UI for CAUT Deception Detection

Backend modelling

In [1]:
```
#mediapipe processing 1 video
import os
import cv2
import traceback
import numpy as np
import mediapipe as mp
import math

from MediaPipe_Processing_single_video import *
```

In [2]:
```
#openFace processing one video
def process_video_openface(vid_path):
    test_video_path = vid_path
    video_prediction = detector.detect_video(test_video_path, skip_frames=24)
    vid_mean = video_prediction.mean()
    vid_mean_df = vid_mean.to_frame()
    vid_mean_df = vid_mean_df.transpose()
    vid_mean_df = vid_mean_df[['AU01','AU02','AU04','AU05','AU06','AU07','AU09','AU10','AU
    return vid_mean_df
```

In [3]:
```
#predictions for the video
import _pickle as cPickle

def DetectDeception(vid_path,mode):
    if mode == "OpenFace":
        new_X = process_video_openface(vid_path)
        with open('C:\\Work\\606Capstone\\Video_chunks\\Models\\OpenFaceAverage_RFR.pickle
            rf = cPickle.load(f)
    else:
        new_X = process_video_mediapipe(vid_path, required_fps=90)
        with open('C:\\Work\\606Capstone\\Video_chunks\\Models\\MediaPipeSequential_RFR.pi
            rf = cPickle.load(f)

    preds = rf.predict(new_X)
    return preds[0]
```

In [4]:
```
#plot the graph for emotions
def Plot_Emotions(vid_path):
    test_video_path = vid_path
    video_prediction = detector.detect_video(test_video_path, skip_frames=24)
    vid_mean = video_prediction.mean()
    vid_mean_df = vid_mean.to_frame()
    vid_mean_df = vid_mean_df.transpose()
    vid_to_plot = vid_mean_df[['anger','disgust','fear','happiness','sadness','surprise',
    trace = go.Bar(x=vid_to_plot[vid_to_plot.columns[0]], y=vid_to_plot[vid_to_plot.columr
                marker={'color': vid_to_plot[vid_to_plot.columns[1]], 'colorscale': 'Blugrr
    layout = go.Layout(title='Emotions in the Video', width=450, height=400)
    fig = go.Figure(data=[trace], layout=layout)
    return fig
```

In [5]:
```
#detector for Openface
from tqdm import tqdm
```

```
from feat import Detector

detector = Detector()
detector
```

Out[5]:
```
feat.detector.Detector(face_model=retinaface, landmark_model=mobilefacenet, au_model=xgb,
emotion_model=resmasknet, facepose_model=img2pose)
```

User Interface

In [6]:
```
import dash
from dash import dcc, html, Input, Output, State
import dash_daq as daq
import base64
import os
from werkzeug.utils import secure_filename
import dash_bootstrap_components as dbc
import time
```

In [7]:
```
#card 1 the selection options
card1 = dbc.Card(
    dbc.CardBody([
        html.H6("Video Mode", className="card-title"),
        dbc.RadioItems(
            id='video-selector',
            options=[
                {'label': 'Select', 'value': 'dropdown'},
                {'label': 'Upload', 'value': 'upload'},
            ],
            style={'display': 'block'},
        ),
        html.Div([
            dcc.Dropdown(id='file-list',style={'width': '250px'},placeholder="Select a Vid
        ], id="dropdown-div", style={'display': 'none'}),
        html.Div([
            dcc.Upload(
                id='upload-video',
                children=html.Div([
                    'Drag and Drop or ',
                    html.A('Select a Video')
                ]),
                style={
                    'width': '100%',
                    'height': '60px',
                    'lineHeight': '60px',
                    'borderWidth': '1px',
                    'borderStyle': 'dashed',
                    'borderRadius': '5px',
                    'textAlign': 'center',
                    'margin': '10px'
                },
                multiple=False
            )
        ], id="upload-div", style={'display': 'none'}),
        html.Br(),
        html.Div([
            html.P("Select a Detector:"),
            dbc.RadioItems(
                id='radio_items',
                options=[
                    {'label': 'MediaPipe', 'value': 'MediaPipe'},
                    {'label': 'OpenFace', 'value': 'OpenFace'},
                ],
```

```
                value='MediaPipe',
                style={'display': 'block'},
                switch = True
            )
        ], id="toggle-div", style={'display': 'none'})
    ]), className="mt-4 shadow"
)
```

In [8]:

```
#card 2 is for the graph
card2 = html.Div([dbc.Card(
    dbc.CardBody([
        html.Div([
            dcc.Loading(
                id="loading-1",
                type="default",
                children=html.Div(id='output-graph')
            )
        ])
    ]), className="mt-4 shadow"
)], id='card2', style={'display': 'none'})
```

In [9]:

```
#card 3 is for the video and the detect button
card3 = html.Div([dbc.Card(
    dbc.CardBody([
        html.Div([
            html.Div([
                html.Video(id='video-player', controls=True, style={'height': '425px', 'wi
            ], className="mx-auto d-block")
        ], id='video-div', style={'display': 'none'}, className=""),
        html.Br(),
        html.Div([
            dbc.Button(
                "", id="play-button", color="#1c4a60", className="mr-1"
            ),
            html.Span(id="boolean-switch-output", style={"vertical-align": "middle"}),
        ], className="text-center", id="detect-div", style={'display': 'none'})
    ]), className="mt-4 shadow"
)], id='card3', style={'display': 'none'})
```

In [10]:

```
#card 4 is for Result
card4 = html.Div([dbc.Card(
    dbc.CardBody([
        html.H6("Result", className="card-title"),
        html.Div(id='text-output-container', style={'display': 'none'}),
        dcc.Loading(
            id="loading-2",
            type="default",
            children=[
                html.Div(id='background-box', style={'display': 'none'}, children=[
                html.P(id='text-output', style={
                    'margin': '20px',
                    'padding': '20px',
                    'border': '1px solid #ddd',
                    'border-radius': '10px',
                    "position": "absolute",
                    "left": "55%",
                    "top": "600px"
                }),]
            ),
        ])
```

```
    ]), className="mt-4 shadow"
)], id='card4', style={'display': 'none'})
```

In [11]:

```
# setting the css stylesheets
ss = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
FONT_AWESOME = "https://use.fontawesome.com/releases/v5.10.2/css/all.css"

#initiating the app
app = dash.Dash(__name__, external_stylesheets=[
                dbc.themes.BOOTSTRAP, FONT_AWESOME, ss])

ASSET_DIR = "assets"

#////////////////////////////////////////// UI Layout ////////////////////////////////
app.layout = dbc.Container(
    [
        html.H1("DECEPTION DETECTION", style={'textAlign': 'center', 'font-size': '30px'}),
        dbc.Row(
            [
                dbc.Col([card1, card2], width=4),
                dbc.Col([card3, card4], width=8),
            ],
            align="start",
            className="mt-4 align-items-start",
        ),
    ],
    fluid=True,
)

#////////////////////////////////////////// Callback Functions ////////////////////////
# Define the callback to list the files in the asset folder
@app.callback(
    Output('file-list', 'options'),
    [Input('file-list', 'contents')])
def update_file_list(contents):
    # List the files in the asset folder
    file_list = os.listdir(ASSET_DIR)
    options = [{'label': f, 'value': f} for f in file_list]
    return options

#callback to display the video block once source path is updated
@app.callback(
    [Output('video-div', 'style'),
     Output('video-div', 'className')],
    [Input('video-player', 'src')])
def update_video_src(value):
    if value:
        return ({"display" : "inline-block"},"mx-auto d-block")
    else:
        return ({"display" : "none"},"")

#Callback to display the mediapipe or openface switch
@app.callback(
    Output('toggle-div', 'style'),
    [Input('video-player', 'src')])
def update_video_src(value):
    if value:
        return {"display" : "inline-block"}
    else:
        return {"display" : "none"}

#Callback to display cards
@app.callback(
    [Output('card2', 'style'),
```

```
     Output('card3', 'style'),
     Output('card4', 'style')],
    [Input('video-player', 'src')])
def update_video_src(value):
    if value:
        return ({"display" : "block"},{"display" : "block"},{"display" : "block"})
    else:
        return ({"display" : "none"},{"display" : "none"},{"display" : "none"})

#callback to update the path of the video
@app.callback(
    Output('video-player', 'src'),
    [Input('video-selector', 'value'),
     Input('file-list', 'value'),
     Input('upload-video', 'contents')],
    State('upload-video', 'filename')
)
def upload_file(value, filelistvalue, content, filename):
    if value == "dropdown":
        if filelistvalue:
            src = os.path.join(ASSET_DIR, secure_filename(filelistvalue))
            return src
        else:
            return ""
    else:
        if content is not None:
            video_path = os.path.join(ASSET_DIR, secure_filename(filename))
            content_type, content_string = content.split(',')
            decoded_content = base64.b64decode(content_string)
            with open(video_path, 'wb') as f:
                f.write(decoded_content)
            return video_path
        else:
            return ""

#callback to either display dropdown or display upload option
@app.callback(
    [Output('dropdown-div', 'style'),Output('upload-div', 'style')],
    [Input('video-selector', 'value')]
)
def update_video_src(value):
    if value == "dropdown":
        return ({"display" : "inline-block"},{"display" : "none"})
    else:
        return ({"display" : "none"},{"display" : "inline-block"})

#call back for displaying the graph
@app.callback(
    Output('output-graph', 'children'),
    [Input('video-player', 'src')]
)
def update_graph(input_value):
    time.sleep(5)
    fig = Plot_Emotions(input_value)
    return dcc.Graph(figure=fig)

#callback to display the detect button
@app.callback(
    Output('detect-div', 'style'),
    [Input('video-player', 'src')])
def update_video_src(value):
    if value:
        return {'textAlign': 'center',"display" : "block"}
    else:
        return {'textAlign': 'center',"display" : "none"}
```

```
# Reset n_clicks when a different video is selected
@app.callback(
    Output('play-button', 'n_clicks'),
    [Input('video-player', 'src')]
)
def reset_n_clicks(src):
    return None

# Define the callback for the detect button
@app.callback(
    [Output('text-output-container', 'children'),
     Output('background-box', 'style')],
    [Input('play-button', 'n_clicks'),
    Input('video-player', 'src'),
    Input('radio_items', 'value')])
def play_video(n_clicks, src, value):
    if value == "MediaPipe":
        time.sleep(1)
    else:
        time.sleep(5)
    if n_clicks:
        return ['', {'display': 'none'}]
    else:
        print(f"value:{value}\n source:{src}")
        v = DetectDeception(src, value)
        if v:
            return ['The person is lying.', {'display': 'block'}]
        else:
            return ['The person is saying the truth.', {'display': 'block'}]
    return src

# Callback to update the result text
@app.callback(
    Output('text-output', 'children'),
    Input('text-output-container', 'children')
)
def update_text(text):
    return text

#callback to update the color of the background box for the result
@app.callback(
    Output('text-output', 'style'),
    Input('text-output', 'children')
)
def update_text_style(text):
    if text == 'The person is saying the truth.':
        return {'background-color': '#cdffcd','border-radius': '10px','font-size':'24px',
    else:
        return {'background-color': '#ff8080','border-radius': '10px','font-size':'24px',

#////////////////////////////////////////// Launch the App ////////////////////////////
if __name__ == '__main__':
    #app.run_server(debug=True, use_reloader=False)
    app.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/
```

```
INFO:__main__:Dash is running on http://127.0.0.1:8050/
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
```

```
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deploym
ent. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8050
INFO:werkzeug:Press CTRL+C to quit
```