< 텍스트 판별 모델 구현 과정 >

1. pytesseract-ocr을 이용해 텍스트 검출 시도

알약 원본 이미지에서 배경만 제거하여 pytesseract-ocr 모델에 인식시켜 보았다.

>> 너무 명확한 텍스트에 대해서만 검출이 성공하여 인식률이 굉장히 떨어졌고, 다른 방법을 강구해야 했다.

2. 이미지 전처리 후 pytesseract-ocr을 적용하여 텍스트 검출 시도

알약 이미지를 전처리 및 이진화하여 pytesseract-ocr에 인식시켜 보았다.
시도한 전처리 기법은 다음과 같다.

0) 원본 이미지 (프린팅된 텍스트)

1) 배경 제거

2) CLAHE* (Contrast Limited Adaptive Histogram Equalization) 적용

* CLAHE : histogram equalization의 변형된 형태로, 이미지를 (N*M)의 구역들로 나누어 해당 구역에 대하여 각각 histogram equialization을 적용한다. histogram equialization에 비해 어두운 부분은 상대적으로 더 밝게, 밝은 부분은 상대적으로 더 어둡게 하는 효과가 있다.

3) gray scale 적용

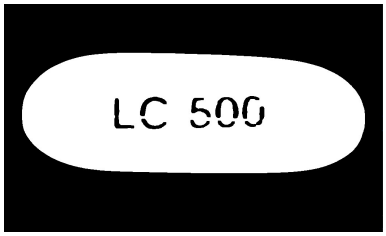
4) gaussian blur 적용

* gaussian blur : (N*M) 블록을 기준으로 matrix의 평균값을 이용해 blur 처리하는 mean blur와 달리, 정규분포를 이용해 blur 처리를 한다. 해상도가 낮은 이미지의 경계를 뭉개는 효과가 있다.

5) cv2.convertScaleAbs()로 밝기(alpha=2.8, beta=20)를 증가시킴 (=밝기 * 2.8 + 20)

6) 고정값 threshold=220 으로 이진화 적용

7) closing* (block size=(5*5)) 기법을 적용하여 이미지의 noise 제거



* closing : erosion* 후에 dilation*을 적용한 것으로, 이미지의 noise를 제거하는 효과가 있다.
* erosion : (N*M) 블록 내에 0이 하나라도 존재하면, 해당 부분에 대하여 0으로 취급한다.
* dilation : (N*M) 블록 내에 1이 하나라도 존재하면, 해당 부분에 대하여 1로 취급한다.
+ opening : dilation 후에 erosion을 적용한 것으로, 이미지 내부에 구멍이 있을 경우 이를 메우는 효과가 있다.

>> 이전보다는 tesseract-ocr의 인식률이 나아졌으나, 여전히 정확도가 낮아 이대로 텍스트 인식에 사용하기엔 부적합하다고 판단하였다. 또한, 알약의 텍스트는 프린팅된 것과 음각된 것 두 종류가 있는데, 음각된 것에 대해서는 전처리 결과가 좋지 못하였다. 따라서 프린팅, 음각 두 경우에 대해 전처리 메서드를 두 가지로 분리하거나, 두 경우 모두에 적용되는 하나의 전처리 메서드를 구현할 필요가 있었다.

3. 텍스트가 음각된 알약의 이미지 전처리

두 가지 방법을 시도하였다.

A.

- 원본 이미지 (음각된 텍스트)

- 전처리 과정

윤곽선 제거 이전 | getPillContours에서 closing 적용 | getPillContours에서 closing 적용 x | erosion(3,3) closing(5,5) 적용

A-1) gray scale > histogram equializaton > adaptive threshold* 를 적용하여 이진화된 이미지를 생성한다.
A-2) getPillContour()*로 윤곽선을 검출해 이진화된 이미지에서 윤곽선을 제거하고, 여기에 추가로 closing(5*5) 을 적용한다.
A-3) A-2에서 closing(5*5)을 제외하고 실행시킨 결과물이다.
A-4) A-2 이후 erosion(3*3), closing(5*5)를 추가 적용시킨 결과물이다.

* adaptive threshold : 기존의 고정값 threshold와 달리, CLAHE와 비슷하게 (N*M) block 내에서 평균적인 임곗값을 설정하고 이진화시킨다. 데이터마다 밝기가 달라도 비교적 균일하게 이진화된다는 장점이 있다. 다만 이진화 결과물에 noise가 많이 섞인다는 단점이 존재한다.
* getPillContour() : gray scale > median blur* > closing(10*10) > cv2.findContours() 를 적용한 커스텀 메서드이다.
* median blur : salt and pepper noise를 제거하는 데 탁월한 blur 기법으로, 이진화된 이미지에 salt and pepper 형태의 noise가 많아 적용하였다.

>> 꽤 괜찮아 보이는 결과물이 나왔지만, 카메라와 조명의 각도에 따라 음영이 달라지기 때문에 데이터마다 결과물이 상이할 수 있다는 문제가 있었다. 따라서 이대로 모델에 학습시키기에는 아직 부족하다고 판단했다. 또한 프린팅된 텍스트에 대해서는 작동하지 않고, 음각된 텍스트에 대해서만 작동하였다. 그리고 우리가 가진 데이터에는 알약 텍스트의 프린팅, 음각 여부를 알려주는 지표가 없었기 때문에 A 방법은 결국 사용하지 못하게 되었다.

B.

black hat(13,13) 적용 | blackhat + thershold(30) | + dilation(3,3)

B-1) gray scale 변환 후, Black hat* (13*13) 연산을 적용한다.
B-2) 고정값 threshold=30 으로 이진화를 적용한다.
B-3) dilation(3*3) 을 적용하였다.
B-4.1) 다른 테스트 데이터에 B-3)을 적용한 결과물이다.
B-4.2) 다른 테스터 데이터에 B-3)을 적용하지 않은 결과물이다.

* black hat : closing 연산 결과물에서 원본 이미지를 뺀 것으로, 어두운 부분을 강조하는 효과가 있다. 음각된 알약의 텍스트 부분은 어두우므로, 이 기법이 효과를 보였다. printing된 텍스트에 대해서도 약간의 효과가 보였다. 이후에 배경 제거 메서드의 처리 시간이 너무 길다는 문제가 수면 위로 떠올라 전처리 과정에서 배경 제거를 사용하지 못하게 되었는데, black hat을 이용하면 배경을 제거하지 않고도 괜찮은 전처리 결과를 보여주어 고정적으로 사용하게 되었다.

>> 음각 텍스트도 만족할 만한 전처리 결과가 나왔으나, 여전히 tesseract-ocr은 이를 제대로 인식하지 못하였다. 또한, adaptive threshold를 적용하니 이진화 결과물에 noise가 너무 많아서 고정값 threshold를 적용했는데, 이미지마다 밝기가 다르므로 결과물들이 데이터마다 다소 상이한 문제가 있었다.

4. 텍스트 area를 추출하는 메서드 구현 시도

이미지 전처리는 하였으나, 이 데이터를 그대로 학습시키니 결과물이 좋지 못하였다. 따라서 text 영역만을 추출하여 학습시키는 방안을 떠올렸다.

여기서 또 두 가지 방안으로 세분화할 수 있다.

a. 텍스트 영역 전체를 추출하여 학습시키는 방법
- class 개수 : 약제의 종류 수
- CNN layer 구상 : convolution > maxPool > convolution > maxPool > softmax

b. 텍스트 문자 각각을 추출하여 해당 문자를 학습시키는 방법
- class 개수 : 약제 텍스트에 쓰여 있는 문자의 모든 종류 수(a~z, 0~9, 기타 특수문자)
- CNN layer 구상 : a.와 동일

a, b 방안 둘 다 일단 텍스트 영역 인식이 가능한 것이 전제이므로, cv.MSER (Maximally Stable Extreme Regions) 와 cv2.findContours 를 사용해 텍스트 영역을 추출하는 메서드 구현을 시도해 보았다.
>> 그러나, 앞서 말했던 이미지 밝기마다 전처리 결과물이 상이한 문제 때문에 이미지에 noise가 너무 많아, cv.MSER와 cv2.findContours를 이용한 text 영역 검출은 좌절되었다.
여러 방법을 찾아보던 중, 텍스트 영역 검출에 적합한 pre-trained 모델을 발견하게 되었다.

black hat(13,13) 적용

blackhat + thershold(30)

> 좋은 전처리 결과물의 예

> noise가 많아 좋지 않은 전처리 결과물의 예

5. CRAFT-pytorch 모델 적용

텍스트 영역 검출에 뛰어난 성능을 보이는 CRAFT-pytorch 모델을 발견하였다.
테스트 결과 굉장히 높은 텍스트 인식률을 보였다.
이 모델을 프로젝트에 적용하기 위해, 모델의 구조와 입출력을 수정할 필요가 있었다.

* 기존 모델 정보
- 입력 : shell을 통해 command를 입력하면, command의 arguments로부터 source 이미지들의 디렉토리 path와 기타 가중치 인자들을 읽어옴
- 출력 : result 디렉토리를 생성하고, bounding box가 적용된 이미지와 bounding box들의 좌표를 각각 jpg, txt 형식으로 result 디렉토리에 저장함

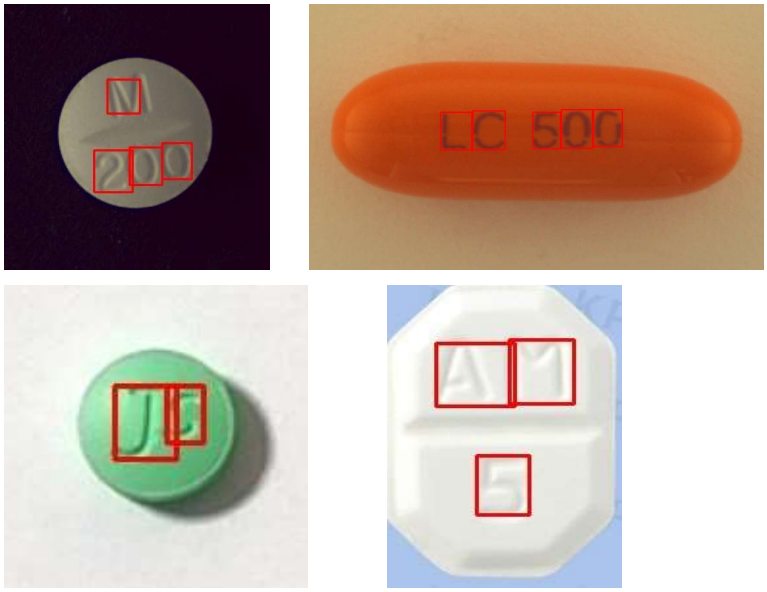
* 문제 : subprocess를 생성해 모델을 실행시키고 file 형식으로 저장된 bounding 적용 결과물들을 다시 읽어와 사용하게 하려 했으나, 이 방식대로면 이미지 하나하나 입력할 때마다 모델을 새로 initialize해야 했는데, 모델의 초기화 시간이 너무 오래 걸려 모델의 구조와 입출력을 수정할 필요가 생김.

* 수정 내역 (./CRAFT/test.py)
- 구조 : command를 통해 단일 파일로 실행되는 구조를, 다른 코드에서 import하여 사용할 수 있도록 class 객체 구조로 변경함.
- 입력 : cv2 형식의 image와 기타 가중치들을 인자로 받도록 수정
- 출력 : 변환한 좌표값들을 이중 list 형태로 반환하도록 수정

CRAFT-pytorch 모델을 적용한 결과물은 다음과 같다.

- word 단위 detection

- character 단위 detection



>> CRAFT-pytorch를 이용한 text bounding 검출은 성공적이었다. 이렇게 추출한 bounding 영역에 해당하는 이미지를 CNN 모델에 학습시키고자 하였다.

6. CNN 모델에 텍스트 학습을 진행

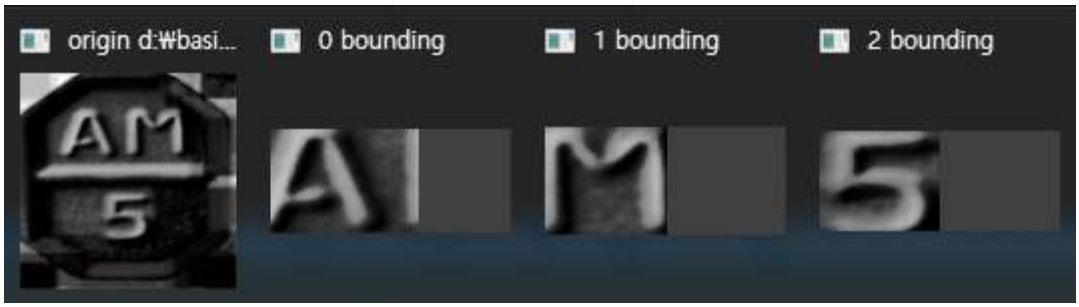
4. 에서 텍스트를 학습시키는 방법에 두 가지가 있다고 언급했다.

a. 텍스트 영역 전체를 추출하여 학습시키는 방법
b. 텍스트 문자 각각을 추출하여 해당 문자에 대해 학습시키는 방법

a. 를 적용한 결과물

- 순서대로 원본 이미지, 전처리 적용 이미지, text area 추출 이미지

b. 를 적용한 결과물



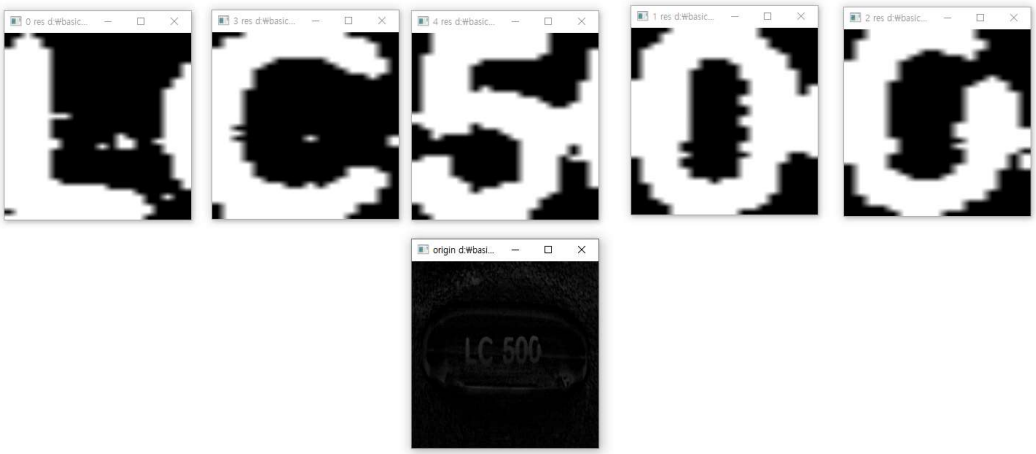
- 순서대로 전처리 적용 이미지, 문자 기준 text area 추출 이미지 (위치에 따른 순서 정렬 적용됨)

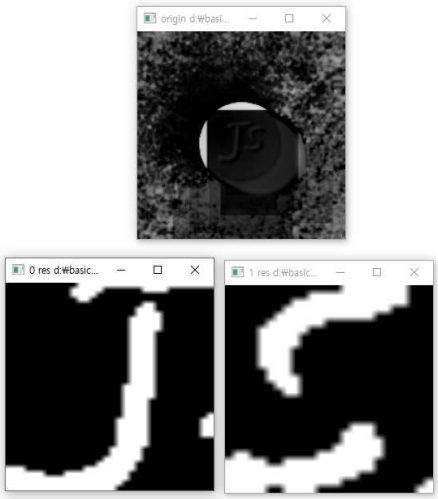
문제가 하나 있었는데, character 기준 검출의 경우 text area를 잘 추출하기는 하나, 검출된 문자들의 배열이 뒤죽박죽이었다. 예를 들어 라벨 데이터가 'AM5' 라면 'A', 'M', '5' 를 순서대로 검출해야 해당 문자들에 대해 학습을 시킬 수 있기 때문에 검출 결과를 정렬해야 했다. 따라서 bounding 좌표들의 x, y, w, h 값을 이용해 순서를 정렬하는 메서드를 구현 및 적용하였다.

정렬 메서드 구현 내용 : 기본적으로 위에 있는 문자를 앞 순번으로 인식하되, tolerance를 적용하여 y좌표가 tolerance 값 이하로 차이나면 같은 줄에 있는 것으로 간주한다.
ex) tolerance=10 이라면, y좌표값 차이가 10 이하인 문자들은 같은 줄에 있는 것으로 간주함.
같은 줄에 있는 문자는 왼쪽부터 앞 순번으로 인식한다.

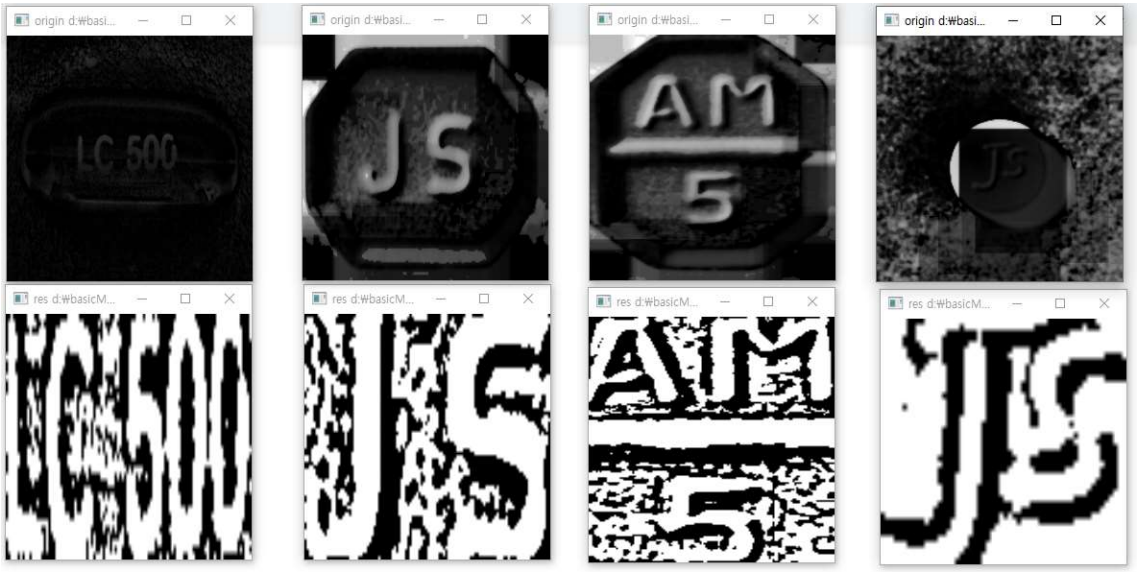
최종적으로 적용된 이미지 전처리 기법은 다음과 같다.
gray scale > gaussian blur > histogram equalization > black hat (51*51) > Craft-pytorch bounding 검출

여기에 이진화를 적용했을 때, 결과물은 다음과 같았다.
- bounding 적용된 image들에 대해 고정값 이진화 (threshold=17) 적용





- bounding 적용된 image들에 대해 adaptive threshold 적용

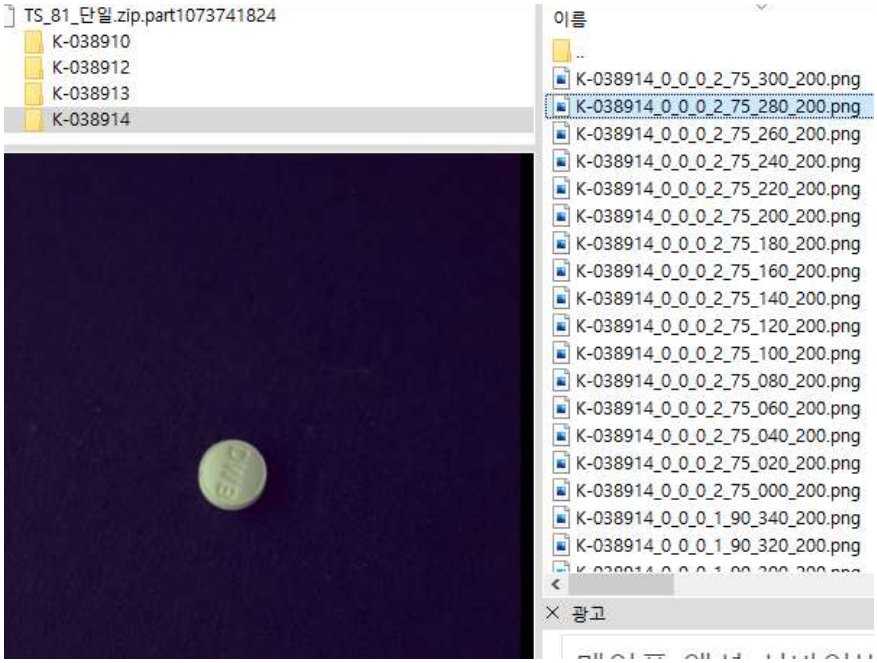


여기까지는 성공적이었으나, 간과한 또다른 문제가 있었다.

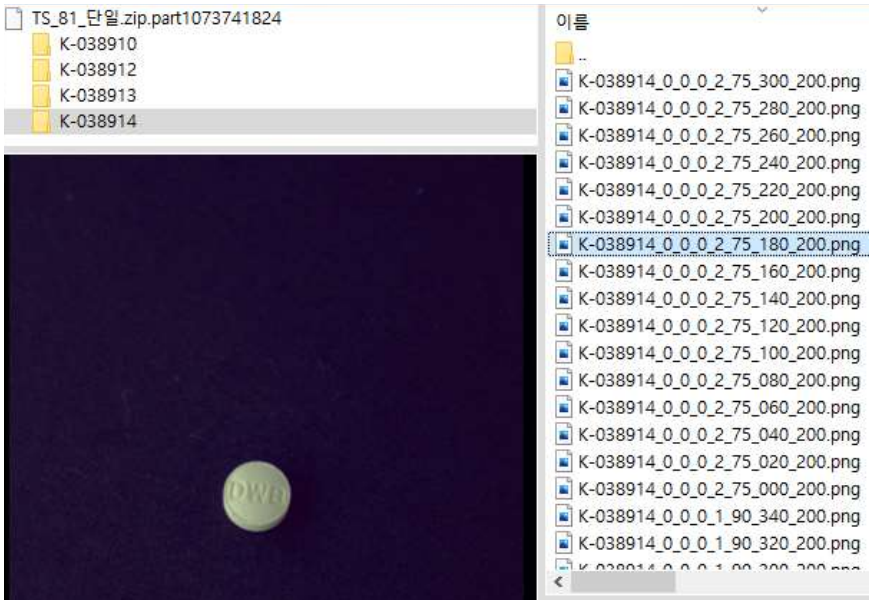
가지고 있는 데이터들 중 회전된 약제 데이터가 무작위로 존재하는데, 약제의 텍스트 또한 마찬가지로 회전되기 때문에 회전된 알약에 대해 CRAFT-pytorch의 텍스트 인식률이 떨어지는 문제가 있었다. 또한 90도, 180도, 270도 돌아간 텍스트에 대해서 텍스트 영역을 검출하더라도 해당 텍스트가 똑바로 되어 있는 것인지, 아니면 회전된 것인지 판단하기 힘들어 그대로 학습시키기에는 큰 지장이 있었다.
따라서 약제 데이터 중 회전되지 않은 알약 이미지만을 선별하는 작업을 수작업으로 진행하기로 하였고, 대략 3,000여장의 이미지를 선별하였다.
선별한 이미지들을 CRAFT-pytorch에 인식시켜 각각 문자 단위로 분리하고, 전처리를 거쳐 문자 학습을 위한 데이터를 생성하였고 총 4,500여장의 문자 학습 데이터를 수집할 수 있었다.
수집한 데이터들 중에서도 인식이 제대로 이루어지지 못한 이미지 등으로 인해 outlier가 존재했는데(10장 중 3~4장 꼴), 이 또한 수작업으로 라벨링을 진행하여 noise를 최대한 줄이고자 하였다.

1. 회전되지 않은 알약 이미지 선별

- 회전된 알약 이미지 (선별 대상 아님)

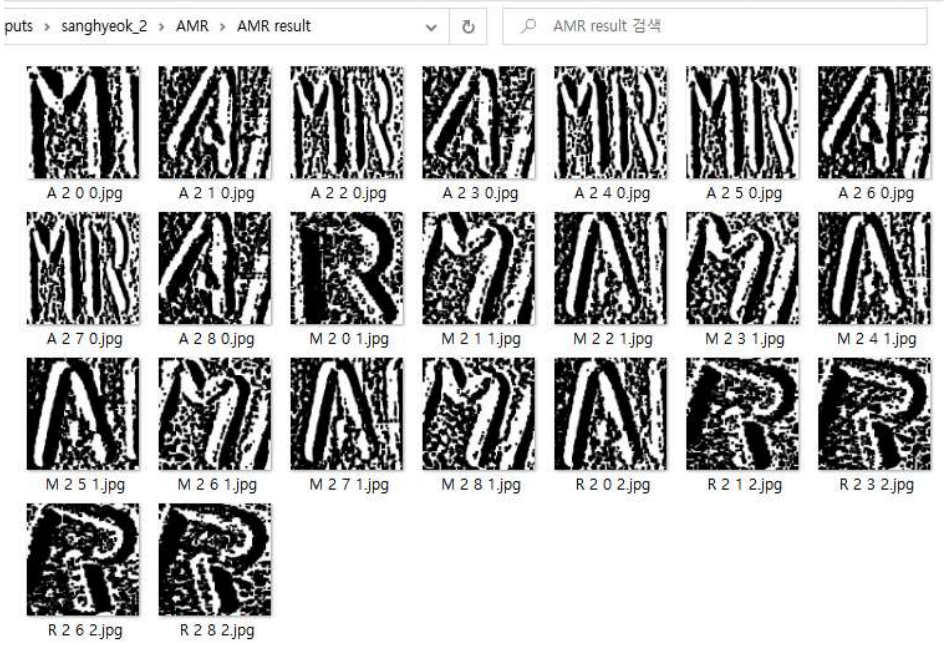


- 회전되지 않은 알약 이미지 (선별 대상)



2. 라벨링

- 라벨링(outlier 제거) 전 데이터의 모습



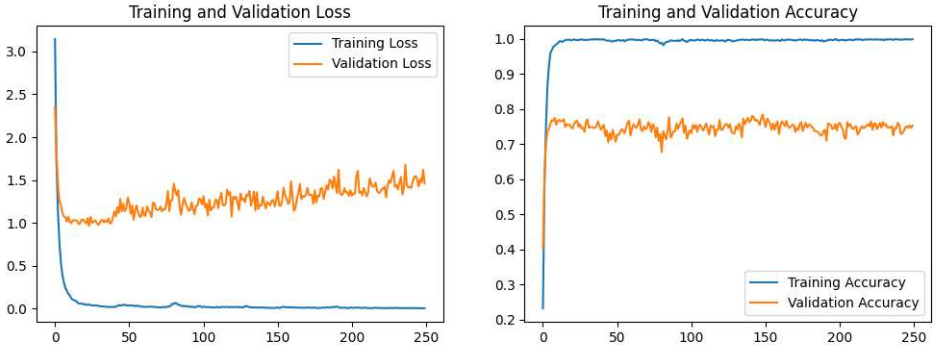
알아볼 수 없는 이미지, 두 개 이상의 문자가 포함된 이미지, 모호한 이미지를 제거한다.
라벨링이 잘못된 이미지에 대해 라벨을 수정해준다.

- 라벨링(outlier 제거) 후 데이터의 모습

- 모델 정보

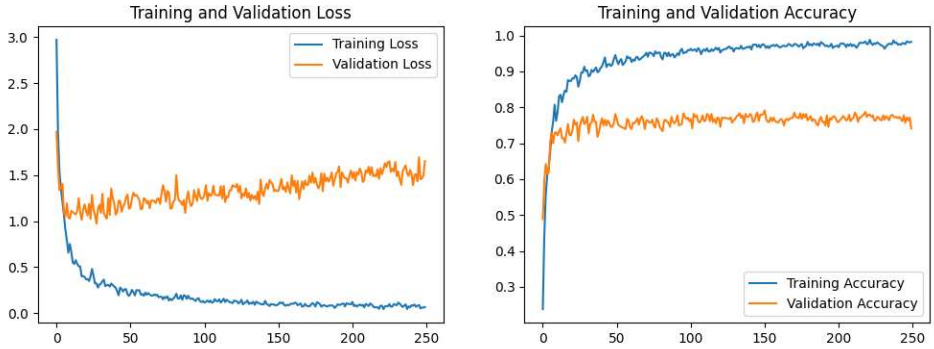
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(256, 256, 1)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.6),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

- 모델 학습 결과



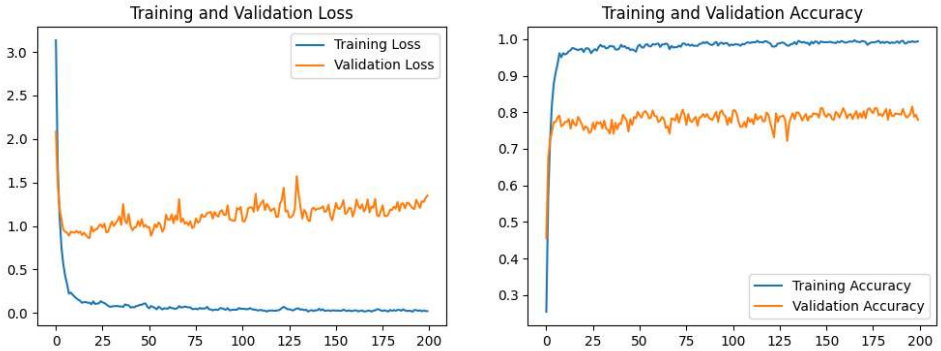
learning rate : 0.0004, epoch : 250, batch size : 32

> validation accuracy : 0.7535



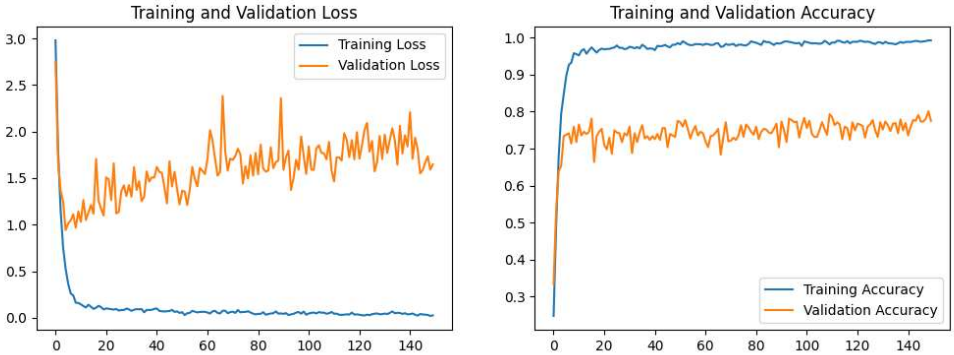
learning rate : 0.004, epoch : 250, batch size : 8

> validation accuracy : 0.7416



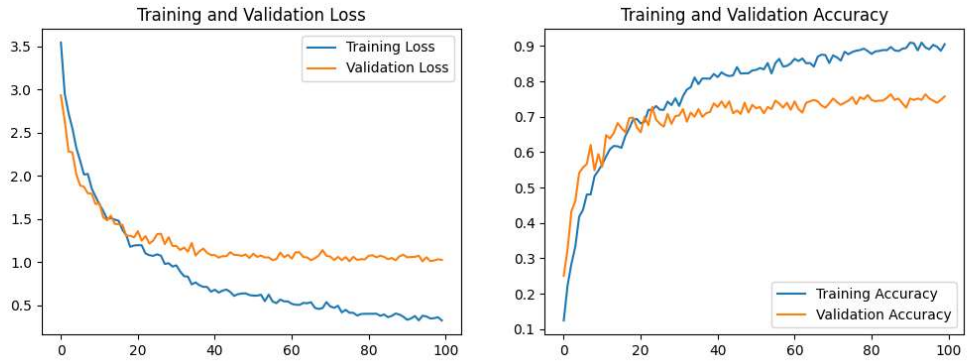
learning rate : 0.001, epoch : 200, batch size : 16

> validation accuracy : 0.7793



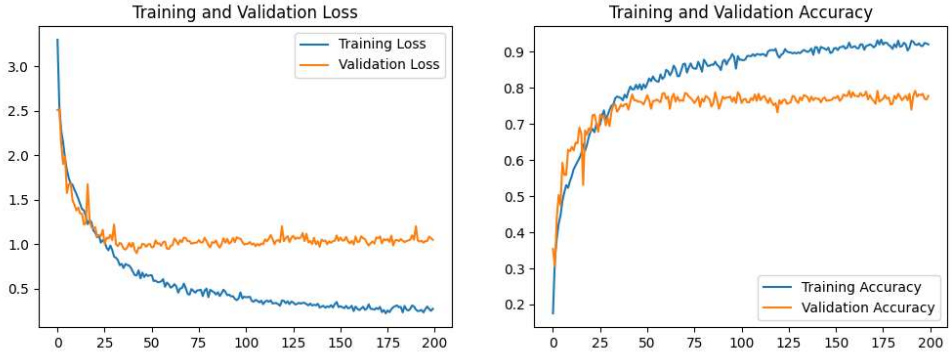
learning rate : 0.006, epoch : 150, batch size : 32

> validation accuracy : 0.7753



learning rate : 0.0001, epoch : 100, batch size : 4

> validation accuracy : 0.7525



learning rate : 0.0005, epoch : 200, batch size : 4

> validation accuracy : 0.773

대체로 0.75 정도로 비슷한 정확도를 보였다.
batch size가 작을수록, loss 값이 비교적 안정적으로 줄어드는 모습을 보였다.
텍스트 판별 모델로는 마지막 결과를 보여준 모델을 채택하기로 하였다.

실제 데이터로 test해 본 결과, 0.3 ~ 0.5 정도의 정확도를 보였다.