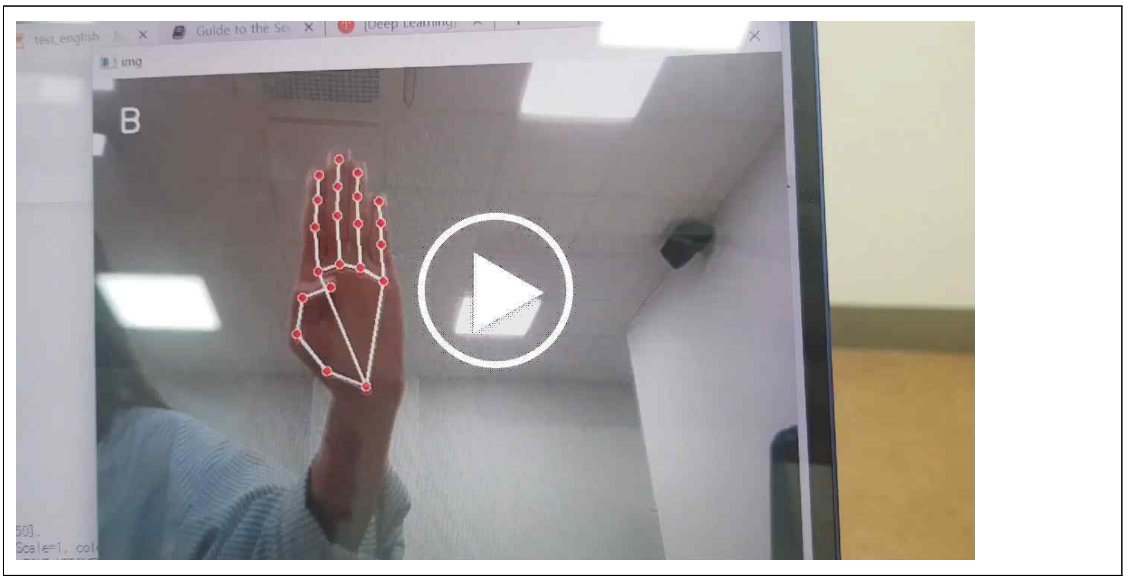


출품 분야 : 모바일SW, 임베디드SW, 게임SW, 기타(인공지능 AI)

1. 작품명

<sign_language 번역기>

2. 팀 구성

팀장 - 소프트웨어학부 김민경
팀원 - 소프트웨어학부 이정연
소프트웨어학부 정세은

3. 기획의도

최근 각광받는 분야인 AI 머신러닝에 대해 학습하며, 머신러닝의 학습능력이 유용하게 활용될 수 있는 분야를 모색하였다. 그 중 청각장애인을 위한 서비스 개발을 기획하게 되었고, 서비스를 통해 장애인과 (수화를 모르는) 비장애인의 경계를 허물고자 한다. 많은 사람이 불편함 없이 일상생활을 보내도록 돕겠다는 주된 목표 하에 프로젝트를 진행한다.

4. 작품 설명

가. 작품 개요

[개발 목표]

손의 수화(지문자) 모션을 텍스트로 변환하는 sign_language 번역기 개발

[개발 개요]

1. 데이터셋 수집(mediapipe, opencv, numpy, PIL 활용)
2. 모델 생성 및 학습(numpy, os, sklearn, tensorflow 활용)
3. 프로그램 제작(mediapipe, opencv, tensorflow, numpy, PIL 활용)

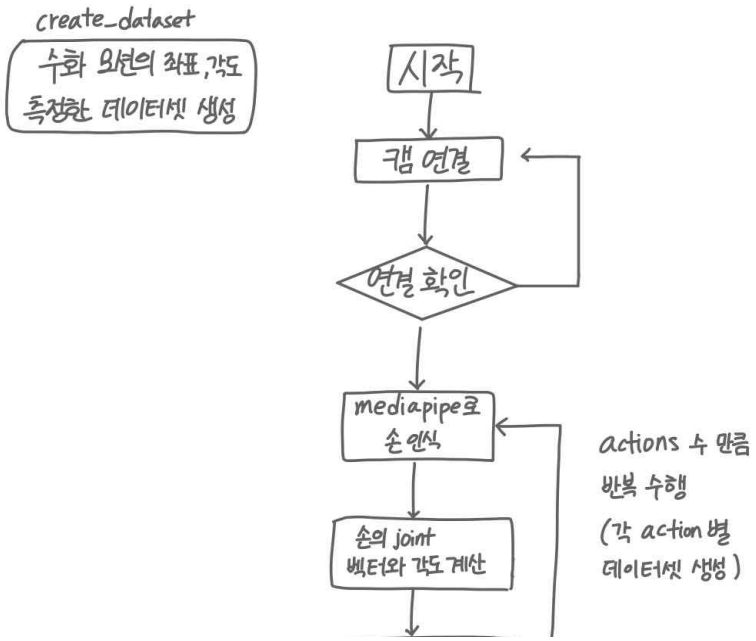
[기술 배경]

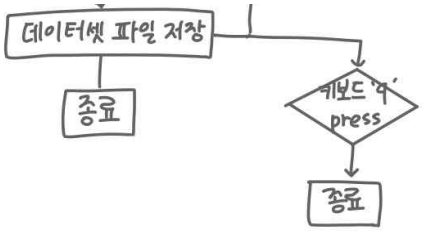
OpenCV : 실시간 컴퓨터 비전을 목적으로 인텔에서 개발한 라이브러리
Mediapipe : 비디오 형식 데이터에 대한 다양한 비전 추론을 수행하기 위해 파이프라인을 구축하는 구글에서 제공하는 AI 프레임워크
RNN(Recurrent Neural Networks) : 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는 인공신경망의 한 종류
LSTM(Long Short-Term Memory models) : RNN의 일종으로, 여러 개의 layer가 서로 정보를 주고받는(의존관계) 학습 수행 능력을 가진 인공신경망

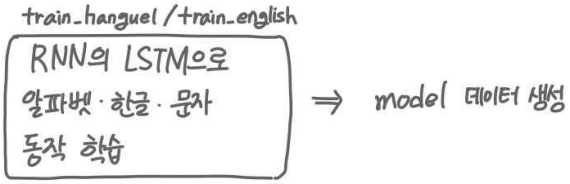
[데이터 범주(저장된 sign_language)]

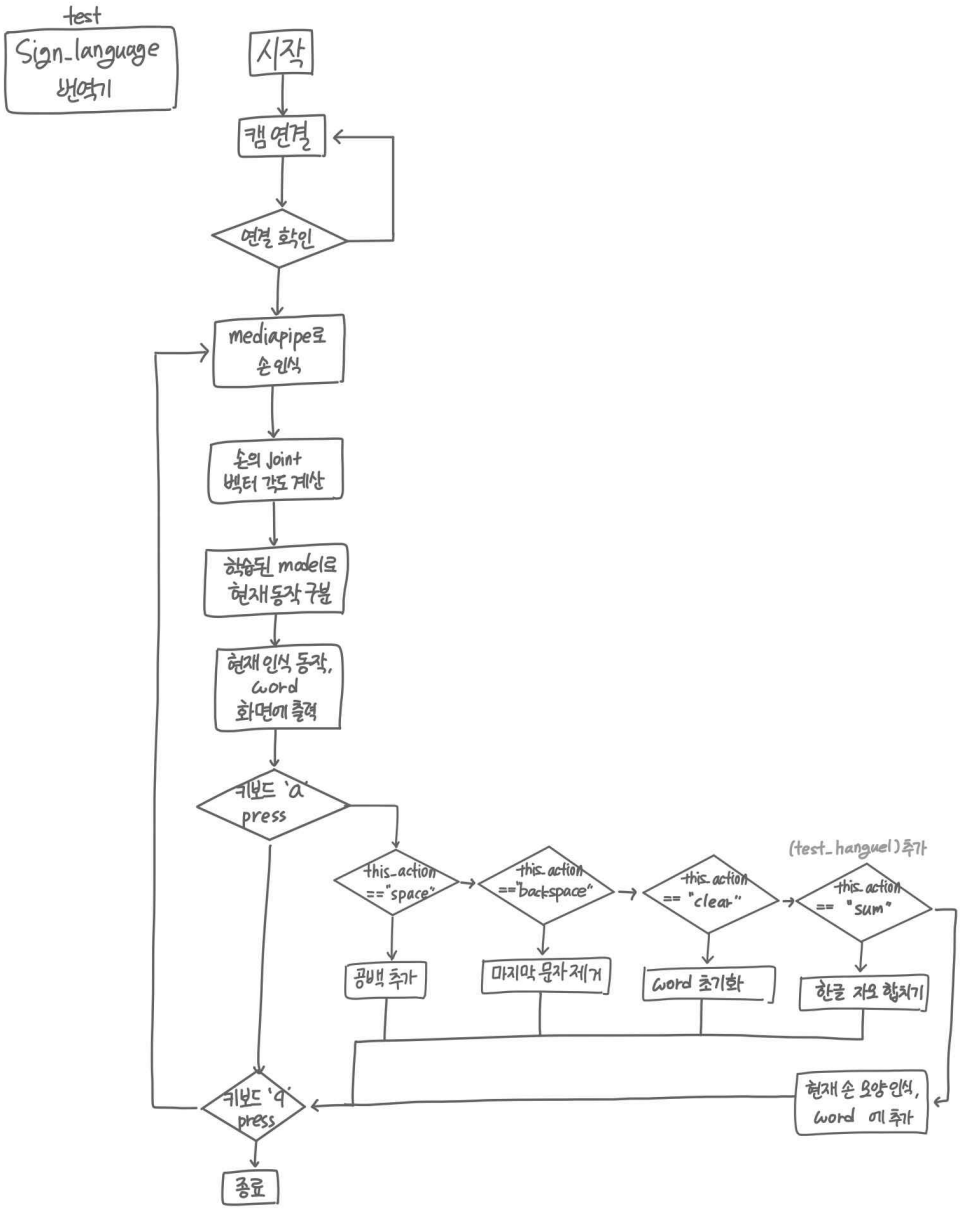
1. 알파벳(26자)
2. 한글(자음 14자 + 모음 17자)
3. 특수 문자(space, backspace, clear, sum)

나. 시스템 구성도




데이터셋 파일 저장
종료
키보드 'q' press
종료


train_hanguel / train_english
RNN의 LSTM으로
알파벳 · 한글 · 문자
동작 학습
⇒ model 데이터 생성


test
Sign_language
번역기
시작
캠 연결
연결 확인
mediapipe로
손 인식
손의 joint
벡터 각도 계산
학습된 model로
현재 동작 구분
현재 인식 동작,
word
화면에 출력
키보드 'a'
press
this_action
== "space"
this_action
== "backspace"
this_action
== "clear"
this_action
== "sum"
(test_hanguel) 추가
공백 추가
마지막 문자 제거
word 초기화
한글 자모 합치기
현재 손 모양 인식,
word 에 추가
키보드 'q'
press
종료

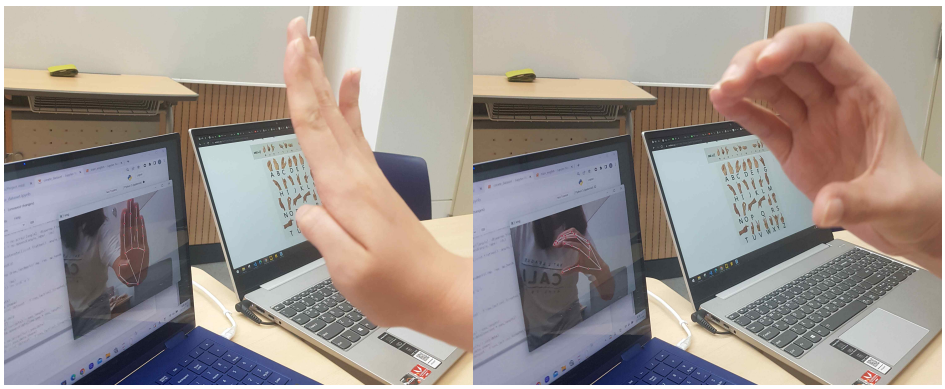
다. 구현기능

[프로그램 개발 단계]

데이터를 직접 생성하여 기계에 학습시키는 sign_language 번역기 프로젝트는 크게 3단계로 나눠 진행된다.

(1단계) 각각의 수화(지문자) 모션과 글자 일치시키는 데이터셋 생성
해당 파일 : create_dataset.py / dataset폴더

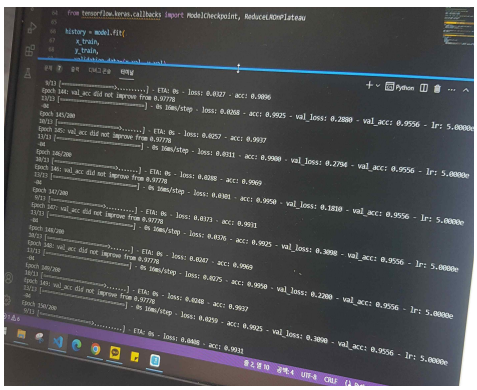
Mediapipe 라이브러리가 제공하는 joint를 활용하여 벡터 사이의 각도, 길이 차를 계산한 데이터셋을 저장한다.



- OpenCV 기능을 활용해 각 모션 별로 joint 데이터를 생성하는 모습
(각 모션별로 15초씩 녹화, 200~400개의 데이터 수집)

(2단계) 수집한 데이터 기반으로 기계 학습한 모델 생성
해당 파일 : train_hanguel.py / train_english.py / dataset폴더

데이터셋에 있는 데이터 중 seq(나열된) 데이터들을 소스로 학습시킨다.



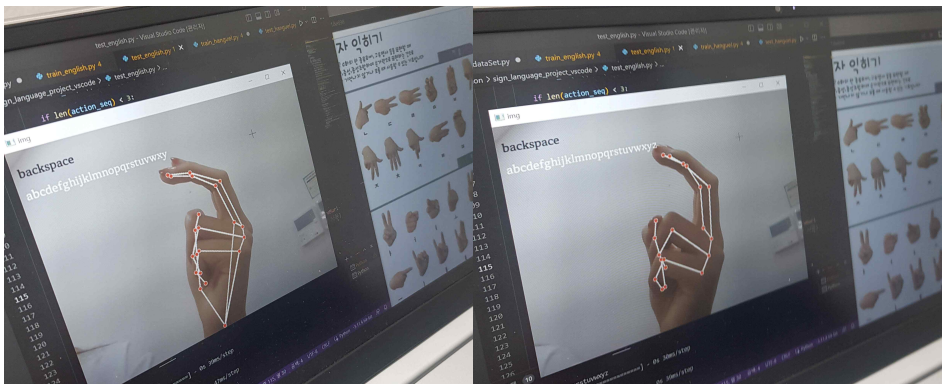
- model을 생성하는데 걸리는 시간은 유동성이 크며 10~40분 소요됨

(3단계) 번역기 프로그램 개발

해당 파일 : test_hanguel.py / test_english.py / models폴더

Medaipipe와 OpenCV를 이용하여 (1단계)에서 저장한 데이터 기준과 동일하게 사용자의 손 모션 판단 자료를 생성하고, 해당 모션의 비전 및 의미를 추론한다.

특수문자(space, backspace, clear, sum)에 대한 예외처리를 하여 번역기 프로그램에 기능을 추가한다. 또한, OpenCV는 한글 지원이 안 되기 때문에 별도의 한글 폰트와 PIL, 분리되어 출력된 자모를 합치도록 하는 unicode 라이브러리를 준비하여 프로그램의 완성도를 높인다.



- backspace 포즈를 취하고 'a'키를 누르면 word의 마지막 문자가 삭제되는 모습

[문제해결 과정]

1. 프로그램 정확도 향상 - 데이터 가공

1-1. Dataset 추출과정

해당 파일 : create_data.py

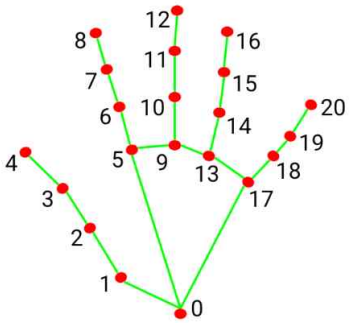
웹캡에서 frame 단위로 비디오를 읽으면 Mediapipe 통해 얻은 Landmark (Mediapipe에서 지정한 손의 각 포인트) 기반으로 각 frame에서의 손 모양의 특징을 추출한다. 특징 및 여러 변수 케이스에 대한 가공 가정을 거침으로써 정확성이 높아진 프로그램을 제작할 수 있었다.

(어떤 정보를 추출했는지는 아래 기재)

이후 추출된 데이터는 'd' 에 차곡차곡 프레임마다 저장된 후, 시퀀스 과정을 거쳐 최종적으로 'seq_{action}_{created_time}.npy' 파일로 저장된다. 이때 최종 파일은 (총 시퀀스 개수, 시퀀스 길이, 특징 수) 3차원으로 구성된다.

1-2. MEDIAPIPE Hand landmarks detection

각 프레임의 손 모양 특징은 다양한 정보로 구성된다. 이러한 특징을 활용하여 특정 정보들만의 추출하여 가공하였다.



각 랜드마크에는 x좌표, y좌표, z좌표, 가시성(visibility) 4개의 정보가 존재한다. 해당 정보 중 x, y 좌표를 중점적으로 데이터를 추출했다.

1-3. data의 feature

(Feature Index 기준)

0~39 : landmark 간의 x좌표 y좌표를 차이 값
-> landmark 사이 거리를 수치로 확인하기 위한 값

40 : landmark index 기준으로 5번과 17번 x좌표를 비교한 값
-> 현재 상태가 손등/손바닥 중에 어떤 부분인지 구분하기 위한 값

41 : landmark index 기준으로 5번과 0번 x좌표를 비교한 값
-> 손목 좌우 기울어짐을 확인하기 위한 값

42~46 : landmark index 기준으로 손의 시작점인 0과 각 손가락의 끝부분(4, 8, 12, 16, 20)의 차이 값
-> 각 손가락마다의 접힘 유무를 확인하기 위한 값

47~54 : landmark index 기준으로 4-8, 8-12, 12-16, 16-20 사이 차이 값
-> 옆에 있는 손가락과 거리를 비교해서 붙어있는지 확인

55 : landmark index 기준으로 0-5 y좌표 간의 차이 값
-> 현재 손이 위, 아래로 뒤집어져 있는지 확인하기 위한 값

56~57 : landmark index 기준으로 4-5 사이 간의 차이 값
-> 위에 있는 정보만으로는 엄지가 손바닥 안쪽 또는 바깥쪽에 있는지 확인하기 어려움이 존재해서 추가로 확인하기 위해서 넣음
-> 엄지의 위치를 구분하기 위한 값

58~59 : landmark index 기준으로 8-17 사이 간의 차이 값
-> 한글 같은 경우, 검지 손가락의 사용빈도가 높으므로 검지 끝(8)과

소지 손가락의 시작점(17)를 이용하여 검지 손가락을 좀 더 정확하게 측정 가능
-> 추가로 손바닥과 손등 구분을 위한 값

60~74 : 상단의 0~39 인덱스에서 구한 차이 값을 활용해서 각 벡터 사이를 내적하여 각도를 계산
(위에서 landmark 사이의 좌표 차이를 계산함. 그 후, v 배열에서 index (0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 14, 16, 17, 18) / (1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 17, 18, 19)에 해당하는 행들을 추출하고, 이를 np.einsum() 함수를 사용하여 두 개의 배열 v1과 v2의 내적(dot product)을 계산한 각 벡터 각도를 1차원 배열에 넣음)
-> 각 관절이 어느 정도로 꺾여 있는지 확인하기 위한 값

1-4. 상태 구분 데이터의 비트화

위의 feature 중 index 기준으로 40, 55, 56, 57은 결과가 0 또는 1로 저장되도록 데이터를 2차적으로 다듬었다.

40번은 현재 카메라에 비친 손등과 손바닥을 구분하기 위한 값, 55번은 현재 손이 위아래로 뒤집어진 상태인지 구분하기 위한 값, 56번과 57번은 엄지의 위치가 손바닥 안쪽인지 바깥쪽인지 확인하기 위한 값이다.

앞선 3가지 데이터는 구체적인 데이터 값보다는 손등/손바닥, 정방향/역방향, 안쪽/바깥쪽이라는 두 상태 중 어느 쪽에 위치하는지 상태 구분을 위해 사용된다. 따라서 나온 데이터를 그대로 쓰지 않고 2차 가공을 통해 0과 1의 상태만 가지도록 정규화함으로써 불필요한 데이터를 줄여 더욱 효율적인 학습이 이루어지도록 했다.

2. 프로그램 정확도 향상 - 훈련

해당 파일 : train_hanguel.ipynb / train_english.ipynb

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import tensorflow as tf

model = Sequential([
    LSTM(16, activation='relu', input_shape=x_train.shape[1:3]),
    Dense(512, activation='relu'),
    Dense(len(actions), activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 16)                6592

 dense (Dense)               (None, 512)               8704

 dense_1 (Dense)             (None, 29)                14877

=================================================================
Total params: 30,173
Trainable params: 30,173
Non-trainable params: 0
_________________________________________________________________
```

상단의 train 코드는 Sequential / TensorFlow / Keras를 사용해 구현된 신경망 모델이다. 높은 정확도를 달성하는 훈련을 하기 위해서는 먼저 모델에 대해 배경 지식이 필요하다.

2-1. 모델과 구성요소

모델(Model)은 LSTM layer, 두 개의 Dense layer로 구성되어 있다.

LSTM layer : 활성화 함수로는 relu를 사용하고, x_train은 학습 데이터로 사용되는 입력 데이터 배열이며, 이 데이터 배열의 두 번째와 세 번째 차원의 크기를 입력데이터로 지정한다.
Dense layer : 완전연결레이어로써, LSTM layer의 출력을 입력으로 받는다. 마지막 Dense layer는 actions 배열의 길이와 동일한 수의 유닛을 가지고 있으며, 활성화 함수로는 softmax를 사용한다.
(참조로, softmax함수는 0~1 사이로 변환하여 출력하지만, 출력값들의 합이 1이 되도록 하는 함수이다. 다중 클래스 분류 문제에서 각 클래스에 대한 확률 분포를 출력하도록 도와준다. 즉, 다중 분류의 최종 활성화 함수로 사용된다.)

2-2. 최적화와 손실함수

모델(Model)은 컴파일 단계에서 최적화를 위해 'adam' 옵티마이저를, 손실함수로 ‘categorical_crossentropy‘를 선택했다.

Adaptive Moment Estimation(Adam) : 딥러닝 최적화 기법으로, 학습의 방향과 크기를 모두 개선하여 딥러닝에서 가장 많이 사용되는 알고리즘이다.
'categorical_crossentropy' : 다중 클래스 분류 문제에서 사용되는 일반적인 손실 함수이다. 또한 정확도(metrics=['acc'])를 모델의 평가 지표로 설정했다.

2-3. 모델에 대한 정리

따라서 위 모델(Model)은 시계열 데이터를 입력으로 받아 LSTM 레이어를 통해 중요한 시퀀스 정보를 학습하고, 이후 완전 연결 레이어를 통해 분류 작업을 수행한다. 모델을 컴파일한 후, 주어진 데이터와 수화에 맞게 모델을 학습시키고 평가하는 과정을 수행하는 것이다.

활성화 함수로 'relu'을 사용한 이유는 다른 활성화 함수(ex. sigmoid, tanh)보다 학습이 빠르고, 연산 비용이 적고, 구현이 매우 간단하기 때문이다.
활성화 함수로 'softmax'을 사용한 이유는 확률의 총합이 1이고, 어떤 분류에 속할 확률이 가장 높은지를 쉽게 인지할 수 있기 때문이다.
손실 함수로 'categorical_crossentropy'을 사용한 이유는 반응 변수가 범주형일 때 특히 유용하며, 주로 분류 문제에 적용되기 때문이다.

2-4. 훈련 정보

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

history = model.fit(
    x_train,
    y_train,
    validation_data=(x_val, y_val),
    epochs=200,
    callbacks=[
        ModelCheckpoint('models/model.h5', monitor='val_acc', verbose=1, save_best_only=True, mode='auto'),
        ReduceLROnPlateau(monitor='val_acc', factor=0.5, patience=50, verbose=1, mode='auto')
    ]
)
```

```
Epoch 1/200
341/343 [============================>.] - ETA: 0s - loss: 2.5473 - acc: 0.3657
Epoch 1: val_acc improved from -inf to 0.54672, saving model to models₩model.h5
343/343 [==============================] - 6s 12ms/step - loss: 2.5417 - acc: 0.3663 - val_loss: 1.4704 - val_acc: 0.5467 - lr: 0.0
010
Epoch 2/200
339/343 [============================>.] - ETA: 0s - loss: 1.4706 - acc: 0.5496
Epoch 2: val_acc did not improve from 0.54672
343/343 [==============================] - 4s 11ms/step - loss: 1.4729 - acc: 0.5487 - val_loss: 1.5462 - val_acc: 0.5139 - lr: 0.0
010
```

상단의 코드와 같이 실행했을 때 model.fit() 함수는 학습을 수행하고, 학습 과정에서 발생한 손실(loss)과 정확도(accuracy)를 기록한 객체(history 변수에 저장)이다. history 객체는 딕셔너리(dictionary) 형태로 되어 있으며, 훈련 데이터와 검증 데이터에 대한 손실과 정확도를 기록하고 있다.
loss : 훈련 데이터에 대한 손실값
val_loss : 검증 데이터에 대한 손실값
accuracy : 훈련 데이터에 대한 정확도
val_accuracy : 검증 데이터에 대한 정확도

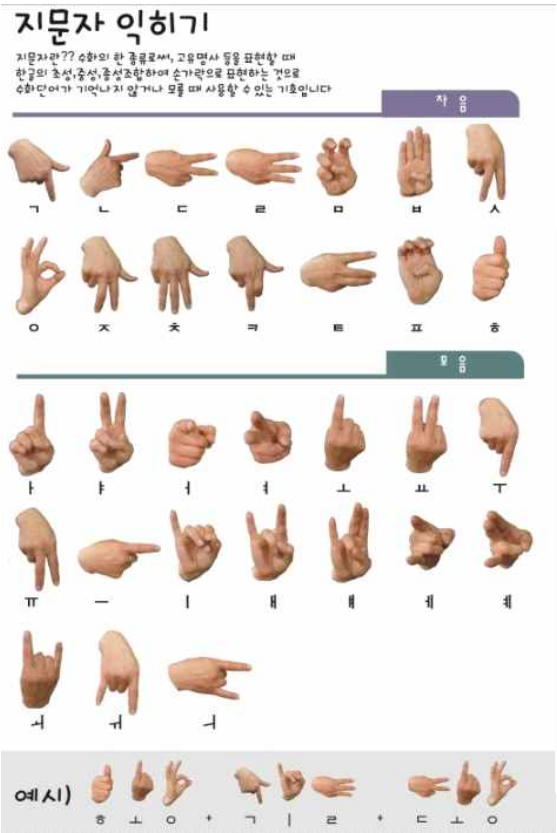
2-5. 정확도를 높이기 위한 방법

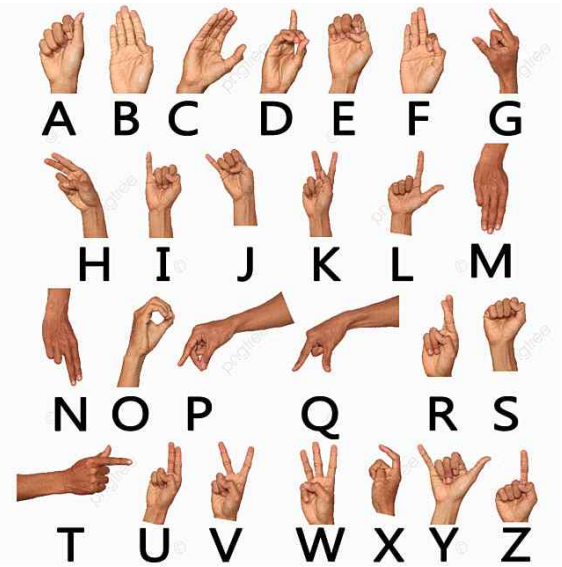
LSTM 레이어에서 정확도를 높이기 위해서는 LSTM 레이어의 유닛수를 경험이나 실험을 통해 결정할 수 있다. 네트워크의 깊이와 다른 레이어의 구성에 따라서도 최적의 유닛수가 달라질 수 있기 때문이다.

Dense 레이어에서는 모델의 복잡성과 훈련 데이터의 양에 따라 달라지는 완전 연결 층의 유닛수를 검증 데이터를 사용해 조정하는 것이 중요하다. 최적의 유닛 수를 찾기 위해서는 여러 실험과 검증 과정을 통해 결정해야 한다.

2-6. 여러 실험과 검증 과정으로 훈련한 모델의 최종 LSTM, Dense
(한글) LSTM layer 유닛 수 : 32 / 완전 연결 층의 유닛 수 : 1024
(영어) LSTM layer 유닛 수 : 16 / 완전 연결 층의 유닛 수 : 512

3. 프로그램 정확도 향상 - 동작의 차별성

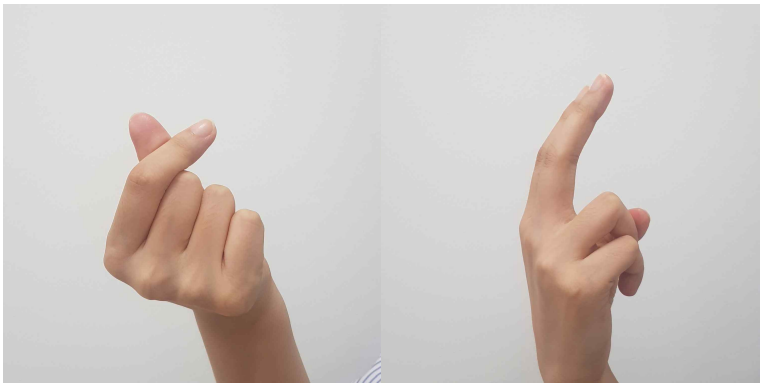




일상에 존재하는 수화의 한 종류인 지문자를 활용하여 데이터를 학습시켰다. 또한, 특수문자 4개를 추가로 임의 모션을 정해 학습시킴으로써 수화 번역기

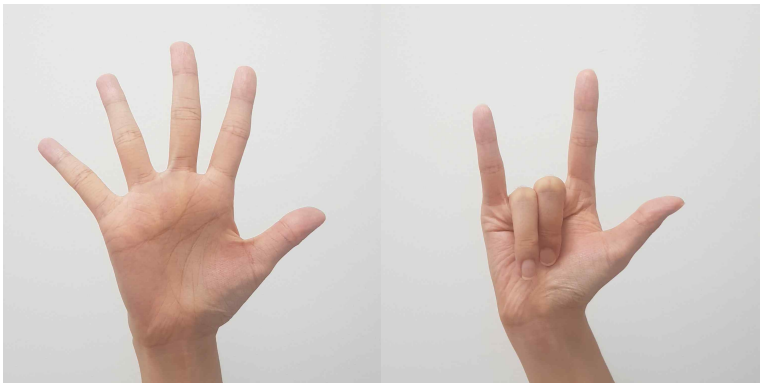
프로그램의 기능 범위를 확장하고자 하였다. 그러나 훈련하고자 하는 모션이 많아질수록 동작 간의 차이를 구분하지 못해 프로그램의 정확도가 떨어지는 경향이 있었다. 문제를 해결하기 위해 기존 지문자와 겹치지 않는 동작을 고안해 내었고, 동작에 차별성을 두어 데이터를 저장하였다.

예시로, 한글 자음 'ㅅ'과 모음 'ㅠ'의 동작이 유사하다. 모션의 일치도가 높기 때문에 'ㅅ'은 펼쳐진 두 손가락을 비틂으로써, 'ㅠ'는 펼쳐진 두 손가락을 붙임으로써 변형된 데이터를 학습시켰다.



- 왼쪽부터 space, backspace

(해당 동작을 취하고 'a'키보드를 누르면 space는 word에 공백을 추가하고, backspace는 word의 마지막 문자를 하나 제거함)



- 왼쪽부터 clear, sum

(해당 동작을 취하고 'a'키보드를 누르면 clear는 word에 저장된 문자를 모두 지우고, backspace는 word의 분리된 한글 자모들을 합침)

4. 한글 폰트 적용

OpenCV는 한글에 대한 호환성 문제가 있다. OpenCV의 putText()는 기본으로 제공하는 폰트가 있지만, 한글 폰트는 제외되며 기본 폰트 형식이 index로 저장되어 있다. 따라서 PIL 라이브러리와 외부에서 가져온 MaruBuri-Bold 폰트 파일로 한글을 출력하였다.

create_dataset.py 코드 일부

```
ret, img = cap.read()
img = cv2.flip(img, 1) # 좌우 반전
img = Image.fromarray(img) #img배열을 PIL이 처리가능하게 변환

draw = ImageDraw.Draw(img)
myfont = ImageFont.truetype('D:/python/sign_language_project_vscode/
org=(10,30)# 글자 표시할 위치

draw.text(org, f'Waiting for collecting {action.upper()} action...',
img = np.array(img) # numpy가 처리할 수 있도록 다시 변환
```

웹캠의 화면을 읽는 img의 배열은 numpy가 처리할 수 있도록 되어 있다. 그러나 위의 형식은 PIL에서 처리할 수 없으므로 Image.fromarray()함수로 PIL이 처리하도록 임시로 변환한 다음, 글자 출력 후에 np.array()함수를 이용하여 다시 numpy가 처리 가능하도록 변환해주어야 한다.

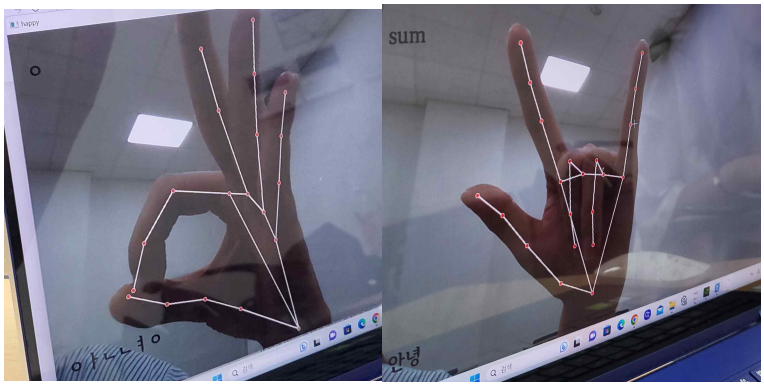
5. 한글 자모 합치기

한글은 자음과 모음을 조합해 하나의 글자를 구성하는 언어이다. 사용자가 입력한 분리된 자모 정보들을 합쳐주는 과정이 필요하다. 그러나 한글의 자모를 합치기 위해서는 고려해야 하는 경우의 수가 많다. 기계학습이라는 학습 목표에 집중하고 간결한 코드를 위해 unicode 라이브러리에서 지원하는 join_jamos() 함수를 활용해 기능을 구현함으로써 프로그램의 완성도를 높였다.

unicode 파일 위치는 test_hanguel.py와 동일한 디렉토리에 존재해야 하며,

```
from unicode import join_jamos
```

코드를 추가해 함수를 불러올 수 있다.



- ‘ㅇㅏㄴㄴㅕㅇ’이 ‘안녕’으로 합쳐지는 모습

[느낀점 및 개선하고 싶은 점]

약간씩 동작이 달라도 제대로 모션을 추론할 수 있도록 다양한 각도로 데이터를 저장했지만, 데이터 수집 및 훈련 시간이 오래 걸리고 고려해야 하는 변수가 많아 어려움이 있었다. 그럼에도 데이터 가공 등의 노력을 기울이면 효과가 현저히 나타남에 뿌듯함을 느낄 수 있었다. 다음에는 좀 더 동작이 큰 (또는 두 손을 이용하는)모션에 대해서도 기능을 확장해 나가고 싶다.

5. 개발환경

[운영체제]
window

[IDE(통합개발환경)]
jupyter / vscode

[개발언어]
python

[오픈 라이브러리]
계산 : numpy
기계학습 : sklearn, tensorflow, Keras
화면 : opencv
손 인식 : mediapipe
한글출력 : PIL

운영체제 : os
시간 : time

기타 : 한글폰트(MaruBuri-Bold.ttf), 자모합치기(unicode.py)

[전체 프로그램 소스코드]
https://github.com/kmg22/23Summer_SoftProject