mediapipe를 활용한

수화번역 시스템

2022 컴퓨터영상신호처리
대구가톨릭대학교 컴퓨터공학과 19120462 김은조

목 차

1. 개발 목적

신한은행은 올해 2022년 4월 20일인 '장애인의 날'을 맞이해 시중은행 최초로 청각 장애인과 고령자의 금융 편의를 높이기 위해 '글로 보는 상담 서비스'를 시행하였다. 이는 영업점을 방문한 청각 장애인과 고령자에게 직원과의 상담내용을 전용 태블릿을 통해 실시간 자막으로 전환, 원활한 상담이 가능하도록 지원하는 서비스이다. '글로 보는 상담 서비스'에 활용되는 전용 태블릿은 실시간 음성을 텍스트로 전환해주는 인공지능 기술이 탑재되어 있고, 필담(서로 말이 통하지 않거나 입으로 말을 할 수 없는 경우에, 글을 써 가며 의견이나 생각을 주고받는 것) 기능 추가로 양방향 의사소통이 가능해 원활한 상담을 지원한다.

하지만 이는 "음성"을 인식해서 텍스트로 전환하는 것이므로 청각장애인의 경우 앞서 말한 필담 기능을 사용해야 하는데 만약 타자를 치는 것이 미숙한 고령자이거나 타자 속도가 느린 사람일 경우 상담 시간이 꽤나 길어져 서로 답답한 상황이 일어날 것 같았다.

그래서 본 태블릿에 수화를 번역할 수 있는 기능이 추가되면 이를 해결할 수 있다고 생각하여 본 프로젝트를 진행하게 되었다.

참고 기사: https://www.thedailypost.kr/news/articleView.html?idxno=86863

2. 개발 과정

1) 프로젝트 개발 환경

- 하드웨어

  LG전자 2019 그램 15Z990-VA56K, 64비트 운영 체제, x64 기반 프로세서

- 소프트웨어

  Windows11(OS), Python(IDE: Visual Studio Code)

  Python Library(Tkinter, OpenCV, Tensorflow, Mediapipe)

2) 사용 프로그램 및 라이브러리

본 프로젝트는 OpenCV를 활용하여 카메라를 열고 Mediapipe를 통해 손을 인식하고, Tensorflow의 딥 러닝 모델을 사용하여 수화를 분류한다. 적용될 수 있는 장비는 카메라가 설치되어 있는 단말기에 해당한다. 본 글에서는 운영체제가 Windows이고 카메라가 설치된 노트북 단말기를 기준으로 작성되었다.

- 사용된 라이브러리: 2-8 프로그램 전체 코드 import문 참고

- 라이브러리 버전:

* 버전 확인 방법: cmd 창에서 python 버전 확인 = python —version, python pakage 설치 목록 확인 = pip list

python == 3.7.9

mediapipe == 0.8.10

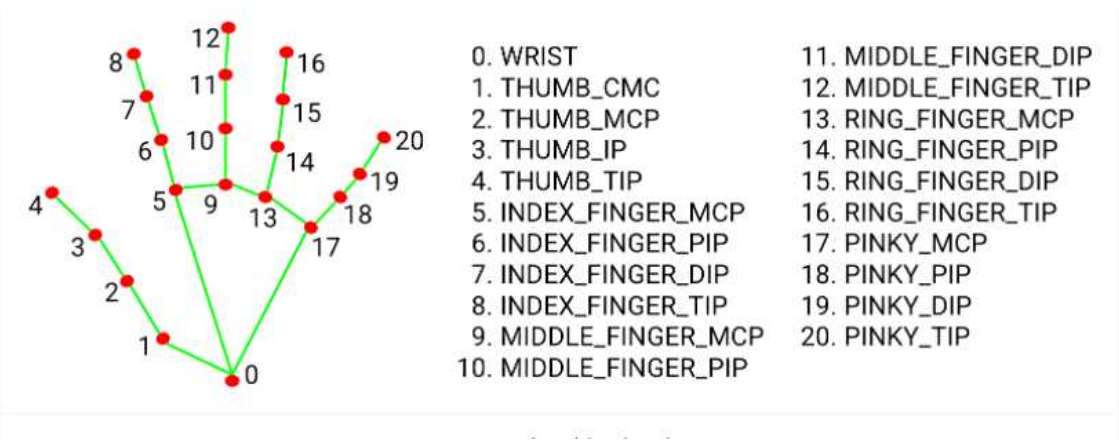
tensorflow == 2.4.0

opencv-python==4.5.5.64

외의 라이브러리는 버전 생략

3) Mediapipe를 활용한 손 인식 구현

본 프로젝트는 Mediapipe의 hands 모듈을 사용하여 손을 인식하고 인식된 손의 랜드마크를 추출해낸다.



아래의 코드는 create_dataset.py 파일의 일부이다.

```
# MediaPipe hands model initialize
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
hands = mp_hands.Hands(
    max_num_hands=2,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5)
```

Mediapipe hands 모델을 initialize하는 코드이다. 양손을 인식해야 하므로 max_num_hands는 2, min_detection_confidence,min_tracking_confidence는 0.5로 설정한다.

4) opencv를 이용한 dataset 수집

본 프로젝트에서는 OpenCV를 통해 dataset 수집을 진행한다. 카메라는 노트북에 내장된 카메라를 사용하므로 Videocapture는 0으로 설정한다.

```
cap = cv2.VideoCapture(0)
cap.set(3, 1080) # set video widht
cap.set(4, 720) # set video height

created_time = int(time.time())
os.makedirs('dataset', exist_ok=True) # 데이터셋 저장 폴더

action = word

actions = []

data = []

with open(file_path, 'r', encoding="UTF-8") as f:
    actions = f.read().splitlines()
    print(actions)

if action not in actions:
    with open(file_path, 'a', encoding="UTF-8") as f:
            f.write(f"{word}\n")
            id = len(actions)

    if cap.isOpened():
        ret, img = cap.read()

        img = cv2.flip(img, 1)
```

데이터 수집을 위해 classset.txt 파일을 생성하여 사용한다. 데이터셋을 추가 할 때 main.py에서 추가할 말(단어나 동사 등)을 입력받아 입력값을 create_dataset.py로 넘겨 라벨을 생성한다. 이 때 입력값은 classset.txt 파일에 한 줄 추가가 된다. 이는 학습 코드를 실행 시 시퀀스 파일을 불러오기 편하도록 하기 위함이다. 이는 다음 챕터인 2-5에서 확인할 수 있다.

word는 main에서 받아온 입력값(라벨명이 될 값)이고 이는 action 변수에 적재한다. actions라는 리스트를 만들어 classset.txt 파일을 불러와 안에 내용(라벨명)을 한 줄씩 읽어들여 적재한다. 이후 actions 리스트의 값들 중에 action의 값과 동일한 값이 있는지 비교하고 없으면 action의 값을 actions에 append 한다. 새로 추가할 데이터의 라벨값을 저장할 id 변수에는 actions 리스트의 길이로 적재한다. 이는 라벨값이 0부터 시작하기 때문에 만약 3개의 데이터 셋이 들어가있을 경우 3가지 라벨값은 각각 0, 1, 2를 부여받아있고 새로 들어올 데이터의 라벨값이 3이 되는 것이다.

```
start_time = time.time()

while time.time() - start_time < secs_for_action: # 30초 동안 녹화
    ret, img = cap.read()

    img = cv2.flip(img, 1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    result = hands.process(img) # 하나씩 프레임을 읽어서 미디어 파이브에 넣어줌
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    if result.multi_hand_landmarks is not None: # 결과를 가지고 각도를 뽑아내는 과정
        for res in result.multi_hand_landmarks:
            joint = np.zeros((21, 4))
            for j, lm in enumerate(res.landmark):
                joint[j] = [lm.x, lm.y, lm.z, lm.visibility] # 손가락 각도가 이미지 상에서 보이는지 안보이는지를 판단

            # 손가락 관절사이에 각도를 구하는 코드
            # Compute angles between joints
            v1 = joint[[0,1,2,3,0,5,6,7,0,9,10,11,0,13,14,15,0,17,18,19], :3] # Parent joint
            v2 = joint[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20], :3] # Child joint
            v = v2 - v1 # [20, 3]
            # Normalize v
            v = v / np.linalg.norm(v, axis=1)[:, np.newaxis]

            # Get angle using arcos of dot product
            angle = np.arccos(np.einsum('nt,nt->n',
                v[[0,1,2,4,5,6,8,9,10,12,13,14,16,17,18],:],
                v[[1,2,3,5,6,7,9,10,11,13,14,15,17,18,19],:])) # [15,]

            angle = np.degrees(angle) # Convert radian to degree

            # 라벨(id)를 넣음
            angle_label = np.array([angle], dtype=np.float32)
            angle_label = np.append(angle_label, id)

            # 수집한 좌표값들을 펼쳐서 concatenate -> 100개짜리 행렬
            d = np.concatenate([joint.flatten(), angle_label])

            data.append(d)

            mp_drawing.draw_landmarks(img, res, mp_hands.HAND_CONNECTIONS)

    cv2.imshow('img', img)
    if cv2.waitKey(1) == ord('q'):
        cap.release()
        break
```

녹화는 30초동안 진행된다. 이 때 녹화는 영상 자체를 녹화하는 것이 아니라 제스처랑 랜드마크를 녹화하는 것을 의미한다. 즉, 점의 위치와 각도를 녹화하여 파일에 저장하는 과정이다. 녹화는 30초간 진행되며 성공적으로 완료가 되면 (수집된 데이터 수, LSTM의

시퀀스 길이, 포인트 수) 형태로 출력이 된다. 본 프로젝트에서는 시퀀스 길이를 30으로 지정했으며 포인트는 손가락 각도 15개, 랜드마크 63개, 랜드마크 visibillity 21개, 정답 라벨 1개 총 100개의 포인트를 의미한다. 즉, 30 사이즈의 윈도우를 가진 100개의 포인트를 수집된 데이터 수만큼 데이터 셋이 저장된다.

Mediapipe의 hands 모듈을 사용하여 각 랜드마크의 x,y,z값을 구하고 랜드마크가 인식이 되는지 안되는지를 판단하는 visibillity값도 구한다. 이를 토대로 변화하는 각도를 구한다. 랜드마크 사이의 거리를 선(관절)으로 출력하여 잘 인식이 되는지 시각적으로 확인할 수 있도록 하였다.

```
# numpy array 형태로 변환
data = np.array(data)
print(action, data.shape)

# npy형태로 저장
np.save(os.path.join('dataset', f'raw_{id}_{action}'), data)

# Create sequence data
full_seq_data = []
for seq in range(len(data) - seq_length):
    full_seq_data.append(data[seq:seq + seq_length])

full_seq_data = np.array(full_seq_data)
print(action, full_seq_data.shape)
np.save(os.path.join('dataset', f'seq_{id}_{action}'), full_seq_data)

cap.release()
```

모든 작업이 완료되면 npy 형태로 데이터를 dataset 폴더에 저장한다. 또 시퀀스 데이터로 저장하여 최종적으로 한 라벨 당 두 가지 파일이 저장된다.

5) tensorflow를 이용한 학습 알고리즘 구현

```
# 시퀀스 데이터만 사용하여 학습

def train():
    os.environ['CUDA_VISIBLE_DEVICES'] = '1'
    os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

#

    # 라벨명 추출
    with open("classset.txt", 'r', encoding="UTF-8") as f:
        actions = f.read().splitlines()

    data = []

    # 시퀀스 데이터 로드하여 하나로 합침
    for i in range(len(actions)):
        data.append(np.load(f'dataset/seq_{i}_{actions[i]}.npy'))

    data = np.concatenate(data)

    data.shape
```

학습은 시퀀스 데이터만을 사용하여 학습한다. 앞서 2-4에서 언급했듯 데이터셋 수집 과정에서 저장된 classset.txt 파일을 불러와 안에 저장된 라벨명을 actions 리스트에 저장하고 이를 np.load하여 하나로 합쳐(concatenate) data에 저장한다.

```
    # 마지막 값에 라벨값이 저장되어 있으므로 분리
    x_data = data[:, :, :-1]
    labels = data[:, 0, -1]

    print(x_data.shape)
    print(labels.shape)

#
    # one-hot encoding
    y_data = to_categorical(labels, num_classes=len(actions))
    y_data.shape
```

data의 마지막 행은 포인트이다. 포인트의 마지막 값은 라벨값(인덱스값)이므로 분리시킨다. 분리 시키고 나면 마지막 값은 100이 아니라 99가 나온다. 이후 to_categorical을 사용해 one-hot encoding을 진행한다.

```
# training set validation set으로 나눔
x_data = x_data.astype(np.float32)
y_data = y_data.astype(np.float32)

x_train, x_val, y_train, y_val = train_test_split(x_data, y_data, test_size=0.1, random_state=2021)

print(x_train.shape, y_train.shape)
print(x_val.shape, y_val.shape)
```

training set과 validation set으로 나눈다.

```
# 모델 정의
model = Sequential([
    LSTM(64, activation='relu', input_shape=x_train.shape[1:3]),
    Dense(32, activation='relu'),
    Dense(len(actions), activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
model.summary()
```

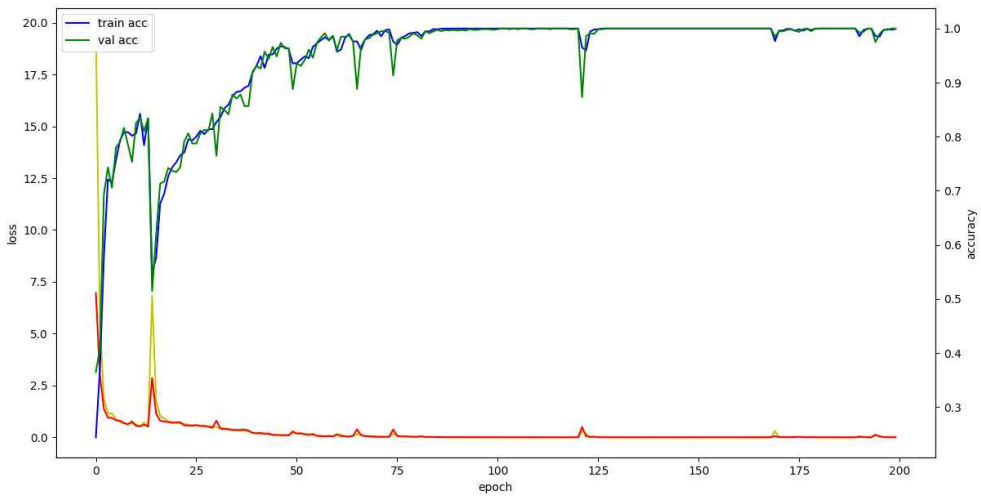
모델을 간단하게 정의한다. seqential api를 사용하여 LSTM과 Dense 2개를 연결한다.

```
# 200번 학습
history = model.fit(
    x_train,
    y_train,
    # batch_size=12,
    validation_data=(x_val, y_val),
    epochs=200,
    callbacks=[
        ModelCheckpoint('./models/model2_1.0.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')
    ]
)
```

학습을 진행한다. epochs값은 200을 주었다. 학습된 모델은 models/model2_1.0.h5 파일에 저장한다.

```
fig, loss_ax = plt.subplots(figsize=(16, 10))
acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')
loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
loss_ax.legend(loc='upper left')

acc_ax.plot(history.history['acc'], 'b', label='train acc')
acc_ax.plot(history.history['val_acc'], 'g', label='val acc')
acc_ax.set_ylabel('accuracy')
acc_ax.legend(loc='upper left')

plt.show()
```

학습이 완료되면 잘 되었는지 그래프로 확인할 수 있다. 초록색, 파란색은 training과 validation accuracy이고, 노랑, 빨강은 loss이다. accuracy가 100%가 되는 것을 확인할 수 있다.



6) 수화 번역 프로그램 구현

main.py 파일은 tkinterface로 구현된 간단한 UI가 정의된 파일이다. main 코드를 실행하면 아래와 같은 화면이 나온다.

entry에 데이터를 추가할 말을 입력하고 수집 버튼을 클릭하면 create_dataset.py 의 create 함수를 실행하고 학습 버튼을 클릭하면 train.py 의 train 함수가 실행된다. 또 수화 번역 버튼을 클릭하면 test.py 의 test 함수를 실행한다.

create와 train은 2-4, 2-5 챕터를 참고하면 된다. 본 챕터에서는 test.py 파일의 test 함수의 구현을 설명한다.

```
def test():
    file_path = "classset.txt"

    with open(file_path, 'r', encoding="UTF-8") as f:
        actions = f.read().splitlines()
        actions.append(' ')
        print(actions)

    seq_length = 30

    model = load_model('models/model2_1.0.h5')
```

학습 모델이 저장된 models/model2_1.0.h5을 로드한다. actions 리스트를 불러오는

과정은 중복 설명이므로 생략한다.

```
input_data = np.expand_dims(np.array(seq[-seq_length:], dtype=np.float32), axis=0)

# 인퍼런스한 결과를 뽑아냄
y_pred = model.predict(input_data).squeeze()

# argmax로 인덱스 뽑아냄
# 그것의 confidence를 뽑아냄
i_pred = int(np.argmax(y_pred))
conf = y_pred[i_pred]

# confidence가 90% 이하면 확실하지 않다 판단해 넘김
if conf < 0.9:
    continue

# 90% 이상이면 action 라벨명 추출
action = actions[i_pred+1]

# 출력
img_pil=Image.fromarray(img)

# PIL 이미지에 한글 입력
draw = ImageDraw.Draw(img_pil)
draw.text((40, 450),  f"{action}", font=ImageFont.truetype("./malgun.ttf", 36), fill=(255,255,255))

# PIL 이미지 -> cv2 Mat 타입으로 변경
img = np.array(img_pil)
```

판단 로직은 위의 코드와 같다. 인퍼런스한 결과를 y_pred 변수에 적재한다. argmax로 인덱스를 추출하여 그것의 confidence를 추출해낸다. 만약 confidence 값이 90% 이하면 해당 동작이 명확하게 판별되지 않았다고 판단해 넘긴다. 90% 이상이라면 action 라벨명을 추출한다.

opencv putText를 사용하면 한글이 깨진다. 하지만 번역을 위해선 반드시 한글을 사용해야 하므로 pillow 라이브러리를 사용하여 한글을 출력한다.

7) 프로그램 전체 코드

main.py

```
import tkinter as tk
from tkinter import *
from create_dataset import *
from test import *
from train import *

fontstyle = 'Ebrima'
bg_color = 'white'
bt_color = 'green'
root = tk.Tk()
root.geometry('800x400')
root.title('Sign language translator')
root.configure(bg=bg_color)

word = StringVar()
def return_word():
    get_word = word.get()
    create(get_word)



# 센터 정렬을 위한 빈 라벨
tk.Label(root, text="", width=15, height=10, bg=bg_color).grid(row=0, column=0)
# 타이틀
tk.Label(root, text="수화번역기", font=(fontstyle, 24), bg=bg_color).grid(row=0,
column=2)
# 추가할 데이터셋 이름 입력 받기
tk.Label(root, text="학습할 말: ", font=(fontstyle, 18), bg=bg_color).grid(row=1,
column=1)
tk.Entry(root, width=25, textvariable=word, font=(fontstyle, 18),
bg=bg_color).grid(row=1, column=2)
# 데이터 수집 버튼
tk.Button(root, text="수집", font=(fontstyle, 15), width=10, command=return_word,
bg=bt_color, fg=bg_color).grid(row=1, column=3)
# 학습 버튼
tk.Button(root, text="학습", font=(fontstyle, 15), width=20, bg=bt_color,
```

```
command=train, fg=bg_color).grid(row=2, column=2, pady=5)
# 번역 버튼
tk.Button(root, text="수화 번역", font=(fontstyle, 15), width=20, command=test,
bg=bt_color, fg=bg_color).grid(row=3, column=2, pady=5)

root.mainloop()
```

create_dataset.py

```
import cv2
import mediapipe as mp
import numpy as np
import time, os

# 대화 예시 ########################
# 안녕하세요 /
# 통장 / 새로 / 만드려고 / 합니다 /
# 신용 / 대출 / 가능한가요? /
# 네 / 아니요 /
# 감사합니다
###################################

def create(word):
    file_path = "classset.txt"# 라벨명
    seq_length = 30 # 윈도우 사이즈 30
    secs_for_action = 30 # 녹화 시간 30초
    # MediaPipe hands model initialize
    mp_hands = mp.solutions.hands
    mp_drawing = mp.solutions.drawing_utils
    hands = mp_hands.Hands(
        max_num_hands=2,
        min_detection_confidence=0.5,
        min_tracking_confidence=0.5)
    cap = cv2.VideoCapture(0)
    cap.set(3, 1080) # set video widht
    cap.set(4, 720) # set video height
    created_time = int(time.time())
```

```
    os.makedirs('dataset', exist_ok=True) # 데이터셋 저장 폴더

    action = word

    actions = []

    data = []

    with open(file_path, 'r', encoding="UTF-8") as f:
        actions = f.read().splitlines()
        print(actions)

    if action not in actions:
        with open(file_path, 'a', encoding="UTF-8") as f:
                f.write(f"{word}₩n")
                id = len(actions)

        if cap.isOpened():
            ret, img = cap.read()
            img = cv2.flip(img, 1)
            cv2.imshow('img', img)
            cv2.waitKey(3000)
            start_time = time.time()
            while time.time() - start_time < secs_for_action: # 30초 동안 녹화
                ret, img = cap.read()
                img = cv2.flip(img, 1)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                result = hands.process(img) # 하나씩 프레임을 읽어서 미디어 파이브에 넣
어줌
                img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
                 if result.multi_hand_landmarks is not None: # 결과를 가지고 각도를 뽑
아내는 과정
                    for res in result.multi_hand_landmarks:
                        joint = np.zeros((21, 4))
                        for j, lm in enumerate(res.landmark):
                            joint[j] = [lm.x, lm.y, lm.z, lm.visibility] # 손가락 각도가 이미
지 상에서 보이는지 안보이는지를 판단
                        # 손가락 관절사이에 각도를 구하는 코드
                        # Compute angles between joints
```

```
                    v1 = joint[[0,1,2,3,0,5,6,7,0,9,10,11,0,13,14,15,0,17,18,19], :3]
# Parent joint
                    v2 = joint[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],
:3] # Child joint
                    v = v2 - v1 # [20, 3]
                    # Normalize v
                    v = v / np.linalg.norm(v, axis=1)[:, np.newaxis]
                    # Get angle using arcos of dot product
                    angle = np.arccos(np.einsum('nt,nt->n',
                        v[[0,1,2,4,5,6,8,9,10,12,13,14,16,17,18],:],
                        v[[1,2,3,5,6,7,9,10,11,13,14,15,17,18,19],:])) # [15,]
                    angle = np.degrees(angle) # Convert radian to degree
                    # 라벨(id)을 넣음
                    angle_label = np.array([angle], dtype=np.float32)
                    angle_label = np.append(angle_label, id)
                    # 수집한 좌표값들을 펼쳐서 concatenate -> 100개짜리 행렬
                    d = np.concatenate([joint.flatten(), angle_label])
                    data.append(d)
                                            mp_drawing.draw_landmarks(img,    res,
mp_hands.HAND_CONNECTIONS)
                cv2.imshow('img', img)
                if cv2.waitKey(1) == ord('q'):
                    cap.release()
                    break

            # numpy array 형태로 변환
            data = np.array(data)
            print(action, data.shape)

            # npy형태로 저장
            np.save(os.path.join('dataset', f'raw_{id}_{action}'), data)
            # Create sequence data
            full_seq_data = []
            for seq in range(len(data) - seq_length):
                full_seq_data.append(data[seq:seq + seq_length])
            full_seq_data = np.array(full_seq_data)
            print(action, full_seq_data.shape)
            np.save(os.path.join('dataset', f'seq_{id}_{action}'), full_seq_data)
```

```
            cap.release()

    else:
        print("exist class!")
```

train.py

```
import numpy as np
import os
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import load_model


# 시퀀스 데이터만 사용하여 학습
def train():
    os.environ['CUDA_VISIBLE_DEVICES'] = '1'
    os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
#
    # 라벨명 추출
    with open("classset.txt", 'r', encoding="UTF-8") as f:
        actions = f.read().splitlines()
    data = []

    # 시퀀스 데이터 로드하여 하나로 합침
    for i in range(len(actions)):
        data.append(np.load(f'dataset/seq_{i}_{actions[i]}.npy'))

    data = np.concatenate(data)
    data.shape
#
```

```
    # 마지막 값에 라벨값이 저장되어 있으므로 분리
    x_data = data[:, :, :-1]
    labels = data[:, 0, -1]
    print(x_data.shape)
    print(labels.shape)
#
    # one-hot encoding
    y_data = to_categorical(labels, num_classes=len(actions))
    y_data.shape
#
    # training set validation set으로 나눔
    x_data = x_data.astype(np.float32)
    y_data = y_data.astype(np.float32)
     x_train, x_val, y_train, y_val = train_test_split(x_data, y_data, test_size=0.1,
random_state=2021)
    print(x_train.shape, y_train.shape)
    print(x_val.shape, y_val.shape)
#
    # 모델 정의
    model = Sequential([
        LSTM(64, activation='relu', input_shape=x_train.shape[1:3]),
        Dense(32, activation='relu'),
        Dense(len(actions), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
    model.summary()
#
    # 200번 학습
    history = model.fit(
        x_train,
        y_train,
        # batch_size=12,
        validation_data=(x_val, y_val),
        epochs=200,
        callbacks=[
            ModelCheckpoint('./models/model2_1.0.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')
```

```
        ]
    )
#
    # 학습 완료 그래프
    # 초록,파랑은 training이랑 validation accuracy
    # 노랑, 빨강은 loss
    fig, loss_ax = plt.subplots(figsize=(16, 10))
    acc_ax = loss_ax.twinx()
    loss_ax.plot(history.history['loss'], 'y', label='train loss')
    loss_ax.plot(history.history['val_loss'], 'r', label='val loss')
    loss_ax.set_xlabel('epoch')
    loss_ax.set_ylabel('loss')
    loss_ax.legend(loc='upper left')
    acc_ax.plot(history.history['acc'], 'b', label='train acc')
    acc_ax.plot(history.history['val_acc'], 'g', label='val acc')
    acc_ax.set_ylabel('accuracy')
    acc_ax.legend(loc='upper left')
    plt.show()
#
    model = load_model('models/model2_1.0.h5')
    y_pred = model.predict(x_val)
    confusion_matrix(np.argmax(y_val, axis=1), np.argmax(y_pred, axis=1))
```

test.py

```
import cv2
import mediapipe as mp
import numpy as np
from tensorflow.keras.models import load_model
from PIL import ImageFont, ImageDraw, Image
def test():
    file_path = "classset.txt"

    with open(file_path, 'r', encoding="UTF-8") as f:
        actions = f.read().splitlines()
        actions.append(' ')
```

```
        print(actions)

    seq_length = 30
    model = load_model('models/model2_1.0.h5')
    # MediaPipe hands model
    mp_hands = mp.solutions.hands
    mp_drawing = mp.solutions.drawing_utils
    hands = mp_hands.Hands(
        max_num_hands=2,
        min_detection_confidence=0.5,
        min_tracking_confidence=0.5)
    cap = cv2.VideoCapture(0)
    cap.set(3, 1080) # set video widht
    cap.set(4, 720) # set video height
    seq = []
    while cap.isOpened():
        ret, img = cap.read()
        img0 = img.copy()

        img = cv2.flip(img, 1)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        result = hands.process(img)
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        if result.multi_hand_landmarks is not None:
            for res in result.multi_hand_landmarks:
                joint = np.zeros((21, 4))
                for j, lm in enumerate(res.landmark):
                    joint[j] = [lm.x, lm.y, lm.z, lm.visibility]
                # Compute angles between joints
                    v1 = joint[[0,1,2,3,0,5,6,7,0,9,10,11,0,13,14,15,0,17,18,19], :3] #
Parent joint
                v2 = joint[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20], :3] #
Child joint
                v = v2 - v1 # [20, 3]
                # Normalize v
                v = v / np.linalg.norm(v, axis=1)[:, np.newaxis]
                # Get angle using arcos of dot product
                angle = np.arccos(np.einsum('nt,nt->n',
                    v[[0,1,2,4,5,6,8,9,10,12,13,14,16,17,18],:],
```

```
                v[[1,2,3,5,6,7,9,10,11,13,14,15,17,18,19],:])) # [15,]
            angle = np.degrees(angle) # Convert radian to degree
            d = np.concatenate([joint.flatten(), angle])
            seq.append(d)
                                        mp_drawing.draw_landmarks(img,      res,
mp_hands.HAND_CONNECTIONS)
            if len(seq) < seq_length:
                continue
                        input_data   =   np.expand_dims(np.array(seq[-seq_length:],
dtype=np.float32), axis=0)
            # 인퍼런스한 결과를 뽑아냄
            y_pred = model.predict(input_data).squeeze()
            # argmax로 인덱스 뽑아냄
            # 그것의 confidence를 뽑아냄
            i_pred = int(np.argmax(y_pred))
            conf = y_pred[i_pred]
            # confidence가 90% 이하면 확실하지 않다 판단해 넘김
            if conf < 0.9:
                continue

            # 90% 이상이면 action 라벨명 추출
            action = actions[i_pred+1]

            # 출력
            img_pil=Image.fromarray(img)
            # PIL 이미지에 한글 입력
            draw = ImageDraw.Draw(img_pil)
                                        draw.text((40,      450),      f"{action}",
font=ImageFont.truetype("./malgun.ttf", 36), fill=(255,255,255))
            # PIL 이미지 -> cv2 Mat 타입으로 변경
            img = np.array(img_pil)


        cv2.imshow('img', img)
        if cv2.waitKey(1) == ord('q'):
            break
```

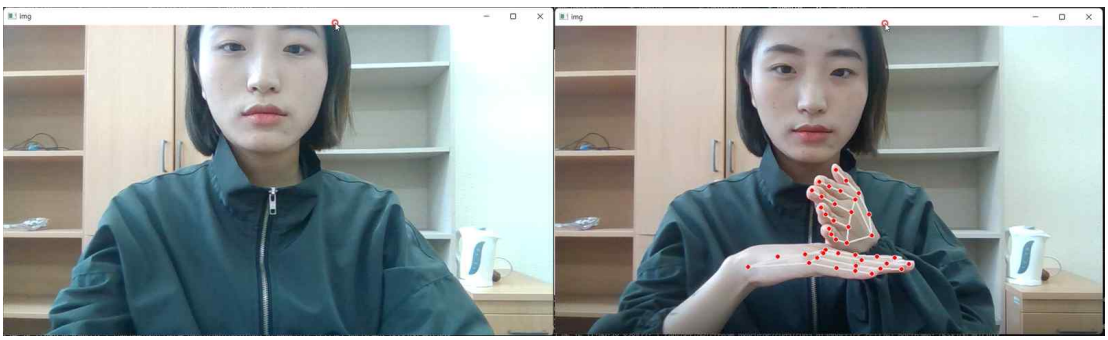
참고 코드: https://github.com/kairess/gesture-recognition

3. 과제 수행 결과

1) dataset 수집 과정

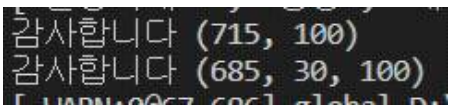
예시로 '감사합니다' 데이터를 수집하는 과정을 아래에 설명한다.

main 코드를 실행시켜서 인터페이스를 띄운다. entry에 '감사합니다'라고 쓰고 수집 버튼을 클릭한다.



처음 카메라가 켜지면 3초간 멈춰있는데 이는 정상적으로 실행되는 것이다. 3초간 멈춰있을 동안 자세를 준비하면 된다.

30초동안 같은 제스처를 입력해주면 된다. 30초가 지나면 화면이 멈추는데 이는 완료되었다는 뜻이다.



추가 완료 후 터미널 창을 확인해보면 라벨명이 감사합니다이고 수집한 training set과 vaildation set 값이 출력된다. 위가 vaildation set, 아래가 training set이다.

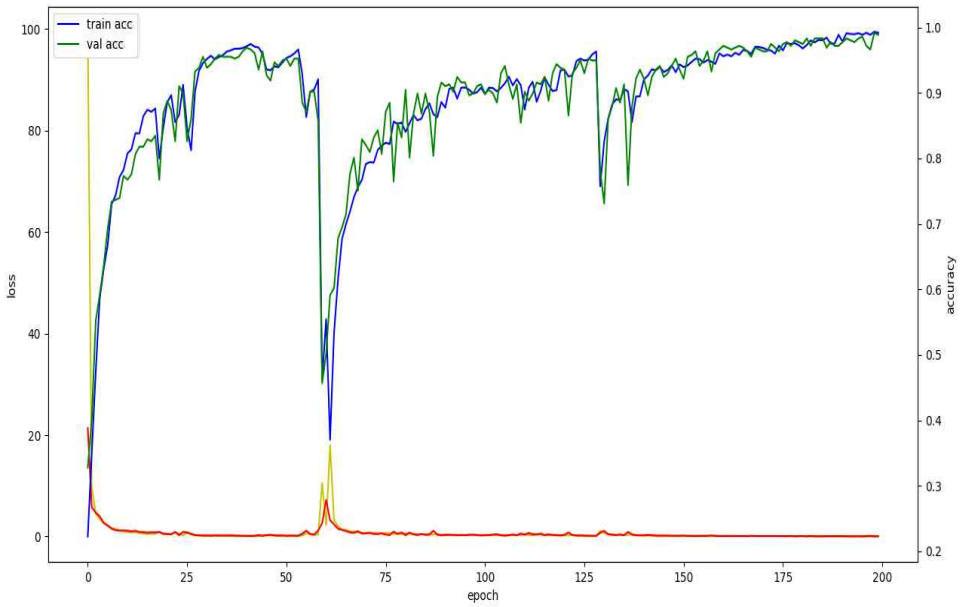
2) 학습 알고리즘 실행

```
(357, 30, 99) (357, 7)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 64)                41984
_________________________________________________________________
dense (Dense)                (None, 32)                2080
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 231
=================================================================
Total params: 44,295
Trainable params: 44,295
Non-trainable params: 0
```

학습 코드 실행 시 터미널 창이다. 위 사진대로 라면 7개의 라벨로 분류가 되었고 44,295 파라미터를 학습시키는 것이다.

```
Epoch 00194: val_acc did not improve from 0.98319
Epoch 195/200
101/101 [==============================] - 2s 22ms/step - loss: 0.0249 - acc: 0.9947 - val_loss: 0.0346 - val_acc: 0.9832

Epoch 00195: val_acc did not improve from 0.98319
Epoch 196/200
101/101 [==============================] - 2s 23ms/step - loss: 0.0325 - acc: 0.9879 - val_loss: 0.0349 - val_acc: 0.9860

Epoch 00196: val_acc improved from 0.98319 to 0.98599, saving model to ./models\model2_1.0.h5
Epoch 197/200
101/101 [==============================] - 2s 23ms/step - loss: 0.0203 - acc: 0.9935 - val_loss: 0.0808 - val_acc: 0.9720

Epoch 00197: val_acc did not improve from 0.98599
Epoch 198/200
101/101 [==============================] - 2s 24ms/step - loss: 0.0316 - acc: 0.9883 - val_loss: 0.1319 - val_acc: 0.9664

Epoch 00198: val_acc did not improve from 0.98599
Epoch 199/200
101/101 [==============================] - 3s 26ms/step - loss: 0.0356 - acc: 0.9910 - val_loss: 0.0410 - val_acc: 0.9916

Epoch 00199: val_acc improved from 0.98599 to 0.99160, saving model to ./models\model2_1.0.h5
Epoch 200/200
101/101 [==============================] - 2s 24ms/step - loss: 0.0241 - acc: 0.9911 - val_loss: 0.0461 - val_acc: 0.9888

Epoch 00200: val_acc did not improve from 0.99160
```

학습이 완료된 터미널 창이다. accuracy가 0.99160이 나왔다. 파일 서버를 구축하여 돌리는 것이 아닌 개인 저사양 노트북에서 돌리다보니 100%가 나오기가 어렵다.



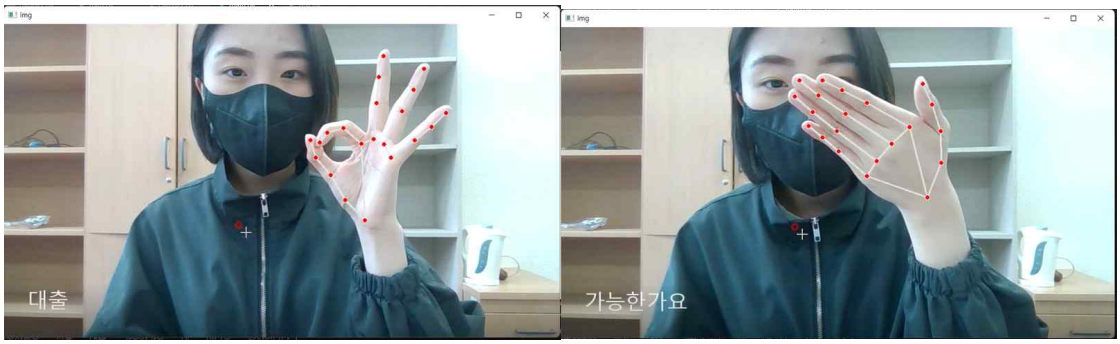
그래프를 출력해보면 그래도 accuracy가 100%에 가깝게 나온 걸 확인할 수 있다.

3) 수화 번역

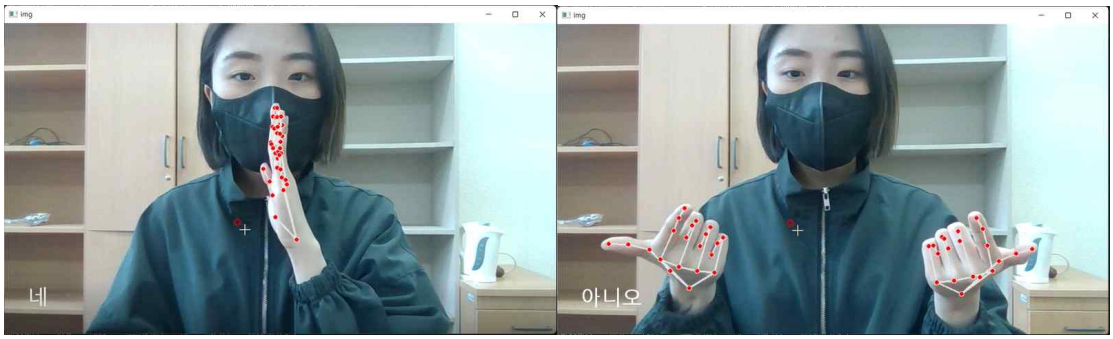
성공적으로 모델이 학습되어 '감사합니다' 수화를 인식하고 출력해내는 것을 확인할 수 있다.

외에도 '안녕하세요', '신용', '대출', '가능한가요', '네', '아니오' 6가지의 수화를 데이터셋에 추가해놓았다.

'안녕하세요'와 '신용'을 인식하는 모습이다.



‘대출’와 ‘가능한가요’를 인식하는 모습이다.



‘네’와 ‘아니오’를 인식하는 모습이다.

바로바로 인식하지는 않지만 잠시 제스처를 유지하거나 2번 정도 제스처를 취하면 인식이 잘 된다.