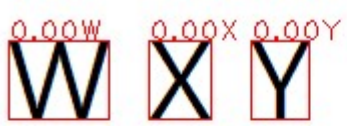
기말 프로젝트 중간보고

사진 속의 텍스트를 추출하는 프로그램 OCR의 중간보고입니다.

원래 기존의 계획은 cv2.findContours()함수와 cv2.matchShape()의 매칭점수를 이용하여 사진의 텍스트를 라이브러리와 비교하여 텍스트를 추출하려고 했습니다.

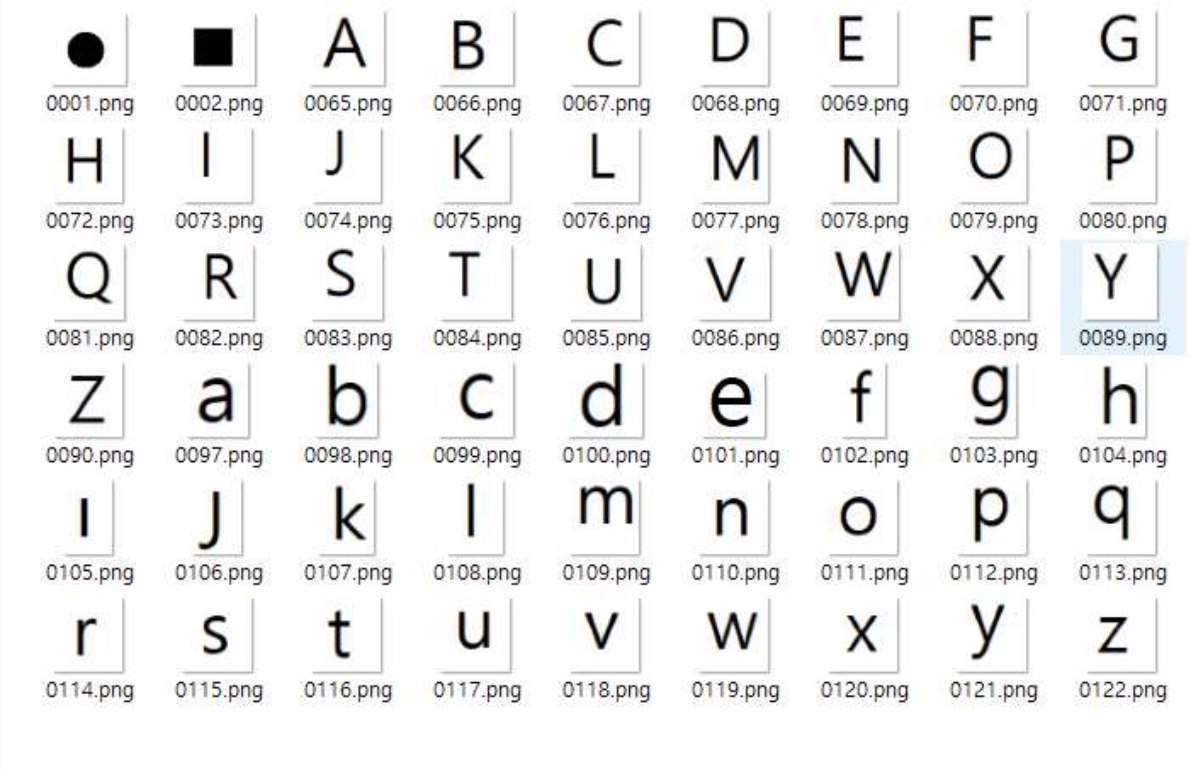
텍스트(글자 도형)의 구별을 위해서 네모상자를 씌어서 비교했습니다.

네모상자는 컨투어의 좌표를 이용해서 x좌표, y좌표의 최소, 최대를 이용해서 씌었습니다.



그리고 해당 네모를 관심영역을 지정해서 라이브러리와 비교를 진행했습니다.

(라이브러리는 아직 이정도 밖에 만들지 못했습니다. 이유는 글자를 컨투어 함수를 이용해서 비교하면 인식이 잘 안되기 때문에 비교 방법을 변경하는데 시간을 보냈습니다.)



라이브러리의 텍스트 그림 파일도 마찬가지로 컨투어의 좌표를 이용하여 roi관심영역을 지정해서 도형의 필요한 부분만 이용해서 비교하도록 했습니다.

비교하는 방법을 바꾼 두번째 이유는 한글의 경우 자음 모음 받침을 나눠서 인식했다 하더라도 문법을 구현하는데 소요되는 시간이 많을 것 같기 때문입니다. 또 소문자 i 와 j 는 위 아래 도형 2개로 되어있는데 이 문제를 해결하고자 방법을 변경했습니다.

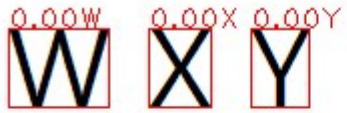
변경하게 되면 단점이 라이브러리를 아래 같이 많이 만들어야 하는 점이 있지만 인식률이 높아질 것 같습니다

원래 중간보고때 글자들과 라이브러리를 만들고

이후 최종에 글자구분, 글의 순서 등 사진을 입력했을 때 텍스트가 출력되게 만드는 거였습니다.

현재 완성된 것은 변경된 비교 방법, i나 j같이 떨어진 문자를 한 구역으로 구별하는 법, 한글 같은 경우 자음, 모음, 받침, 떨어진 글자들을 한 글자로 묶는 방법까지 구현했습니다.

1> 변경된 비교 방법입니다.



기존 컨투어를 사용해 비교하는 방법은 글자를 늘리거나, 회전, 대칭이동 등을 고려하지 않기 때문에 인식률이 많이 저하됩니다.

변경된 방법입니다.

컨투어의 특성상 맨 밑의 도형부터 잡힙니다.

살짝 찌그러진 소문자 e를 예시로 들었습니다.

일단 e를 흑백 이미지로 변경 후에 이진화를 진행했습니다.

그리고 라이브러리를 하나씩 비교합니다.

두개의 크기를 같게 설정해두기 때문에 찌그러진 문자도 인식됩니다.

이진화된 이미지와 라이브러리를 bitwise_xor 연산을 수행하면 일치하지 않는 부분에 비례해서 흰색의 이미지가 출력됩니다.

결과 이미지에서 np.sum()을 이용해서 픽셀값을 다 더한다음에 결과이미지의 크기로 나뉘면

흰색이 차지하는 비율이 나옵니다. 값이 작을수록 라이브러리와 유사합니다.

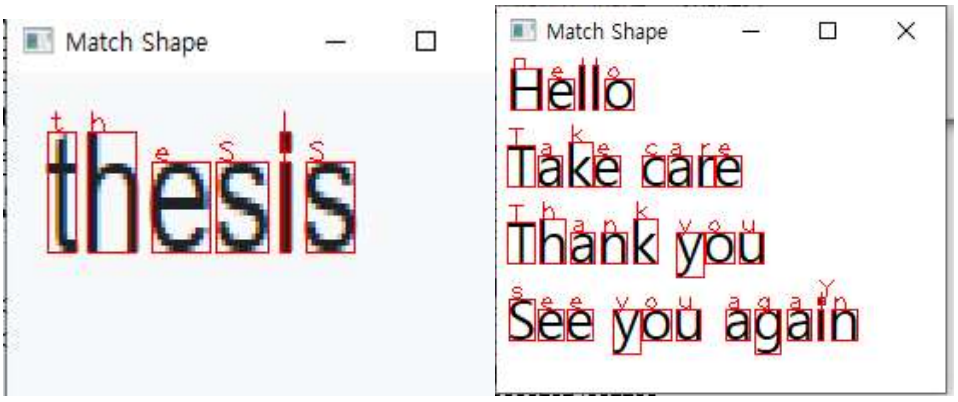
제일 유사한 소문자 e는 거의 검정색입니다.

유사성을 결정하는 점수는 아래와 같습니다.

point = white_ratio(흰색 비율) + aspect_ratio(가로 세로 비율차이)

aspect_ratio은 알파벳은 모두 가로 세로 비율이 다르기 때문에 특히 소문자 j와 대문자J의 인식률을 높이기 위해서 추가했습니다.

예제의 결과입니다. 다른 그림파일의 결과입니다.



다른 문자는 잘 인식되는데, 소문자 i와 j의 윗부분이 따로 놀기 때문에 인식이 잘 안되어서 또 방법을 변경하기로 했습니다.

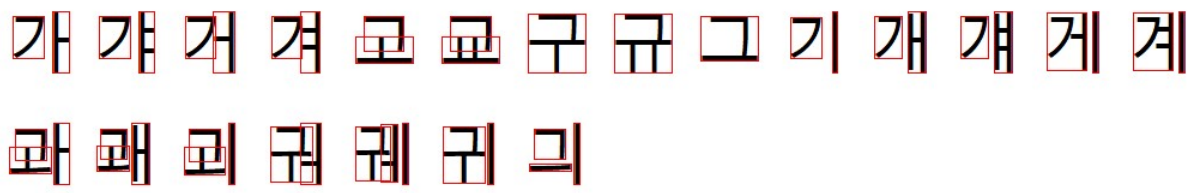
그림과 같이 상단부분과 하단부분을 동시에 관심영역을 지정하게 구현했습니다.

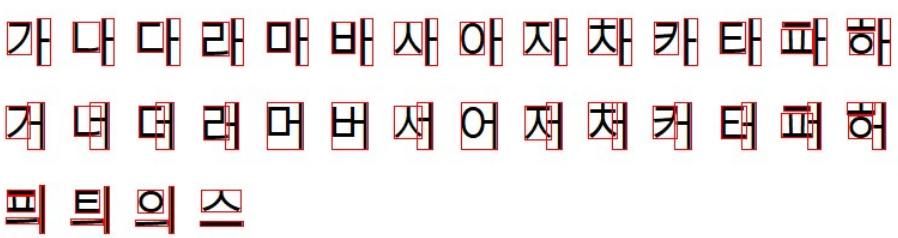
자음과 모음이 붙어있는 한글 같은 경우 어떻게 나눌까 고민해 보았는데 해답을 찾지 못했는데

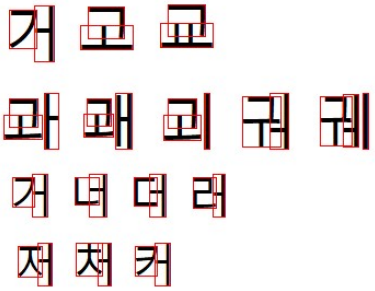
이런 방법을 사용하면 같은 경우에도 나누지 않고 한글자로 모은 다음 처리하기 때문에 이 문제 또한 해결되었습니다.

문자를 붙이는 단계를 3단계로 나누어서 구현했습니다.

처음관심영역이 지정된 모형들의 영역이 겹쳐 있으면 한 글자안에 포함되는 것을 나타냅니다.







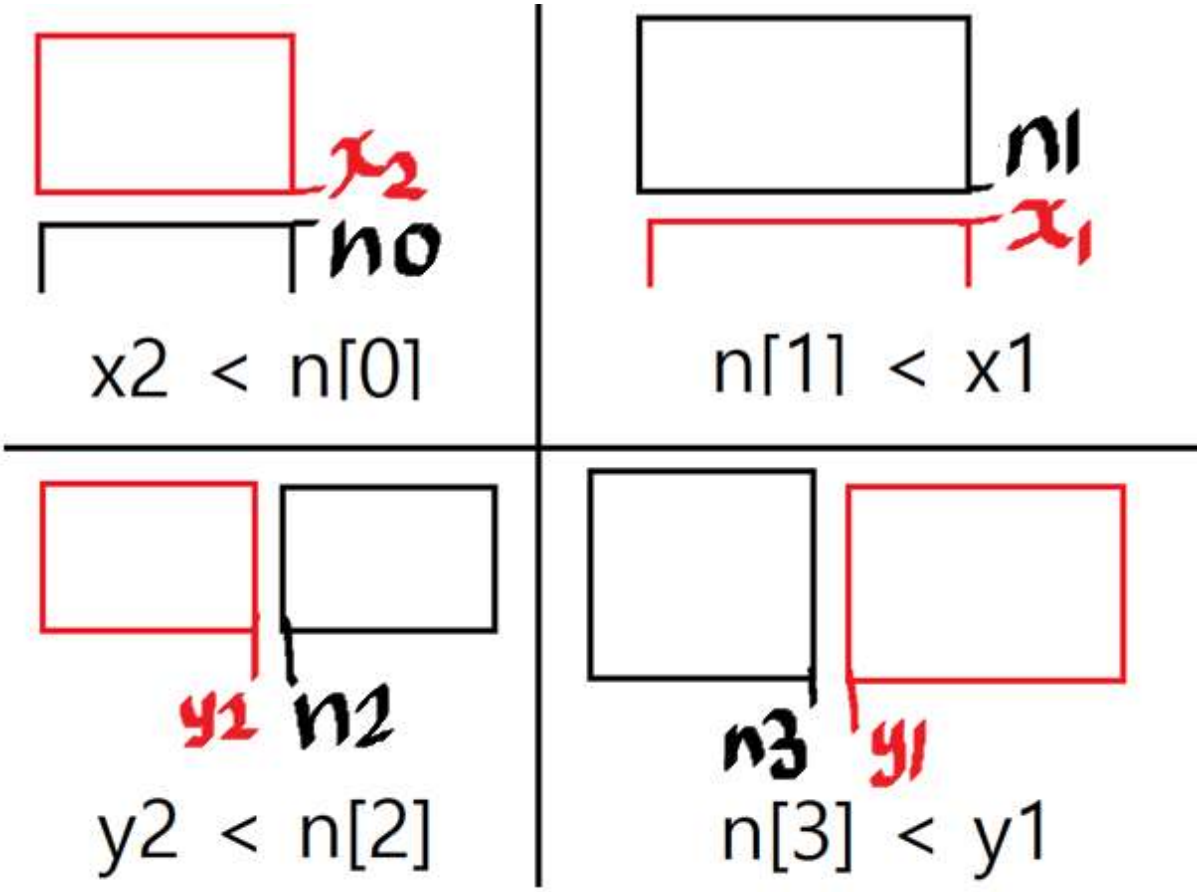
위 글자들은 관심영역이 겹쳐 있는 글자들 입니다

2개의 관심영역의 좌표를 표현할 때 x좌표,y좌표 최소최대를 이용해서 나타냈습니다.

관심영역1(x1,x2,y1,y2) **비교할 관심영역(n[0],n[1],n[2],n[3])**

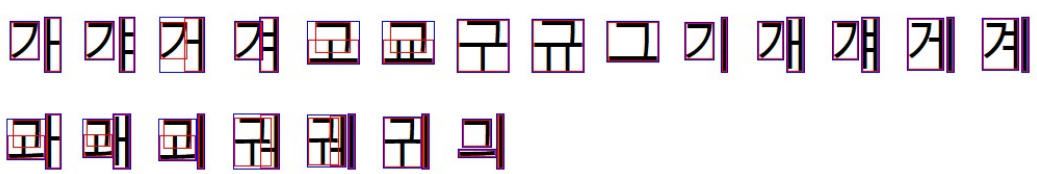
If 조건문을 사용해서 구별했는데 하나도 겹치지 않을 때의 조건을 구해서 부정을 이용해서 조건을 만들었습니다.

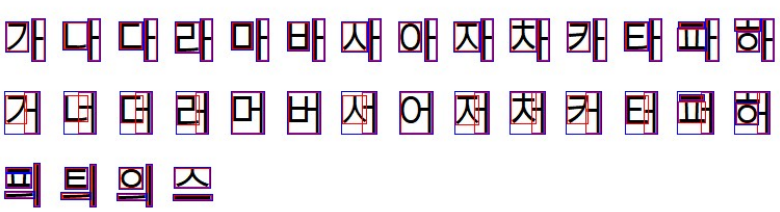
if ((x2 < n[0]) | (x1 > n[1]) | (y2 < n[2]) | (y1 > n[3])) == False:



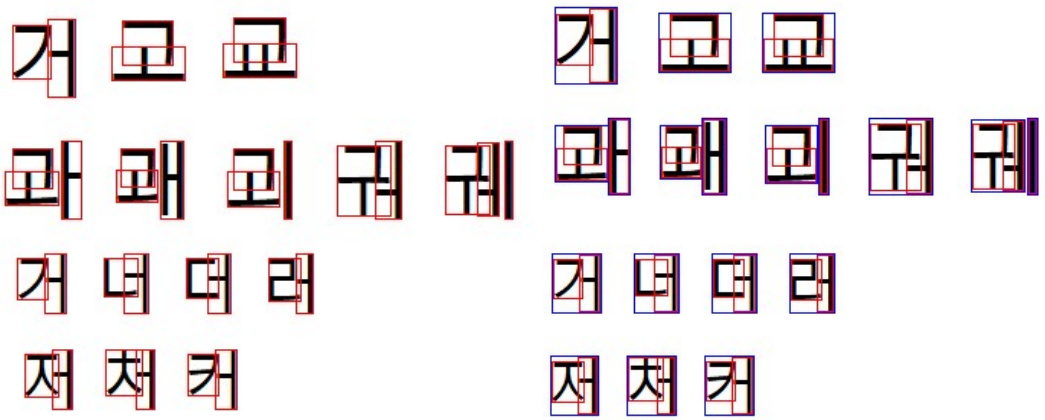
조건을 적용시키고 글자를 합치서 관심영역을 다시 지정했습니다

(min(n[0],x1), max(n[1],x2), min(n[2],y1), max(n[3],y2))





파란색은 새로 만들어진 관심영역의 범위입니다. 결과입니다

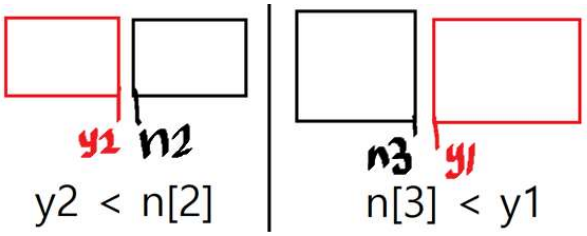


모음이 여러 개로 구성되어 있는 것을 제외하곤 해결되었습니다.

다음은 ㅍ, ㅎ 처럼 떨어져 있는 자음과 그 자음과 ㅡ 같은 모음과 받침을 한 문자로 묶어주도록 하겠습니다.

if (y2 < n[2]) | (y1 > n[3]) == False: / if n[0]-x2 <= space_size_x:

두 조건을 중첩해서 사용했습니다. 붙이려는 모형들은 가로선상에 좌표가 겹치는 부분이 있어야 되기 때문에 겹치지 않을 때의 부정조건을 사용했고,

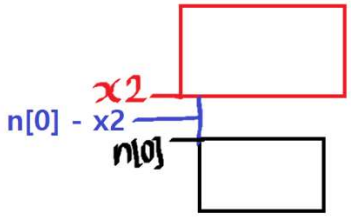


if n[0]-x2 <= space_size_x: 에서

길이의 제한을 두지 않으면 모든 가로선상 겹치는 모든 그림들이 합쳐져서 최대 공백의 크기를

나타내는 변수space_size_x을 사용했습니다.

컨투어의 특성상 findContours 함수를 사용하면 그림의 아래쪽의 오른쪽부터 생성됩니다. 나중에 생성된 그림이 위쪽으로 두 그림 사이의 위치 차이는 기존 그림의 제일 큰 세로 위치 값n[0]에서 현재 제일 작은 세로 값x2를 뺀 n[0] – x2가 됩니다.



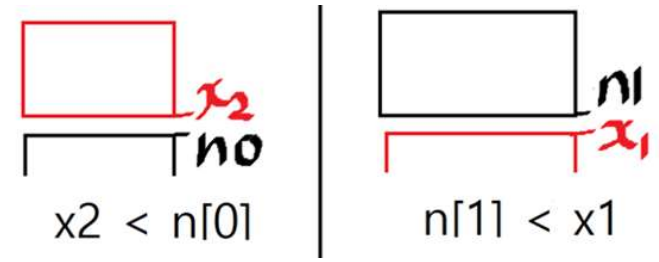
조건의 실행 결과는 다음과 같습니다.

다음은 좌우 떨어진 글자를 합치는 조건식입니다.

if (x2 < n[0]) | (x1 > n[1]) == False:

if y1-n[3] <= space_size_y and n[2]-y2 <= space_size_y:

마찬가지로 중첩으로 사용했습니다.

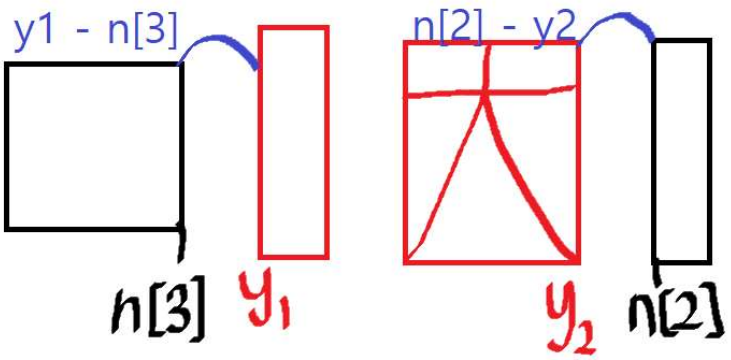


마찬가지로 같은 세로선상에 있어야 되므로 겹치지 않을 때의 부정조건을 사용했고,

if y1-n[3] <= space_size_y and n[2]-y2 <= space_size_y:

좌우 합치는 건 예외 케이스가 있어서 조건이 두 개입니다. 보통 findContours 함수를 사용하면

자음에 ㅏ ㅔ ㅣ ㅑ 같은 모음들 붙을 때 세로의 시작 좌표가 더 큰 자음들이 먼저 함수를 통과하는데 자음의 ㅊ 같은 경우는 상단부분이 길어서 세로의 시작 좌표가 작아서 나중에 통과하게 됩니다.



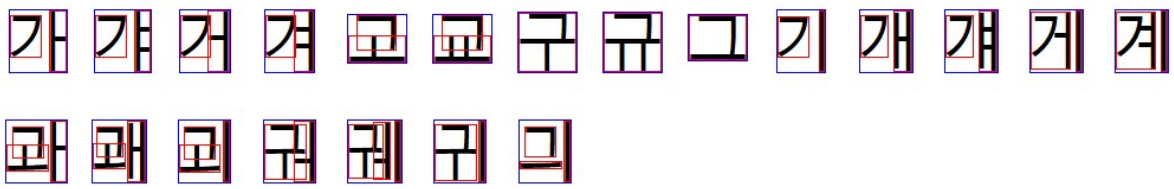
여기서 수식 사이에 and 인이유는

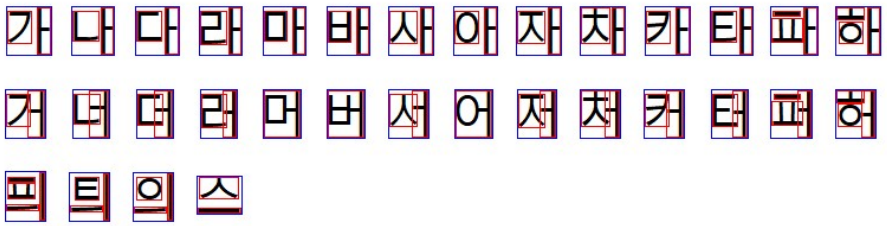
or로 하게 되면 수식에서 세로 공백의 최대 크기를 나타내는 변수 **space_size_y**가 양수이기 때문에 수식 음수인 경우 거리의 제한이 사라지기 때문에 모든 도형이 하나로 합쳐진다.

또 앞에 수식이 참인 경우, 뒤의 수식이 음수가 나오므로 뒤에 수식도 자동으로 참이다.

반대의 경우도 마찬가지다.

세 조건을 적용한 결과이다.





이후에는 라이브러리 제작과, 라이브러리를 비교할 때 bitwise_xor 연산할 때에도 한 글자씩 할 수 있도록 세 조건을 적용시키고 글자의 순서를 정하는 법을 구현해서 최종 제출하겠습니다.