한글 인쇄체 인식 모델 개발

2020-07-25 2014231183 최찬호

한글 인쇄체 이미지로부터 지정된 한글 텍스트 문자를 얻는 소프트웨어를 개발하였습니다. 이미지 인식은 CNN 구조의 학습 모델을 이용하였습니다. 하나의 한글 인쇄체 글자 이미지를 입력으로 받으며, 입력 받은 이미지를 한글 완성형 음절(총 11,172자) 중 하나로 구분합니다.

소프트웨어는 Python 3.7(64bit) 환경에서 작성되었으며, 학습 모델 구현은 Tensorflow 및 Keras 오픈소스 라이브러리를 이용하였습니다. CNN 구조의 학습 모델은 Resnet (50 layers)를 사용했습니다. Resnet의 구현은 Keras 라이브러리 built-in model을 이용했습니다. Resnet model은 Imagenet dataset를 이용하여 학습된 가중치를 초기값으로 사용했습니다.

학습에 사용된 데이터는 ETRI 한글 인쇄체 학습 데이터 (532,659자)와 14개의 TTF 형식 글꼴로 생성한 한글 인쇄체 학습 데이터 (156,408자) 입니다. 한글 인쇄체를 효율적으로 인식하기 위하여 각각 한글의 초성, 중성, 종성을 인식하는 3개의 model을 학습시킨 후, 결합하는 방식을 사용하였습니다.

결과적으로 학습된 모델은 validation set에서 99.858%의 정확도를 가질 수 있었습니다. 또한, 임의의 문서 이미지로부터 추출된 글자들을 복원하는데 91.75%의 정확도를 가집니다.

목차

1. **소프트웨어**
   1. 개발 환경
      1. 소스코드 (Git hub)
      2. 라이브러리 버전 및 사용
      3. 모듈 구성 (소스 파일)
2. **학습 데이터 구성**
   1. UTF-16 형식 한글 완성형 문자 구성
      1. UTF-16 Code Table
   2. 학습 데이터 정보
      1. ETRI Dataset
      2. 한글 인쇄체 학습 데이터 생성
   3. Image augmentation
      1. Randomized Morphological Transform
3. **학습 방법**
   1. Model
      1. Output Model
      2. Backbone Model – Resnet (50 layers)
   2. 학습 모델 구현
      1. Built-in Model in Keras
      2. Functional model API
      3. 모델 및 가중치 저장
4. **결과**
   1. Validation Set
   2. 문서 스캔 이미지
5. **소프트웨어**
   1. 개발 환경

한글 인쇄체 인식 모델은 Python 3.7(64bit) 환경에서 개발되었습니다. 프로그램 작성을 위한 IDE로는 PyCharm community를 사용하였으며, Anaconda 3를 이용한 가상환경을 구성하였습니다.

* + 1. 소스코드 (Git hub)

한글 인쇄체 데이터 생성, 인식 모델 학습, 학습된 모델을 이용한 추정 등이 구현된 소스코드입니다.

<https://github.com/chanho096/Hangul_OCR>

* + 1. 라이브러리 버전 및 사용
* tensorflow 2.0.1: 인공지능 모델 구현 및 학습
* numpy 1.19.0: 각종 연산 처리
* Keras-Applications 1.0.8: built-model 사용 (Resnet50)
* opencv-python 4.3.0.36: 이미지 데이터 로드, Image augmentation
* Pillow 7.2.0: 한글 인쇄체 학습 데이터 생성 (TTF 파일 이용)
* sklearn 0.0: 학습 데이터 무작위 추출 및 분할
  + 1. 모듈 구성 (소스 파일)
* agent.py: 학습 모듈 (model 구현 및 hyper parameter 설정)
* aug.py: Image augmentation 구현
* gen.py: 문자 데이터 변환 장치 (UTF-16 형식 이용)
* hangul.py: 한글 문자 데이터 정보 추출
* loader.py: 학습에 사용될 데이터 경로들을 추출

1. **학습 데이터 구성**

학습에 사용된 데이터는 ETRI에서 제공받은 한글 인쇄체 이미지 (532,659자) 와 14개의 TTF 형식 글꼴로 생성한 한글 인쇄체 이미지 (156,408자), 도합 689,067자의 한글 인쇄체 이미지 데이터입니다.

ETRI 학습 데이터의 label은 String 형식의 한글 한 글자로 구성되어 있습니다. String의 한글을 UTF-16 형식을 기준으로 정수와 대응시켰습니다. 또한, 각 정수 값은 output layer의 출력과 대응되도록 one-hot encoding 됩니다.

689,067개의 한글 이미지가 한 번에 처리될 수 없으므로, batch 단위로 이미지를 메모리에 적재하는 Custom Generator를 구현하였습니다. 또한 훈련 모델이 overfitting 되지 않고, 여러가지 바람직한 특성들을 학습할 수 있도록 Image augmentation 기법을 사용하였습니다.



[ETRI 한글 인쇄체 데이터]



[TTF 파일로부터 생성한 한글 인쇄체 데이터 – 네이버 나눔 고딕]

* 1. UTF-16 형식의 한글 완성형 문자 구성

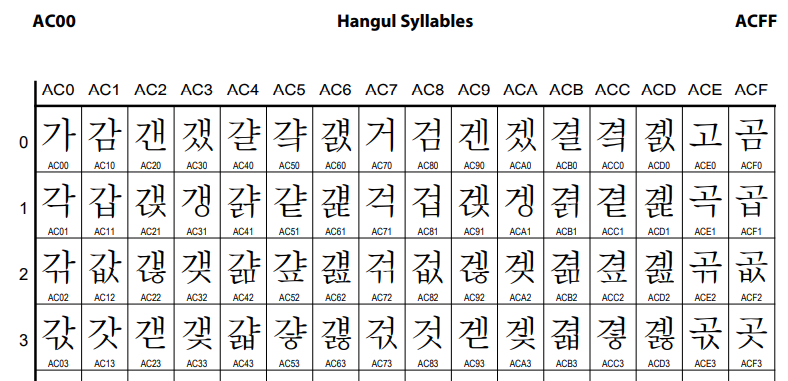
한글 인쇄체 데이터들을 학습에 사용하기 위하여, 먼저 한글 문자 데이터를 처리하는 형식을 지정할 필요가 있습니다. 해당 소프트웨어는 UTF-16의 한글 완성형 형식을 이용하여 한글 문자(음절 단위)를 바이트 코드로 변환합니다.

1~4 byte 가변 길이로 한글을 표현하는 UTF-8 형식과 다르게, UTF-16 형식은 2 byte고정 길이로 한글, 알파벳 대소문자, 숫자, ASCII 특수기호 등을 표현할 수 있습니다. 따라서 더 변환하기 간편한 UTF-16 형식을 선택하였습니다.

* + 1. UTF-16 Code Table

UTF-16 형식에서는 2 byte로 한글 완성형 문자를 표현할 수 있습니다. 한글 완성형 문자 11,172자는 0xAC00 ~ 0xD7AF로 표현됩니다.

한글 완성형 문자는 초성 19자, 중성 21자, 종성 27자로 구성되어 있습니다. UTF-16 Code Table은 0xAC00 지점의 “가” 부터 시작하여, 초성 – 중성 – 종성 순서대로 정렬된 한글 음절을 표현합니다.



[UTF-16 Code Table – Hangul Syllables]

* https://rudhar.com/lingtics/uniclnkl.htm
* 변환 알고리즘

String 자료 형식으로 주어지는 한글 음절을 다음과 같은 방법을 이용하여 학습에 사용하기 적합한 벡터로 변환합니다.

1. String 형의 한글 음절을 UTF-16 형식으로 Decoding
2. Decoding 값을 정수형으로 형 변환
3. 변환된 정수형 값에 0xAC00을 차감
4. 정수형 값에 One-Hot Encoding 수행

* Output-Layer의 출력과 같은 차원의 벡터 생성
  1. 학습 데이터 정보

ETRI에서 제공받은 한글 인쇄체 이미지 (532,659자) 와 14개의 TTF 형식 글꼴로 생성한 한글 인쇄체 이미지 (156,408자), 도합 689,067자의 한글 인쇄체 이미지 데이터를 학습에 사용하였습니다.

* + 1. ETRI Dataset

<http://www.aihub.or.kr/aidata> - Korean Font image AI Training dataset

한글 인쇄체 이미지 Dataset 중 현대 한글 약 55만자를 이용하였습니다. 다양한 글씨체로 구성된 한글 음절 이미지들이 포함되어 있습니다. 이미지에 대한 구체적인 정보는 별도의 json 파일에 포함되어 있습니다. json 파일을 읽어서 이미지 크기 및 Text 정보 등을 추출하고 학습에 사용하였습니다.

* + 1. 한글 인쇄체 학습 데이터 생성

더 많은 한글 인쇄체 데이터를 얻기 위하여 직접 학습 데이터를 생성하였습니다. Pillow 라이브러리를 이용하여 직접 TTF 형식의 글꼴을 읽어서 임의 크기의 한글 음절 이미지를 생성합니다. 사용된 글꼴은 서로 다른 14개 종류로, 총 11,172 \* 14 = 156,408 자의 학습 데이터를 추가로 확보할 수 있었습니다.

* 사용된 글꼴 이름

바탕, 굴림, HY그래픽M, 함초롱바탕, 함초롱돋움, 휴먼편지체, 휴먼아미체, 맑은 고딕, 맑은 고딕 Semilight, NanumBarunGothic, 나눔바른펜, 나눔고딕, 나눔명조, 새굴림

- 네이버 나눔 글씨체: <https://hangeul.naver.com/2017/nanum>

* 1. Image Augmentation

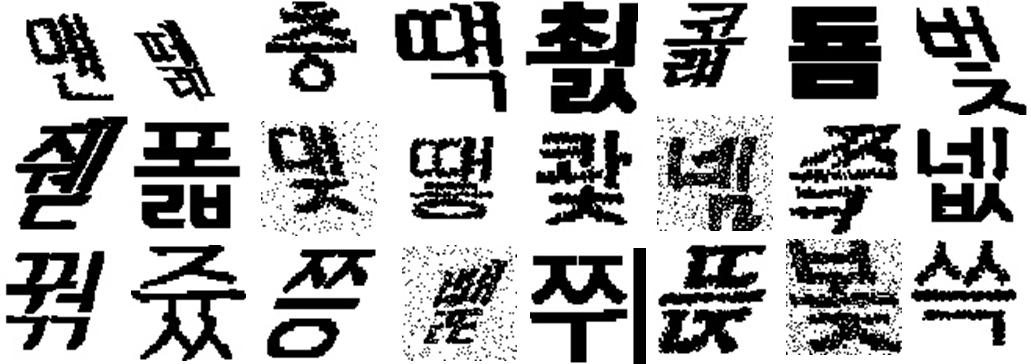
학습되는 모델이 다양한 환경에서도 문자 인식을 수행할 수 있도록, 학습 데이터를 전 처리하여 학습을 수행하는 Image Augmentation 기법을 사용하였습니다. 곧, 학습 데이터들을 지정된 규칙에 따라서 무작위적으로 변형합니다.

문자 데이터가 지나치게 손상되어 학습에 나쁜 영향을 주지 않도록, 다양한 전 처리 함수들을 Opencv 라이브러리를 이용하여 직접 생성하고 실험하여 적용하였습니다. 적용된 전 처리 함수들은 다음과 같습니다.

* Shearing (문자 이미지 전단)
* Median Blurring
* 무작위 문자 크기 조정 및 이동
* Random Padding
* Morphological Transform
* Noise 생성

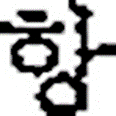
[Morphological Transform] [Shearing]



[Image Augmentation Example]

* + 1. Randomized Morphological Transform

학습 데이터로 사용되는 한글 인쇄체 데이터는 글꼴로부터 생성되어, 글자들이 비교적 뚜렷한 직선 모양을 가지고 있습니다. 반면 문서 스캔 이미지에서 추출된 글자는 이러한 직선 모양이 뚜렷하게 나타나지 않습니다.

[문서 스캔 이미지 “항”] [학습 데이터 “항”]

위와 같은 이미지 특징 차이로 인하여, 학습 데이터에서는 높은 인식률을 보유한 모델이 문서 스캔 이미지에서 추출된 글자를 잘 인식하지 못하는 문제점이 있었습니다. 이러한 문제점을 학습 데이터 추가 없이 해결하기 위하여, 새로운 Image Augmentation 전 처리 함수를 고안하였습니다.

Randomized Morphological Transform은 위의 문제를 해결하기 위해 고안된 전 처리 함수입니다. 기존 Morphological Transform은 이미지 전체를 하나의 고정된 커널을 이용하여 변환했던 반면에, Randomized Morphological Transform은 이미지 각 부분마다 무작위 커널을 생성하여 변환합니다.

[Randomized Morphological Transform을 적용한 학습 데이터 “항”]

Randomized Morphological Transform은 Numpy 라이브러리를 이용하여, 병렬 처리가 가능하도록 구현되었습니다. 따라서 학습 시간에 큰 영향을 주지 않고서 적용 가능합니다.

1. **학습 방법**
   1. Model

CNN 구조의 딥러닝 모델을 이용하여, 입력으로 주어진 한글 인쇄체 문자 이미지를 11,172자의 한글 완성형 문자 중 하나로 구분하는 능력을 학습시킵니다.

입력으로 주어지는 이미지는 74 \* 74, 8bit 크기의 3 channel 이미지입니다. 이미지는 흰 바탕에 검정색 글자 성분으로 구성되어 있습니다. 입력 이미지는 이진 이미지로 변환된 후, 두 색상만으로 나타나도록 변환됩니다. 딥러닝 모델에 입력 될 때에는 각 값이 0과 1의 사이 값을 가지도록 정규화 됩니다.

학습 모델은 크게 두 부분으로 나누어져 있습니다. CNN 구조로 이미지로부터 특징을 추출하는 Backbone Model, 완전 연결망 구조로 추출된 특징으로부터 Classification을 수행하는 Output Model으로 구분됩니다.

Backbone Model은 Resnet(50 layer)로 구성됩니다. 또한, Backbone Model은 Imagenet 데이터로 학습된 pre-trained Model을 이용하였습니다.

* He\_\_Deep Residual Learning for Image Recognition\_\_2015

Output Model은 동일한 Backbone Model의 출력을 이용하는 독립된 세 개의 Model들로 구성되어 있습니다. 세 종류의 Model은 각각 한글의 초성, 중성, 종성을 인식합니다. Output Model은 softmax 활성함수의 output layer를 가집니다.

학습에는 Cross-Entropy Loss Function과 Adam Optimizer가 사용되었습니다. 아래는 구체적인 Hyperparameter 설정 값입니다.

|  |  |  |  |
| --- | --- | --- | --- |
| Batch Size | Learning Rate | Beta\_1 | Beta\_2 |
| 64 | 0.0005 | 0.9 | 0.999 |

* + 1. Output Model

하나의 Output Model을 구성하여, 11,172자의 한글 음절을 구분하는 방법은 좋은 성능을 내지 못하였습니다. 따라서, 새로운 Output Model을 고안하였습니다.

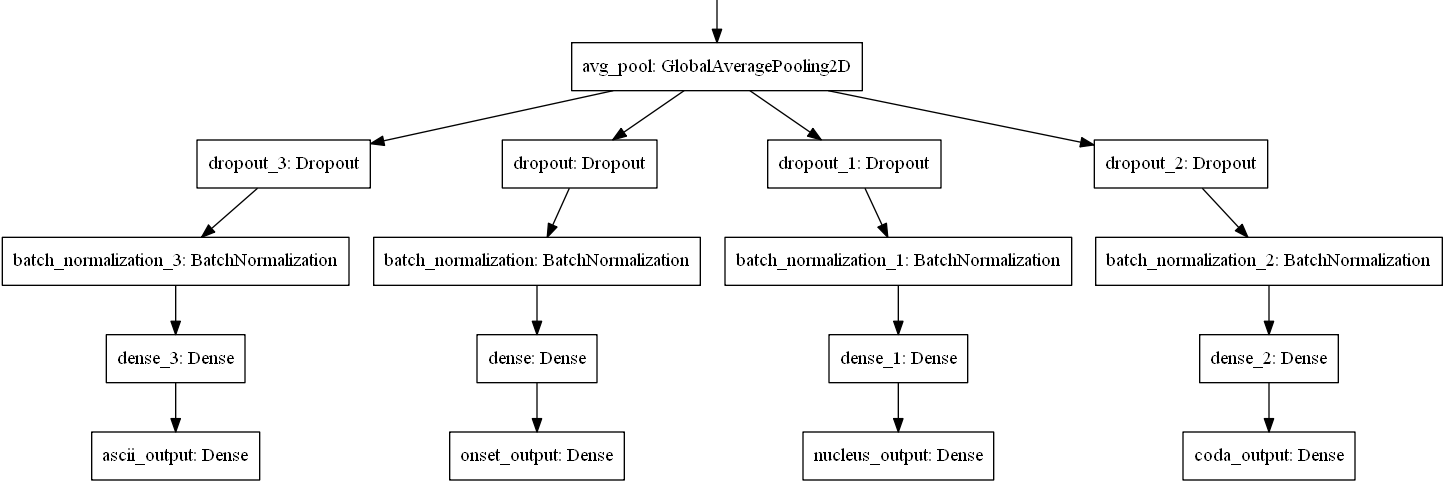
선행 연구 [1]에서, 딥러닝 모델이 한글 낱 글자를 구분하는데 좋은 성능을 거두지는 못했지만 (67.6%), 한글 음절의 구조를 인식하는데 뛰어난 성능을 거두었음을 확인하였습니다. 이 점에서 착안하여, 한글 글자 인식 문제를 하나의 Classification 문제로 해석하는 것이 아니라, 초성 – 중성 – 종성에 대한 Multi-Labeled Classification 문제로 해석하여 재구성하였습니다.

곧, 각 초성 – 중성 – 종성을 인식하는 3개의 독립된 Output Model을 구성하였습니다. 각각의 Output Model을 동시에 학습하며, 한글을 인식할 때에도 각 Output Model의 결과값을 결합하여 최종 결과를 산출합니다. 그 결과 한글 낱 글자 인식에서도 좋은 성능을 거둘 수 있었습니다.

* 참조 논문

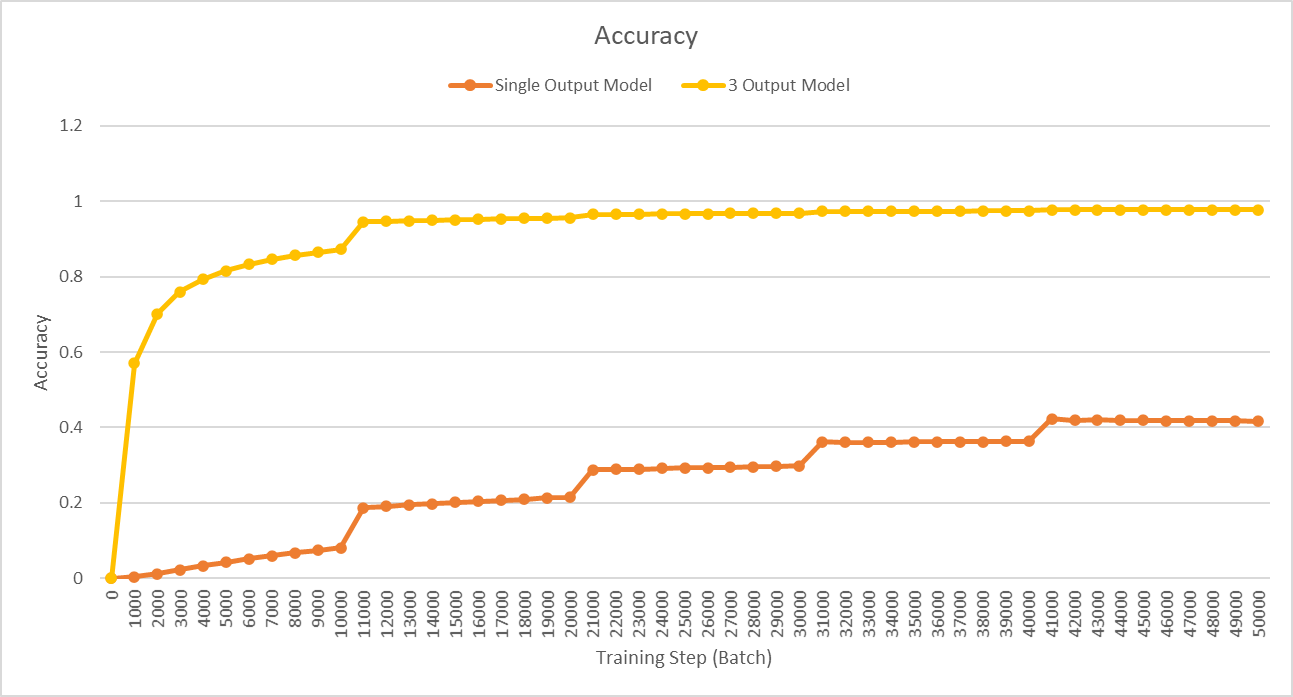
[1] Hangul Handwriting Recognition using Recurrent Neural Networks, 2017

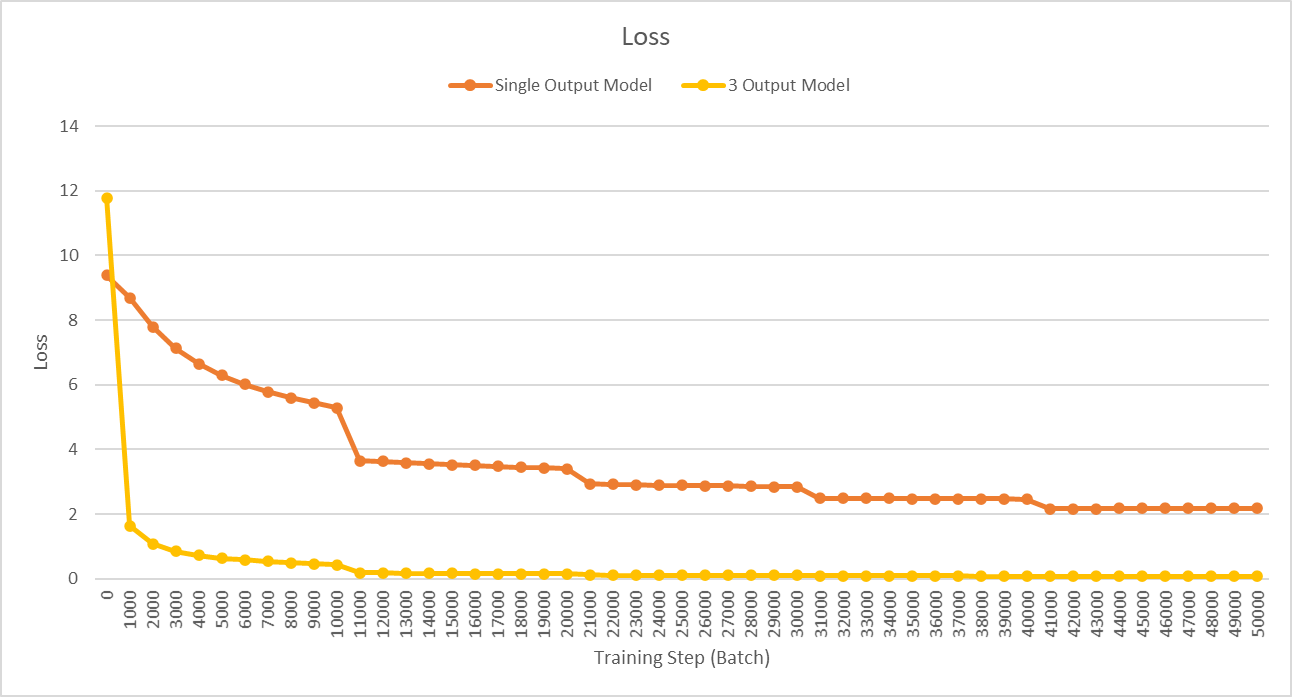
[2] Handwritten Hangul recognition using deep convolutional neural networks, 2014



[Output Model 구조]

* Output Model 은 정규화를 위한 Drop Out 및 Batch Normalization Layer를 포함하고 있습니다.
* Output Model 비교





* Single Output Model: 11,172 개의 Classification 수행
* 3 Output Model: 각각 20, 22, 28개의 Classification 수행

(각각 초성 – 중성 – 종성 19, 21, 27개에 공백 분류 추가)

* 3 Output Model의 Accuracy는 각 Output Model의 Accuracy의 곱
* Single Model은 Backbone Model을 학습하지 않았다. (Transfer learning)
* 복잡한 Image Augmentation 적용 또는 Backbone Model 학습 시 Single Model은 적절하게 학습되지 않았습니다.
  + 1. Backbone Model – Resnet (50 layer)

Backbone Model은 입력 이미지로부터 특징을 추출하는 기능을 수행합니다. 한글 인식의 경우 입력 이미지의 크기는 비교적 작은 편이지만, 11,172자를 구분하는 작업을 수행하기 위하여 깊은 신경망 학습이 필요하다고 판단했습니다. 해당 소프트웨어에서의 Backbone Model로는 50 layer의 Resnet을 사용했습니다.

* Resnet

Resnet은 2015년에 제안된 CNN 구조의 딥러닝 모델 입니다. Resnet은 degradation 의 해결책으로, 딥러닝 모델이 Identity Mapping을 쉽게 학습할 수 있는 구조를 제안합니다. (직관적으로, 딥러닝 모델이 Identity Mapping을 쉽게 학습할 수 있다면 최소한 신경망이 깊어지면서 성능이 감소하는 degradation 현상이 발생하지 않을 것 입니다.)

Resnet은 Skip Connection을 이용하여 딥러닝 모델이 블록 단위의 잔차를 학습하도록 구조를 변경하였습니다. 만약 특정 블록이 Identity Mapping을 학습하고자 한다면, 단순히 잔차가 0이 되도록 학습하면 됩니다. 결과적으로 Identity Mapping을 쉽게 학습하여, 깊은 신경망에서도 학습이 원활하게 진행될 수 있는 성과를 거두었습니다.

* Deep Residual Learning for Image Recognition, 2015

Backbone Model로 사용된 Resnet은 Imagenet 데이터로 학습된 pre-trained Model입니다. keras에서 제공하는 built-in model을 이용하여 구현되었습니다. 총 학습 parameter의 개수는 23,534,592개입니다.

Backbone Model의 Output은 global average pooling을 이용하여 1차원 벡터로 변환됩니다. 해당 1차원 벡터는 개별 Output Model의 입력 값으로 사용됩니다.

* 1. 학습 모델 구현
     1. Built-in Model in Keras

Backbone Model로 사용된 Resnet은 Keras-Applications 라이브러리의 built-in model로 구현되었습니다.

* <https://keras.io/api/applications/>

사용된 50 layer Resnet은 Imagennet validation dataset에서 0.749의 Top-1 accuracy, 0.921의 Top-5 accuracy를 달성하였으며, 해당 pre-trained model을 직접 Backbone Model로 사용하였습니다.

* + 1. Functional Model API

3개의 Output Model을 동시에 학습시키기 위하여, Keras의 Functional Model API를 이용하여 구현하였습니다. Loss는 각 Output Model Loss의 가중 합 (weight = 1.0)으로 계산되며, Accuracy는 각 Output Model Accuracy의 곱(Output Model간의 독립 가정)으로 계산됩니다.

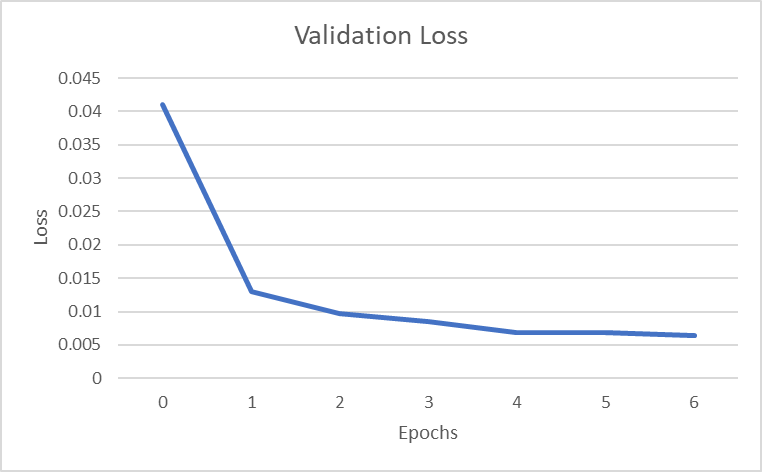
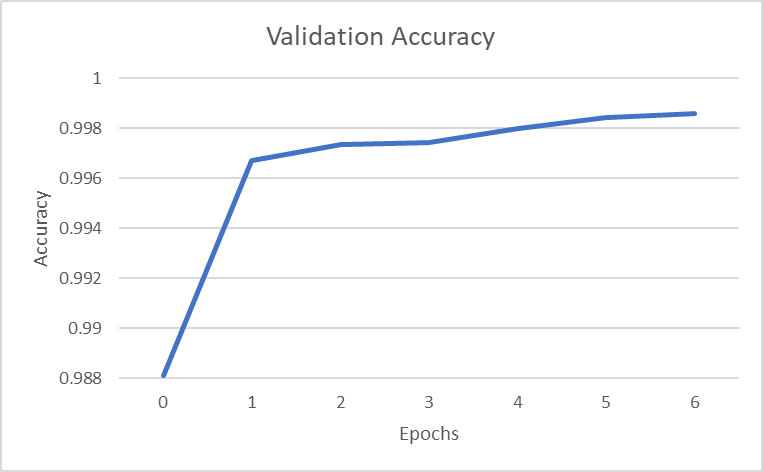
3개의 Output Model과 Backbone Model 전부 동시에 학습하였습니다.

* + 1. 모델 및 가중치 저장

모델 구조와 학습된 parameter는 하나의 HDF (Hierarchical Data Format) 형식 파일로 저장됩니다. Python 개발 환경에서는 h5py 라이브러리를 이용하여 손쉽게 HDF 파일을 읽어, 저장된 모델을 불러올 수 있습니다.

1. **결과**
   1. Validation Set

학습 평가를 위한 validation set은 전체 689,067개의 학습 데이터 중 5%에 해당되는 약 34,000개의 학습 데이터를 무작위 추출하여 구성하였습니다.



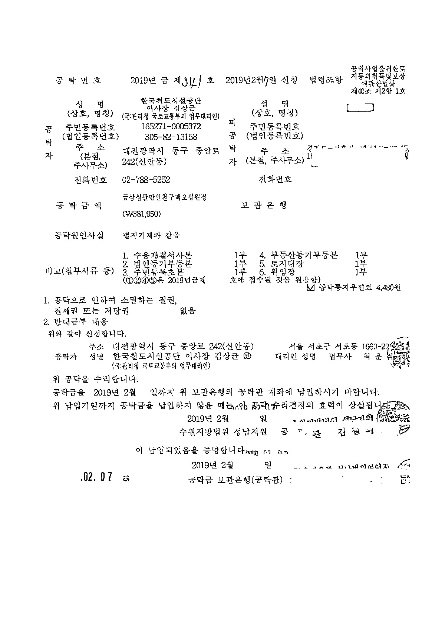
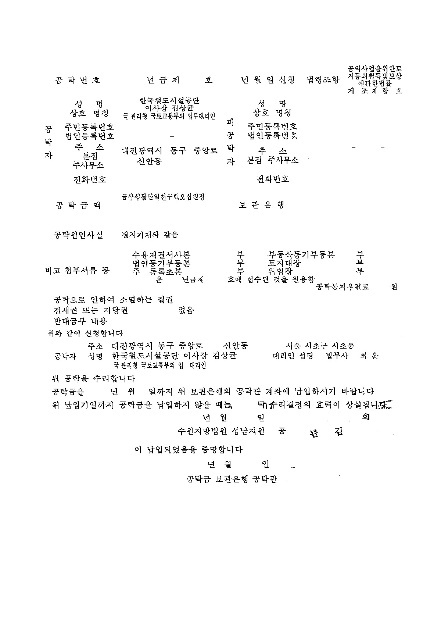
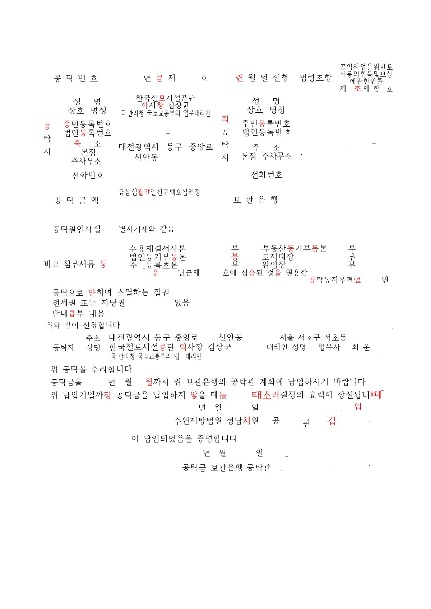
[Validation Accuracy / Loss]

총 6 Epoch 학습 결과, validation set에서 한글 낱 글자를 인식할 때 99.858% accuracy를 가질 수 있었습니다.

* Validation Accuracy 와 Loss 또한 마찬가지로, 3 Output Layer Accuracy의 곱, Loss의 가중 합으로 계산되었습니다.
  1. 문서 스캔 이미지

학습된 모델이 일반적인 환경에서도 한글 인쇄체를 정확하게 인식하는지 확인하기 위하여, 추가적인 Test 환경을 구성하였습니다.

임의의 스캔 된 문서 이미지로부터 추출된 473개의 한글 음절을 Test Set으로 구성하여 정확도를 측정한 결과, 91.75%의 정확도를 얻을 수 있었습니다.

[원본 이미지] [테스트 이미지] [테스트 결과]

* 학습용 한글 음절 이미지는 가로, 세로가 74 pixel 크기인 반면, 문서로부터 추출된 문자 이미지는 약 15~20 pixel 크기입니다.
* 문서로부터 추출된 문자 이미지를 Binary Image로 변환하는 Threshold 값에 따라서 결과가 크게 변합니다. (잘못된 Threshold 값이 지정될 경우, 문자의 형태가 지나치게 훼손됩니다.)
* 문서의 특성 상, 특정 문자가 반복하여 등장합니다. (정확한 테스트를 위해서는 더 많은 테스트 문서 데이터가 필요합니다.)
* 인식 실패 사례

 Label: “년”, Predict: “련”  Label: “등”, Predict: “둥”

 Label: “피”, Predict: “픠”  Label: “만”, Predict: “먄”