딥러닝을 활용한 지문자 인식 및 음성화 (TTS)

1118 이현수, 1314 이건희

지도 교사 : 송석리

Big Data를 이용한 Deep learning은 현대 정보 분야에서의 유망 분야 중 하나이다. 이 중에서도 Tensorflow는 딥러닝을 보다 잘 구현할 수 있게 해 주고, 이와 CNN Algorithm을 이용해서 이미지 등을 인식하는 기술이 급증[1] 하고 있다. 이 연구는, 다량의 이미지를 쉽게 압축하고 분류할 수 있는 CNN을 활용하여 기존에 연구되어 있지 않은 '한글 지문자 인식' 에 대해 연구하였다. 한글 지문자들을 학습시키고 인식시켜 그 문장을 음성으로 변환해주는 (TTS) 프로그램을 개발하였다. 이와 관련하여 그 인식률을 높이는 방법에 대해서도 고찰을 진행하였으며, 앞으로 연구를 확장해 영상에서 이미지를 추출해 인식하는 기능까지 구현할 수 있을 것이다. 이 연구의 결과물을 통해, 장애인과 비장애인과의 거리감이 줄어들고 보다 소통이 원활해지는 선순환을 일으킬 수 있다.

1. 서론

1.1. 연구 동기

최근, 정보 분야에서 빅 데이터와 딥러닝이 많은 인기를 받고 있으며, 이 분야들에 대해서 연구들도 활발하게 진행되고 있다. 딥러닝의 경우 활용될 수 있는 곳이 상당히 많기 때문에, 앞으로도 유망한 분야로 각광받고 있다.

본 연구의 연구원들은 딥러닝을 통해 청각 장애인들이 쓰는 지문자를 인식하고, 그 결과를 출력해 음성으로 변환시켜 주는 모델을 설계하였다. 지문자는 다른 수어들보다 보다 간단하였고, 한글의 모든 자음 (쌍자음 포함)과 모음을 표현할 수 있기 때문이었다.

지문자란, 수화보다 비교적 간단하고 단순한, 손을 이용한 문자이다. 아래는 그림.1. 은 한글 지문자 표현법이다. 수화로 표현할 수 없는 말에 사용되며, 주로 고유명사나 외래어를 표현할 때 사용된다.

대부분의 청각 장애인들이, 지문자를 애용한다고 한다. 그 이유는 청각 장애인들에게는 비장애인들이 사용하는 일반 문자보다, 지문자가 훨씬 사용하기 편할뿐더러, 익숙하게 느껴지기 때문이다.

따라서, 청각 장애인이 수어 (지문자) 로 표현하는 말을 비장애인들이 쓰는 말로 번역해 청각장애인들의 의사소통을 보다 원활하게 하였으며, 이 과정에서 CNN 알고리즘을 사용하여서 이미지 인식을 조금 더 간단하게 구현하였다. 기존 연구들의 경우 한글 지문자 인식은 거의 구현되어 있지 않았기에, 이런 동기까지 더해져, 연구를 시작하게 되었다.

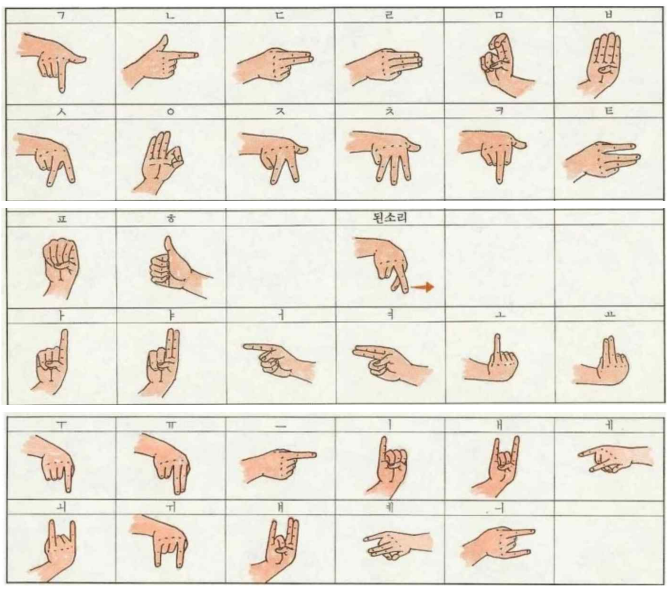

그림. 1. 한글 지문자 자음, 모음 표기법 [1)]

1.2. 연구의 필요성

이 연구를 통해서, 청각 장애인들의 의사소통을 조금 더 원활하게 만들 수 있다. 연구를 통해 개발한 프로그램은 청각 장애인들이 지문자로 자신이 하고 싶은 말을 표현하면, 이를 자동으로 인식하여서

1) 출처 https://goo.gl/413ZuH

문장으로 변환하고, 비장애인들이 사용하는 언어(음성) 으로 자동 변환해주는 과정을 모두 포함하고 있다. 결론적으로, 장애인과 비장애인과의 거리감 역시 줄어들 수 있다.

2. 이론적 배경

2.1. 딥러닝 기본이론

가. Deeplearning

딥러닝이란 주어진 데이터들을 스스로 학습하고, 그 학습 내용을 바탕으로 컴퓨터 스스로 생각 및 사고 (판단)을 하여서 알맞은 결론을 만들어 내는 것이다. 대표적으로 딥러닝을 구현한 예로는 구글의 '알파고' 가 있으며, 바둑 기수를 지속적으로 학습하여서 자신이 이기기 위해서 바둑을 어디에 두어야 할지 판단한다.

즉, 딥러닝은 머신러닝을 실현하기 위한 하나의 방법이다. 딥러닝을 활용한 연구들로는 사진 인식, 음성 인식, 글자 인식, 자동 번역 및 단어완성 등이 있다.

나. 신경망 설계

딥러닝 알고리즘의 구조는 기본적으로 '인공 신경망 (NeuralNetwork)' 로 이루어져 있다. 신경망은 입력 노드로부터 데이터들이 들어온 후, 히든 노드에서 학습을 진행하고 판단을 세운 후, 출력 노드에서 최종적인 값을 출력한다.

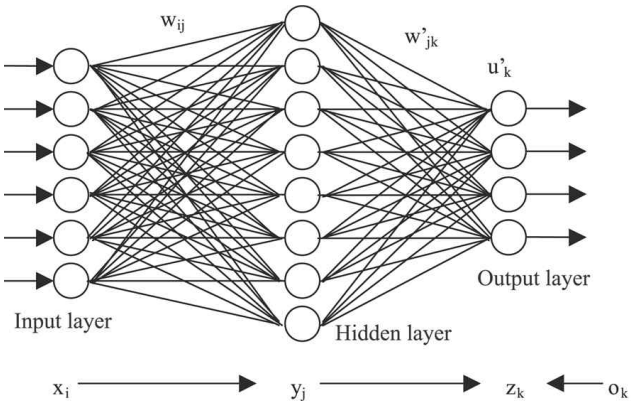

그림. 2. 인공 신경망의 구조 [2]

신경망은, 학습을 통해 신경망이 만들어낸 추측값과 정답의 일치율을 높여야 하므로, 그 차이 (오차)를 최소화 시키는 과정을 따라 딥러닝이 진행된다.

오차 최소화를 위해 사용하는 방법으로는 '최소 제곱법' 이 가장 널리 쓰이는데, 이는 주로 비교적 적은 양의, 단순한 데이터들을 이용해서 그 관계를 찾을 때 사용된다. 예시로 주어져 있는 아래 데이터에서 x와 y는 선형 관계를 가지고 있다고 가정하고, 그 관계식을 $y = ax + b$라고 해 보자.

표1. 선형 관계를 가지는 데이터 예시

x	y
1	2
2	5
3	7
4	9
5	13
6	15
7	17
8	20

먼저 아래와 같은 수식을 통해서 기울기와 절편 a, b의 값을 정할 수 있다.

$$a = \frac{\sum_{k=1}^{n}(x_k - x_{평균})(y_k - y_{평균})}{\sum_{k=1}^{n}(x_k - x_{평균})} \cdots (1)$$

$$b = y_{평균} - a \times x_{평균} \cdots (2)$$

이때 기울기 a의 값을 '가중치(W)' 라고 하고, 절편의 값 b를 '편향' 이라고 한다. 가중치와 편향이 변수별로 다르다면 가중치 행렬과 편향 행렬을 생각할 수 있는데, 이때의 선형 관계식은 아래와 같이 나타낼 수 있다.

$$y = x \cdot W + b \cdots (3)$$

즉, 행렬곱으로 표현할 수 있다.

다음으로는 오차를 구해서 그 오차를 최소화 시키는 과정에 대한 설명이다. 데이터 양이 많거나 그 관계가 복잡할 경우 (3) 만으로는 가중치와 편향을 결정할 수 없다. 따라서 오차를 계산해야 하는데, 오차로는 '평균 제곱근 오차(RMSE)' 를 가장 많이 사용한다.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - Y_i)^2} \cdots (4)$$

이때 X_i는 예측 값을, Y_i는 실제 값 (정답)을 의미한다.

오차를 줄여가는 방법으로는 '경사 하강법' 을 이용한다. 아래 절에서 소개하였다.

다. 경사 하강법

경사 하강법이란, 매우 조금씩 그 값 (W, b)을 바꿔 가면서 오차가 최소가 되는 최적의 점을 찾는 방법이다. 그래프에서 보자면, x좌표를 조금씩 바꿔 가면서 최솟값을 가지는 점을 찾는 과정이다. 해당 점에서 미분 계수를 구해서, 그 기울기 부호의 반대 방향으로 이동한다. 예를 들어, 해당 점에서의 미분계수가 양수라면, 음의 방향으로 ($-x$ 방향) 조금 더 이동하고, 이동한 점에서 다시 미분을 해서 이 과정을 반복하는 것이다.

2) 출처 https://goo.gl/EgJorY

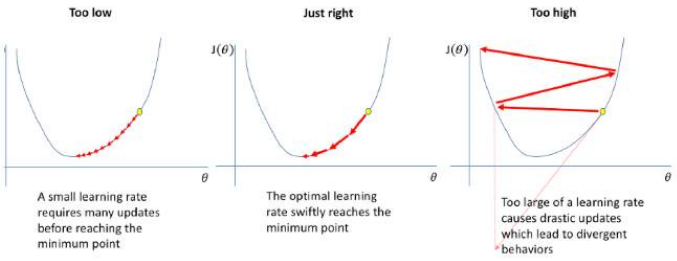

그림. 3. 경사 하강법의 원리 [3]

이때 그래프 상에서 '얼마만큼' 움직일지를 정하는 변수가 바로 '학습률(learning rate)' 이다. 학습률이 크다는 것은 이동 폭이 크다는 것이고, 학습률이 작다는 것은 그 반대이다. 적절한 값의 학습률을 설정해야 오차를 최소화 할 수 있다.

2.2. CNN

CNN이란, Convolution Neural Network의 약자로, 이미지 인식 등을 보다 효율적으로 도와주는 딥러닝의 한 방법 중 하나이다. 기존 신경망 알고리즘을 이용하면 사진 데이터 (3차원)을 1차원 평면배열로 변환시키는 과정에서 데이터 손실이 일어나게 되는데, 이를 보완해서 그 정확도를 높이는 것이 바로 CNN의 핵심 원리이다. CNN 알고리즘의 크게 컨볼루션 계층, 풀링 계층, 그리고 판단 및 출력계층으로 구성된다.

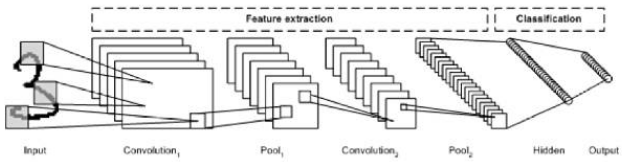

그림. 4. CNN의 원리 [4]

먼저 하나의 이미지가 입력되면 (Input), '컨볼루션 계층(Convolution$_{1,2}$)' 에서 이 이미지의 특징적 정보들을 추출해 내어서 압축시킨다. 이후 한번 더 압축 과정을 거치는데, 이것이 바로 '풀링(Pool$_{1,2}$)' 이다. 주로 '맥스 풀링' 기법을 이용해서 풀링을 진행하며, 이 연구 역시 '맥스 풀링' 기법을 사용하였다. 이 과정을 설정한 노드 개수만큼 반복하여서 최종적으로 분류를 진행한 후 (Classification), 판단을 내린다.

2.3. Dropout (드롭아웃)

적은 데이터로 학습을 시도하면 그 정확도가 매우 낮다. 데이터 양이 부족하기 때문에 신경망이 충분한 학습을 진행하지 못하였기 때문이다.

하지만, 그렇다고 해서 너무 많은 데이터가 있어도 '과적합 (Overfitting)' 이 발생하여서 정확도가 낮아진다. 학습 횟수 역시 마찬가지다. 과도하게 학습이 진행되어서, 신경망이 판단에 혼란을 겪는 것이다. 즉, 적절한 양의 데이터와 학습 횟수를 설정해야 최적의 정확도를 얻을 수 있다.

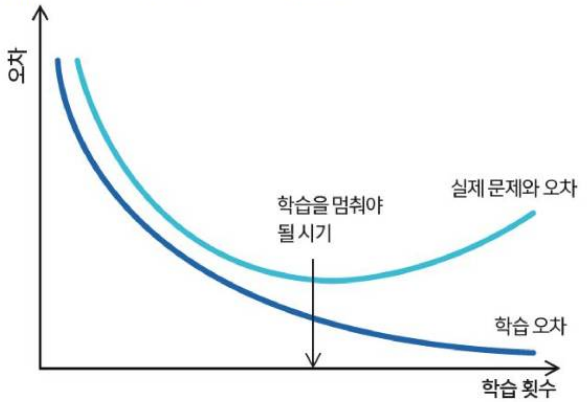

그림. 5. 학습 횟수가 너무 많아지면 오히려 오차가 증가하는 과적합 현상 [5]

과적합을 방지하기 위해서는, '드롭아웃' 기법이 사용된다. 이 드롭아웃 기법은, 학습 데이터의 양이나 학습 횟수가 과적합을 불러올 정도로 많아도 자동적으로 줄여 준다. 즉, 데이터 중의 일부분만 다음 학습 단계로 가져가는 것이다. 이 연구에서도 드롭아웃 기법을 이용해서 과적합을 방지하였다. (<3. 연구 방법> 참고)

2.4. Activation Function (활성화 함수)

신경망에서 입력 노드에서 입력된 데이터들은 특정한 함수를 거쳐 처리된 후 출력된다. 이때 이 처리를 담당하는 함수를 활성화 함수라고 한다.

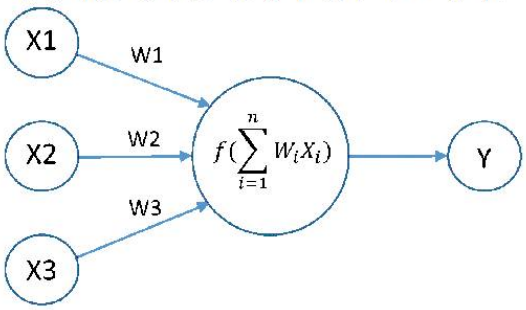

그림. 6. 활성화 함수의 작용과정 [6]
위 그림에서 f가 활성화 함수이다. 함수 f는 Input 데이터 (X · W)를 입력받아서 처리한 후, Output 데이터 (Y)로 반환시킨다.

활성화 함수는 주로 비선형 함수이다. 신경망에 입력되는 노드(신호) 는 비선형이며 아날로그 식이기 때문이다. 활성화 함수는 여러 가지 종류가 있는데, 대표적인 종류는 아래와 같다.

3) 출처 머신러닝단기집중과정
4) 출처 https://goo.gl/fkphjL
5) 출처 http://dongascience.donga.com/news.php?idx=11225
6) 출처 https://goo.gl/6zG4UK

가. 계단 함수

계단 함수란, 입력값이 음수일 때는 0을 반환하고, 입력값이 양수일 때는 1을 반환하는 함수이다.

$$f(x)=\begin{cases}0 & (x<0)\\ 1 & (x\geq 0)\end{cases} \cdots (5)$$

또는 아래와 같이 표현할 수도 있다.

$$f(x)=\max\left(0,\frac{x}{|x|}\right) \cdots (6)$$

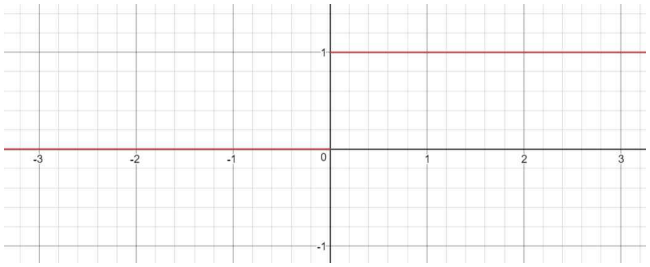

그림. 7. 계단 함수

① 계단 함수는 최초의 활성화 함수로,
② $x=0$에서 불연속이고 함숫값이 0, 1 2개밖에 존재하지 않기 때문에, 많은 데이터들을 다양하게 분류하기에는 부적합할 수 있다. ③ 2진 분류의 경우 계단 함수를 이용한다.

이 연구의 경우, 출력 계층이 다양하고 입력 데이터의 양 역시 많기 때문에 계단 함수는 부적절하다고 판단하였다.

나. 시그모이드 함수

위의 계단 함수의 문제점을 보완하기 위해 제안된 활성화 함수가, 지수함수 꼴로 표현되어 있는 시그모이드 함수 (Sigmoid Function) 이다. 이 함수의 함숫값은 $(0,1)$ 내의 임의의 값을 가진다. 그 수식은 아래와 같다.

$$f(x)=\frac{1}{1+e^{-x}} \cdots (7)$$

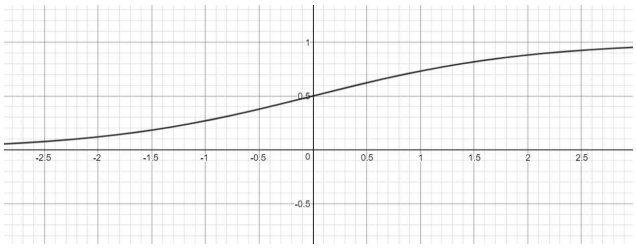

그림. 8. 시그모이드 함수

하지만, 시그모이드 함수 역시 여러 가지 문제점이 있다. 먼저, ① 점근선 $y=0, y=1$에 가까워 질수록 변화량 Δy가 작아지기 때문에, 그 학습이 매우 느리게 진행된다. 또한, ② 결과값이 항상 양수이기 때문에 오차 계산 시 (역전파) 부호 문제 역시 발생하게 된다. ③ 수식 역시 다른 함수에 비해 복잡해서, 컴퓨터가 계산하는 시간도 오래 걸리게 되고 그 결과 학습이 느려진다.

다. 임계논리 (ReLU) 함수

최종적으로 개발 된, 매우 간단한 활성화 함수가 ReLU 함수이다. 입력 값이 양수일 때는 그 값을 그대로, 음수일 때는 0을 반환한다.

$$f(x)=\max(0,x)=\begin{cases}x & (x>0)\\ 0 & (x\leqq 0)\end{cases} \cdots (8)$$

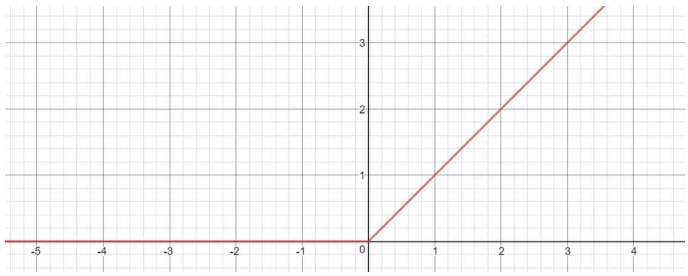

그림. 9. 임계논리 함수

ReLU 함수는 다양한 장점을 가지고 있다. 먼저, ① 그 모양과 식이 매우 간단하기 때문에 계산 시간이 줄어든다. ② $x\geqq 0$일 때 선형이기 때문에, 시그모이드 함수에 비해 Δy가 작아지는 문제가 발생하지 않고, 따라서 학습이 느려지는 일도 발생하지 않고, 효율이 증가한다.

하지만, 이에 비해 단점 역시 가지고 있다. 바로 ③ 출력값이 음수여서 다음 단계의 입력값이 계속 음수인 경우, 함숫값이 계속 0이 되는 문제가 발생할 수 있다는 것이다. 이 경우 가중치 $W=0$이 되어서 학습이 더 이상 진행되지 않을 수 있다.

2.5. 선행 연구 논문

선행 연구 논문의 경우 한글 지문자 인식에 관한 내용은 없었으며, 주로 찾아본 논문들은 '숫자 인식' 또는 '사물 인식' 을 위주로 진행한 논문들이었다 (해당 내용은 논문리뷰에도 포함되어 있다).

하지만 이 논문들의 경우, 인식보다는 손 영역 추출과 손 모양 그 자체를 추출하고 YCbCr을 이용하고, 고급진 기능을 구현하는 경우 HMM (Hidden Markov Mode)을 사용하는 등[2][3] (SLR 논문[4]의 경우에도 HMM을 이용하였다 우리가 진행하려는 연구와는 방향에서 차이가 있다고 판단하였다.

따라서 딥러닝을 공부할 때 사용했던 자료들[8][9] 이나 온라인에서 얻을 수 있는 교육 자료들, 그리고 간단한 CNN 논문들[6][7]을 참고하기로 하였고, 그 결과 여러 가지 딥러닝 설명서와 온라인 사이트들 및 github을 이용 [10][11][12] 해서 딥러닝을 공부하고 모델을 구현하였다.

선행 연구 논문들은 현재 이 연구에서 구현하는 모델과는 관련성이 크지 않더라도, 향후 과제에서는

클 수 있다. 실시간[2] 또는 CNN을 이용해서 영상[5]에서의 인식을 다룬 논문도 있었고, 임의의 배경에서 손을 추출해서 학습 시키는 기능을 추가할 때도 역시 이와 비슷한 연구를 진행한 논문[3] 역시 존재하였기 때문이다.

3. 연구의 목적

이 연구는 최종적으로 청각 장애인들의 지문자를 인식해서 문장으로 출력해 주고, 이를 음성으로 읽어 주어서 (Text To Speech, TTS) 비장애인들과의 소통을 원활하게 하려는 목표를 가지고 있다. 딥러닝과 CNN, Tensorflow를 이용해서 이를 구현하였으며, 결론적으로 장애인, 비장애인 간의 언어 장벽을 없앨 수 있을 것이다.

4. 연구 방법

본 연구는 2018년 10월 22일부터 2018년 11월 5일까지 진행되었다.

4.1. 개발 환경

이 연구에서 최종적으로 목표하는 것은 사진 인식과 TTS 기능이다. CNN 알고리즘이 가장 적합하다 판단하였기에, Python에서 Tensorflow를 이용하여서 개발하기로 하였다. 아래는 정확한 개발 환경이다.

Windows - 64bit Anaconda, Tensorflow (텐서플로) 기반 Python 3.7 Pycharm, Jupyter Notebook 동시 사용

전체 코드는 논문 맨 뒤 <부록>에 첨부하였으며, 자세한 결과물들은 GitHub에 업로드 하였다. GitHub Url 역시 <부록> 에 첨부되어 있다.

4.2. 인공신경망 구성

가. 신경망 구성

이 연구에서 우리가 구성한 신경망 (Neural Network)을 소개하도록 하겠다. 크게

이름	순서	구성 (개수)
입력 노드	↓	28×28 1층
히든 노드 1	↓	→ 28×28 32층 → 14×14 32층
히든 노드 2	↓	→ 14×14 64층 → 7×7 64층
히든 노드 3	↓	→ 256개
출력 노드	↓	→ 31개

로 구성되어 있다. 입력 노드에는 28 × 28 size의 이미지가 입력된다. 구체적인 내용 및 처리방법은 아래 <나. 학습 알고리즘> 에 소개하였다.

나. 학습 알고리즘

이 연구에서는 신경망을 학습시키는 알고리즘으로 CNN을 선택하였다. 그 이유는, 많은 기존 논문들이 이미지 및 영상 인식을 CNN을 이용해서 진행하고 있어 이미지 인식 분야에서 가장 대표적으로 쓰이는 방법 중 하나이고, 실제로 그 알고리즘의 동작 역시 이미지 압축, 특징 추출 등에 특화되어 있기 때문이다. 따라서 우리는 손 모양의 특징을 잘 추출해 낼 수 있는 CNN을 통해서 그 정확도를 가장 높일 수 있다 생각하였다.

CNN 에서는, 컨볼루션은 3×3의 크기를 가진 커널들을 이용하였다. 이는 내장 함수 conv2d를 이용하였다. 이후 풀링 시에는 strides를 [1, 2, 2, 1] 로 설정해서 2×2 맥스 풀링을 진행하였고, 역시 내장 함수인 nn.max_pool을 사용해서 구현하였다. 이와 같은 과정을 총 2번 반복하였고, 결론적으로 256개의 1차원 배열로 변환하였다. 마지막으로 31개의 출력 계층을 거친다. 한글 자음과 모음 (쌍자음 제외)의 개수가 31개이기 때문에, 출력 노드의 개수는 31개로 설정하였다. 아래 그림은 우리가 설계한 신경망의 처리 과정을 정리한 것이다.

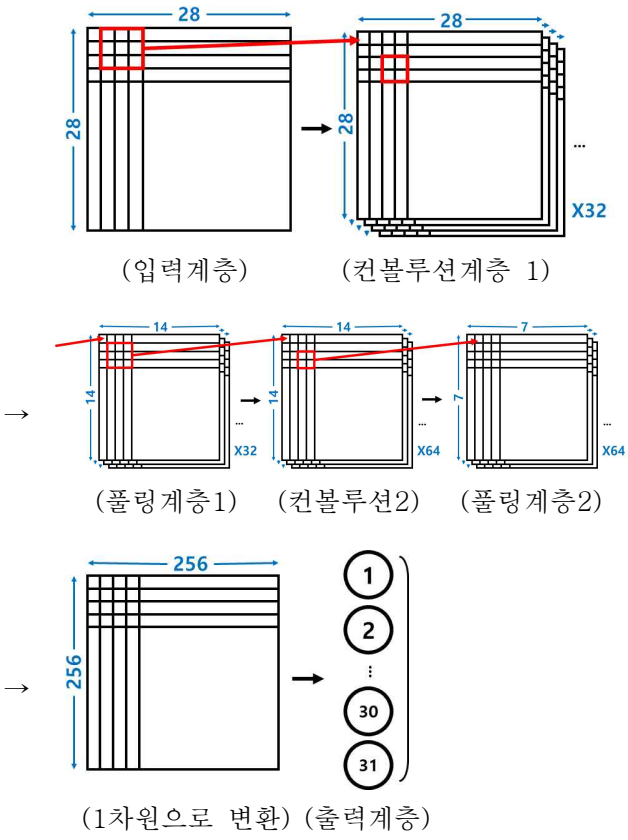

그림. 10. 설계한 신경망의 이미지 처리과정 도식화

다. 활성화 함수 및 오차함수

앞서, <2.4 Activation Function>에서 활성화 함수의 종류에 대해 설명하였다. 이 함수들 중에서, 우리는 Relu Function (임계논리 함수)를 사용하기로 하였다.

그 이유는, 다른 활성화 함수에 비해 훨씬 안정적이고 학습 속도 역시 빨라서, 많은 데이터를 다루는 이 연구에 가장 효율적이라는 판단을 내렸다. 결론적으로 이 연구에서는 활성화 함수로 Relu 함수를 사용하였다.

비록, 출력값이 음수여서 다음 단계의 입력값이 계속 음수인 경우, 함숫값이 계속 0이 되는 문제가 발생할 수 있지만, 이 연구에서는 이 경우를 검사하여서 가중치가 0이 되는 경우를 방지하였다. 자세한 내용은 <4. 정확도 높이기> 에 소개하였다.

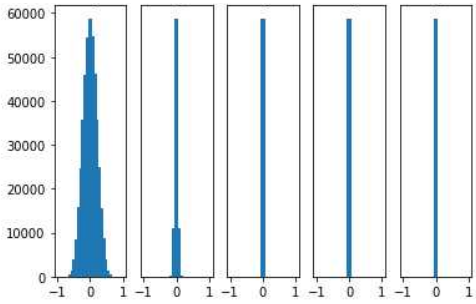

그림. 11. 갈수록 학습률이 0으로 줄어드는 문제점.[7]

오차 함수의 경우, '교차 엔트로피 (CEE)' 를 사용하였다. 교차 엔트로피 함수는 전형적으로 딥러닝에서 사용되는 함수로, 그 계산은 아래와 같다.

$$COST = -\sum_{i=1}^{n} t_k \log o_k \cdots (9)$$

여기서 t_k, o_k는 각각 신경망이 추론한 답과 실제 정답을 의미한다.

4.3. 모델 개발

우리는 최종적으로 원하는 결과물을 얻기 위해 다음 절차에 따라서 연구(개발)을 진행하였고 검토 역시 병행하였다. 아래는 본 연구 시 설치하거나 불러온 library 목록이다.

```
import tensorflow
import os
import cv2
import numpy

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from numpy import array
from gtts import gTTS
```

① 데이터 생성하기

: 한글 지문자 데이터의 경우, 빅 데이터를 얻을 수 있는 사이트 (Kaggle.com) 에서도 찾을 수 없었다. 따라서 직접 촬영하기로 하였으며, 검정색 바탕을 배경으로 모든 지문자에 대해서 촬영 하였다.

또한, 사람마다 손 모양이나 색깔 등이 약간씩 다르므로 연구자 2명 외에도 주변인의 손 사진 역시 찍어서 학습 데이터에 수시로 추가함으로써 그 정확도를 높였다.

② 이미지 읽어오기 및 처리

: 기존 CNN 알고리즘에서 손글씨를 인식할 때는, Tensorflow 에서 기본적으로 MNIST 데이터를 제공하였기 때문에 이미지를 변환할 필요가 없었다. 하지만 앞에서 언급한 대로, 현재 인위적으로 데이터를 생성하였기 때문에 (직접 촬영), 신경망의 input node 에서 이를 인식 및 변환하기 위해서는 28pixel × 28pixel 사이즈로 바꾸어야 한다.

㉠ 촬영한 사진을 28pixel × 28pixel 로 변환한다(Resize). 이때는 사진 편집 프로그램 (PhotoScape) 을 이용하여서 일괄적으로 변환하였다.

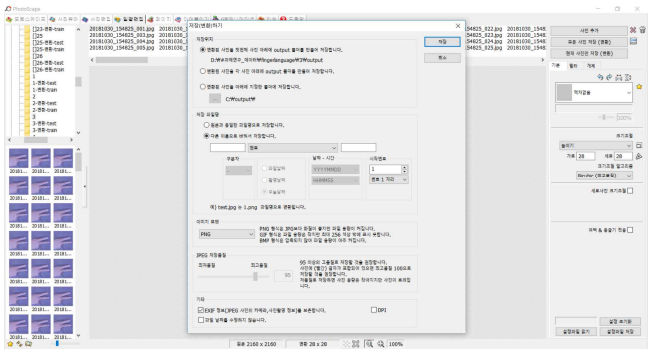

그림. 12. 이미지를 28 × 28 사이즈로 변환하는 과정

그 결과는 아래와 같다. 각 폴더 내에 들어있는 이미지들의 예시이다.

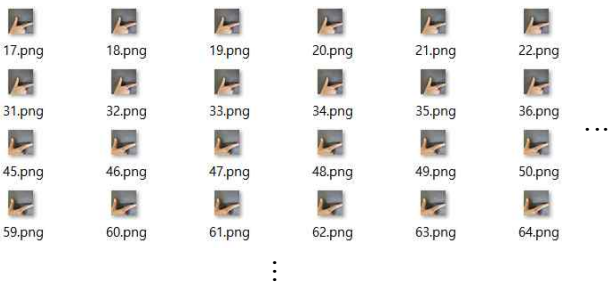

그림. 13. 28 × 28 사이즈로 변환된 이미지들

위 이미지들은 모두 ㄴ을 나타내고 있다.

㉡ Resize 시킨 이미지를 숫자 배열로 변환하고, train data (학습 데이터) 와 test data (테스트 데이터)로 구분한 뒤, 이를 CNN 인식 데이터에 첨부시킨다. 학습 데이터는 input이라는 변수에, 테스트 데이터의 정답은 label 에 저장하였다.

7) 출처 http://excelsior-cjh.tistory.com/177

㉢ 첨부된 사진들을 정답과 함께 배열에 최종적으로 저장하고, 인식을 시작한다.

위 ㉡, ㉢을 위한 핵심적인 코드는 아래와 같다.

코드 1

```
for index in range(len(train_folder_list)):
    path = os.path.join(TRAIN_DIR, train_folder_list[index]) + '/'
    img_list = os.listdir(path)
    for img in img_list:
        img_path = os.path.join(path, img)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        train_input.append([np.array(img)])
        train_label.append([np.array(onehot_encoded[index])])

train_input = np.reshape(train_input, (-1, 784))
train_label = np.reshape(train_label, (-1, output_nodes))
```

이미지를 배열로 변환시킨 결과는 다음과 같다.

```
[[72 74 70 59 61 64 78 164 190 202 180 182 197 186 188 193 194 188 185 177 178 151 56 46 51 54 56 56]
 [72 74 66 61 61 59 93 175 196 195 181 191 198 184 188 195 195 189 182 175 176 166 80 46 53 50 53 56]
 [74 74 66 66 65 59 111 184 199 190 187 199 197 185 190 196 194 190 180 175 175 182 117 43 56 53 53 56]
 [74 74 70 66 66 66 133 189 196 192 194 203 196 190 195 199 195 190 180 175 175 186 149 53 53 53 53 53]
 [74 74 72 66 64 68 158 198 192 194 199 205 197 197 199 201 195 188 183 174 173 182 173 82 46 55 56 56]
 [76 78 68 66 59 78 180 199 187 193 200 208 199 200 202 204 197 189 185 174 174 176 186 127 50 56 59 56]
 [78 78 72 68 59 114 194 192 183 190 200 210 199 204 209 206 200 194 188 179 178 178 192 170 66 53 59 59]
 [80 76 80 70 61 151 203 184 179 190 200 210 195 207 216 205 203 197 190 182 183 181 192 195 85 54 59 61]
 [78 78 82 78 68 158 206 177 170 202 203 206 195 209 212 210 200 198 194 194 189 197 203 206 127 50 61 64]
 [74 76 76 78 68 140 200 160 186 203 206 189 193 214 207 209 198 194 202 194 183 194 206 204 175 68 56 66]
 [76 78 80 80 68 111 189 190 210 206 196 186 199 210 213 207 203 199 194 210 124 84 195 199 197 109 56 61]
 [76 84 80 78 80 80 102 126 212 217 201 179 203 221 217 205 213 197 193 192 59 21 139 185 192 156 70 61]
 [78 76 85 89 84 82 74 66 98 144 162 189 226 223 218 212 209 205 213 146 39 21 53 173 192 180 82 66]
 [76 85 84 84 80 80 78 76 72 46 106 203 236 232 223 206 214 216 200 93 53 43 28 116 200 196 98 61]
 [78 84 82 87 80 80 82 78 78 61 102 195 224 186 184 198 210 214 161 66 68 68 53 66 187 224 151 66]
 [78 82 82 80 80 80 80 78 76 74 59 85 84 46 106 192 208 212 106 64 78 78 74 66 103 204 192 76]
 [80 80 80 80 80 82 80 80 76 70 56 34 28 50 126 187 220 157 70 84 78 84 80 78 72 89 92 80]
 [78 78 78 78 78 78 78 76 74 76 56 39 28 59 145 195 204 120 70 84 84 87 85 84 84 78 74 76]]
```

⋮

(이후 생략)

그림. 14. 28 × 28 사이즈의 배열로 변환된 이미지

㉣ 테스트 데이터에 대해서도 위와 같은 과정을 수행한. 방법은 동일하므로 테스트 데이터에 대해 실행하는 과정은 생략하였다.

③ 학습횟수 (epoch) 설정하기

데이터 개수와 양이 많이 때문에, 미니 배치(batch)를 사용하였으며 총 배치의 개수는 100개로 하였다. 학습 반복 횟수는 일단은 기본적으로 50회로 설정하였다. 이후 이 값을 '40번' 부터 '70번' 까지 변화시켜 나가며 최적의 값을 찾아 나갔는데, 그 과정과 결과는 <4.4 모델 개선> 에 기록하였다. 아래는 학습을 실행하는 코드 중 핵심적인 부분이다.

코드 2

```
all_epoch = 50
        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs, Y: batch_ys})
        total_cost = total_cost + cost_val #오차값 구하기
    print('Epoch:', '%04d' % (epoch + 1), 'Avg. cost = ', '{:3f}'.format(total_cost / total_batch))
```

학습 결과 테스트 데이터로 확인을 하고, 그 정확도를 출력해 준다. 정확도를 출력하는 코드를 포함한 모든 코드는 '부록' 전체코드에 포함되어 있다.

④ 학습률 (learning_rate) 설정하기

학습률은 기본적으로 0.001로 설정해 놓았다. 학습률은 작아질수록 오차가 계속해서 줄어들었다. 하지만, 너무 학습률이 작아지면 학습에 소요되는 시간이 크게 늘어났다. 따라서 가장 적절한 값인 0.001로 설정하였다.

⑤ 인식한 결과 한글로 출력하기

지문자 데이터들 (Test Data)을 인식하였다면, 그 인식한 결과는 현재 숫자로 되어 있으므로 한글로 변환하는 과정이 필요하였다. result라는 변수에 숫자로 만들어진 인식결과를 저장한 후, 이를 dictionary를 이용해서 한글 자/모음으로 변환하였다.

dictionary 구성은 아래와 같이 하였다.

```
char = {0 : 'ㄱ', 1 : 'ㄴ', 2 : 'ㄷ', 3 : 'ㄹ', 4 : 'ㅁ', 5 : 'ㅂ', 6 : 'ㅅ', 7 : 'ㅇ', 8 : 'ㅈ', 9 : 'ㅊ', 10 : 'ㅋ', 11 : 'ㅌ', 12 : 'ㅍ', 13 : 'ㅎ', 14 : 'ㅏ', 15 : 'ㅑ', 16 : 'ㅓ', 17 : 'ㅕ', 18 : 'ㅗ', 19 : 'ㅛ', 20 : 'ㅜ', 21 : 'ㅠ', 22 : 'ㅡ', 23 : 'ㅣ', 24 : 'ㅐ', 25 : 'ㅒ', 26 : 'ㅔ', 27 : 'ㅖ', 28 : 'ㅚ', 29 : 'ㅟ', 30 : 'ㅢ'}
```

그 원리는 다음과 같다. char이라는 dictionary에 'ㄱ' 부터 'ㅢ' 까지 각각의 문자에 대해서 번호를 부여하였다. 인식 결과, 배열에서 가장 높은 값을 가지는 index가 인식 결과이므로 그 index를 key로 가지는 value를 알아내는 것이다.

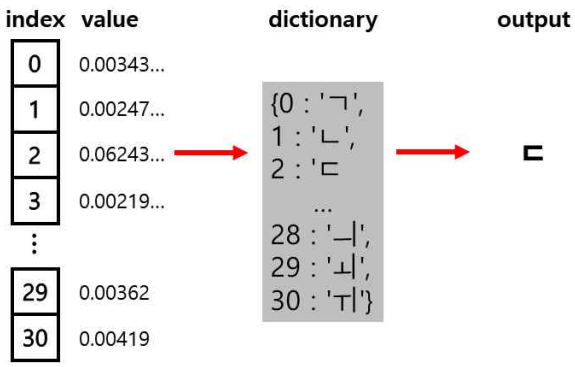

그림. 15. 인식한 이미지 한글로 변환하는 과정 (원리)

⑥ TTS를 위한 과정 - **임의 데이터 인식하기**

이 연구에서 최종적으로 목표하는 바는 임의의 지문자 사진을 이용해 하나의 글자를 합성하고, 그 글자들로 이루어진 문장을 음성으로 읽어 주는 것이다. 따라서 먼저 임의의 지문자 데이터들을 인식시키는 알고리즘부터 설계하였다.

데이터들이 있는 폴더들 중 ttss라는 폴더에 인식을 원하는 순서대로 아래 그림과 같이 데이터들을 추가하였다.

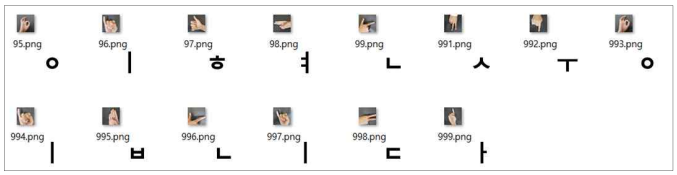

그림. 16. TTS인식을 위한 이미지 데이터들
해당 이미지들은 '**이현수입니다**' 라는 문장을 나타낸다.

그리고 이 사진들을 숫자 배열로 가공 하였는데, 이는 위의 학습 데이터들을 변환시키는 ***코드 1*** 과 일치한다.

하지만, 이미지를 변환시킨 후 하나씩 인식시킨 결과 출력 문제가 발생하였는데, 이미지 인식 결과 그 결과들의 index값이 일정하게 밀려서 출력 되었다. 처음에는 인식 오류하고 생각하였으나, 학습 데이터들을 그대로 인식시켜도 다른 문자로 나오는 것을 보아서 인식 오류는 아니라고 생각하였고, index 처리의 오류라고 최종 판단을 내렸다. 이를 위해서 index를 인위적으로 변환하는 작업을 수행해 주었다.

⑦ TTS를 위한 과정

- 한글 초성, 중성, 종성 합쳐서 한 글자로 만들기

임의의 지문자 사진 데이터들을 인식 하였다면, 그 인식 결과는 초성으로 ('ㄱ', 'ㅏ', 'ㄴ' 처럼) 출력되고 저장된다. 따라서 이들을 하나의 글자로 만들어 주어야 한다.

한글 글자를 합성할 때는, 글자의 시작과 끝을 알아내는 것 역시 필요하다. 예를 들어 인식 결과가 'ㅈㅓㅇㅂㅗ (정보)' 일 때, 'ㅇ' 에서 글자가 바뀐다는 것을 인식해야 한다. 따라서, 글자가 바뀌는 경우를 아래와 같이 2개로 나누어 보았다.

㉠ 모음-자음-자음 순서일 때 :
중성 - 종성(받침) - 초성 순서이다.
㉡ 모음-자음-모음 순서일 때:
중성 - (받침없이) 초성 - 중성 순서이다.

전체 한글 인식 코드는 분량이 매우 길기 때문에 부록에서 첨부하였다. 최종적으로 초성들을 모아 하나의 글자로 합성한 뒤 배열에 저장하고, 그 글자들을 모두 합쳐서 하나의 문장으로 완성하였다.

⑧ TTS를 위한 과정 - **음성파일로 변환시키기**

⑦에서 만든 문장을 python 의 gtts 라이브러리를 통해서 문장을 읽는 음성 파일로 변환하였다. 이로써 최종적으로 지문자 인식부터 TTS 변환까지 코드를 완성하였다. 이 과정을 그림으로 요약한 것을 아래에 첨부하였다. 코드의 결과는 <5. 연구 결과> 에 첨부하였다.

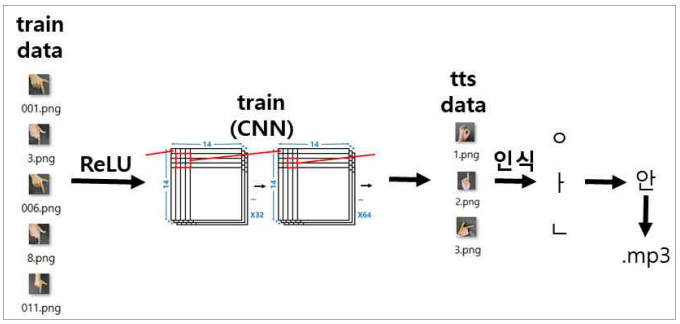

그림. 17. 이 연구의 전반적인 과정

⑨ [향후과제] 영상 인식하기

이 연구를 조금 더 발전 시킨다면, 영상에서 이미지를 추출해 내서, 그 이미지들을 인식시키는 것까지 구현할 수 있을 것이다. (관련 내용은 <6.2 후속연구 제안> 에도 나와 있다). 그 뼈대를 잡아 보았다.

먼저 카메라를 켜서, 영상을 촬영한다. 이후 촬영한 영상은 video.avi 라는 파일로 저장이 된다. 이 영상을 읽어와서, 5프레임 마다 이미지를 추출 (캡처) 해서 저장해 준다.

코드 4

```
cam = cv2.VideoCapture('video.avi')
while(cam.isOpened()):
    ret, image = cam.read()
    fps = cam.get(cv2.CAP_PROP_FPS)
    if (int(cam.get(1)) % 5 == 0):
        cv2.imwrite("frames/frame%d.png" %
cntvid, image)
```

이 코드를 실행하면, 미리 저장해 둔 영상에서 5프레임 마다 자동적으로 이미지를 추출해서 Number 순서로 저장해 준다.

4.4. 모델 개선

모델을 개발한 후, 정확도를 높이기 위해 개선 작업을 실시하였다. 개선 작업의 종류는 여러 가지가 있었는데, 앞서 설명한 ① 학습 반복횟수 조정, 이 외에도 ② 드롭아웃 기법, ③ 데이터 추가 생성 등이 있다.

가. 드롭아웃 (Dropout) 기법사용

데이터와 학습 횟수가 많을 때, 과적합을 방지하기 위해서 드롭아웃 기법을 추가하였다. 다음 계층으로 넘어갈 때 유지할 데이터 비율을 57% 로

설정하였다. 변수 keep_prob = 0.57 로 설정하였다. 따라서 아래와 같이 코드를 수정하였다.

코드 2 수정

```
#드롭아웃 시 사용할 변수 keep_prob
keep_prob = tf.placeholder(tf.float32)
L3 = tf.nn.dropout(L3, keep_prob)
_, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs, Y: batch_ys,
      keep_prob : 0.57})
```

나. 학습 반복횟수 (Epoch) 조정

기존의 반복횟수는 50번이었다. 하지만, 드롭아웃 기법을 적용한 상태에서, 학습반복횟수는 45번, 40번, 45번... 65번, 70번까지 시도해 본 결과 '55번' 이 가장 적당하다는 결론을 내렸다. 학습 횟수에 따른 오차와 정확도를 아래 표와 그래프로 나타내었다. 오차와 정확도는 모두 3번 실행한 결과를 평균 내서 계산하였다.

표2. 학습반복횟수에 따른 평균 오차

학습횟수	오차1	오차2	오차3	Average.
40회	3.136142	3.364775	1.515386	**2.672101**
45회	0.162218	0.065653	0.176199	**0.135690**
50회	0.024073	0.041014	0.031186	**0.032091**
55회	0.020345	0.038316	0.041253	**0.033305**
60회	0.118730	0.017540	0.028536	**0.054935**
65회	0.043643	0.028473	0.019956	**0.030691**
70회	0.018349	0.058489	0.030560	**0.035799**

표3. 학습반복횟수에 따른 평균 정확도

학습횟수	정확도1	정확도2	정확도3	Average.
40회	3.226%	3.226%	58.71%	**21.72%**
45회	100%	100%	100%	**100%**
50회	97.42%	100%	100%	**99.14%**
55회	100%	100%	99.35%	**99.78%**
60회	99.35%	100%	100%	**99.78%**
65회	100%	100%	100%	**100%**
70회	100%	100%	100%	**100%**

이를 그래프로 나타내면 아래와 같다.

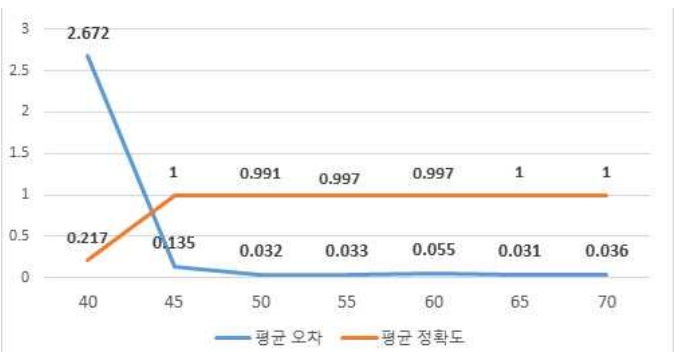

그림. 18. 학습횟수에 따른 평균오차와 정확도

그래프에서 알 수 있듯이, 40회의 경우 오차가 매우 크고 정확도가 매우 낮아 부적격하다고 생각하였다. 가장 좋은 2개의 값은 50과 55이었다. 물론 오차는 65번에서 가장 최소였고 정확도도 1 (100%) 이었지만, 그 차이가 매우 작고 65번의 경우 학습에 소요되는 시간이 너무 길어졌다. 따라서, 50번과 55번 중 정확도가 조금이라도 더 높은 55번을 선택하였다.

다. 데이터 추가생성

또한, 후에도 수시로 손 모양의 사진을 추가 함으로써 추가 학습 / 테스트 데이터들를 생성하였고, 그 결과 기존에는 잘 인식이 되지 않던 사진들도 잘 인식되었다. 이때 추가한 손 사진의 경우 연구원들의 가족이나 지인의 손 사진을 찍어서 첨부하였으며, 다양한 사람들의 손 모양 데이터들을 모았다.

5. 연구 결과

5.1. 실행 결과

가. 기본 실행

위 <4. 연구 방법> 에 나와 있는 코드들을 통해 최종적으로 프로그램을 실행시켰다. Train Data 들을 ***코드 2***을 이용해 학습 시키고 미리 생성해 둔 Test Data로 진행한 '점검 (Confirm) 과정' 의 정확도를 출력한 결과는 아래와 같다. 학습률 0.001, 반복횟수 55번, 드롭아웃을 사용한 상태에서 진행하였다.

실행 결과 1

```
학습 시작!
Epoch: 0001 Avg. cost =  3.414526
Epoch: 0002 Avg. cost =  3.433837
Epoch: 0003 Avg. cost =  3.450250
                  ⋮
Epoch: 0012 Avg. cost =  2.995491
Epoch: 0013 Avg. cost =  2.704387
Epoch: 0014 Avg. cost =  2.435333
                  ⋮
Epoch: 0033 Avg. cost =  0.100858
Epoch: 0034 Avg. cost =  0.089678
Epoch: 0035 Avg. cost =  0.074343
                  ⋮
Epoch: 0053 Avg. cost =  0.018192
Epoch: 0054 Avg. cost =  0.011818
Epoch: 0055 Avg. cost =  0.013027
학습 완료!
```

정확도: 1.0 *#정확도 1은 100%를 의미한다.*

초기에는 오차가 크고 잘 줄어들지 않지만, 학습 진행 횟수 (Epoch)가 증가할수록 (특히 epoch = 11 부터), 그 오차 (cost)가 점점 줄어드는 것을 알 수 있다. 이후 테스트 데이터들을 인식시켰다.

100%의 정확도는 비정상정인 값이라고 생각될 수 있다. 하지만, 찍은 여러 개의 사진들 중 임의로 5개 ~ 10개를 골라 테스트 데이터로, 나머지는 학습 데이터도 분류해 놓았기 때문에 이렇게 나왔을 수 있다고 해석하였다. 즉, 전체적인 모양은 구분 및 인식이 된다는 것은 알 수 있었다. 향후에 데이터를 더 추가하였는데, 이에 대해서는 <4.4 모델 개선> 에 설명하였다.

최종적으로, 테스트 데이터들을 인식한 결과를 한글로 변환하였다. (실제로는 세로로 출력되지만 가로로 변환하였음.)

실행 결과 2

ㄱ ㄱ ㄱ ㄱ ㄱ ㄴ ㄴ ㄴ ㄴ ㄴ ㄷ ㄷ ㄷ … ㅚ ㅚ ㅚ ㅚ ㅟ ㅟ ㅟ ㅟ ㅟ

나. TTS 데이터 인식 실행

먼저, 위 <4.3 모델 개발>에서 소개한 '이현수입니다' 데이터들을 인식시켜 보았다 (그림.__ 참고). ***코드 3***을 포함하여, 인식 및 출력코드를 실행하면, 인식한 초성이 담겨져 있는 배열이 출력된다.

실행 결과 3

#실제로는 세로로 출력되지만 가로로 변환

ㅇ, ㅣ, ㅎ, ㅕ, ㄴ, ㅅ, ㅜ, ㅇ, ㅣ, ㅂ, ㄴ, ㅣ, ㄷ, ㅏ

['ㅇ', 'ㅣ', 'ㅎ', 'ㅕ', 'ㄴ', 'ㅅ', 'ㅜ', 'ㅇ', 'ㅣ', 'ㅂ', 'ㄴ', 'ㅣ', 'ㄷ', 'ㅏ']

이후, ***코드 7*** 과 ***코드 8*** 까지 실행시키면 아래와 같이 결과가 출력되고, 해당 문자을 읽어주는 음성 파일이 'FinalTTSFile.mp3' 로 저장된다.

실행 결과 4

이현수입니다
저장 완료!

#저장된 파일

FinalTTSFile.mp3 2018-11-09 오후 1... MP3 파일

이 외에도 '안녕하세요이현수입니다', '안녕하세요' 등의 문장들 역시 인식시켜 보았고, 성공하였다. 위 과정들의 실행 결과는 모두 시연영상으로 녹화하였다. (부록 1 참고)

5.2. [향후과제] 영상 인식 실행 결과

영상 인식의 경우 완성된 기능이 아니지만, 현재 이미지를 추출해 각각을 인식시키는 기능을 갖추고 있다. 앞으로 개선시키거나 추가시켜야 할 점에 대해서는 <6.2 후속연구 제안 및 활용방안> 에 기록하였다. 먼저, ***코드 9*** 를 통해 카메라를 실행하고 영상을 저장한 결과이다.

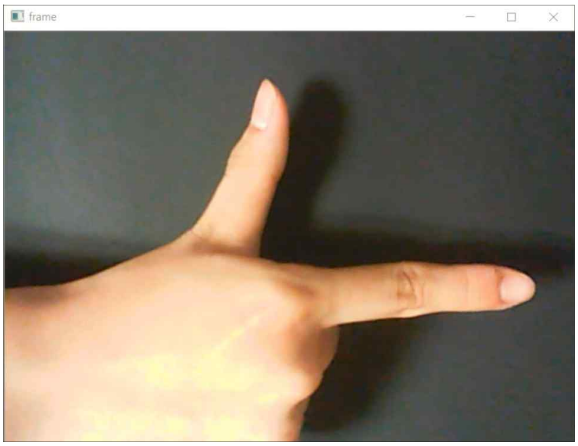

그림. 19. 카메라 실행

카메라 실행 시 지문자를 보여주면, 아래와 같이 video.avi 라는 영상 파일이 저장된다.

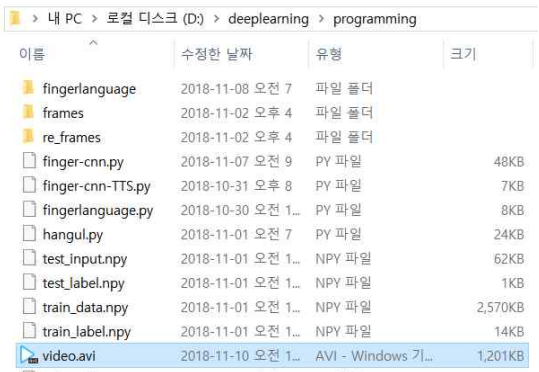

그림. 20. 저장된 동영상 파일

이후 ***코드 4를*** 통해서 5프레임 마다 이미지를 추출해 내면, frames 라는 폴더에 아래와 같이 저장된다. (이때 연구원들이 미리 영상으로 지문자 동작을 촬영하였었다)

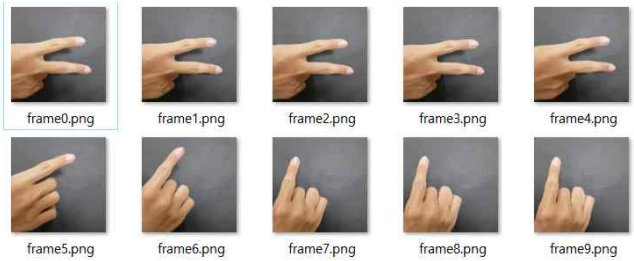

그림. 21. 프레임별로 추출된 이미지들

그 다음으로, 이 이미지들을 python에서 인식할 수 있게 28pixel × 28pixel 로 변환이 되고 (역시 ***코드 4*** 에 변환과정까지 포함되어 있다) 이는 re_frames 라는 폴더에 아래와 같이 저장이 된다.

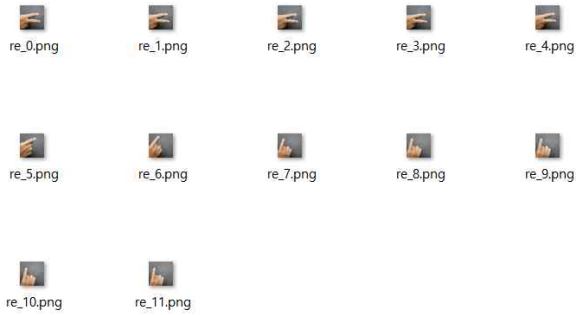

그림. 22. 28 × 28 size로 변환된 추출된 이미지들

이후 과정 부터는, 앞으로 해결해야 할 문제들이다. 구체적 내용은 아래 <6.2 후속연구 제안 및 활용방안> 에 서술하였다.

6. 결론

6.1. 연구 최종정리

이 연구는 청각 장애인의 의사소통을 위한 지문자 자동 인식 프로그램을 개발하였다. 임의의 한글 지문자를 손으로 나타내면 자동적으로 인식하고, 이를 모아서 하나의 문장과 음성파일로 변환 해 주는 기능까지 가지고 있다. 기존에 한글 지문자 인식은 많이 연구되어 있지 않았고, 음성으로 변환해 주는 과정의 경우 시현되어 있지 않다. 따라서 이 연구에서 이를 구현하였으며, 앞으로 그 활용 분야 역시 넓을 것으로 예상된다. 특히, 청각 장애인과 비장애인과의 소통의 벽을 허물어 이를 원활하게 해 줄 수 있을 것이다.

6.2. 후속연구 제안 및 활용방안

이 연구는 특히 청각 장애인들에게 많은 도움이 될 것이다. 그동안은 수화 또는 지문자를 아는 비장애인들은 적었다. 하지만 이 연구의 결과물을 통해서 이를 모르는 비장애인들도 청각 장애인들이 하는 말을 인식하여서 상호 작용을 할 수 있다. 따라서 의사 소통에 많은 활용이 될 것이고, 나아가서 영상에서의 인식까지 구현한다면 영상 통화 등에서도 이용될 수 있을 것이다.

후속 연구로는, 영상에서의 인식이 남아 있다. 현재 영상에서 이미지를 추출하고 각 이미지별로 인식하는 기능까지는 구현이 되어 있다. 앞으로 해결해야 할 과제는, 해당 이미지들의 인식 결과 중 '실제 결과 값' 만 저장하는 부분이다. 즉, 영상의 여러 프레임 중에서 실제로 사용자가 원하는 부분만 인식되어서 음성화 되기 위한 과정에 대한 연구가 더욱 필요하다.

참고문헌

[1] 김정진, 조성욱, 지영민, "CNN을 이용한 이미지 분류", 2017 한국정보기술학회 한국디지털콘텐츠학회 하계공동학술대회 논문집, 452-453 (2017)
[2] 하정요, 김계영, 최형일, "실시간 핸드 제스처 추적 및 인식", 한국컴퓨터정보학회 학술발표논문집, 141-144 (2010)
[3] 하정요, 이민호, 최형일, HMM(Hidden Markov Model)을 이용한 핸드 제스처인식, 한국디지털콘텐츠학회 논문지 10(2), 291-298 (2009)
[4] Helen Cooper, Brian Holt and Richard Bowden, "Sign Language Recognition" (2011)
[5] 변영현, 곽근창, "로봇환경에서 3차원 CNN을 이용한 비디오 기반 얼굴 인식", Proceedings of KIIT Summer Conference, 26-29 (2014)
[6] 류장욱, 이상협, 이종덕, 배태호, 오병우, "CNN 기법을 이용한 차량 종류 식별", Proceedings of KIIT Summer Conference, 236-237 (2018)
[7] 유소홍, 이지상, 노현주, 박효근, 손홍규, "CNN 기법을 이용한 공간정보 자동분류연구", 한국지형공간정보학회 학술대회, 167-168 (2017)
[8] 김진중, 골빈해커의 3분 딥러닝 텐서플로맛, 한빛미디어
[9] 조태호, 모두의 딥러닝, 길벗
[10] birc, 실제 이미지 데이터를 활용한 CNN 모델 구현하기, https://goo.gl/CPQVc1 (2018)
[11] OpenCV Python 예제 - 동영상 다루기, https://webnautes.tistory.com/577 (2018)
[12] 알파카친구, Python을 이용하여 동영상으로부터 이미지추출, https://goo.gl/NNBStM (2017)

부록

부록1. Github 레파지토리 주소

ID : frogyunmax
Repository : 2018_2_research
URL : http://github.com/frogyunmax/2018_2_research
시연영상 : https://goo.gl/QSgt3q

※ Github에서는 시연 영상과 느낀점, 주석처리 (설명)가 되어있는 코드 역시 확인하실 수 있습니다.

부록2는 전체 코드 모음입니다.

[1] 주 코드 (인식 + TTS인식) 코드

```
import tensorflow as tf
import os
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from numpy import array
from gtts import gTTS

TRAIN_DIR = 'D:/deeplearning/programming/fingerlanguage/traindata'
train_folder_list = array(os.listdir(TRAIN_DIR))

train_input = []
train_label = []
output_nodes = 31

label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(train_folder_list)

onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)

for index in range(len(train_folder_list)):
    path = os.path.join(TRAIN_DIR, train_folder_list[index]) + '/'
    img_list = os.listdir(path)
    for img in img_list:
        img_path = os.path.join(path, img)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        train_input.append([np.array(img)])
        train_label.append([np.array(onehot_encoded[index])])

train_input = np.reshape(train_input, (-1, 784))
train_label = np.reshape(train_label, (-1, output_nodes))
train_input = np.array(train_input).astype(np.float32)
train_label = np.array(train_label).astype(np.float32)
np.save("train_data.npy", train_input)
np.save("train_label.npy", train_label)

TEST_DIR = 'D:/deeplearning/programming/fingerlanguage/testdata'
test_folder_list = array(os.listdir(TEST_DIR))
```

```
test_input = []
test_label = []

label_encoder2 = LabelEncoder()
integer_encoded2 = label_encoder.fit_transform(test_folder_list)

onehot_encoder2 = OneHotEncoder(sparse=False)
integer_encoded2 = integer_encoded2.reshape(len(integer_encoded2), 1)
onehot_encoded2 = onehot_encoder2.fit_transform(integer_encoded2)

path = ''
for index in range(len(test_folder_list)):
    path = os.path.join(TEST_DIR, test_folder_list[index])
    path = path + '/'
    img_list = os.listdir(path)
    for img in img_list:
        img_path = os.path.join(path, img)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        test_input.append([np.array(img)])
        test_label.append([np.array(onehot_encoded2[index])])

test_input = np.reshape(test_input, (-1, 784))
test_label = np.reshape(test_label, (-1, output_nodes))
test_input = np.array(test_input).astype(np.float32)
test_label = np.array(test_label).astype(np.float32)
np.save("test_input.npy", test_input)
np.save("test_label.npy", test_label)

X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])
Y = tf.placeholder(tf.float32, [None, output_nodes])  # 출력값
keep_prob = tf.placeholder(tf.float32)
learning_rate = 0.001

W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

```
W3 = tf.Variable(tf.random_normal([7 * 7 * 64, 256], stddev=0.01))
L3 = tf.reshape(L2, [-1, 7 * 7 * 64])
L3 = tf.matmul(L3, W3)
L3 = tf.nn.relu(L3)
L3 = tf.nn.dropout(L3, keep_prob)

W4 = tf.Variable(tf.random_normal([256, output_nodes], stddev=0.01))
model = tf.matmul(L3, W4)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

batch_size = 100
total_batch = int(len(train_input) / batch_size)
all_epoch = 55
before_cost = 0

print('학습 시작!')
for epoch in range(all_epoch):
    total_cost = 0
    for i in range(total_batch):
        start = ((i + 1) * batch_size) - batch_size
        end = ((i + 1) * batch_size)
        batch_xs = train_input[start:end]
        batch_ys = train_label[start:end]
        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs, Y: batch_ys, keep_prob:
0.57})
        total_cost = total_cost + cost_val
    print('Epoch:', '%04d' % (epoch + 1), 'Avg. cost = ', '{:3f}'.format(total_cost / total_batch))
    if before_cost == total_cost / total_batch: total_cost = total_cost + 1
    before_cost = total_cost / total_batch
print('학습 완료!')

is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도:', sess.run(accuracy, feed_dict = {X: test_input, Y: test_label, keep_prob : 1}))
result = sess.run(model, feed_dict = {X: test_input, Y: test_label, keep_prob : 1})
char = {0 : 'ㄱ', 1 : 'ㄴ', 2 : 'ㄷ', 3 : 'ㄹ', 4 : 'ㅁ', 5 : 'ㅂ', 6 : 'ㅅ', 7 : 'ㅇ', 8 : 'ㅈ', 9 : 'ㅊ', 10 : 'ㅋ',
        11 : 'ㅌ', 12 : 'ㅍ', 13 : 'ㅎ', 14 : 'ㅏ', 15 : 'ㅑ', 16 : 'ㅓ', 17 : 'ㅕ', 18 : 'ㅗ',
       19 : 'ㅛ', 20 : 'ㅜ', 21 : 'ㅠ', 22 : 'ㅡ', 23 : 'ㅣ', 24 : 'ㅐ', 25 : 'ㅒ', 26 : 'ㅔ', 27 : 'ㅖ', 28 : 'ㅢ
',
```

```
        29 : '괴', 30 : '귀'}
for i in range(len(result)):
    maxi = -100
    ind = 0
    for j in range(0, output_nodes):
        if result[i][j] > maxi:
            maxi = result[i][j]
            ind = j
    print(char[ind])

TTS_DIR = 'D:/deeplearning/programming/fingerlanguage/ttss'
tts_input = []
img_list = os.listdir(TTS_DIR) + '/'
img_list = os.listdir(TTS_DIR)
for img in img_list:
    img_path = os.path.join(TTS_DIR, img)
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    tts_input.append([np.array(img)])
tts_input = np.reshape(tts_input, (-1, 784))
tts_input = np.array(tts_input).astype(np.float32)
han_result = []
ttsresult = sess.run(model, feed_dict = {X: tts_input, keep_prob : 1})

for i in range(len(ttsresult)):
    maxi = -1000
    ind = 0
    for j in range(0, len(ttsresult[i])):
        if ttsresult[i][j] > maxi:
            maxi = ttsresult[i][j]
            ind = j
    if ind == 6 :
        ind = ind - 5
        print(char[ind])
        han_result.append(char[ind])
        continue
    if 25 <= ind and ind <= 30 :
        ind = ind - 22
        print(char[ind])
        han_result.append(char[ind])
        continue
    if 1 <= ind and ind <= 5:
        ind = ind + 8
        print(char[ind])
```

```
            han_result.append(char[ind])
            continue
        if 7 <= ind and ind <= 16:
            ind = ind +  7
            print(char[ind])
            han_result.append(char[ind])
            continue
        if ind == 17 :
            ind = ind - 15
            print(char[ind])
            han_result.append(char[ind])
            continue
        if 18 <= ind and ind <= 19:
            ind = ind +  6
            print(char[ind])
            han_result.append(char[ind])
            continue
        if 20 <= ind and ind <= 24:
            ind = ind +  6
            print(char[ind])
            han_result.append( char[ind])
            continue
        print(char[ind])
        han_result.append(char[ind])
print(han_result)

M = '가각갂갃간갅…ힻힼힽힾힿ'
#길어서 생략

previous_letters_number = 0
Consonant_number = [2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1]
Consonants_number = [1, 3, 3, 1, 8, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1]
Consonant = [['ㄱ', 'ㄲ'], ['ㄴ'], ['ㄷ', 'ㄸ'], ['ㄹ'], ['ㅁ'], ['ㅂ', 'ㅃ'], ['ㅅ', 'ㅆ'], ['ㅇ'], ['ㅈ', 'ㅉ'], ['ㅊ'], ['ㅋ'], ['ㅌ'], ['ㅍ'], ['ㅎ']]
Consonants = [[' '], ['ㄱ', 'ㄲ', 'ㄳ'], ['ㄴ', 'ㄵ', 'ㄶ'], ['ㄷ'], ['ㄹ', 'ㄺ', 'ㄻ', 'ㄼ', 'ㄽ', 'ㄾ', 'ㄿ', 'ㅀ'], ['ㅁ'], ['ㅂ', 'ㅄ'], ['ㅅ', 'ㅆ'], ['ㅇ'], ['ㅈ', 'ㅉ'], ['ㅊ'], ['ㅋ'], ['ㅌ'], ['ㅍ'], ['ㅎ']]
Vowel = ['ㅏ', 'ㅐ', 'ㅑ', 'ㅒ', 'ㅓ', 'ㅔ', 'ㅕ', 'ㅖ', 'ㅗ', 'ㅘ', 'ㅙ', 'ㅚ', 'ㅛ', 'ㅜ', 'ㅝ', 'ㅞ', 'ㅟ', 'ㅠ', 'ㅡ', 'ㅢ', 'ㅣ']

Z = ['ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ', 'ㅇ', 'ㅈ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ']
cntind = 0
char_result = []
flag = False
```

```
while (True):
    # print(cntind)
    if flag: break
    a = ''
    b = ''
    c = ''
    cnt = 0
    if cntind == len(han_result): break
    if cntind == len(han_result) - 1:
        if han_result[cntind] in Z:
            a = han_result[cntind - 2]
            b = han_result[cntind - 1]
            c = han_result[cntind]
        elif han_result[cntind] in Vowel:
            a = han_result[cntind - 1]
            b = han_result[cntind]
            c = ''
    elif len(han_result) == 2:
        a = han_result[0]
        b = han_result[1]
        c = ''
        flag = True
    elif len(han_result) == 3:
        a = han_result[0]
        b = han_result[1]
        c = han_result[2]
        flag = True
    elif cntind >= 1 and han_result[cntind - 1] in Vowel and han_result[cntind] in Z and
han_result[cntind + 1] in Z:
        a = han_result[cntind - 2]
        b = han_result[cntind - 1]
        c = han_result[cntind]
    elif cntind >= 1 and han_result[cntind - 1] in Vowel and han_result[cntind] in Z and han_result[
        cntind + 1] in Vowel:
        a = han_result[cntind - 2]
        b = han_result[cntind - 1]
        c = ''

    if a != '' and b != '':
        letter = [a, b, c]
        for i in range(len(Consonant)):
            if letter[0] in Consonant[i]:
                for x in range(0, i):
```

```
                    cnt += Consonant_number[x] * 588

                for j in range(len(Consonant[i])):
                    if letter[0] == Consonant[i][j]:
                        cnt += j * 588

        for k in range(len(Vowel)):
            if letter[1] == Vowel[k]:
                cnt += (k) * 28
        for l in range(0, len(Consonants)):
            if letter[2] in Consonants[l]:
                for m in range(0, l):
                    cnt += Consonants_number[m]
                for n in range(0, len(Consonants[l])):
                    if letter[2] == Consonants[l][n]:
                        cnt += n
        char_result.append(M[int(cnt)])
    cntind = cntind + 1

total_char = ''
for i in range(len(char_result)):
    total_char = total_char + char_result[i]
print(total_char)

tts_file = gTTS(text=total_char, lang='ko')
tts_file.save("FinalTTSFile.mp3")
print('저장 완료!')
```

---

[2] 영상 촬영 코드

```
#참고한 코드 https://webnautes.tistory.com/577
import cv2 as cv
cam = cv.VideoCapture(0)

r = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('video.avi', r, 30.0, (640, 480))
while(cam.isOpened()):
    ret, frame = cam.read()
    if ret == False:
        continue
    out.write(frame)
    cv.imshow('frame', frame)
    if cv.waitKey(1) & 0xFF == 27:
```

```
        break
cam.relase()
out.release()
cv.destroyAllWindows()
```

---

[3] 영상 인식 코드

```
#참고한 코드 https://webnautes.tistory.com/577 , https://goo.gl/NNBStM
import cv2
from PIL import Image
cam = cv2.VideoCapture('video.mp4')
cntvid = 0
while(cam.isOpened()):
    ret, image = cam.read()
    fps = cam.get(cv2.CAP_PROP_FPS)
    if (int(cam.get(1)) % 5 == 0):
        cv2.imwrite("frames/frame%d.png" % cntvid, image)
        cntvid += 1
    if cntvid == 12 : break
cam.release()
cv2.destroyAllWindows()

for i in range(cntvid):
    path = 'frames/frame' + str(i) + '.png'
    save_path = 're_frames/re_' + str(i) + '.png'
    im_r = Image.open(path)
    size = (28, 28)
    im_r.thumbnail(size)
    im_r.save(save_path)
```
