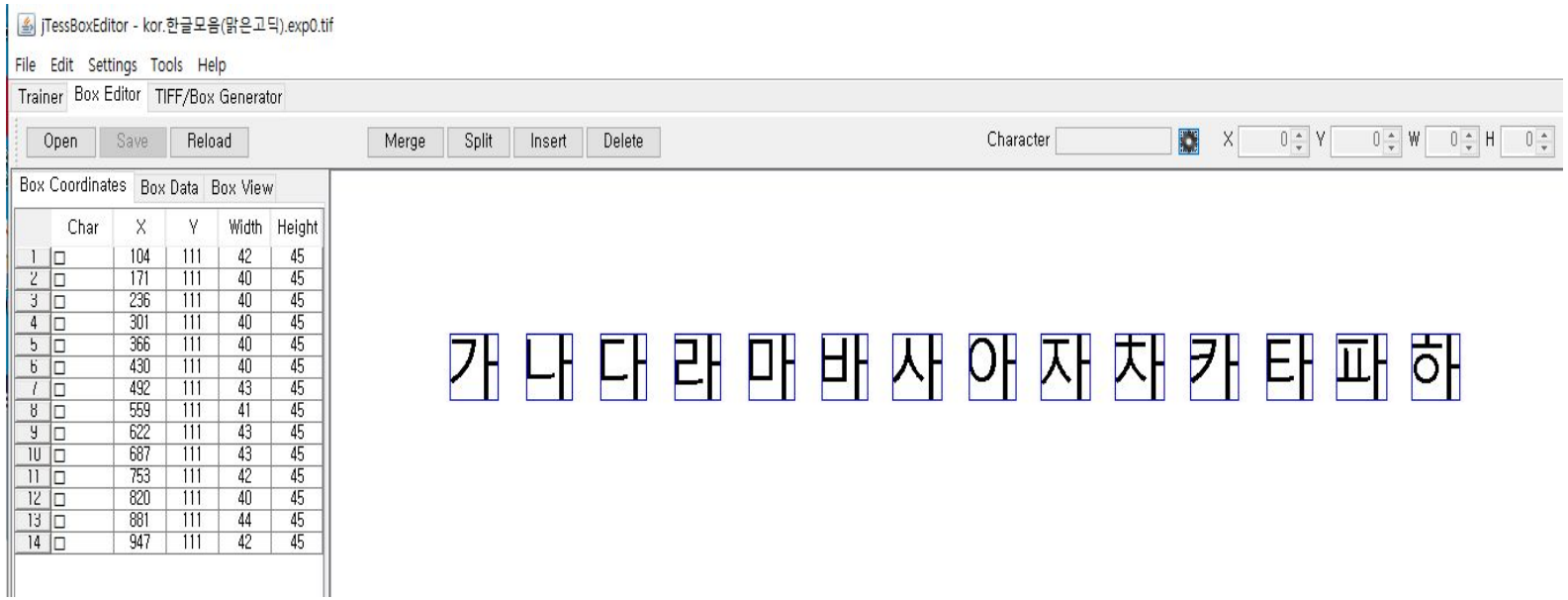
영수증에서 글자 부분 검출하기
Pytersseract 적용

- 이미지 전처리와 검출이 되었다면 해당 ROI들에 있는 내용을 모두 Pytesseract를 이용해 글자를 출력한다.

한글 데이터 학습 방법

❖ JTessBoxeditor

- 학습하려하는 한글을 JTessBoxeditor로 tif 및 box파일화 시킨다.
- 각 글자마다 box로 구분이 되는지 확인 후 저장한다.
- 저장한 파일을 Train을 시키면 traindata파일이 만들어지며 이를 Tesseract-OCR 파일에있는 traindata와 교체한다.



한글 데이터 학습 방법

❖ Tesseract 자체 엔진 학습

- 가상화 소프트웨어를 통해 Ubuntu 가상환경을 구축한다.
- 관련 라이브러리를 설치하고 Tesseract-OCR을 설치한다.
- 처음부터 다시 학습이 아닌 미세조정 방법을 사용할 것이기 때문에 Github에서 배포중인 traindata파일을 학습 파일로 사용한다.
- 학습데이터 생성을 마치고 기존의 모델을 기반으로 학습을 진행한다.
- 원하는 결과를 얻을때 까지 튜닝을 반복한다.

앞으로의 진행 방향

❖ 이미지 전처리
- 앞에서 언급했던 기법들을 사용하여 이미지를 사용하기 좋은 형태로 만든다.

❖ 데이터 학습
- JTessBoxeditor같은 도구를 이용한 학습이나 가상환경에서 tesseract 엔진을 직접 학습 시키는 방법 중 하나를 선택하여 데이터를 학습시키고 원하는 결과를 얻을때 까지 튜닝한다.

❖ 기존 Tesseract-OCR과 비교
- 이미지 전처리, 데이터 학습의 방법이 한글 인식률에 어떠한 영향을 주었는지 판단하기 위해 기존의 tesseract-OCR의 한글 인식률과 개선을 거친 tesseract-OCR의 한글 인식률을 비교한다.