레포트 7 : 서명인식

과목	생체인식 및 응용
제출 일자	2020년 05월 29일
담당 교수	권영빈 교수님
학 과	컴퓨터공학과
학 번	20172129
이 름	박예솔

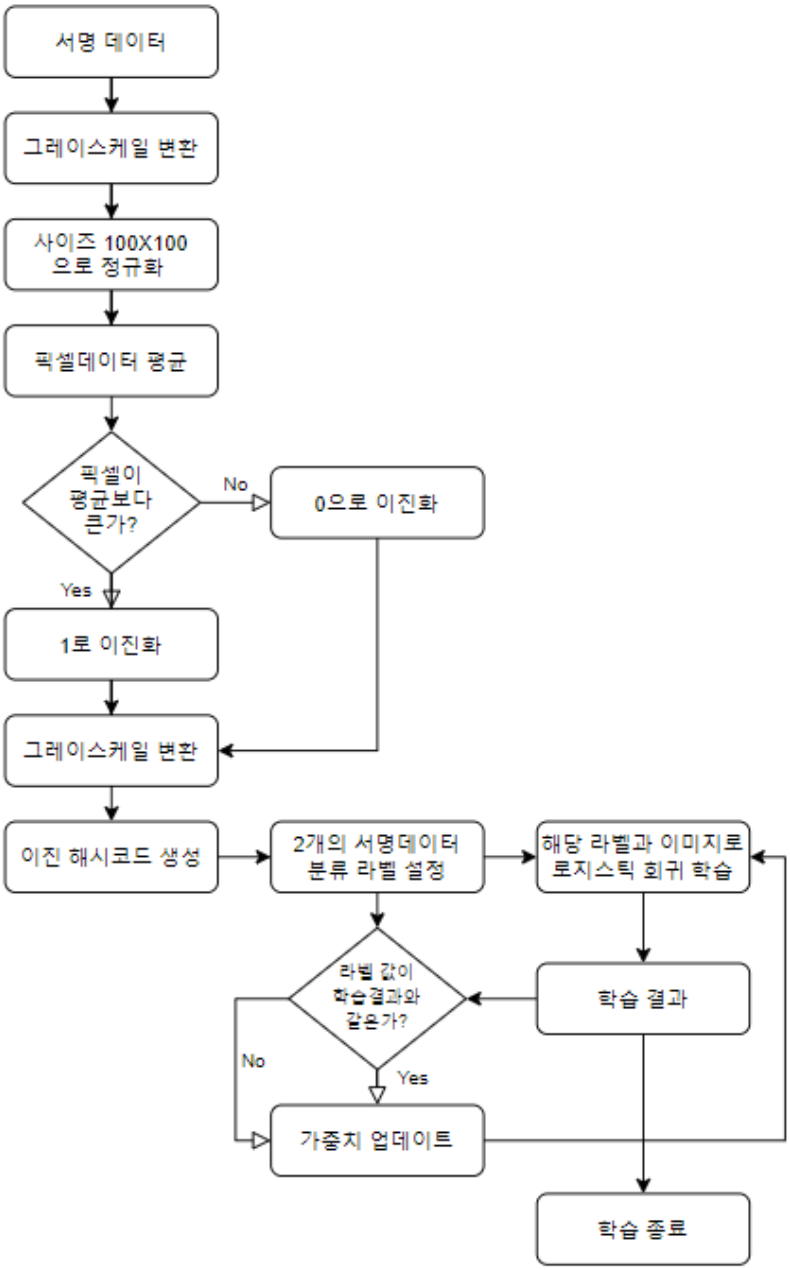
Ⅰ. 프로그램의 개요

1. 구현 언어 및 실행프로그램:

- Python, 구글 Colab

- <**Level 3**> 까지 구현

2. flow chart



3. 오픈 소프트웨어 URL / 참고 코드

* 파이썬을 이용한 머신러닝, 딥러닝 실전 개발 입문/예제 코드/ch7

  https://wikibook.co.kr/python-machine-learning/

* 다층 퍼셉트론 구현

  https://m.blog.naver.com/PostView.nhn?blogId=samsjang&logNo=221037822576&proxyReferer=https:%2F%2Fwww.google.com%2F

4. 본인이 작성한 연결 프로그램 설명

인식 과정을 자세히 설명하기에 앞서 간략한 프로그램 구현 방향에 대해 설명하려고 한다. 우선 수업 시간에 홍채인식 과정에서 이진 홍채코드를 만든 것에서 아이디어를 얻어왔다. 각 레벨 별 폴더에서 이미지 데이터를 가져와 우선 그레이 스케일로 변환하고, 픽셀 데이터를 가져와 이진 해시로 변환하였다. 이 과정은 앞서 올려놓은 '파이썬을 이용한 머신러닝, 딥러닝 실전 개발' 책의 예제코드를 사용하여 진행하였다. 책에서는 후에 해밍거리를 계산해서 매칭을 시켰는데, 실제로 해보니 정확도가 아주 낮아서 사용할 수가 없었다. 이에 대해 고민하다가 최근 머신러닝 수업에서 배웠던 숫자 인식(Mnist) CNN 학습 방법이 생각나 머신러닝 학습으로 진행방향을 향하게 되었다. 그러나, 널리 쓰이는 파이토치, 텐서플로우 같은 라이브러리는 데이터 수가 너무 적어서인지, 인식률이 매우 낮게 나왔고 방법을 고안하다가 직접 파이썬의 numpy를 이용해서 로지스틱 회귀 공식을 구현해보았다. 이때 수식이나, 구현 방법에 있어 많이 참고를 한 사이트가 2번째 URL인 다층 퍼셉트론 구현 블로그이다. 직접 구현한 로지스틱 회귀 방법으로 대략 23번의 학습을 진행하니 Level3 기준으로 100%의 정확도를 보이는 것을 알 수 있었다. 그럼 다음으로 각 진행과정에 대해 설명하겠다.

5. 전처리

서명 이미지를 이진화하여 CNN 학습을 시킬 것이기 때문에 이번 프로젝트에서 전처리 과정은 굉장히 가볍게 진행되었다. 우선 각각의 서명 이미지를 그레이 스케일로 변환한 후, 각기 다른 픽셀 크기를 고려하여 100X100의 동일한 크기로 바꿔주었다. 이렇게 사이즈를 변환한 이미지에서 픽셀 데이터를 가져와 평균을 구하고, 평균보다 크면 1, 작으면 0으로 변환하는 이진화를 진행하였다. 사실 해시코드를 생각하기 이전에 Opencv를 활용해서 가우시안 블러도 씌워 이진화를 진행한 후 세선화를 해보려고 하였으나, 세선화 알고리즘이 뼈대를 추출하는 것이라 그런지 제대로 된 결과를 얻을 수 없었다. 아쉬운 대로 이진화 코드를 이용해 데이터의 정확도를 비교하는 방식을 사용하였고, 2개의 사인에는 충분히 잘 구분을 해주었으나 실제로 위조 서명을 찾기에는 미흡할 것이라고 생각해 아쉬움이 들었다.

6. 특징 추출

로지스틱 회귀 방식은 시그모이드 함수를 이용하여 0 또는 1의 값으로 데이터를 구분해 줄 수 있다. 서명 데이터가 각 레벨 별 2개라는 점에서 프로젝트에 잘 적용이 될 수 있었던 것 같다. 우선 앞서 이진화 한 각각의 데이터(2차원 배열)를 1차원의 이진 코드로 변환시켜 준다. 앞서 100X100의 크기로 변환하였으므로 데이터의 길이는 10000이 될 것이다. 10000개의 이진 데이터가 각 레벨 별 20~25개 정도 있으므로 대충 (20 * 10000)의 2차원 배열로 전체 데이터를 모두 로지스틱 회귀 공식을 이용하여 반복학습 시켜준다. 이때 레이어는 총 3개를 사용하였다. 오차가 낮아지는 방향으로 학습을 진행시키면 어느 순간 이미지가 100% 일치하여 학습이 종료된다. 후에 주어진 DB말고 다른 데이터가 들어와도 학습에서 구한 가중치(세타)값으로 계산을 해주면 해당 사인이 어디에 속하는 지 확인할 수 있다. 그러나 이번 프로젝트에서는 새로운 데이터가 들어오는 것이 아니므로 학습하는 것까지 구현하였다.

7. 매칭

기본적으로 학습의 정확도를 알기 위해서는 해당 학습 결과가 일치하는 지 확인해 줄 값이 필요하다. 그러나 주어진 이진 이미지 데이터에는 분류를 해줄 수 있는 데이터가 없으므로, 직접 각각의 이미지의 이름을 토큰으로 분류해서 구별해주었다. 예를 들어 파일의 이름 "L301-2320~"에서 첫 번째 L과 뒤에 ' - '를 기준으로 끊어내어 '301'만 추출해 낼 수 있도록 하였다. 따라서 실제로 코드를 돌릴 때, 파일의 이름과 파일 경로가 중요하게 작용한다. 또한 서명을 인식하여 이름을 출력하는 것이 프로젝트의 조건에 있는데 Level3 와 같이 흘려 쓴 서명 데이터는 한글을 인식하기에는 불가능 하다고 생각했기에, 앞에서 추출해 낸 각 이미지의 고유 번호에 맞게 프로그램 내에서 직접 이름을 넣어주었다. 예를 들어 320~340 까지의 숫자는 '박예솔'이라고 저장되는 것이다.

기본적으로 비교할 수 있는 파일명을 기준으로 2개의 서명 데이터를 수동으로 0, 1의 값을 부여주어 이미지 구분 데이터를 만들어 준다. 이렇게 만들어진 데이터는 모델을 트레이닝 하면서 오차 값을 구하는데 사용되면서 오차가 낮아지는 방향으로 가중치를 업데이트 하도록 이끌어준다. 또한 이 실제 구분 값을 이용해서 시그모이드 함수를 통해 얻어낸 최종 학습 데이터와 비교하여 정확도를 알아낼 수 있다. 여기서 시그모이드 함수의 결과 값이 0.5보다 크면 1, 작으면 0으로 정해 정확도를 구하는 데 사용되었다. 실제로 100% 인식이 성공했을 때, 평균 오차는 1~5% 정도로 꽤 정확한 인식률을 보이는 것을 알 수 있었다.

Ⅱ. 프로그램 결과

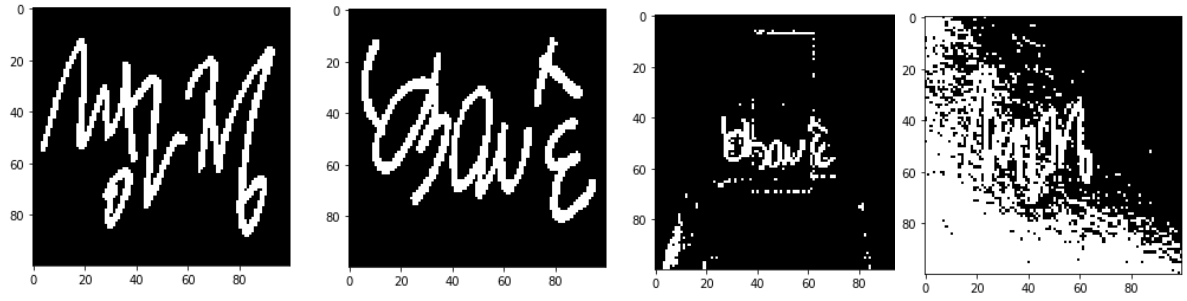
1. 전처리 (그레이스케일, 크기 변환, 이진화) 후 얻은 이진 해시 코드

→ 중간에 생략이 되어있지만, 100X100의 이진 코드이다.

```
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]]
[[1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
```

2. 특징 구분

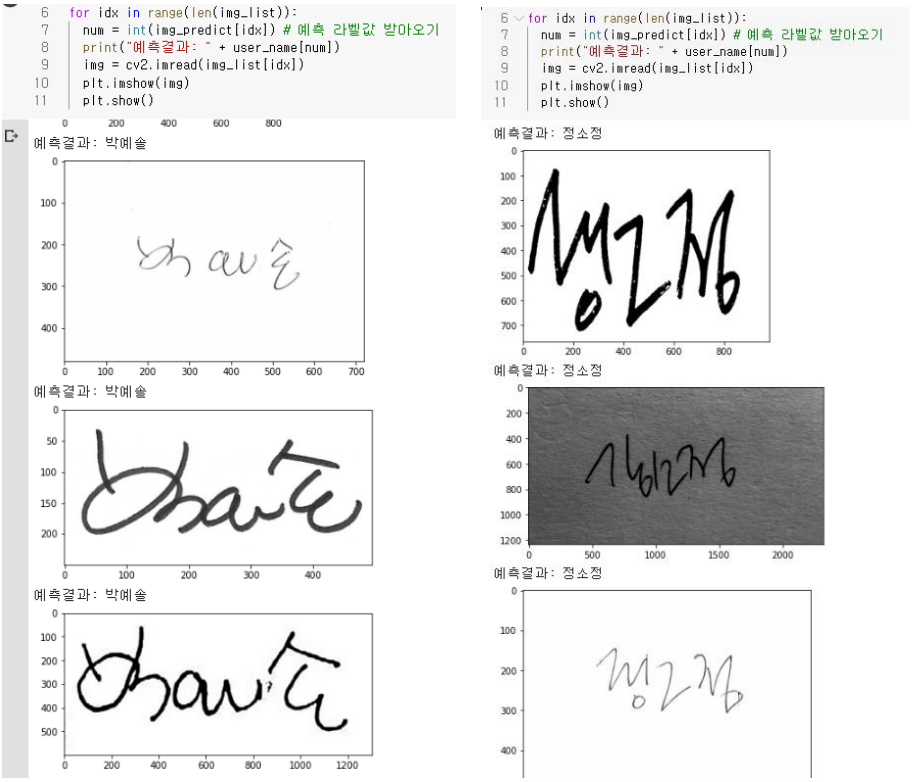
→ 앞서 이진 해시 코드로 변환한 데이터를 출력하면 다음과 같이 나온다. 꽤 깔끔한 이미지가 대부분이지만, 그렇지 않은 이미지도 있었다.



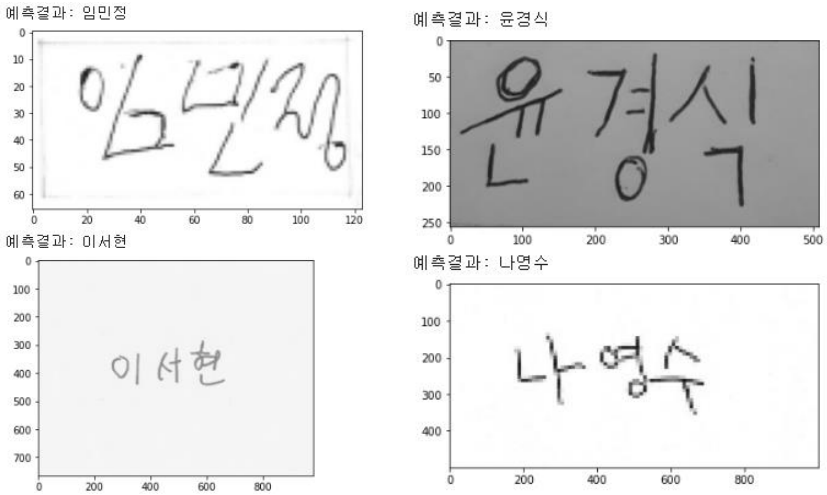
3. 인식 결과

```
========================== Training Start ==========================
Epoch:      1, train_cost:  55.400937, train_accuracy:   0.500000
Epoch:      2, train_cost:  36.033315, train_accuracy:   0.500000
Epoch:      3, train_cost:  25.457107, train_accuracy:   0.625000
Epoch:      4, train_cost:  18.106425, train_accuracy:   0.750000
Epoch:      5, train_cost:  14.277484, train_accuracy:   0.791667
Epoch:      6, train_cost:  11.620666, train_accuracy:   0.791667
Epoch:      7, train_cost:   9.828752, train_accuracy:   0.833333
Epoch:      8, train_cost:   8.805113, train_accuracy:   0.833333
Epoch:      9, train_cost:   7.932382, train_accuracy:   0.833333
Epoch:     10, train_cost:   7.153281, train_accuracy:   0.875000
Epoch:     11, train_cost:   6.393415, train_accuracy:   0.875000
Epoch:     12, train_cost:   5.598497, train_accuracy:   0.916667
Epoch:     13, train_cost:   5.081409, train_accuracy:   0.916667
Epoch:     14, train_cost:   4.699538, train_accuracy:   0.916667
Epoch:     15, train_cost:   4.382945, train_accuracy:   0.916667
Epoch:     16, train_cost:   4.127106, train_accuracy:   0.958333
Epoch:     17, train_cost:   3.923215, train_accuracy:   0.958333
Epoch:     18, train_cost:   3.756875, train_accuracy:   1.000000
cost is converged
Epoch:     18, train_cost:   3.756875, train_accuracy:   1.000000
img_label > [1 0 1 1 0 1 1 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0]
predict > [1. 0. 1. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0.]
```

위의 학습 결과를 보면 18번의 학습 끝에 100% 인식률을 보였음을 알 수 있다. 사실 predict는 완벽하게 1, 0 이 아니라 대략 0.97 나 0.02 정도의 값을 나타낸다. 그러나 이미지를 출력할 때, 라벨 값을 0, 1 로 떨어지는 정수 값을 받아야 하기 때문에 편의상 반올림을 하였다.



마찬가지로 Level2, Level3 에서도 잘 인식되는 것을 알 수 있었다.



4. 결과에서 미진한 점 및 개선점 설명

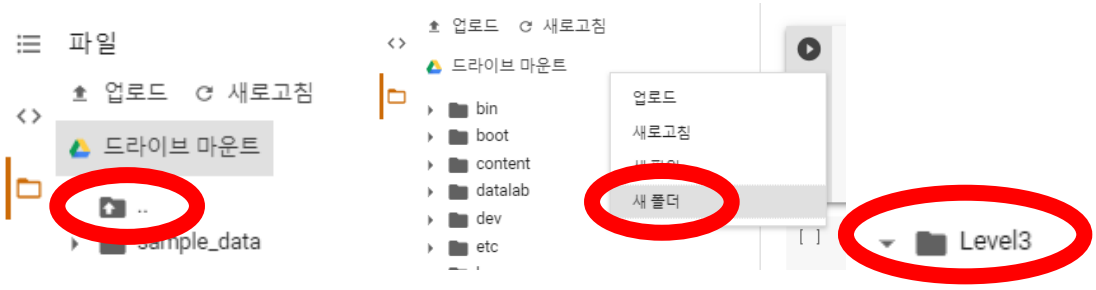
결과 자체는 100% 정확하게 나왔지만, 전처리에서 좀 더 명확한 해결방법을 찾지 못한 점이 아쉽다. 제대로 된 세선화 알고리즘을 찾아 적용할 수 있다면 실제로 위조 서명까지 가려낼 수 있지 않을까 생각한다.

Ⅲ. 프로그램 실행 방법

1. 구글 코랩 사이트에 들어간다.
https://colab.research.google.com/notebooks/welcome.ipynb?hl=ko

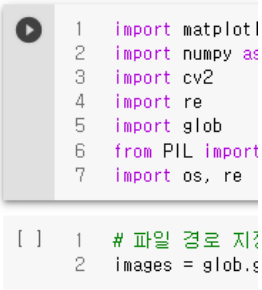
2. 첨부한 'signature_recognition.ipynb' 파일을 좌상단 위의 (파일>노트 열기> 업로드)에 넣는다.

3. 파일이 열리고 조금 기다리면 좌측에 파일을 업로드 할 수 있는 화면이 생기는데 이때 아래와 같이 새 폴더를 생성한 후 해당 폴더 안에 서명 이미지를 넣어준다. 파일 이름에 따리 인식률을 계산할 수 있도록 코드를 짰기 때문에 파일명과 경로가 아주 중요하다.



4. 우측의 그림과 같이 [ ] 표시에 마우스를 가져다 대면 실행할 수 있는 화살표가 나온다 위에서부터 아래로 순서대로 모두 실행시켜주면 된다.

오른쪽 그림을 보면 그림이 잘려 있기는 하지만, 2번째 셀에 파일 경로를 입력할 수 있는 코드가 있다. 만약에 폴더 경로를 확인하고 싶다면 폴더에서 마우스 우측 버튼을 누르고 경로를 복사하여 붙여 넣으면 된다.



5. 데스크탑에서 바로 DB 폴더를 업로드하려고 하였으나 업로드가 안되었다.

(라이브러리를 import하는 번거로움을 없애려고 Colab을 사용했는데, 이미지 데이터를 업로드하는 번거로움을 드려서 죄송합니다.)