CNN 모델을 통한 한글 필기체 분류

김자현[1] 데이터 전처리, 프로그래밍, PPT 작성, 보고서 작성

나상우[2] 데이터 전처리, 프로그래밍, PPT 작성, 보고서 작성

요약

본 연구는 CNN(Convolutional Neural Networks) 모델을 통한 한글 손글씨 예측 정확도 향상을 목표로 한다. 185개 한글 문자에 대해 학습을 진행하였고, 46개의 폰트체와 152명의 필기체를 샘플로 사용했으며, 총 31,677개의 구성돼 있다. 여러 개의 모델을 시험해보았고, 그 중 오버피팅, 언더피팅, 정확도 낮음 등의 문제가 발생한 모델들이 있었다. 시도해 본 여러 모델 중 본 프로젝트에서는 성능이 좋은 모델 두 가지와 그렇지 못한 모델 두 가지로 총 4개의 분류모델을 제안했으며, 그중 1개의 모델을 최종 선택하였다. 선택한 최적 모델을 학습 정확도 96%, 검증 정확도 935, 시험 정확도 94%에 도달하였다. 더 나아가 더 많은 손글씨 예측 모델을 만들어 정확도를 높여 기술 발전 근간에 도움이 되기를 기대한다.

주요용어 : CNN, 한글 인식, 손글씨, 분류.

1. 서론

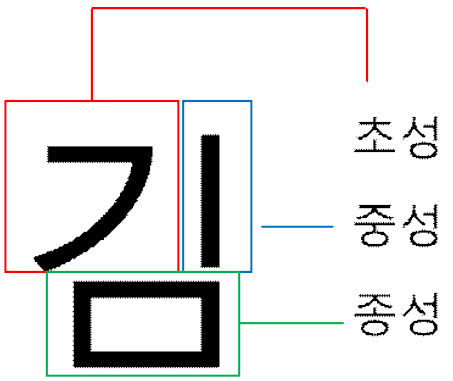
영어의 경우 알파벳이 26자이다. 일본어 경우에도 총 100개로 이루어져 있다. 그에 비해 한글은 글자를 구성할 수 있는 경우의 수가 많다. 초성 19개, 중성 21개, 종성 27개로 이루어져 있다. 초성x중성=399개, 초성x중성x종성=10,773개이다. 즉, 총 11,172개로 이루어져 있어 영어, 일본어와 다르게 손글씨 예측 정확도가 다소 떨어지는 편이다. 영어와 일본어 인식률에 관한 연구에 따르면 영어의 경우 99%에 달하는 정확도[1]를 보유하고 있고, 일본어 경우에도 필기체 인식률이 99.52%[2]에 달한다. 그에 반해 한글 상용 OCR 프로그램들은 66.95%에서 83.17 분류 성공률[3]을 보인다. 다른 언어에 비해

[1]30019 세종시 세종로 2511, 고려대학교 공공정책대학 경제통계학부 빅데이터전공 학사과정, E-mail : kjh00808@korea.ac.kr

[2]30019 세종시 세종로 2511, 고려대학교 공공정책대학 경제통계학부 빅데이터전공 학사과정, E-mail : ra0622@korea.ac.kr

상당히 낮은 인식률을 보여 어려움을 겪고 있다. 따라서 직접 CNN 모델을 구현하여 한글 필기체 인식률을 높이고자 한다.

본 연구는 1장에서 연구의 필요성을 소개한 후 2장에서 데이터의 출처와 전처리 과정을 설명한다. 그 후 본론에 해당하는 3장에서 분석에 사용된 모델 CNN 모델 구조를 보여주며 분석 결과를 정리한다. 마지막 4장에서 결론으로 마무리한다.



<그림. 1 > 초성, 중성, 종성 표현

2. 데이터

인공지능 학습을 위한 데이터를 제공해주는 AI Hub 자연어 처리 파트의 '다양한 형태의 한글 문자 OCR 소개'에서 152명의 손글씨 데이터를 구하였다. 개인당 완성형 2,350자의 손글씨를 썼으며, 누락 된 데이터도 존재한다. 'GitHub - ios 모바일에서 한글 손글씨 인식하기'에서 총 46개 폰트로 980자를 작성한 데이터를 얻었다. 980자의 기준은 국립국어원에서 2004년 12월에 발표한 "한국어 학습용 어휘" 6,000개 낱말 가운데 고유명사를 뺀 자주 쓰이는 한국어 5,888개 기초 낱말 중 중복을 제외한 낱말들이다.

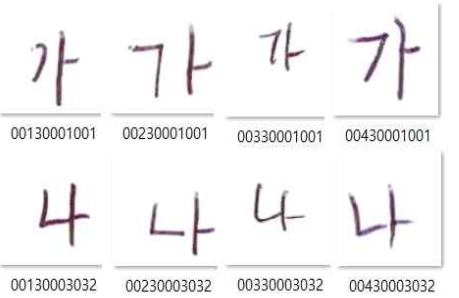
초기 데이터 수집 단계에서 이 980자를 출력층으로 사용하려 하였으나, 출력층을 모두 사용할 경우 학습시간이 상당히 길 것이라고 예상되었다. 따라서 임의로 출력

가각간갇갈감갑갓강갖갗같갚갛거고구
나낙난낟날남납낫낭낮낯낱낳너노누다
닥단닫달담답닷당닺닻닿더도두라락란
랄람랍랏랑랒랖랗러로루마막만맏말맘
맙맛망맞맡맣머모무바박반받발밤밥밧
방밭버보부사삭산삳살삼삽삿상샅서소
수아악안알암압앗앙앝앞어오우자작잔
잗잘잠잡잣장잦저조주차착찬찰참찹찻
창찾처초추카칵칸칼캄캅캇캉커코쿠타
탁탄탈탐탑탓탕터토투파팍판팔팜팝팟
팡퍼포푸하학한할함합핫항허호후

<그림. 2> 출력층 185자

층을 선정했다. ‘ㄱ’~‘ㅎ’를 초성, 종성으로 두었고 ‘ㅏ’를 중성으로 두는 것으로 조합하여 출력층을 만들었다. 끝으로 모음이 한 개만 있을 때 모음에 대한 예측력을 볼 수 없으므로 ‘ㅓ’, ‘ㅗ’, ‘ㅜ’를 추가하였다. 총 252자의 조합이 만들어진다. 원래 데이터가 실생활에 자주 쓰이는 글자만 모아 놓은 것으로, 잘 안 쓰이는 단어는 제외됐다. 따라서 <그림. 2>에서 보는 거와 같이 총 185자의 출력층이 만들어진다.

우리의 데이터는 글씨를 작성한 사람별로 폴더가 나누어져 있었다. 폴더 내의 손글씨 이미지는 각각의 글자별로 동일한 고유 번호가 붙어있으며, <그림. 3>에서 확인할 수 있다. 먼저, 파이썬을 이용해 출력층 개수에 해당하는 185개의 폴더를 만들었다. 폴더명은 1-185로 순서대로 ‘가’, ‘각’,…. ‘후’가 된다. 이 이미지들을 파이썬



<그림. 3> 이미지별 고유 번호

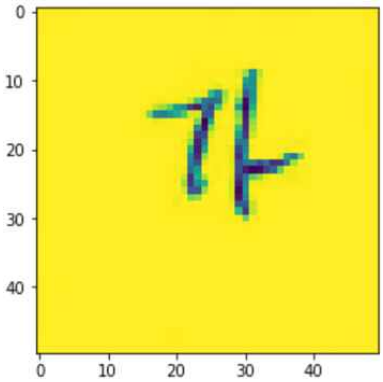
<그림. 4> 폴더 내 이미지 이름

코드를 사용해 각각의 글자에 해당하는 폴더로 이동시킨다. 이로써 31,677개에 해당하는 데이터를 각 폴더별로 옮겼다. 한 폴더에 대한 평균 이미지 개수는 171개다. 이미지를 쉽게 불러오기 위해 각 폴더의 이미지들의 이름을 1, 2, 3, …, 1nn으로 변경시켰다.

3. 본론

3.1 데이터 전처리

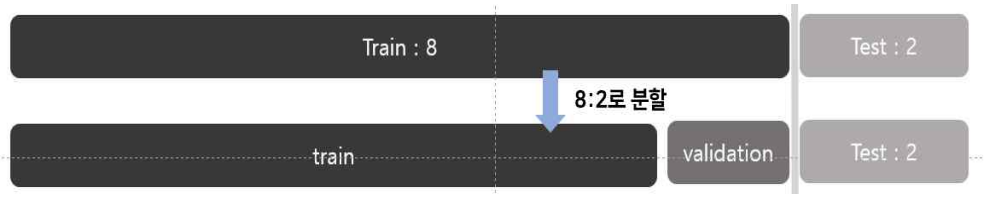
위에서 작업했던 이미지들을 불러와 각각의 폴더를 목적변수 Labels로, 이미지의 픽셀값을 설명변수 pixels로 리스트에 넣는다. 이때 이미지의 크기를 50x50으로 통일시킨다. 픽셀값을 1차원 배열로 평탄화시켜 255로 나누어 정규화시켜준 후, 다시 50x50 배열로 바꾸는 reshape 작업을 진행했다. <그림. 5>는 표준화를 거친 후의 이미지 모습이다.



<그림. 5> 표준화 거친 후의 이미지

3.2 학습, 검증, 시험 데이터 나누기

먼저 학습데이터와 시험데이터의 비율을 8:2로 분할한다. 그 후 학습데이터를 다시 학습데이터와 검증데이터 8:2 비율로 분할한다. 결과적으로 총 31,677개의 이미지 데이터는 학습데이터의 개수는 20,272개, 검증데이터의 개수는 5,069개 그리고 시험데이터의 개수는 6,336개로 분할되었다. <그림. 6>은 이 과정을 시각화한 모습이다.



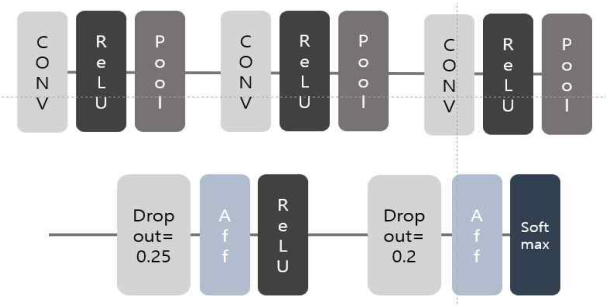
<그림. 6> 데이터 분할

<표. 1> 데이터 개수

Name	Count
Train	20,272
Validation	5,069
Test	6,336
Total	31,677

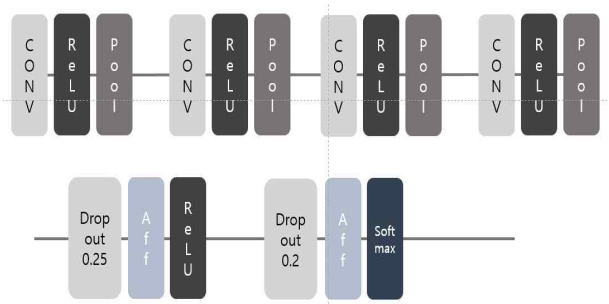
3.3 모델 구성하기

1) 첫 번째 모델은 convolution2D, ReLU, Pooling 계층을 3층으로 쌓았고, 출력 계층에서 softmax를 사용했다. 층이 깊어질수록 채널 수가 32-64-128로 늘어나는 구조이다.



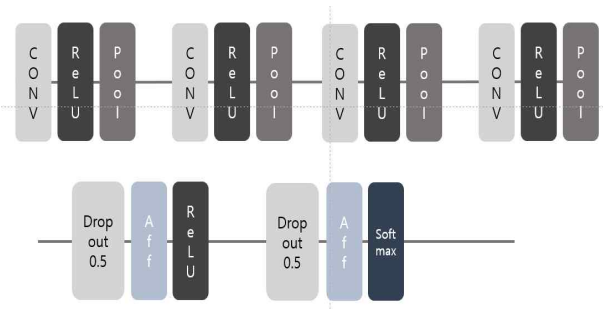
<그림. 7 > 1번 신경망 모델

2) 두 번째 모델은 첫 번째 모델에서 convolution2D, ReLU, Pooling 계층을 추가해 더 깊은 신경망을 구성한 모델이다.



<그림. 8> 2번 신경망 모델

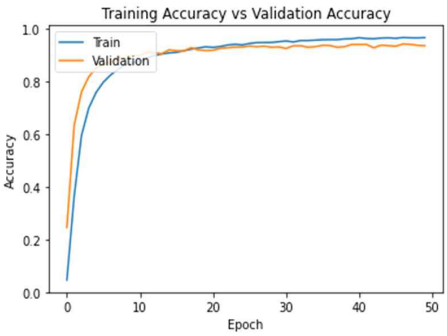
3) 세 번째 모델은 두 번째 모델과 깊이와 계층은 동일하지만 과적합 방지를 위한 dropout을 0.5로 조정한 모델이다.



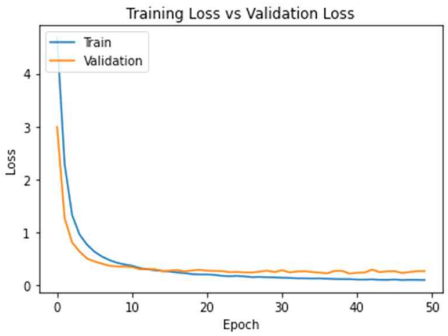
<그림. 9> 3번 신경망 모델

3.4 각각 모델 적용시키기

1) 첫 번째 모델에서 Adam을 사용해 최적화하고 배치크기는 32, 에폭수는 50으로 설정한 경우이다. 정확도와 손실을 시각화한 그래프를 보면 에폭이 증가함에 따라 학습이 잘되어가는 것을 볼 수 있다.



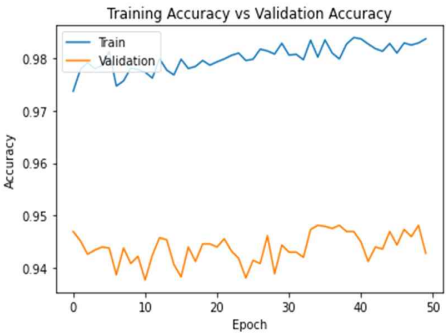
<그림. 10 > 경우1 정확도 시각화



<그림. 11 > 경우1 손실 시각화

학습데이터의 정확도는 0.9646, 손실은 0.1058이고 검증데이터의 정확도는 0.9339, 손실은 0.2738, 그리고 시험데이터의 정확도는 0.9371, 손실은 0.2704의 결과가 나왔다. 대체로 좋은 정확도 값을 가진 학습을 잘 시킨 모델이라고 평가할 수 있다.

2) 첫 번째 모델에서 Adam을 사용해 최적화하고 배치 크기는 60, 에폭 수는 50으로 설정한 경우이다. 정확도와 손실을 시각화한 그래프를 보면 에폭이 증가함에 따라 학습이 잘 이루어지지 않고, 일정 범위 내에서 심하게 변동하는 것을 볼 수 있다.

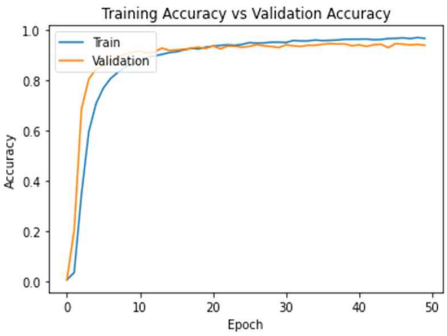


<그림. 12> 경우2 정확도 시각화

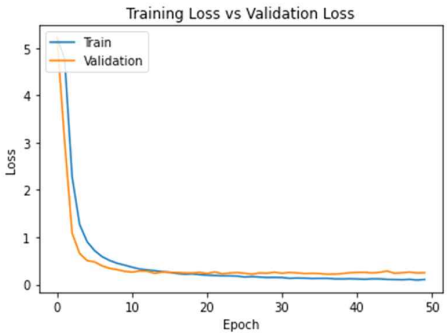
<그림. 13> 경우2 손실 시각화

학습데이터의 정확도는 0.9838, 손실은 0.0516이고 검증데이터의 정확도는 0.9428, 손실은 0.2439, 그리고 시험데이터의 정확도는 0.9434, 손실은 0.2639의 결과가 나왔다. 대체로 좋은 정확도 값을 가진 모델이지만, 한 에폭 사이의 변동이 심하기 때문에 좋은 모델로 평가할 수 없다.

3) 두 번째 모델에서 Adam을 사용해 최적화하고 배치 크기는 32, 에폭 수는 50으로 설정한 경우이다. 정확도와 손실을 시각화한 그래프를 보면 에폭이 증가함에 따라 학습이 잘 되어가는 것을 볼 수 있다.



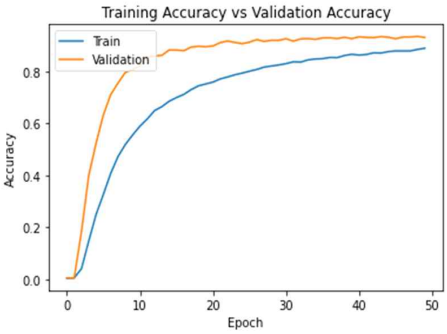
<그림. 14> 경우3 정확도 시각화



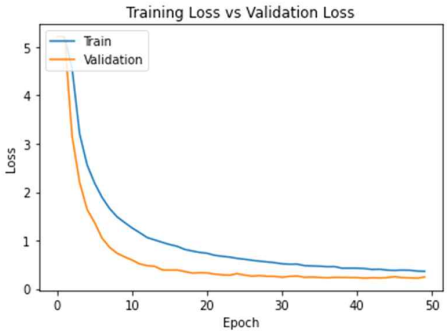
<그림. 15> 경우3 손실 시각화

학습데이터의 정확도는 0.9668, 손실은 0.1041이고 검증데이터의 정확도는 0.9394, 손실은 0.2470, 그리고 시험데이터의 정확도는 0.9434, 손실은 0.2639의 결과가 나왔다. 대체로 좋은 정확도 값을 가진 학습을 잘 시킨 모델이라고 평가할 수 있다. 또한, 첫 번째 경우와 비교해보았을 때, 학습데이터와 검증데이터의 정확도는 유사하지만 깊은 신경망을 가진 해당 경우가 시험데이터의 정확도가 높아졌음을 알 수 있다.

4) 세 번째 모델에서 Adam을 사용해 최적화하고 배치 크기는 32, 에폭 수는 50으로 설정한 경우이다. 정확도와 손실을 시각화한 그래프를 보면 에폭이 증가함에 따라 학습이 되어가고 있지만, 학습데이터의 정확도보다 검증데이터의 정확도가 매우 높게 나타나는 언더피팅이 발생하였음을 알 수 있다.



<그림. 16> 경우4 정확도 시각화



<그림. 17> 경우4 손실 시각화

학습데이터의 정확도는 0.8896, 손실은 0.3607이고 검증 데이터의 정확도는 0.9312, 손실은 0.2440, 그리고 시험데이터의 정확도는 0.9343, 손실은 0.2394의 결과가 나왔다. 위에서 설명했던 것처럼 학습데이터의 정확도가 검증 데이터의 정확도보다 낮은 언더피팅의 결과가 나왔으므로 좋은 모델이라고 평가할 수 없다.

3.5 모델 비교

네 가지 경우를 비교해본 결과, 4번 경우를 제외하고는 모두 좋은 정확도를 가진 모델임을 알 수 있고, 정확도만 보았을 때는 2번 경우가 가장 좋다. 하지만 2번 경우는 에폭의 증가에 따라 정확도와 손실의 노이즈가 심하게 있었다. 1번과 3번 경우는 학습과 검증 데이터가 유사한 정확도를 보인다. 하지만 시험데이터의 정확도는 3번의 경우가 더 높음을 알 수 있다.

<표. 2> 경우1 수치 비교

#1	accuracy	loss
Train	0.9646	0.1058
Validation	0.9339	0.2738
Test	0.9371	0.2704

<표. 3> 경우2 수치 비교

#2	accuracy	loss
Train	0.9838	0.0516
Validation	0.9428	0.2439
Test	0.9434	0.2639

<표. 4> 경우3 수치 비교

#3	accuracy	loss
Train	0.9668	0.1041
Validation	0.9394	0.2470
Test	0.9434	0.2639

<표. 5> 경우4 수치 비교

#4	accuracy	loss
Train	0.8896	0.3607
Validation	0.9312	0.2440
Test	0.9343	0.2394

3.6 최적의 모델 선정하기 및 결과

본 프로젝트에서는 한글 필기체 이미지 데이터를 사용하여 프로젝트를 진행하였다. 총 185개의 클래스를 가지고 진행했으며, 각 클래스당 평균 171개의 데이터셋을 포함하고 있다. 학습데이터와 시험데이터 비율은 8:2로 가져갔고, 데이터 수가 적절하다고 생각되어 시험데이터 비율을 적절하게 채택했다. 학습을 위한 모델은 4개를 구성했으며, 한 번의 학습당 대략 1시간이 소요됐다. 3.5 모델 비교에서 언급했던 이유들로 우리는 최적의 모델로 3번 경우를 선정하였다. 3번의 경우는 훈련 데이터 정확도와 검증 데이터 정확도 값이 적절하게 큰 결과로 잘 나왔고, 시험데이터 정확도 또한 대략 94%로 높은 결과가 나왔기 때문에 3번을 최적의 모델로 선정하였다.

4. 결론

이번 프로젝트에서 우리는 CNN 신경망 구조를 활용하여 높은 정확도의 한글 필기체 분류 모델을 만들었다. 출력층이 185개로 한글 문자 경우의 수에는 한참 부족한 수 이지만 더 보완한다면 이 프로젝트의 초기 목표였던 일상에서 가장 많이 쓰이는 글자 980자 분류모델 또한 충분히 구현할 수 있을 것이다. 980자에 해당하는 글자 또한 높은 정확도를 보인다면 상용 OCR의 정확도가 낮은 상황에서 새로운 OCR 기법의 대안으로 활용이 가능할 것이다.

끝으로 이번 프로젝트를 기반으로 초성, 중성, 종성 필기 순서에 맞추어 작성한 필기체를 인식할 수 있는 모델을 구현하여 실시간으로 작성하는 필기체를 인식하는 프로그램을 만들고자 하는 목표를 가지게 되었다.

참고문헌

[1]김진호. (2020). 딥러닝 신경망을 이용한 문자 및 단어 단위의 영문 차량 번호판 인식. (사)디지털산업정보학회 논문지, vol.16, no.4, 19-28.

[2]Hyunwoo Kim, Yoojin Chung. (2018). Improved Handwritten Hangeul Recognition using Deep Learning based on GoogLenet. The Journal of the Korea Contents Association, vol.18, no7, 495-502

[3]Kim, Y., Cha, E. (2016). Streamlined GoogLeNet Algorithm Based on CNN for Korean Character Recognition. Journal of the Korea Institute of Information and Communication

Engineering, vol.20, no.9, 1657-1665.
[4]아다치 하루카. (2020), 백견불여일타 머신러닝 데이터 전처리 입문 실습하며 배우는 데이터 전처리 입문서. 로드북.