영상 음성인식 및 표정(제스쳐) 인식

Face&Motion regognition And Speaker Diarization with emotion Analysis

프로젝트 개요

google speech to text기반 음성인식

hugging face 모델 기반 화자분리 (화자분리 데이터 기반으로 화자별 내용 분리)

openAi LLM API 활용 텍스트 요약

SpeechBrain(word2Vec기반) 화자별 음성 감정분석 및 화자 음성데이터 기반 유사도 검사

OpenCV 및 mediaPipe기반 표정인식, 제스쳐인식

모듈 설명

1. google speech to text기반 음성인식

```
from pyannote.audio import Pipeline
from google.cloud import speech_v1p1beta1 as speech
from pydub import AudioSegment
import io
import wave
import torchaudio
import pandas as pd
import torch

def get_sample_rate(file_path):
    """WAV 파일의 샘플 레이트를 확인합니다."""
    with wave.open(file_path, 'rb') as wf:
        sample_rate = wf.getframerate()
    return sample_rate

def convert_to_mono(audio):
    """오디오 파일을 모노로 변환합니다."""
    if audio.channels != 1:
        audio = audio.set_channels(1)
    return audio

def convert_to_16bit(audio):
    """WAV 파일을 16비트 샘플로 변환합니다."""
    return audio.set_sample_width(2)  # 16비트 샘플

def transcribe_audio_chunk(audio_chunk, sample_rate):
    """Google Cloud Speech-to-Text API를 사용하여 음성을 텍스트로 변환합니다."""
    client =
speech.SpeechClient.from_service_account_file('../apiKey/myKey.json')

    # 오디오 조각을 메모리에서 처리
    with io.BytesIO() as audio_file:
        audio_chunk.export(audio_file, format="wav")
        audio_file.seek(0)
        content = audio_file.read()
```

```
    audio = speech.RecognitionAudio(content=content)
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=sample_rate,
        language_code="ko-KR",
    )

    # 파일이 길 경우, long_running_recognize 사용
    operation = client.long_running_recognize(config=config, audio=audio)
    print('Waiting for operation to complete...')
    response = operation.result(timeout=90)

    transcripts = []
    for result in response.results:
        transcripts.append(result.alternatives[0].transcript)

    return transcripts

def transcribe_audio_file(file_path, diarization_results):
    """오디오 파일을 텍스트로 변환하고 화자별로 대화 내용을 저장합니다."""
    # 오디오 파일의 샘플 레이트 확인
    sample_rate = get_sample_rate(file_path)

    # 오디오 파일 로드 및 변환
    audio = AudioSegment.from_file(file_path)
    audio = convert_to_mono(audio)
    audio = convert_to_16bit(audio)

    output_data = []

    # 화자 별로 음성을 인식
    for segment in diarization_results:
        start_time = segment['start'] * 1000  # milliseconds
        stop_time = segment['stop'] * 1000  # milliseconds
        speaker = segment['speaker']

        # 해당 구간의 오디오 조각 추출
        audio_chunk = audio[start_time:stop_time]

        # 오디오 조각 텍스트 변환
        print(f"Transcribing {speaker} from {segment['start']} to
{segment['stop']} seconds...")
        transcripts = transcribe_audio_chunk(audio_chunk, sample_rate)

        # 대화 내용을 문자열로 합침
        dialog_content = ' '.join(transcripts)

        # 데이터 추가
        output_data.append({
            'start': segment['start'],
            'stop': segment['stop'],
            'speaker': speaker,
            'dialogue': dialog_content
        })

    return output_data
```

```python
def diarize_audio(file_path, num_speakers):
    """화자 분리 수행 후 결과를 반환합니다."""
    access_token = "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"

    # 사전 훈련된 화자 분리 모델 로드
    pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization-3.1",
use_auth_token=access_token)
    pipeline.to(torch.device("cuda"))

    # 오디오 파일 로드
    waveform, sample_rate = torchaudio.load(file_path)

    # 화자 분리 수행
    diarization = pipeline({"waveform": waveform, "sample_rate":
sample_rate}, num_speakers=num_speakers)

    results = []
    for turn, _, speaker in diarization.itertracks(yield_label=True):
        results.append({
            'start': turn.start,
            'stop': turn.end,
            'speaker': f'speaker_{speaker}'
        })

    return results

# 사용 예제
audio_file_path = '../resource/audio.wav'

# 화자 수 입력 받기
num_speakers = int(input("화자 수를 입력하세요: "))  # 예: 2

# 화자 분리 수행
diarization_results = diarize_audio(audio_file_path, num_speakers)

# 대화 내용을 텍스트로 변환
output_data = transcribe_audio_file(audio_file_path, diarization_results)

# DataFrame 생성
df = pd.DataFrame(output_data)

# DataFrame을 엑셀 파일로 저장
output_file = "../resource/fullText.xlsx"
df.to_excel(output_file, index=False)

print(f"화자별 대화 내용이 {output_file}에 저장되었습니다.")
```

```
(videoAnalysis) C:₩Windows₩System32₩videoAnalysis₩module>python speech2text2.py
C:₩WINDOWS₩system32₩videoAnalysis₩Lib₩site-packages₩pydub₩utils.py:170: RuntimeWarning: Couldn't find ffmpeg or avconv -
 defaulting to ffmpeg, but may not work
  warn("Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may not work", RuntimeWarning)
Transcribing chunk 0...
Waiting for operation to complete...
['세 번째 키워도가 볼게요 소리와 빛의 4일 산타 루돌프 마지막 크리스마스에 관련된 건데 크리스마스에 상징인 산타클로스 루
돌프 하루에 전세계 모든 어린이들이 사는 집에 선물을 배달하는 루돌프의 속도와 썰매 무게는 얼마나 될까요 일단은 배달 시간
을 먼저 가정을 해야 되는데 밤 10시부터 새벽 6시 까지로 가정 해요 그 일반 시간이니까 안 되는 거죠 근데 8시간']
```

mp4파일을 moviepy라이브러리를 통해 wav형식으로 변환한 후, Google Api중 speech to text를 활용해 음성을 텍스트로 추출한다.

2. hugging face pyannote 모델 기반 화자분리 (화자분리 데이터 기반으로 화자별 내용 분리)

```
from pyannote.audio import Pipeline
import torchaudio
import torch
from pyannote.audio.pipelines.utils.hook import ProgressHook
import pandas as pd  # pandas 라이브러리 추가

# 여기에 본인의 액세스 토큰을 입력하세요
access_token = "-"

# 사전 훈련된 화자 분리 모델 로드
pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization-3.1",
use_auth_token=access_token)
pipeline.to(torch.device("cuda"))

# 사용자로부터 화자 수 입력 받기
num_speakers = int(input("화자 수를 입력하세요: "))  # 예: 2

# 오디오 파일 로드
waveform, sample_rate = torchaudio.load("../resource/audio.wav")

# 진행 상황을 확인하기 위해 ProgressHook 사용
with ProgressHook() as hook:
    # 전체 파일에 대해 추론 실행
    diarization = pipeline({"waveform": waveform, "sample_rate":
sample_rate},
                           hook=hook,
                           num_speakers=num_speakers)

# 결과를 저장할 리스트 초기화
results = []

# 화자 분리 결과를 리스트에 저장
for turn, _, speaker in diarization.itertracks(yield_label=True):
    result_line = {
        'start': turn.start,
        'stop': turn.end,
        'speaker': f'speaker_{speaker}'
    }
    results.append(result_line)

# DataFrame 생성
df = pd.DataFrame(results)

# DataFrame을 엑셀 파일로 저장
output_file = "../resource/fullText.xlsx"
df.to_excel(output_file, index=False)  # 인덱스 없이 저장

print(f"화자 분리 결과가 {output_file}에 저장되었습니다.")
```

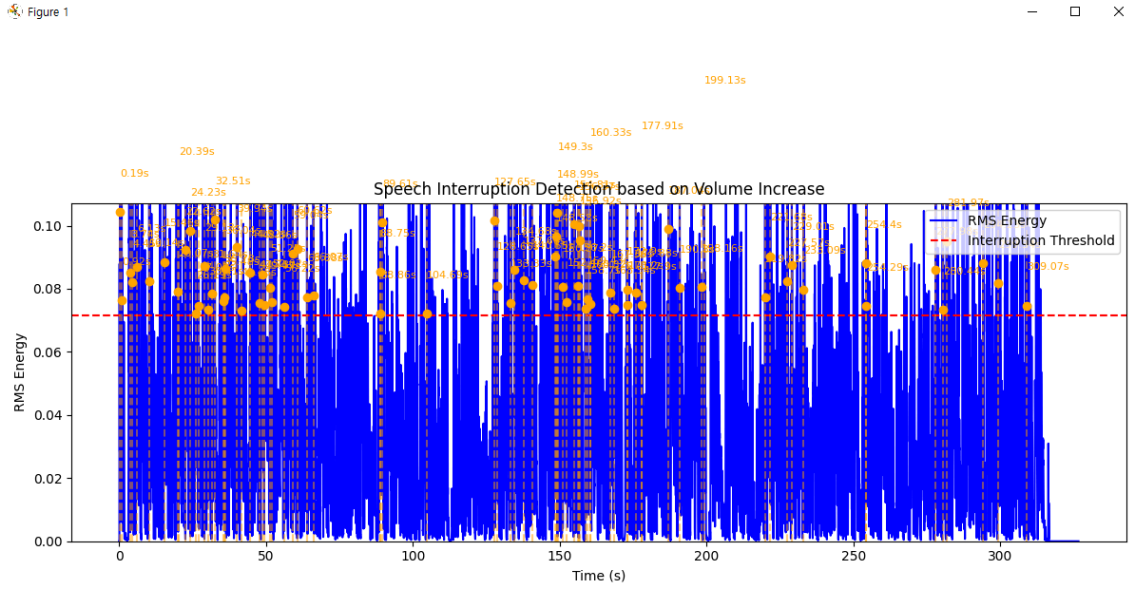
```
(videoAnalysis) C:₩Windows₩System32₩videoAnalysis₩module>python diarization.py
C:₩Users₩bandl₩AppData₩Local₩Programs₩Python₩Python311₩Lib₩inspect.py:992: UserWarning: Module 'speechbrain.pretrained'
was deprecated, redirecting to 'speechbrain.inference'. Please update your script. This is a change from SpeechBrain 1.0
. See: https://github.com/speechbrain/speechbrain/releases/tag/v1.0.0
  if ismodule(module) and hasattr(module, '__file__'):
화자 수를 입력하세요: 2
C:₩WINDOWS₩system32₩videoAnalysis₩Lib₩site-packages₩pyannote₩audio₩utils₩reproducibility.py:74: ReproducibilityWarning:
TensorFloat-32 (TF32) has been disabled as it might lead to reproducibility issues and lower accuracy.
It can be re-enabled by calling
   >>> import torch
   >>> torch.backends.cuda.matmul.allow_tf32 = True
   >>> torch.backends.cudnn.allow_tf32 = True
See https://github.com/pyannote/pyannote-audio/issues/1370 for more details.

  warnings.warn(
C:₩WINDOWS₩system32₩videoAnalysis₩Lib₩site-packages₩pyannote₩audio₩models₩blocks₩pooling.py:104: UserWarning: std():
degrees of freedom is <= 0. Correction should be strictly less than the reduction factor (input numel divided by output
numel). (Triggered internally at
C:₩actions-runner₩_work₩pytorch₩pytorch₩builder₩windows₩pytorch₩aten₩src₩ATen₩native₩ReduceOps.cpp:1823.)
  std = sequences.std(dim=-1, correction=1)
segmentation      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00
speaker_counting  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00
embeddings        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  60% 0:00:04
```

hugging face 에 업로드 되어있는 pyannote speak Diarization 모델의 Pretrainned 모델을 활용해 화자수 입력데이터를 기반으로 화자분리를 진행한다.

	A	B	C	D	E	F	G	H	I
1	start	stop	speaker	dialogue					
2	0.030969	1.718469	speaker_SPEAKER_01	세 번째 키워도가 볼게요					
3	3.068469	7.034094	speaker_SPEAKER_01	소리와 빛의 사이 산타는 루돌프가 잡아야 돼 크리스마스에 관련					
4	6.645969	7.287219	speaker_SPEAKER_00						
5	7.371594	17.66534	speaker_SPEAKER_01	크리스마스의 상징인 산타클로스 루돌프 하루에 전세계 모든 어					
6	17.78347	18.72847	speaker_SPEAKER_00	재밌죠 이거					
7	18.40784	19.65659	speaker_SPEAKER_01	내일					
8	18.98159	33.47722	speaker_SPEAKER_00	일단은 싼타페 배달 시간을 먼저 가정을 해야 되는데 밤 10시부터					
9	24.93847	25.03972	speaker_SPEAKER_01						
10	25.19159	25.42784	speaker_SPEAKER_01						
11	27.84097	28.95472	speaker_SPEAKER_01						

이처럼 타임스탬프와 더불어 얘기한 내용을 연계하여 결과물로 얻는다. 이를 기반으로 start시간과 stop시간 계산을 통해 대화중 끼어드는 사람또한 찾을 수있다. 주황색점이 여러명의 화자가 얘기해 RMS에너지가 높아진 경우이며 이러한 경우에 타임스탬프를 통해 누가 끼어들었는지를 확인할 수 있다. 이를 통해 화자가 얘기한 내용을 더욱 정확히 분리할 수 있다.



3. openAi LLM API 활용 텍스트 요약

```
import os
from openai import OpenAI

client = OpenAI(api_key="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")  # 여기에서 YOUR_API_KEY를 실제 API 키로 변경하세요
```

```
def summarize_text(file_path):
    # 텍스트 파일 읽기
    with open(file_path, "r", encoding="utf-8") as f:
        text = f.read()

    # OpenAI API를 사용하여 텍스트 요약하기
    chat_completion = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": f"한국어 회의 내용을 주요사건 및 논의점을 요약해서 설명
해:\n\n{text}"
            }
        ],
        model="gpt-3.5-turbo",
        max_tokens=150,  # 요약의 최대 토큰 수
        temperature=0.5,  # 창의성 조절
    )

    # 요약 결과 추출
    summary = chat_completion.choices[0].message.content
    return summary

# 사용 예제
file_path = "../resource/fullText.txt"
summary = summarize_text(file_path)

# 요약 결과 출력
print("요약 결과:")
print(summary)

# 요약 결과를 파일에 저장
with open("../resource/summary.txt", "w", encoding="utf-8") as f:
    f.write(summary)

print("요약이 summary.txt에 저장되었습니다.")
```

```
(videoAnalysis) C:₩Windows₩System32₩videoAnalysis₩module>python LLMsumurize.py
요약 결과:
이 회의는 크리스마스와 관련된 주요 사건과 논의점을 다루었습니다. 산타클로스와 루돌프가 선물을 배달하는 과정에서 필요한 시간과 무게에 대한 논의가 진행되었으며, 선물의 양과 가격, 레이저로 선물을 전달하는 방법 등이 다뤄졌습니다. 또한, 산타
요약이 summary.txt에 저장되었습니다.
```

이처럼 추출한 텍스트를 기반으로 요약을 생성한다.

4. 화자별 음성데이터 기반 음성 감정 분석 및 화자 검증

```
import json
import torch
from pydub import AudioSegment
from speechbrain.inference.diarization import Speech_Emotion_Diarization
import os

# 모델 불러오기 (GPU 사용 설정 포함)
classifier = Speech_Emotion_Diarization.from_hparams(
    source="speechbrain/emotion-diarization-wavlm-large",
    run_opts={"device": "cuda"}  # GPU 사용 설정
```

```
)

# 오디오 파일을 분할하고 감정 분석을 수행할 함수 정의
def analyze_emotion_segments(audio_path, segment_duration=30):
    # 오디오 파일 로드
    audio = AudioSegment.from_wav(audio_path)

    # 분할한 감정 분석 결과 저장할 리스트
    results = []

    # 30초 단위로 오디오 분할 및 감정 분석
    for i in range(0, len(audio), segment_duration * 1000):  # pydub은 ms 단위
사용
        segment = audio[i:i + segment_duration * 1000]

        # 각 세그먼트의 시간 보정
        base_time = i / 1000  # ms를 초로 변환

        # 세그먼트를 분석에 사용 (임시 파일을 현재 작업 디렉토리에 저장)
        temp_segment_path = "../resource/temp_segment.wav"
        segment.export(temp_segment_path, format="wav")
        diary = classifier.diarize_file(temp_segment_path)

        # 분석 결과에 시간 보정을 적용하여 저장
        for entry in diary[temp_segment_path]:
            adjusted_entry = {
                'start': entry['start'] + base_time,
                'end': entry['end'] + base_time,
                'emotion': entry['emotion']
            }
            results.append(adjusted_entry)

        # GPU 메모리 캐시 정리
        torch.cuda.empty_cache()

    return results

# 오디오 파일 목록
audio_files = ["../resource/audio.wav"]

# 전체 분석 결과 저장할 딕셔너리
emotion_analysis_results = {}

# 각 파일에 대해 감정 분석 수행
for audio_path in audio_files:
    results = analyze_emotion_segments(audio_path)
    emotion_analysis_results[audio_path] = results

    # 분석 결과 출력
    print(f"Results for {audio_path}:")
    for segment in results:
        print(f"Start: {segment['start']}s, End: {segment['end']}s, Emotion:
{segment['emotion']}")
    print("-" * 40)

# 결과를 JSON 파일로 저장
```

```
with open("../result/emotion_analysis_results.txt", "w") as json_file:
    json.dump(emotion_analysis_results, json_file, indent=4)

print("Emotion analysis results saved to 'emotion_analysis_results.txt'")
```

```
Start: 293.08s, End: 293.16s, Emotion: n
Start: 293.16s, End: 296.02s, Emotion: h
Start: 296.02s, End: 297.94s, Emotion: n
Start: 297.94s, End: 298.2s, Emotion: h
Start: 298.2s, End: 298.22s, Emotion: n
Start: 298.22s, End: 298.24s, Emotion: h
Start: 298.24s, End: 298.26s, Emotion: n
Start: 298.26s, End: 298.28s, Emotion: h
Start: 298.28s, End: 298.36s, Emotion: n
Start: 298.36s, End: 298.42s, Emotion: h
Start: 298.42s, End: 299.98s, Emotion: n
Start: 300.0s, End: 303.06s, Emotion: n
Start: 303.06s, End: 303.32s, Emotion: h
Start: 303.32s, End: 306.7s, Emotion: n
Start: 306.7s, End: 306.94s, Emotion: h
Start: 306.94s, End: 306.96s, Emotion: n
Start: 306.96s, End: 307.0s, Emotion: h
Start: 307.0s, End: 307.04s, Emotion: n
Start: 307.04s, End: 307.06s, Emotion: h
Start: 307.06s, End: 308.12s, Emotion: n
Start: 308.12s, End: 312.82s, Emotion: h
Start: 312.82s, End: 314.26s, Emotion: n
Start: 314.26s, End: 314.32s, Emotion: h
Start: 314.32s, End: 314.36s, Emotion: n
Start: 314.36s, End: 316.6s, Emotion: h
Start: 316.6s, End: 326.66s, Emotion: a
----------------------------------------
Emotion analysis results saved to 'emotion_analysis_results.txt'
```

다음과 같이 타임스탬프를 기반으로 감정을 Anger, Sadness, Happiness, Neutral로 구분지어 감정 분석한다.

또한 이전 pyannote모델을 통해 화자분리를 진행하였기에 이 데이터와 연계하고 더불어 Speaker0, Speaker1등의 임시 화자이름을 사용하는 것이 아닌, 화자의 음성데이터 DB를 만들고 평서문 데이터를 기반으로 유사도를 검사해 누가 얘기하고 있는것인지를 예측한다.

```
import torch
from speechbrain.pretrained import SpeakerRecognition

# 1. 모델 로드 (CPU 사용 설정)
device = "cpu"  # CPU 사용
model = SpeakerRecognition.from_hparams(source="speechbrain/spkrec-ecapa-voxceleb", savedir="tmp_model").to(device)

# 2. 음성 파일 목록 설정
audio_files = ["../resource/test1.wav", "../resource/test2.wav"]

# 3. 두 음성 파일의 유사도 계산
if len(audio_files) == 2:
    # 각 음성을 CPU에 맞게 로드
    score, prediction = model.verify_files(audio_files[0], audio_files[1])

    # 결과 출력
```

```
    result = (
        f"Results for comparing {audio_files[0]} and {audio_files[1]}:\n"
        f"Similarity Score: {score}\n"  # 점수를 변환하지 않고 사용
        f"Are the speakers the same? {'Yes' if prediction else 'No'}\n"
        + "-" * 40 + "\n"
    )

    # 결과를 ../result/similarity.txt 파일에 저장
    with open("../result/similarity.txt", "w") as file:
        file.write(result)

    # 콘솔에 결과 출력
    print(result)
    print("Results saved to '../result/similarity.txt'.")
```

```
Results for comparing ../resource/test1.wav and ../resource/test2.wav:
Similarity Score: tensor([0.8506])
Are the speakers the same? Yes
----------------------------------------

Results saved to '../result/similarity.txt'.
```

본인의 음성데이터를 기반으로 테스트 해본 결과이며, 다른 문장을 얘기하더라도 기본적인 음성의 피치 및 톤을 기반으로 유사도를 검출해 Speaker0이 누구인지를 유추할 수 있는 모듈을 구현했다.

5. OpenCV 및 mediaPipe기반 표정인식, 제스쳐인식

```
import cv2
import mediapipe as mp

# MediaPipe 초기화
mp_face_mesh = mp.solutions.face_mesh
mp_hands = mp.solutions.hands

# 비디오 파일 경로
video_file = "path/to/your/video.mp4"  # 사용할 MP4 파일의 경로로 변경하세요

# 비디오 캡처 객체 생성
cap = cv2.VideoCapture(video_file)

# MediaPipe 얼굴 메쉬 및 손 인식 초기화 (GPU 사용)
with mp_face_mesh.FaceMesh(max_num_faces=1, min_detection_confidence=0.5) as
face_mesh, \
     mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5) as hands:

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # 색상 변환 (BGR to RGB)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # 얼굴 메쉬 인식
        face_results = face_mesh.process(rgb_frame)
```

```
        # 손 인식
        hand_results = hands.process(rgb_frame)

        # 얼굴 메쉬 그리기
        if face_results.multi_face_landmarks:
            for face_landmarks in face_results.multi_face_landmarks:
                for landmark in face_landmarks.landmark:
                    h, w, _ = frame.shape
                    x = int(landmark.x * w)
                    y = int(landmark.y * h)
                    cv2.circle(frame, (x, y), 2, (0, 255, 0), -1)

        # 손 제스처 그리기
        if hand_results.multi_hand_landmarks:
            for hand_landmarks in hand_results.multi_hand_landmarks:
                for landmark in hand_landmarks.landmark:
                    h, w, _ = frame.shape
                    x = int(landmark.x * w)
                    y = int(landmark.y * h)
                    cv2.circle(frame, (x, y), 2, (255, 0, 0), -1)

        # 결과 프레임 보여주기
        cv2.imshow('MediaPipe Face and Hand Recognition', frame)

        # 'q' 키를 눌러 종료
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# 비디오 캡처 해제 및 창 닫기
cap.release()
cv2.destroyAllWindows()
```

mediaPipe를기반으로 얼굴 및 손을 인식한다. 이는 OpenCV로도 인식이 가능하며 이러한 인식만으로는 감정분석 결과를 얻을 수 없기에 DeepFace모델을 같이 활용하여 감정분석 결과를 얻는다.

```
import cv2
from deepface import DeepFace

# 비디오 파일 경로
video_file = "path/to/your/video.mp4"  # 사용할 MP4 파일의 경로로 변경하세요

# 비디오 캡처 객체 생성
cap = cv2.VideoCapture(video_file)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # OpenCV를 사용하여 얼굴 인식
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

```
    detected_faces = faces.detectMultiScale(gray_frame, scaleFactor=1.1,
minNeighbors=5)

    # 각 얼굴에 대해 감정 분석 수행
    for (x, y, w, h) in detected_faces:
        face = frame[y:y+h, x:x+w]
        result = DeepFace.analyze(face, actions=['emotion'],
enforce_detection=False)

        # 감정 결과 추출
        emotion = result[0]['dominant_emotion']
        confidence = result[0]['emotion'][emotion]

        # 감정 결과를 프레임에 표시
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(frame, f"{emotion}: {confidence:.2f}", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

    # 결과 프레임 보여주기
    cv2.imshow('OpenCV and DeepFace Emotion Recognition', frame)

    # 'q' 키를 눌러 종료
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# 비디오 캡처 해제 및 창 닫기
cap.release()
cv2.destroyAllWindows()
```

하지만, 아직 인식과 감정분석결과를 위한 개발환경을 설정중에 있으며 아직 유의미한 결과데이터를 얻지 못해 스크린샷은 따로 첨부하지 못했다.

이전에는 Video에서 텍스트를 추출 및 요약하여 Video Context로 활용하였지만,

음성데이터를 기반으로 화자분리 및 추론(누가 무슨 얘기를 했는가)을 진행하고 더불어 감정분석을 진행하여

청각 데이터에 대한 Feature들을 추가하였으며 이를 통해 영상을 더욱 정교하게 이해할 수 있다.

추가로 영상에 등장한 인물의 표정 및 제스쳐 인식또한 개발진행중에 있다. 이를 통해 청각적 데이터 및 시각적 데이터가 추가된 VideoContext를 기반으로 Opinnion Minning결과와 연계한다면 FollowSupport 분석 결과물의 정확성을 더 올릴 수 있을 것이다.