서명 인식

Offline Signature Recognition

REPORT 7

20161856 한지효

2020-01 생체인식 및 응용

프로그램 설명

본인이 개발한 서명 인식 프로그램은 Shubham-maurya 가 Github 에 게시한 ocsvm-signature-verification(https://github.com/shubham-maurya/ocsvm-signature-verification)의 이미지 전처리와 특징 추출 방법을 이용하여 서명 이미지를 입력 받아 그 서명인이 누구인지 검출하여 결과를 출력하는 것을 목적으로 한다. 이용한 데이터베이스는 Group C 의 Level 1, 2, 3 이다.

프로그램의 진행과정은 [document] 폴더 내 **(참조)OfflineSignature.pdf** 파일에서 볼 수 있으며, GUI 를 적용한 코드는 [code] 폴더 내 OfflineSignature_GUI.py 이다.

1. 진행과정 플로우 차트

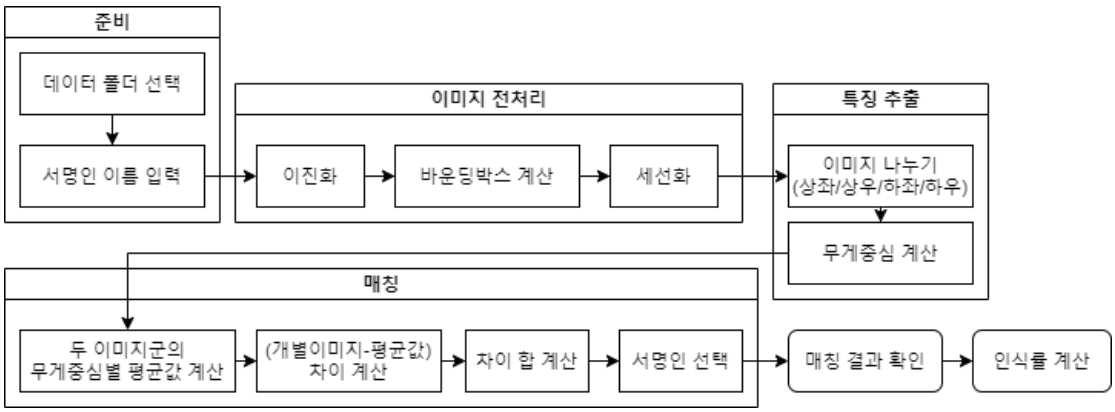


그림 1 프로그램 플로우차트

이미지 전처리, 특징 추출, 매칭에 대한 상세한 설명은 아래에서 설명하도록 한다.

2. 이미지 전처리 과정

1) 이진화 (Binarization)

입력 이미지에서 노이즈를 제거하기 위해 가우시안 블러 필터링(Gaussian Blurring)을 적용한다. 이후 서명 정보와 배경을 분리하기 위해 Otsu 알고리즘을 사용하는데, 이는 원본 이미지의 밝기 분포를 보고 그레이값의 문턱값을 설정하는 방법이다.

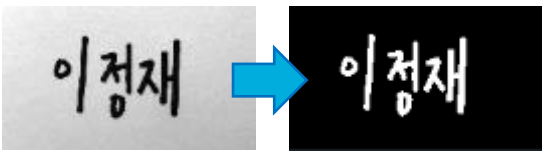


그림 2 이진화(L181.bmp)

2) 바운딩 박스(Bounding box) 계산

이진화 된 이미지에서 유효한 서명의 픽셀만을 구분하기 위해 바운딩 박스를 계산하고 해당 부분만 남겨지도록 크롭(crop)한다.

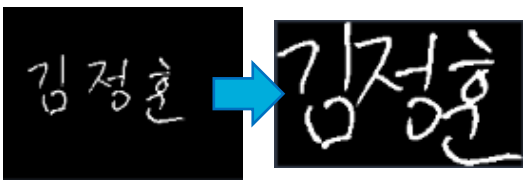


그림 3 바운딩 박스(L121.bmp)

3) 세선화(Thinning)

위에서 크롭된 이미지로부터 서명의 골격을 추출하기 위해 OpenCV 라이브러리를 활용하여 세선화 한다.

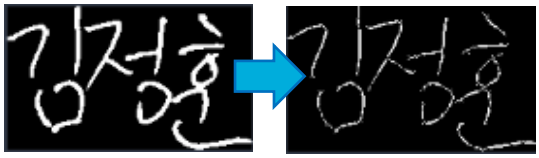


그림 4 세선화(L121.bmp)

3. 특징 추출

전처리된 이미지를 상좌, 상우, 하좌, 하우 네 부분으로 나눠 각각의 무게중심(COG, Centre of Gravity)를 계산한다. 이를 통해 한 이미지당 총 8 개의 좌표(x, y)를 산출해낼 수 있다.

	File	tl_x	tl_y	tr_x	tr_y	bl_x	bl_y	br_x	br_y
0	L121-397x281.bmp	0.565399	0.520333	0.468384	0.347997	0.424293	0.572816	0.470073	0.495879
1	L122-468x265.bmp	0.599389	0.480184	0.465375	0.416656	0.438907	0.530944	0.453929	0.511840
2	L123-464x244.bmp	0.542735	0.539859	0.525118	0.360648	0.432292	0.544648	0.476756	0.498136
3	L124-1500X750.bmp	0.476090	0.483705	0.473150	0.542479	0.475593	0.597720	0.443795	0.471088
4	L125-980X765.bmp	0.545577	0.525698	0.448222	0.536954	0.456920	0.517417	0.471154	0.487526

그림 5 좌-이미지 분할(L121.bmp), 우-특징 추출 결과

4. 매칭

본 프로젝트에서는 한 번의 차시에 두 서명인의 엇비슷한 개수의 이미지만을 비교하기 때문에 각 이미지군의 좌표 별 평균값을 계산해낼 수 있다. 다시 말해, 한 디렉토리에 존재하는 파일의 개수를 반으로 나누었을 때 한쪽은 서명인 A, 다른 한쪽은 서명인 B 의 서명 이미지파일이라고 생각할 수 있다. 이는 사용자의 입력 없이는 미리 각 이미지들의 서명인 이름을 알 수 없어 매칭 이전에 가정한 것이다. 물론 프로그래머가 파일명의 구분을 통해 직접 범위를 설정해 놓을 수도 있지만, 최대한 기계가 자율적으로 판단할 수 있도록 알고리즘을 설계하였다. 해당 사항은 아래 [6. 개선사항]에서 더 자세히 설명하도록 한다.

두 이미지군에 있는 개별 이미지들을 통해 좌표 별 평균값을 계산하였다면 각 좌표별로 A 의 평균값과 B 평균값을 산출할 수 있다. 이를 이용해 다시 각 개별 이미지와의 차이를 계산하면 해당 서명 이미지가 A 와 가까운지, B 와 가까운지 판단할 수 있다.

위 과정을 좌표별로 거친 후에는 해당 이미지가 A 인지 B 인지 판단한 결과값이 8 개 나오게 된다. 이들의 평균값을 이용해 최종적으로 서명인을 선택하게 된다.

ex)
상좌 부분에 해당하는 무게중심의 x 좌표값 평균이 A 는 0.4 이고 B 는 0.6 이라면 해당 좌표값이 0.3 인 서명이미지 1 은 A 에 의해 서명되었다고 추측할 수 있다. 같은 방식으로 서명이미지 1 에 대한 다른 좌표들을 계산하여 산출된 결과가 A, A, A, B, B, A, A 라고 하면 서명이미지 1 은 서명인 A 가 작성한 것으로 판단한다.

매칭에 대한 결과는 아래 [결과 > 2. 특징 구분, 3. 매칭]부분에서 보이도록 한다.

5. 오픈 소프트웨어 및 본인 구현 부분 설명

오픈 소프트웨어에서 가져온 부분은 이미지 전처리 방법과 특징 추출의 일부분이다. 오픈 소프트웨어에서는 위에서 설명한 특징 추출 방법에서 더 나아가 각 좌표별로 역 탄젠트 값을 구하고 이를 벡터화 하여 서명 이미지의 밀도를 구하였으나, 이는 위조서명 판단 개념과 더 맞물려 있는 것 같아 해당 부분은 생략하였다. 생략된 특징 추출 방법을 통해 각 서명인의 평균값을 구하고 차이를 계산하는 방식은 이전 수업에서 배운 홍채 인식 알고리즘 내 해밍 거리 측정 방법에서 착안하였다.

이외에도 프로그램의 구동을 쉬이 하기 위한 GUI 의 개발은 tkinter 라이브러리를 이용하여 본인이 하였다. GUI 가 적용된 프로그램 파일을 실행하면 데이터 폴더를 지정하고 서명인의 이름을 입력하게 되는데, 이미지 처리와 매칭 과정을 시각화하기 위해 이진화, 바운딩박스 처리, 세선화 작업을 일정 시간을 두어 보여준다.

프로그램에서 서명인 판단이 끝난 뒤 각 이미지와 매칭된 서명인이 화면에 표시되고 나면 사용자가 직접 인식률 계산 단계에 참여해야 한다. 이는 매칭 단계에서 프로그램이 임의로 서명 파일을 두 집단으로 나누었기 때문인데, 정확한 결과를 알기 위해서는 사용자가 직접 결과를 보고 몇 개의 판단이 옳았는 지 입력해야 한다. 이후 폴더 내 총 이미지 개수와 사용자가 옳다고 판단한 이미지의 수를 비교하여 정확도를 인식률로 표기한다.

6. 개선 사항

1) 특징 추출

이미지를 상좌, 상우, 하좌, 하우 네 부분으로 나누었을 때 계산할 수 있는 무게중심의 좌표는 다시 x, y 로 나누어 좀 더 많은 특징 정보들을 이용할 수 있도록 설정하였는데, 이는 해당 정보만으로도 서명인을 구분하기에 충분하다고 판단되었기 때문이다. 그러나 오픈 소프트웨어에서 그러했 듯이 서명 픽셀의 밀도 등 다른 정보들을 복합적으로 이용하였으면 완벽에 가까운 인식률을 보였을 것이란 아쉬움이 남는다.

2) 서명인 이름 검출

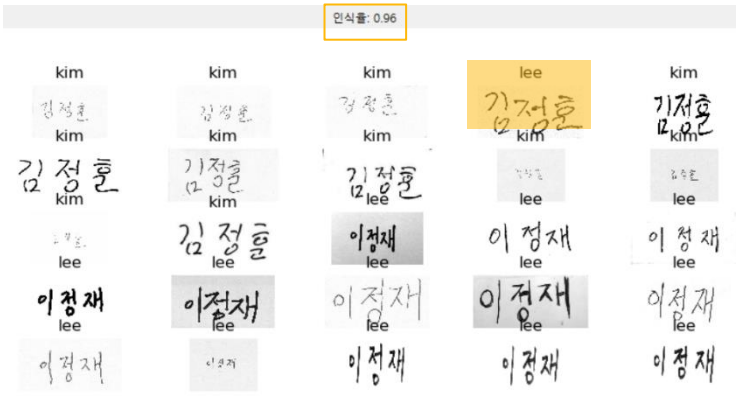
위 [프로그램 설명>5. 오픈소프트웨어 및 본인 구현 부분 설명]에서 언급하였다 싶이 인식률 계산에는 사용자가 개입을 해야 한다. 프로그램 단에서 각 이미지가 누구의 것인지 미리 알지 못하기 때문인데, 이러한 문제는 설계 단계에서 이미지별 라벨링은 진짜 서명과 위조 서명을 구분하는 데에만 필요하다고 생각했기 때문이다. 그러나 보고서 작성 도중, TensorFlow 를 이용해 한글 손글씨를 인식(https://developer.ibm.com/kr/journey/create-a-mobile-handwritten-hangul-translation-app/)할 수 있는 것을 알게 되었다. 현재 본인이 작성한 프로그램은 단순히 이미지의 무게중심 값만을 이용하여 사용자가 후에 입력한 서명인의 이름과 비교를 하지만, 이러한 머신러닝 기술을 적용하여 미리 서명인이 누구인지 판단한 뒤 직접 개발한 알고리즘과 비교한다면 더 개선된 결과를 얻을 수 있을 것이다.

결과

1. 인식률

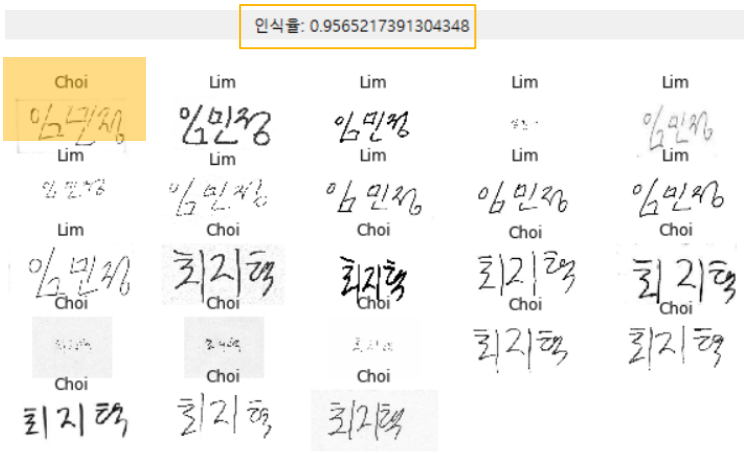
인식률은 사용자판단 대비 기계판단의 정확도를 표기한 것이다.

1) Level 1



인식률: 0.960 (24/25)

2) Level 2



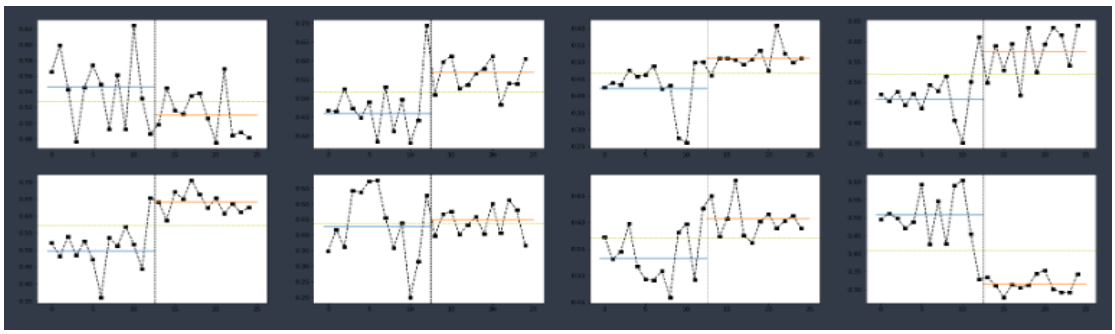
인식률: 0.957 (22/23)

3) Level 3

인식률: 0.834 (20/24)

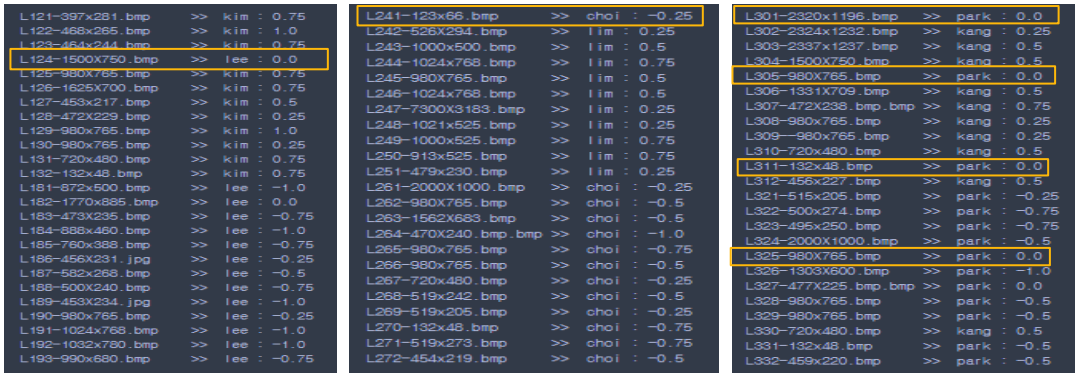
총 인식률: 0.917 (66/72)

2. 특징 구분

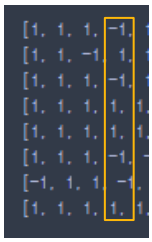


위 그래프는 예시로 Level 1 의 이미지들에 대해 차례대로 상좌 x, 상우 x, 하좌 x, 하우 x, 상좌 y, 상우 y, 하좌 y, 하우 y 값의 분포를 나타낸 것이다. 청색선은 서명인 A 의 평균값이며, 적색선은 서명인 B 의 평균값이고 황색선은 전체 평균값이다.

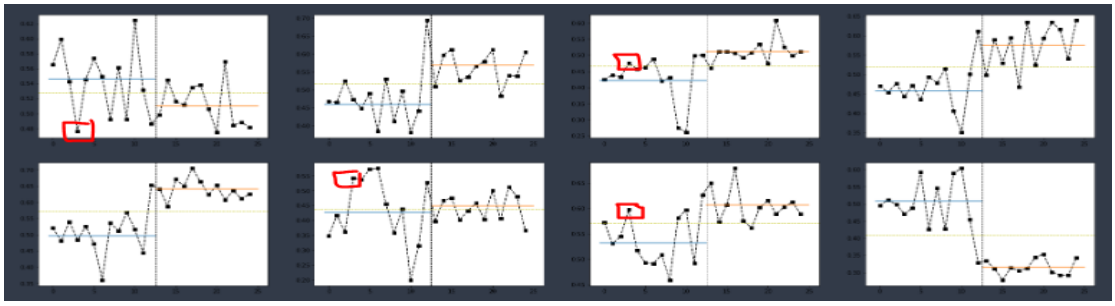
3. 특이 케이스



하나의 이미지의 각 좌표값에 대해 A 라고 판단했을 경우 1, B 라고 판단했을 경우 -1 이라고 설정한 후 총 8 개 값에 대해 평균값을 낸 결과는 레벨별로 위와 같다.



위 Level 1 인식률에서 잘못 판단된 이미지인 L124 파일을 분석하였을 때, 각 좌표값들의 결과는 좌측과 같다. 이는 상좌 x, 하좌 x, 상우 y, 하좌 y 값이 서명인 B 와 더 근접하단 뜻이며, 이는 아래와 같이 그래프에서도 확연히 드러남을 알 수 있다. 이처럼 총 계산값이 0 으로 나올 경우 프로그램 상에서 B 라고 판단하여 이와 같은 경우가 발생하나, A 라고 판단하게 해도 결과는 비슷하다.



평균값을 근거로 하였기에 원본 서명 이미지 파일을 직접 비교할 수는 없지만, 이렇게 종합된 결과를 통해 프로그램이 서명인을 판단함을 알 수 있다.

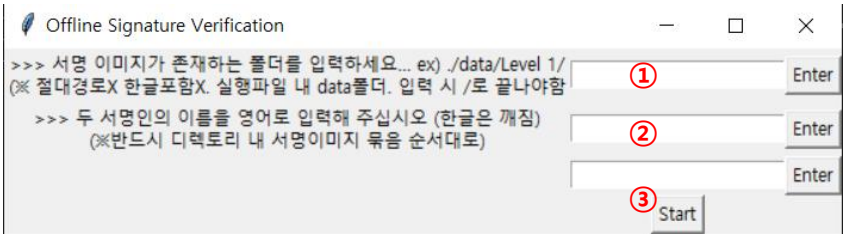
사용자 매뉴얼

프로그램 실행은 아래 두 방법 중 하나를 선택해서 실행할 수 있다.

- [code/dist] 폴더 내 OfflineSignature_GUI.exe 파일 실행: 로딩까지 2 분 남짓 소요
- 명령 프롬프트(cmd) 이용: OfflineSignature_GUI.py 파일이 있는 폴더로 이동 후
  >> python OfflineSignature_GUI.py 명령어 실행

  ※ 이 경우 사용자 PC 에 Python3 포함 여타 라이브러리가 설치되어 있어야 함 (오류 시 콘솔 로그 참조 후 pip install 명령어 등 사용해 모듈 설치)

이때 데이터 폴더는 실행 파일(.exe 혹은 .py)과 같은 폴더 내에 존재해야 하며, 정확한 인식률 계산을 위해 두 서명인의 서명 이미지들의 개수는 같거나 엇비슷해야 한다.

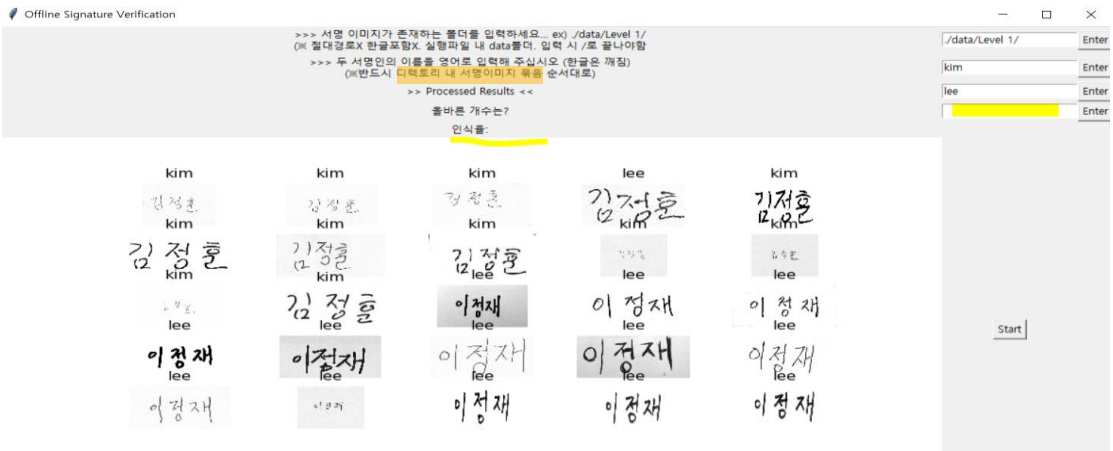


① 비교할 서명 이미지들이 있는 폴더를 반드시 **./data/Level 1/** 과 같은 형식으로 입력한다(절대경로 x, 폴더명 내 한글포함 x). 입력 완료 시 Enter 클릭.

② 서명 이미지를 작성한 두 서명인의 이름을 입력한다. 입력 완료 시 Enter 클릭.

③ 위 항목 모두 문제없이 작성을 마치면 Start 버튼을 클릭한다.
(만일 폴더 내 파일명에 이상이 있을 경우 이미지를 읽어오지 못하므로 콘솔 로그 참조 후 오류가 발생한 파일명을 변경 후 다시 시도한다.)

다음과 같이 '>> Processed Results <<' 메시지 출력 화면이 나오면 비교가 끝난 것이다. (아직 실행 중일 경우 '..Processing..' 메시지가 출력됨)



화면에 표시된 각각의 서명 이미지와 해당 이미지 위에 프로그램이 판단한 서명인의 매치를 보고, 올바르게 서명인을 판단한 경우의 개수를 세어 해당란에 입력후 Enter 버튼을 클릭하면 왼편에 인식률이 계산되어 표시된다.