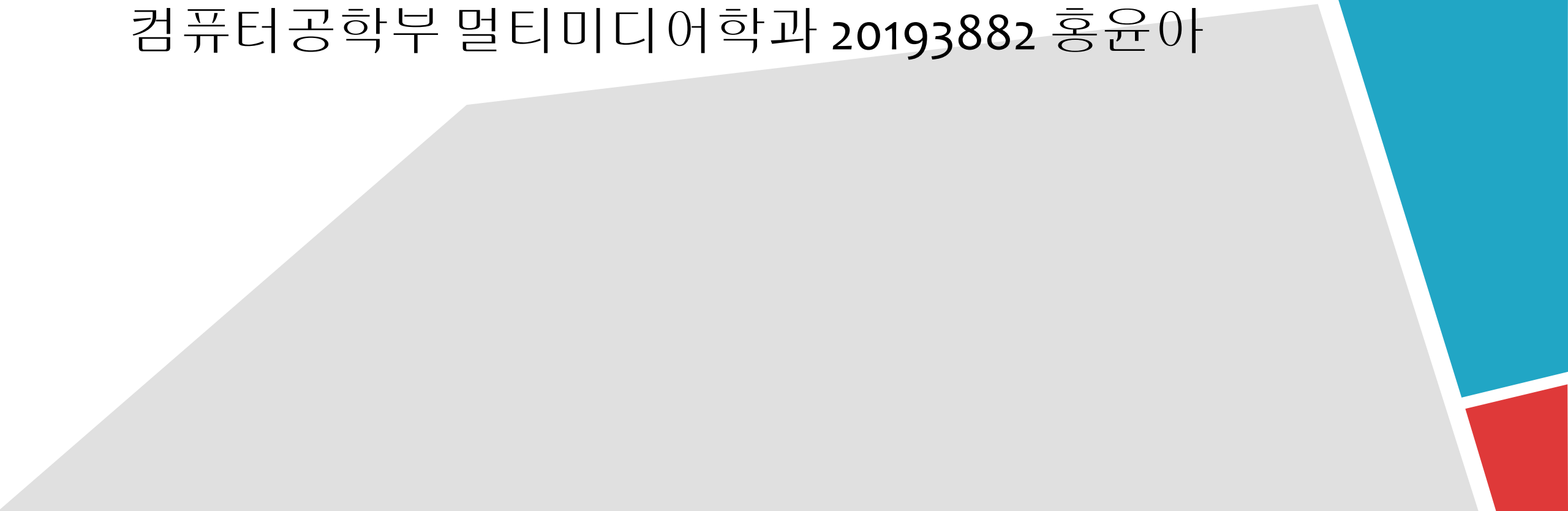




opencv와 딥러닝을 이용한 감정인식

컴퓨터공학부 멀티미디어학과 20193882 홍윤아



목차

- ▶ 주제 선정배경
- ▶ 기능설명
- ▶ 참고사이트
- ▶ 기능개요도
- ▶ UI설명
- ▶ 코드설명(start.py, ui_test.py,
UI_func.py,sound_learning.py,Audio_Video.py)

주제 선정배경

- ▶ 1.딥러닝부문의 음성과 영상처리에 관심이 있음.
- ▶ 2.딥러닝을 이용해 음성의 감정인식하는 프로젝트를 해보았는데 이 프로젝트를 확장시켜서 동시에 영상을 이용한 감정인식도 함께 할 수 있는 프로젝트를 하려는 계획을 세우게 되었다.

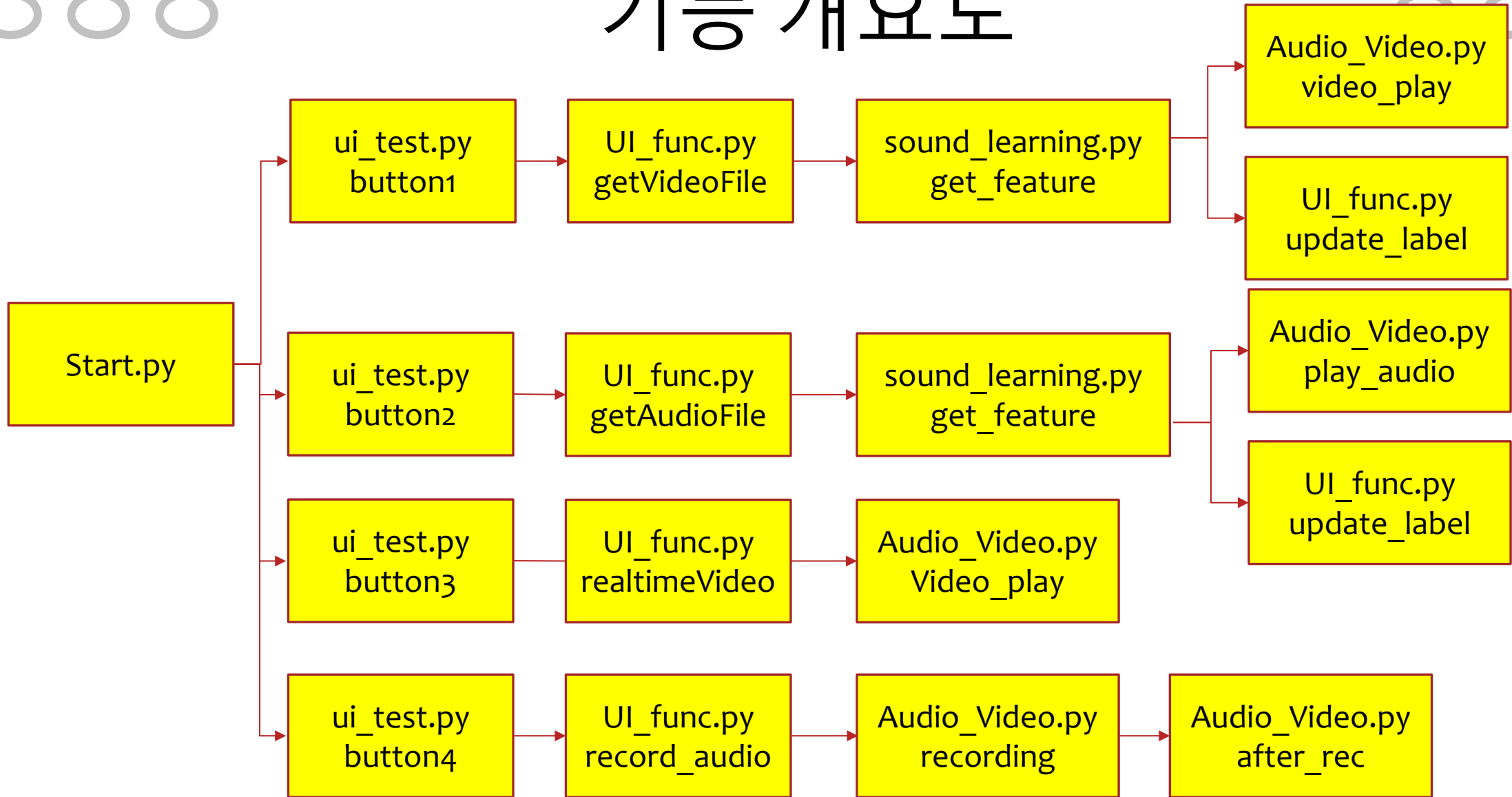
기능설명

- ▶ 1.시작할때마다 다시 학습시킬 것인지 물어본 후 UI실행(학습을 위한 layer을 다시 쌓고 매개변수들은 여러 번 반복해서 돌린 결과 가장 잘 나온 것으로 사용하였다.)
- ▶ 2.영상을 선택해서 음성을 추출한 후 영상 길이에 따라 분리해서 딥러닝으로 학습된 모델로 예측시킨 뒤 opencv로 얼굴인식을, 딥러닝으로 음성인식을 동시에 진행한다. 이 경우 앞면,측면, 기울어진 얼굴 모두 인식 가능하다.
- ▶ 3.음성파일을 선택해서 분리해서 딥러닝으로 예측시킨 뒤 음성인식을 진행한다.처리가 완료되면 음성파일과 결과값이 동시에 나온다. 이 경우 대화,노래(뮤지컬)모두 가능하다.
->영상을 사용할 경우에는 필요한 영상을 미리 다운받아서 audio와 video폴더에 넣어놔야한다.
- ▶ 4.사용자가 직접 음성을 녹음해서 파일의 이름을 지정해서 데이터셋으로 사용할 수 있다.
- ▶ 5.한번에 한번의 기능만 사용할 수 있다. 영상과 음성파일을 응용해서 스스로 데이터셋을 만든다.

참고사이트

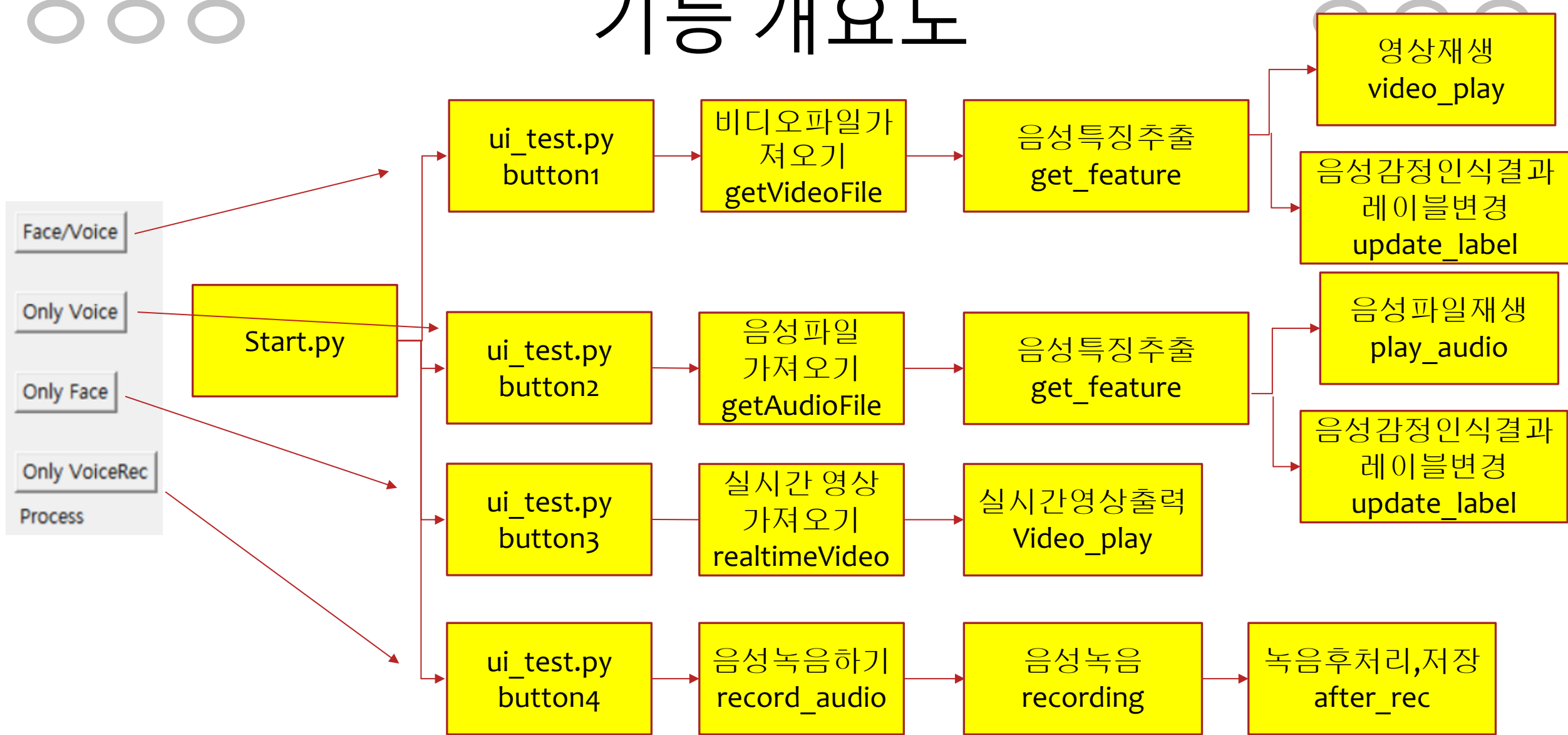
- ▶ Opencv얼굴감정인식을 위한 기본코드+ 음성데이터셋
<https://github.com/omar178/Emotion-recognition>
 - ▶ 딥러닝음성감정인식을 위한 기본코드
<https://medium.com/@raihanh93/speech-emotion-recognition-using-deep-neural-network-part-i-68edb5921229>
 - ▶ 얼굴회전 참조코드
<https://a292run.tistory.com/entry/Face-Alignment-for-Face-Recognition-in-Python-within-OpenCV-1>
- >모두 응용해서 사용

기능 개요도





기능 개요도



UI설명

동영상이나 실시간 영상을 재생하고 얼굴의 감정을 표시한다.

감정을 인식할 Video 파일을 가져오고 파일의 경로를 표시한다. Flag =1이다.

감정을 인식할 음성wav파일을 가져오고 파일의 경로를 표시한다. Flag=2이다.

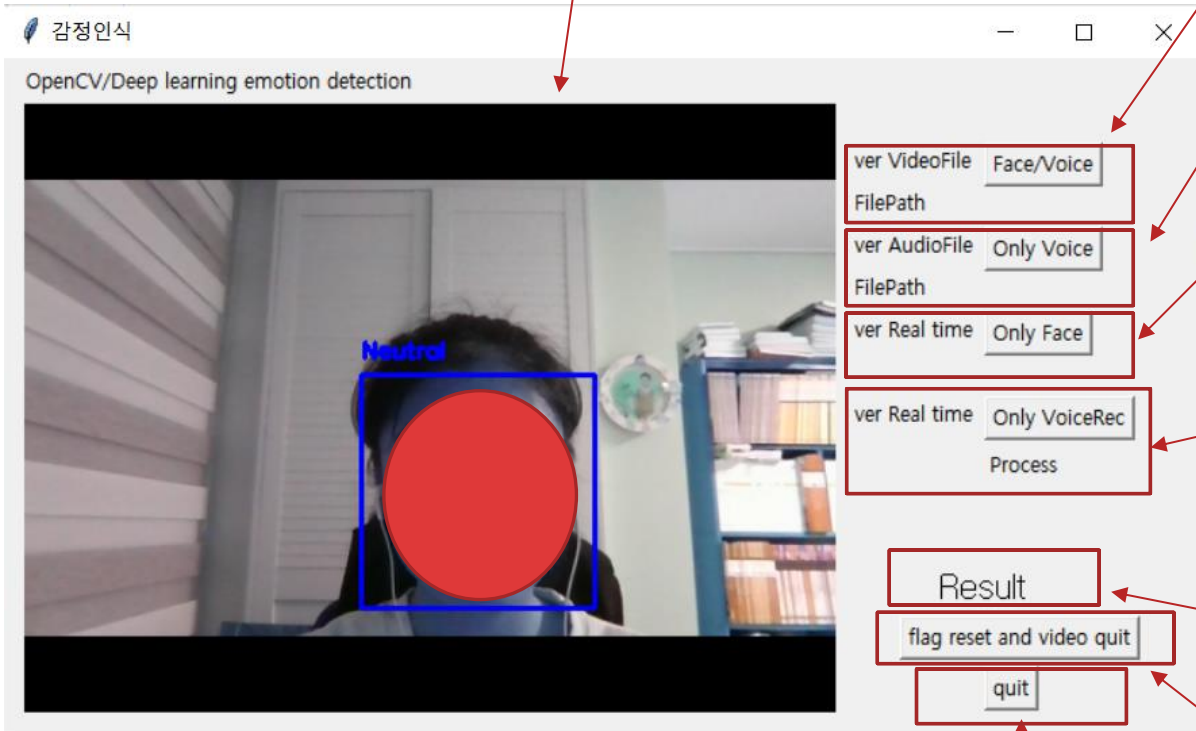
실시간으로 카메라를 통해 영상을 받아와서 실행하고 얼굴이 있으면 얼굴의 감정을 인식한다. Flag=3이다.

사용자가 직접 음성을 녹음해서 테스트파일을 만들고 제목으로 레이블링을 할 수 있다. 녹음이 끝났는지 진행되고 있는지, 몇번째 녹음인지 표시한다. Flag=4이다.

음성파일의 감정인식의 결과를 출력한다.

현재 실행되고 있는 기능을 멈추고 flag의 값을 리셋한다. Flag=0이다.

화면을 종료한다.





- Import 목록 생략

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```
def main():
    cap = cv.VideoCapture(0) # VideoCapture 객체 정의

    window = Tk()
    window.title("감정인식")
    window.geometry("710x400")
    learning = Test()
    rec = Audio_Video()
    func = UI_func(window, rec, learning)
    app = Application(window, func, rec, cap, master=window)

    value = askstring("확인", "다시 학습하시겠습니까?(yes/no)")
    if value == "yes":
        messagebox.showinfo("확인", "학습을 다시 합니다.")
        p1 = Process(target = learning.main)
        p1.start(); p1.join()
    elif value == "no":
        messagebox.showinfo("확인", "학습하지 않습니다.")

    rec.get_elements(func, window, app.lbl1, cap)
    rec.video_play()
    app.mainloop()

if __name__ == "__main__":
    main()
```

코드설명 start.py



- ▶ 처음 시작하는 파일
- ▶ 실시간 카메라를 이용하기 위한 cap 객체 지정
- ▶ 기본적인 Tkinter 선언
- ▶ Askstring을 이용한 학습의 유무 확인
- ▶ Multiprocessing을 이용하여 처리 시간 줄였다.
(68초에서 40초로)
- ▶ 만들어진 클래스 객체들을 Audio_Video의 함수
에 넘겨주게 된다.
- ▶ Tkinter 인터페이스가 실행된다.



코드설명

ui_test.py

```
from tkinter.filedialog import *
```

```
class Application(Frame):
```

```
    global window, cap
```

```
    def __init__(self, window, func, rec, cap, master=None):
```

```
        super().__init__(master)
```

```
        self.master = master
```

```
        self.window = window # Tkinter 객체 정의
```

```
        self.func = func # UI_func 객체 정의
```

```
        self.rec = rec # Audio_Video 객체 정의
```

```
        self.cap = cap # VideoCapture 객체 정의
```

```
        self.pack()
```

```
        self.create_widgets() # 위젯 배치
```

- ▶ Application 클래스
- ▶ Tkinter로 UI를 만드는 클래스
- ▶ `__init__` 함수로 생성자 역할을 하고 클래스 내 변수들을 초기화 한다.
- ▶ 마지막에 `create_widgets()` 함수를 부르면서 위젯을 만들고 배치한다.

```
def create_widgets(self):
    global func, rec, window
    # 라벨 추가
    self.lbl = Label(self.window, text = "OpenCV/Deep learning emotion detection")
    self.lbl.place(x=10, y=3)
    # 프레임 추가
    self.frm= Frame(self.window, bg="white", width=480, height=400)
    self.frm.place(x=10, y=25)
    # 라벨1 추가
    self.lb1 = Label(self.frm)
    self.lb1.grid()
```

```
#label
self.label1 = Label(self.window, text="ver VideoFile")
self.label2 = Label(self.window, text="ver AudioFile")
self.label3 = Label(self.window, text="ver Real time")
self.label4 = Label(self.window, text="ver Real time")
self.label5 = Label(self.window, text="Process")
self.label6 = Label(self.window, text="FilePath")
self.label7 = Label(self.window, text="FilePath")
self.label8 = Label(self.window, text="Result", font=16)

#button
self.button1 = Button(self.window, text="Face/Voice", command=lambda: self.func.getVideoFile(self.label6, self.label8)) # flag = 1
self.button2 = Button(self.window, text="Only Voice", command=lambda: self.func.getAudioFile(self.label7, self.label8)) # flag = 2
self.button3 = Button(self.window, text="Only Face", command = self.func.realtimeVideo) # flag = 3
self.button4 = Button(self.window, text="Only VoiceRec", command=lambda: self.func.record_audio(self.label5)) # flag = 4
self.button5 = Button(self.window, text="flag reset and video quit",
                      command=lambda: self.func.flag_reset(self.label6, self.label7, self.rec.stop, self.cap)) # flag = 0
self.button6 = Button(self.window, text="quit", command=lambda: self.func.quit_UI(self.cap))
```

```
#place labels and buttons
self.label1.place(x=500, y=50)
self.label2.place(x=500, y=100)
self.label3.place(x=500, y=150)
self.label4.place(x=500, y=200)
self.label5.place(x=580, y=230)
self.label6.place(x=500, y=75)
self.label7.place(x=500, y=125)
self.label8.place(x=550, y=300)
```

```
self.button1.place(x=580, y=50)
self.button2.place(x=580, y=100)
self.button3.place(x=580, y=150)
self.button4.place(x=580, y=200)
self.button5.place(x=530, y=330)
self.button6.place(x=580, y=360)
```

코드설명

ui_test.py



- ▶ Application클래스
- ▶ create_widgets함수에서는 위젯들을 만들어서 배치.
- ▶ Tkinter에 영상화면을 배치하기 위해서 레이블과 프레임을 추가한다. Opencv영상을 tkinter에 배치하기 위한 핵심요소이다.

class UI_func:

```
def __init__(self, window, rec, learning):
    self.flag, self.yesno, self.count = 3, 0, 0
    self.window = window
    self.rec = rec
    self.fName = ""
    self.test = learning
```

def record_audio(self, label1):

```
if self.flag != 0 and self.flag != 4:
    msg.showinfo('error', str(self.flag) + '번이 실행 중입니다.')
elif self.flag == 4:
    msg.showinfo('error', '이미 실행 중입니다.')
else:
    self.flag = 4
    if self.yesno == 0:
        self.yesno = 1; self.count += 1
        label1.configure(text=str(self.count) + " recording...")
        a = Thread(target=self.rec.recording)
        a.setDaemon(True); a.start()
    elif self.yesno == 1:
        self.rec.after_rec()
        label1.configure(text=str(self.count) + " finished recording")
        self.yesno = 0
```

def getVideoFile(self, label6, label8):

```
if self.flag != 0 and self.flag != 1:
    msg.showinfo('error', str(self.flag) + '번이 실행 중입니다.')
    label6.configure(text="FilePath")
elif self.flag == 1:
    msg.showinfo('error', '이미 실행 중입니다.')
else:
    self.fName = askopenfilename(parent=self.window, filetypes=(("Mp4 파일", "*.mp4"), ("모든 파일", "*.*")))
    label6.configure(text=str(self.fName))
    if len(label6.cget("text")) == 0:
        msg.showinfo('확인', '파일이 선택되지 않았습니다.')
        label6.configure(text="FilePath")
    else:
        self.flag = 1
        results = []; q = Queue()
        p = Process(target=self.test.get_feature, args=(self.fName, self.flag, q))
        p.start(); p.join()
        while not q.empty():
            results.append(q.get())
        result = results[0].tolist()
        tr = results[2]
        cap = cv.VideoCapture(str(self.fName))
        b2 = Thread(target=self.update_label, args=(result, label8, self.flag, cap, tr)); b2.start()
        self.rec.get_cap(cap)
        self.rec.video_play()
```

코드설명 UI_func.py



- ▶ UI_func클래스
- ▶ Tkinter에서 실행되는 기능들을 수행하는 명령함수클래스
- ▶ __init__ 함수에서 클래스 내 변수를 초기화 한다. 기본 flag=3이므로 실시간 얼굴이 기본적으로 출력된다.
- ▶ 버튼들의 기능을 하는 함수들은 flag가 0 이어만 실행이 된다.
- ▶ record_audio함수에서는 음성녹음을 하는 함수를 쓰레드로 실행하고 음성녹음을 하고 있는지 끝났는지 알려준다.
- ▶ getVideoFile함수에서는 실행할 비디오 mp4파일을 선택하고 음성의 특징을 추출하는 함수를 부르고 처리가 완료되면 비디오를 실행함과 동시에 레이블을 업데이트 하는 함수를 쓰레드로 부른다.

#tkinter과 합치면서 속도 느려져서 속도를 맞추기 위해 프레임수에 맞춰서 레이블 출력

```
def update_label(self, result, label, flag, cap, tr = 1, length = 0):
    emo = {1:'Angry', 2:'Disgust', 3:'Fear', 4:'Happy', 5:'Sad', 6:'Surprise', 7:'Neutral'}
    if flag == 1: #camera or video
        divided = cap.get(cv.CAP_PROP_FRAME_COUNT) / tr
        for i in result:
            label.configure(text=emo[i])
            if cap.get(cv.CAP_PROP_POS_FRAMES) % divided == 0:
                time.sleep(0.1)
            while cap.get(cv.CAP_PROP_POS_FRAMES) % divided != 0:
                if cap.get(cv.CAP_PROP_POS_FRAMES) == (divided * tr):
                    break
        elif flag == 2: #audio
            for a, i in enumerate(result):
                label.configure(text=emo[i])
                time.sleep(length//tr)

def play_audio(self, path):
    winsound.PlaySound(path, winsound.SND_FILENAME)

def getAudioFile(self, label7, label8):
    if self.flag != 0 and self.flag != 2:
        msg.showinfo('error', str(self.flag) + '번이 실행 중입니다.')
    elif self.flag == 2:
        msg.showinfo('error', '이미 실행 중입니다.')
    else:
        self.fName = askopenfilename(parent=self.window, filetypes=(("Wav 파일", "*.wav"), ("모든 파일", "*.*")))
        label7.configure(text=str(self.fName))
        if len(label7.cget("text")) == 0:
            msg.showinfo('확인', '파일이 선택되지 않았습니다.')
            label7.configure(text="FilePath")
        else:
            self.flag = 2
            results = []; q = Queue()
            p2 = Process(target=self.test.get_feature, args=(self.fName, self.flag, q))
            p2.start(); p2.join()
            c = Thread(target=self.play_audio, args=(self.fName,))
            c.start()
            while not q.empty():
                results.append(q.get())
            result = results[0].tolist()
            length = results[1]
            tr = results[2]
            b = Thread(target=self.update_label, args=(result, label8, self.flag, self, tr, length))
            b.start()
```

코드설명 UI_func.py



- ▶ UI_func클래스
- ▶ update_label함수에서는 음성파일이나 동영상파일이 실행될때 실행되는 속도에 맞춰 음성인식의 결과를 나타낸다. 영상인 경우에는 현재프레임과 전체프레임 수의 자른 영상의 개수를 이용하고 음성파일의 경우에는 음성파일의 재생시간과 자른 영상의 개수를 이용해 속도를 맞춘다.
- ▶ play_audio함수에서는 음성파일을 재생한다.
- ▶ getAudioFile에서는 실행할 wav파일을 선택하고 특징을 추출하는 함수를 쓰레드로 부르고 인식결과를 받아서 update_label 함수를 불러 출력한다.

코드설명 UI_func.py

```
def realtimeVideo(self):
    if self.flag != 0 and self.flag != 3:
        msg.showinfo('error', str(self.flag) + '번이 실행 중입니다.')
    elif self.flag == 3:
        msg.showinfo('error', '이미 실행 중입니다.')
    elif self.flag == 0:
        self.flag = 3
        cap = cv.VideoCapture(0)
        self.rec.get_cap(cap)
        self.rec.video_play()

def flag_reset(self, label6, label7, stop, cap):
    if self.flag == 0:
        msg.showinfo('error', '이미 flag=0입니다.')
    else:
        self.flag = 0
        self.window.after_cancel(stop)
        cap.release()
        label6.configure(text="FilePath")
        label7.configure(text="FilePath")

def quit_UI(self, cap):
    cap.release()
    quit()
```

- ▶ UI_func클래스
- ▶ realtimeVideo함수에서는 실시간 영상을 실행하기 위해 videocapture객체를 만들어서 video_play함수를 부르기 위해 먼저 객체를 넘긴 후 함수를 실행한다.
- ▶ flag_reset함수에서는 flag의 값을 초기화라고 영상을 종료하고 레이블 값들을 초기화 시킨다.
- ▶ quit_UI함수는 영상을 release하고 UI를 종료한다.

#(1=Angry, 2=Disgust, 3=Fear, 4=Happy, 5=Sad, 6=Surprise, 7=Neutral)

def extract_feature(file_name): #파일의 음성 특징 추출

X, sample_rate = librosa.load(file_name, sr=96000)

stft = np.abs(librosa.stft(X))

mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)

chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)

mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T, axis=0)

contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T, axis=0)

tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X), sr=sample_rate).T, axis=0)

return mfccs, chroma, mel, contrast, tonnetz

def parse_audio_files(parent_dir, sub_dirs, file_ext="*.wav"): #처음 기본 학습파일의 특징을 추출

features, labels = np.empty((0, 193)), np.empty(0)

for label, sub_dir **in** enumerate(sub_dirs):

for fn **in** glob.glob(os.path.join(parent_dir, sub_dir, file_ext)):

try:

mfccs, chroma, mel, contrast, tonnetz = extract_feature(fn)

except Exception **as** e:

print("Error encountered while parsing file: ", fn)

continue

ext_features = np.hstack([mfccs, chroma, mel, contrast, tonnetz])

features = np.vstack([features, ext_features])

labels = np.append(labels, fn.split("/")[-2].split("-")[-1])

return np.array(features), np.array(labels, dtype = np.int)

def one_hot_encode(labels): #레이블 원핫인코딩

n_labels = len(labels)+1

n_unique_labels = len(np.unique(labels))

one_hot_encode = np.zeros((n_labels, n_unique_labels+1))

one_hot_encode[np.arange(n_labels), labels] = 1

one_hot_encode=np.delete(one_hot_encode, 0, axis=1)

return one_hot_encode

model layers

def create_model(activation_function='relu', init_type='normal', dropout_rate=0.25):

global n_dim, n_classes, n_hidden_units_1, n_hidden_units_2

층마다 차원

model = Sequential()

model.add(Dense(n_hidden_units_1, kernel_regularizer=regularizers.l2(0.004), input_dim=n_dim, init=init_type, activation=activation_function, bias_initializer='zeros')) # 1

model.add(Dense(n_hidden_units_2, kernel_regularizer=regularizers.l2(0.004), init=init_type, activation=activation_function, bias_initializer='zeros')) # 2

model.add(Dropout(dropout_rate))

model.add(Dense(n_classes, init=init_type, activation='softmax')) # output

model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0004), metrics=['accuracy'])

return model

코드설명

sound_learning.py



- ▶ sound_learning.py의 클래스 밖의 함수들
- ▶ extract_feature함수에서는 librosa라이브러리를 이용하여 파일의 음성 특징을 추출한다.
- ▶ parse_audio_files함수에서는 파일이 들어있는 폴더로 들어가서 extract_feature함수를 불러 특징을 추출하고 넘파이 어레이로 features과 labels에 값을 저장한다.
- ▶ one_hot_encode함수는 원핫인코딩을 하여 레이블을 만든다. ex)7개 중 3번이면 [0,0,1,0,0,0,0]로 만든다. 원래는 [0,0,0,1,0,0,0]이나 보기 쉽게 하기 위해서 8개의 값을 만들고 앞부분을 없앤다.
- ▶ create_model함수는 모델의 층을 쌓는 함수이다.

코드설명

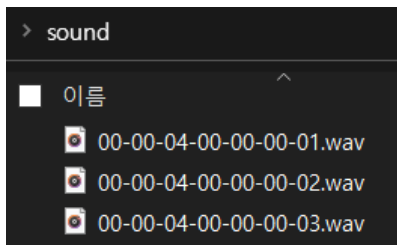
sound_learning.py



```
class Test:
    global saved_model,result
    saved_model = load_model('emotion_result.h5')
    def __init__(self):
        self.i=0
        self.name = []
        self.tr = 1

    def trim_audio(self, audio_file, save_file,length): #오디오 자름
        sr = 96000
        if length <= 30:
            self.tr = 1
        elif length > 30:
            self.tr = int(length//20)
        sec = int(length)//self.tr
        a = 0
        y, sr = librosa.load(audio_file, sr=sr)
        while(sr*(sec*(a+1))< len(y)):
            ny = y[sr*(0+sec*a):sr*(sec*(a+1))]
            librosa.output.write_wav(save_file + str(self.i+1) + '.wav', ny, sr)
            name = save_file + str(self.i + 1) + ".wav"
            self.name.append(name.split("/")[-1])
            print(name + "완료")
            self.i += 1
            a += 1

    def make_test(self,main_dir, sub_dir): #특징추출하고 features,labels 만들기
        i = 0
        features, labels = np.empty((0, 193)), np.empty(0)
        while (i < len(sub_dir)):
            fn = main_dir + sub_dir[i]
            mfccs, chroma, mel, contrast, tonnetz = extract_feature(fn)
            ext_features = np.hstack([mfccs, chroma, mel, contrast, tonnetz])
            features = np.vstack([features, ext_features])
            labels = np.append(labels, fn.split("/")[-1].split("-")[2])
            i += 1
        labels = np.array(labels, dtype=np.int)
        labels = to_categorical(labels - 1, 8)
        return features, labels
```



- ▶ sound_learning.py내의 Test클래스
- ▶ global로 결과를 불러올 변수를 선언하고 음성의 감정을 인식할 저장된 모델을 불러온다.
- ▶ trim_audio함수에서는 비디오를 자르고 자른 파일을 저장한다.
- ▶ make_test에서는 특징을 추출하고 features와 labels를 만든다. 기존의 extract_features와 parse_audio_files의 함수를 필요에 맞게 응용하여 간편하게 사용하도록 하였다.


```

def audio_feature(self):
    # 데이터 추출. 이미 데이터 추출되었으면 시간 오래 걸려서 생략
    # change the main_dir accordingly....
    main_dir = 'D:/MyFile/Audio_speech'
    sub_dir = os.listdir(main_dir)
    print("\ncollecting features and labels...")
    print("\nthis will take some time...")
    features, labels = parse_audio_files(main_dir, sub_dir)
    print("done")
    np.save('X', features) # 저장
    # one hot encoding labels
    labels = one_hot_encode(labels)
    np.save('y', labels)
    labels = one_hot_encode('08')
    print(labels)

def video_feature(self): # 맨 처음 데이터 특징추출하기 위한 부분-npy로 저장
    # 영상
    main_dir = os.getcwd()
    main_dir = main_dir + "/Video/"
    sub_dir = os.listdir(main_dir)
    print("subdir", sub_dir)

    for i in range(0, len(os.listdir(main_dir))):
        head, tail = os.path.split(sub_dir[i])
        clip = mp.VideoFileClip("./video/" + tail)
        length = clip.duration
        print("length:", length)
        print("1-", "./video/" + tail)
        tail = tail[0:2]
        print("2-", tail)
        audio_file = "00-00-" + tail + "-00-00-00-00.wav"
        print("3-", audio_file)
        clip.audio.write_audiofile(audio_file)
        print("4trim-", audio_file)
        save_file = "./sound/" + audio_file.replace("0.wav", "")
        print(save_file)
        self.trim_audio(audio_file, save_file, length)

    main_dir = os.getcwd()
    main_dir = main_dir + "/sound/"
    sub_dir = os.listdir(main_dir)
    print("subdir", sub_dir)
    features, labels = self.make_test(main_dir, sub_dir)
    np.save('f', features)
    np.save('l', labels)

```

코드설명

sound_learning.py



- ▶ test클래스
- ▶ audio_feature함수와 video_feature함수는 이 프로그램에서는 사용하지 않고 그 전에 딥러닝으로 학습하기 위해 특징을 추출하고 npy파일로 저장하기 위해 사용되었다.
- ▶ audio_feature함수에서는 기본데이터인 감정을 담아 말하거나 노래하는 음성파일들을 가지고 특징을 추출하여 저장한다.
- ▶ video_feature함수에서는 비디오를 폴더에서 가져와서 음성을 추출하고 그 음성파일을 잘라서 데이터셋을 만든다음에 npy 파일로 저장한다. 직접 데이터셋을 만들어 진행하였다.

코드 설명

sound_learning.py



```
def musical_feature(self): #맨 처음 뮤지컬 음악 학습할때 필요
    # 뮤지컬
    main_dir = 'D:/MyFile/Musical/'
    sub_dir = os.listdir(main_dir)
    print("\ncollecting features and labels...")
    print("\nthis will take some time...")
    features, labels = self.make_test(main_dir, sub_dir) ##
    print("done")
    np.save('Xmusic', features) # 저장
    np.save('ymusic', labels)
```

```
def get_feature_save(self):
    self.audio_feature()
    self.video_feature()
    self.musical_feature()
```

```
def get_feature(self, fName, flag, q):
    global saved_model, result
    main_dir = fName
    head, tail = os.path.split(main_dir)
    if flag == 1:
        clip = mp.VideoFileClip("./video/" + tail)
        length = clip.duration
        tail = tail[0:2]
    elif flag == 2:
        with contextlib.closing(wave.open(main_dir, 'r')) as f:
            frames = f.getnframes()
            rate = f.getframerate()
            length = frames / float(rate)
        tail = tail[0:2]
    audio_file = "00-00-" + tail + "-00-00-00-00.wav"
    save_file = "./sound/" + audio_file.replace("0.wav", "")
    self.trim_audio(main_dir, save_file, length)
    features, labels = self.make_test("./sound/", self.name)
    predict = saved_model.predict(features, batch_size=4)
    print(predict[0])
    q.put(np.argmax(predict, 1) + 1)
    q.put(length)
    q.put(self.tr)
```

▶ test클래스

▶ musical_feature함수에서는 뮤지컬의 음성을 가지고 특징을 추출하여 데이터셋을 만들고 npy파일로 저장한다. 기본 데이터셋에 노래의 감정을 파악하는 데이터셋이 있어서 추가하였다.

▶ get feature save함수에서는 말하고 노래하는 감정을 담은 데이터파일인 기본데이터와 비디오와 뮤지컬의 감정을 추출하여 저장하는 함수들을 부른다.

▶ get feature함수는 UI의 버튼의 종류에 따라 처리과정이 달라지는데 영상이면 음성추출을 하는 과정이 추가되고 음성파일이면 그대로 사용하며 자르고 특징을 추출해서 미리 학습한 모델로 어떤 감정인지 예측한다. 이 함수는 multiprocessing의 process로 실행되기때문에 매개변수를 return할 수 없으므로 큐를 받아서 필요한 값들을 큐에 넣어 전달한다.



코드설명

sound_learning.py



```
def main(self):
    features = np.load('f.npy') # 영상
    labels = np.load('l.npy')
    Xmusic = np.load('Xmusic.npy') # 뮤지컬
    ymusic = np.load('ymusic.npy')
    X = np.load('X.npy') # 기본데이터
    y = np.load('y.npy')
    train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.1, random_state=60)
    train_x, val_x, train_y, val_y = train_test_split(train_x, train_y, test_size=0.1, random_state=60)
    Xmusic, a, ymusic, b = train_test_split(Xmusic, ymusic, test_size=0.05, random_state=60)
    features, c, labels, d = train_test_split(features, labels, test_size=0.07, random_state=60)
    train_x = np.append(train_x, a, axis=0)
    train_x = np.append(train_x, c, axis=0)
    train_y = np.append(train_y, b, axis=0)
    train_y = np.append(train_y, d, axis=0)

    global n_dim, n_classes, n_hidden_units_1, n_hidden_units_2
    n_dim = train_x.shape[1] # 193
    n_classes = train_y.shape[1] # 8
    n_hidden_units_1 = n_dim
    n_hidden_units_2 = 400 # 400

    model = create_model() # 모델생성
    epoch = 120
    train_history = model.fit(train_x, train_y, epochs=epoch, batch_size=15, validation_data=(val_x, val_y), verbose = 2)
    predict = model.predict(test_x, batch_size=4)
    (test_loss, test_acc) = model.evaluate(test_x, test_y, verbose=0)
```

- ▶ test클래스
- ▶ main함수에서는 저장된 npy의 데이터셋들을 불러오고 그 데이터셋들을 학습데이터셋과 테스트데이터셋, 검증데이터셋으로 나누고 모델을 생성한 다음 학습을 진행한다.

```
print("\n테스트데이터 정확도:", test_acc)
```

```
predict = model.predict(Xmusic)
```

```
print("\n뮤지컬음악 예측값:", np.argmax(predict, 1) + 1, '\n\n', np.argmax(ymusic, 1) + 1)
```

```
(test_loss, test_acc) = model.evaluate(Xmusic, ymusic, verbose=0)
```

```
print("\n뮤지컬음악 테스트 정확도:", test_acc)
```

```
predict = model.predict(features)
```

```
print("\n영상 예측값:", np.argmax(predict, 1) + 1, '\n\n', np.argmax(labels, 1) + 1)
```

```
(test_loss, test_acc) = model.evaluate(features, labels, verbose=0)
```

```
print("\n영상 테스트 정확도:", test_acc)
```

```
model.save('emotion_result.h5')
```

```
# 그래프
```

```
epochs = range(1, epoch + 1)
```

```
accuracy = train_history.history['accuracy']
```

```
val_accuracy = train_history.history['val_accuracy']
```

```
loss = train_history.history['loss']
```

```
val_loss = train_history.history['val_loss']
```

```
plt.subplot(1, 2, 1)
```

```
plt.plot(epochs, accuracy, 'b', label='accuracy')
```

```
plt.plot(epochs, val_accuracy, 'g', label='val_accuracy')
```

```
plt.xlabel('Epochs')
```

```
plt.ylabel('accuracy')
```

```
plt.legend()
```

```
plt.grid()
```

```
plt.subplot(1, 2, 2)
```

```
plt.plot(epochs, loss, 'r', label='loss')
```

```
plt.plot(epochs, val_loss, 'k', label='val_loss')
```

```
plt.xlabel('Epochs')
```

```
plt.ylabel('loss')
```

```
plt.legend()
```

```
plt.grid()
```

```
plt.tight_layout() # 떨어져있게 간격조정
```

```
plt.show()
```

코드설명

Sound_learning.py



- ▶ test클래스
- ▶ 학습이 완료된 모델을 가지고 테스트데이터들의 정확도를 측정한 다음 모델을 저장한다.
- ▶ 모델의 학습과정을 그래프로 그리는데 정확도그래프와 손실그래프를 그리게 된다.
- ▶ `__name__ == "main"`에서는파일들을 직접 불러와서 특징을 추출하는 과정을 거친 후 `main`함수를 실행한다. 전체 특징을 추출하는 과정은 2시간정도 걸리므로 특징들을 `npz`파일을 저장해서 그 시간을 줄였다.

```
if __name__ == "__main__":
```

```
    a = Test()
```

```
    a.get_feature_save()
```

```
    a.main()
```

코드설명 Audio_Video.py

```
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 16000
CHUNK = 1024
#RECORD_SECONDS = 0
WAVE_OUTPUT_FILENAME = "file"

# Face detection XML load and trained model loading
front_path = "./haarcascade_frontalface_default.xml"
profile_path = './haarcascade_profileface.xml'
eye_path = "./haarcascade_eye.xml"
face_detection1 = cv.CascadeClassifier(front_path)
face_detection2 = cv.CascadeClassifier(profile_path)
eyeCascade = cv.CascadeClassifier(eye_path)
emotion_classifier = load_model('./emotion_model.hdf5', compile=False)
EMOTIONS = ["Angry", "Disgusting", "Fearful", "Happy", "Sad", "Surprising", "Neutral"]

class Audio_Video:
    def __init__(self):
        self.audio, self.stream, self.func, self.window, self.lbl1 = 0, 0, 0, 0, 0
        self.faces, self.face_color, self.frame_raw, self.new_img = self, self, self, self
        self.stop, self.cap, self.count = 0, 0, 1
        self.frames = []
        self.eye_1, self.eye_2 = (0,), (0,)

    def newstream(self):
        self.audio = pyaudio.PyAudio()
        # start Recording
        self.stream = self.audio.open(format=pyaudio.paInt16,
                                       channels=CHANNELS,
                                       rate=RATE,
                                       input=True,
                                       input_device_index=2,
                                       frames_per_buffer=CHUNK)

        self.frames = []
```

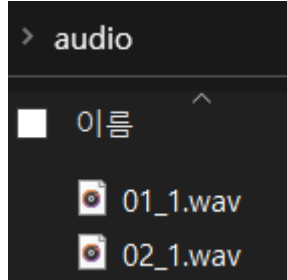
- ▶ Audio_Video클래스
- ▶ Cascade파일들을 먼저 지정하는데 얼굴정면을 인식하는 frontalface, 얼굴측면을 인식하는 profileface, 눈을 인식하는 eye 파일들의 경로를 지정하고 CascadeClassifier객체를 만든다. 그다음 얼굴의 감정을 담은 EMOTIONS리스트를 만든다.
- ▶ Audio_Video클래스의 __init__함수에서는 생성자역할을 하고 필요한 변수들을 초기화 한다.
- ▶ newstream함수에서는 음성녹음을 하기 위한 pyaudio라이브러리로 객체를 만들고 포트를 연 stream을 만든다.

코드설명 Audio_Video.py



```
def recording(self): #파일 녹음
    self.newstream()
    print("recording...")
    self.frames = []
    while self.func.yesno == 1:
        data = self.stream.read(CHUNK)
        self.frames.append(data)
        if self.func.yesno == 0:
            break
    print("finished recording")
    # stop Recording
    self.stream.stop_stream()
    self.stream.close()
    self.audio.terminate()

def after_rec(self): #녹음후처리
    try:
        if not os.path.exists("./audio"):
            os.makedirs("./audio")
    except OSError:
        print("Error: Creating directory. ' + './audio'")
    while True:
        WAVE_OUTPUT_FILENAME = askstring("확인", '저장하고자 하는 파일의 이름을 쓰세요\n("Angry":1,
"Disgusting":2, "Fearful":3, "Happy":4, "Sad":5, "Surpring":6, "Neutral":7)')
        if not WAVE_OUTPUT_FILENAME:
            msg.showinfo("error", "이름을 쓰세요")
        else:
            break
    os.chdir("./audio")
    condition = f"{WAVE_OUTPUT_FILENAME}_*.wav"
    wavfiles = glob.glob(condition)
    if not wavfiles:
        self.count = 1
    else:
        wavfile = wavfiles.pop(-1)
        c = wavfile.split(".")[0].split("_")[-1]
        self.count = int(c) + 1
    waveFile = wave.open(f"{WAVE_OUTPUT_FILENAME}_{self.count}"+".wav", 'wb')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(self.audio.get_sample_size(FORMAT))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b"".join(self.frames))
    waveFile.close()
    self.frames = []
    self.count+=1
    os.chdir("../")
```



- ▶ Audio_Video클래스
- ▶ recording함수는 음성파일을 녹음하는 기능을 한다.
- ▶ after_rec함수는 recording함수를 실행한 후 파일을 처리해서 경로를 통해 저장하는 과정을 수행한다. os라이브러리로 폴더안에 들어가서 저장한 후 그 전 폴더로 다시 나온다.



코드설명 Audio_Video.py



```
def get_elements(self, func, window, lbl1, cap=0):  
    self.func = func  
    self.window = window  
    self.lbl1 = lbl1  
    self.cap = cap  
  
def get_cap(self, cap):  
    self.cap = cap
```

- ▶ Audio_Video 클래스
- ▶ get_elements 함수에서는 처리에 필요한 객체들과 값들을 받는다. 맨 처음 실행하기 전에 필요한 요소를 전달하기 위한 함수이다.
- ▶ get_cap 함수에서는 비디오를 실행하기 위한 cap 값을 받는다. 버튼을 다르게 누를 때마다 동영상 또는 실시간 카메라의 값이 필요하므로 그 값만 받을 수 있는 함수를 만들었다.



코드 설명 Audio_Video.py



```
def video_play(self): # cap에 따라 실시간 동영상 가능
    ret, frame = self.cap.read() # 프레임이 올바르게 읽히면 ret은 True
    frame = cv.flip(frame, 1)

    if frame is None: # 영상 끝나면 영상만 종료
        self.window.after_cancel(self.stop)

    if not ret:
        self.cap.release() # 작업 완료 후 해제
        return

    self.frame_raw = frame.copy()
    frame = imutils.resize(frame, width=480, height=400)
    # Convert color to gray scale
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    rotated = self.detect(gray, frame)
    # Face detection in frame
    faces = face_detection1.detectMultiScale(frame, scaleFactor=1.05, minNeighbors=5, minSize=(30, 30)) # 정면
    if faces is None:
        faces = face_detection1.detectMultiScale(rotated, scaleFactor=1.05, minNeighbors=5, minSize=(30, 30)) # 정면
    if faces is None:
        faces = face_detection2.detectMultiScale(frame, scaleFactor=1.05, minNeighbors=5, minSize=(30, 30)) # 측면

    # Perform emotion recognition only when face is detected
    if len(faces) > 0:
        # For the largest image
        face = sorted(faces, reverse=True, key=lambda x: (x[2] - x[0]) * (x[3] - x[1]))[0]
        (fX, fY, fW, fH) = face
        # Resize the image to 48x48 for neural network
        roi = gray[fY:fY + fH, fX:fX + fW]
        roi = cv.resize(roi, (48, 48))
        roi = roi.astype("float") / 255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi, axis=0)

        # Emotion predict
        preds = emotion_classifier.predict(roi)[0]
        label = EMOTIONS[preds.argmax()]

        # Assign labeling
        cv.putText(frame, label, (fX, fY - 10), cv.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
        cv.rectangle(frame, (fX, fY), (fX + fW, fY + fH), (0, 0, 255), 2)

    img = Image.fromarray(frame) # Image 객체로 변환
    imgtk = ImageTk.PhotoImage(image=img) # ImageTk 객체로 변환
    # OpenCV 동영상
    self.lbl1.imgtk = imgtk
    self.lbl1.configure(image=imgtk)
    self.stop = self.lbl1.after(10, self.video_play)
```

- ▶ Audio_Video 클래스
- ▶ video_play 함수에서는 그 전에 받은 객체인 cap으로 읽어서 frame을 전달받는다.
- ▶ 정면과 측면을 인식하기 위해 detectMultiScale로 객체를 만들고 인식된 얼굴만 잘라서 48x48로 resize한다. 그 자른 얼굴을 가지고 감정을 인식하고 감정을 화면에 표시한다.
- ▶ UI의 frame 부분에 화면을 표시하기 위해 Image 객체를 만들고 ImageTk 객체로 변환하여 configure 함수에 넣고 after 함수를 10ms마다 불러 영상이 실행되도록 한다.

코드설명 Audio_Video.py



```
def euclidean_distance(self, a, b):
    x1 = a[0]
    y1 = a[1]
    x2 = b[0]
    y2 = b[1]
    return math.sqrt(((x2 - x1) * (x2 - x1)) + ((y2 - y1) * (y2 - y1)))

def detect(self, gray, frame):
    global new_img
    # 등록된 Cascade classifier 를 이용 얼굴을 찾음
    faces = face_detection1.detectMultiScale(gray, scaleFactor=1.05, minNeighbors=5, minSize=(100, 100),
                                              flags=cv.CASCADE_SCALE_IMAGE)
    # 얼굴에 사각형을 그리고 눈을 찾자
    for (x, y, w, h) in faces:
        # 얼굴: 이미지 프레임에 (x,y)에서 시작, (x+넓이, y+길이)까지의 사각형을 그림(색 255 0 0, 굵기 2)
        # cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

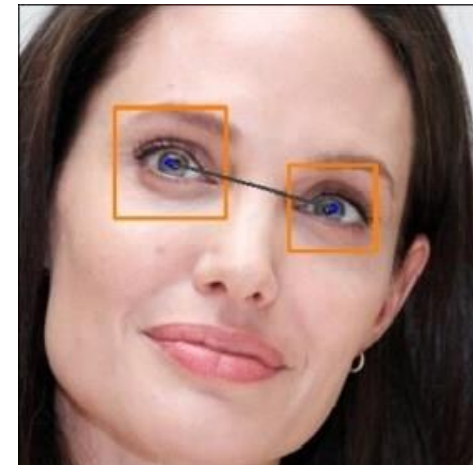
        # 이미지를 얼굴 크기 만큼 잘라서 그레이스케일 이미지와 컬러이미지를 만들
        face_gray = gray[y:y + h, x:x + w]
        self.face_color = frame[y:y + h, x:x + w]

        # 등록된 Cascade classifier 를 이용 눈을 찾음(얼굴 영역에서만)
        eyes = eyeCascade.detectMultiScale(face_gray, 1.1, 3)
        # lefteye = lefteyeCascade.detectMultiScale(face_gray, 1.1, 3)

        # 눈: 이미지 프레임에 (x,y)에서 시작, (x+넓이, y+길이)까지의 사각형을 그림(색 0 255 0, 굵기 2)
        for i, (eye_x, eye_y, eye_w, eye_h) in enumerate(eyes):
            #cv.rectangle(self.face_color, (eye_x, eye_y), (eye_x + eye_w, eye_y + eye_h), (0, 255, 0), 2)
            #cv.circle(self.face_color, (eye_x, eye_y), 2, (0, 0, 255), 2)
            #cv.circle(self.face_color, (eye_x + eye_w, eye_y + eye_h), 2, (0, 0, 255), 2)
            #cv.circle(self.face_color, (eye_x + int(eye_w / 2), eye_y + int(eye_h / 2)), 2, (0, 0, 255), 2)
            if i == 0:
                self.eye_1 = (eye_x, eye_y, eye_w, eye_h)
            elif i == 1:
                self.eye_2 = (eye_x, eye_y, eye_w, eye_h)

    if self.eye_1[0] < self.eye_2[0]:
        left_eye = self.eye_1
        right_eye = self.eye_2
    else:
        left_eye = self.eye_2
        right_eye = self.eye_1
```

- ▶ Audio_Video클래스
- ▶ Euclidean_distance함수는 유클리드 정리로 즉 피타고라스 정의와 같아서 대각선의 길이를 구하는 부분이다.
- ▶ Detect함수는 Cascade classifier을 이용해서 얼굴을 찾은 다음 눈을 찾고 눈의 위치를 저장한다.



코드설명 Audio_Video.py



```
try:
    left_eye_center = (left_eye[0] + int(left_eye[2] / 2), left_eye[1] + int(left_eye[3] / 2))
    left_eye_x = left_eye_center[0]
    left_eye_y = left_eye_center[1]
    right_eye_center = (right_eye[0] + int(right_eye[2] / 2), right_eye[1] + int(right_eye[3] / 2))
    right_eye_x = right_eye_center[0]
    right_eye_y = right_eye_center[1]
except IndexError:
    #print("IndexError")
    return
```

```
try:
    if left_eye_y < right_eye_y:
        point_3rd = (right_eye_x, left_eye_y)
        direction = -1
    else:
        point_3rd = (left_eye_x, right_eye_y)
        direction = 1

    a = self.euclidean_distance(left_eye_center, point_3rd)
    b = self.euclidean_distance(right_eye_center, left_eye_center)
    c = self.euclidean_distance(right_eye_center, point_3rd)
except:
    pass
```

```
try:
    cos_a = (b * b + c * c - a * a) / (2 * b * c)

    angle = np.arccos(cos_a)

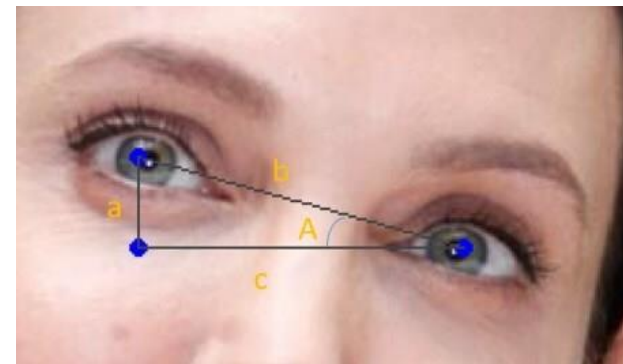
    angle = (angle * 180) / math.pi

    if direction == -1:
        angle = 90 - angle
    if angle < 2:
        angle = 0.1

    new_img = Img.fromarray(self.frame_raw)
    new_img = np.array(new_img.rotate(direction * -angle))
except ZeroDivisionError as e:
    pass
```

```
return new_img # frame
```

- ▶ Audio_Video 클래스
- ▶ 눈의 위치를 저장한 후 각도를 구하기 위해서 point_3rd를 만들고 세 점들을 가지고 코사인을 구해 역코사인을 구하고 그 각도를 저장한다. 그 후 이미지를 각도를 곱해서 얼굴이 기울어진 형태가 아닌 똑바로 정면을 바라보고 있어 얼굴을 인식하기가 편해진다.





감사합니다.

