import sys

import os

sys.path.append("/Users/manfrednde/Library/Python/3.9/lib/python/site-packages")

import io

import datetime

import pandas as pd

from PIL import Image

import requests

import io

import glob, os, sys, time, uuid

from matplotlib.pyplot import imshow

import matplotlib.pyplot as plt

from urllib.parse import urlparse

from io import BytesIO

from PIL import Image, ImageDraw

from video\_indexer import VideoIndexer

from azure.cognitiveservices.vision.face import FaceClient

from azure.cognitiveservices.vision.face.models import TrainingStatusType

from msrest.authentication import CognitiveServicesCredentials

CONFIG = {

'SUBSCRIPTION\_KEY': '621d1daaa9ff4041bc484d4f7b9d5d10',

'LOCATION': 'trial',

'ACCOUNT\_ID': '6723d089-da1a-41ab-ae6a-7074c23d084b'

}

video\_analysis = VideoIndexer(

vi\_subscription\_key=CONFIG['SUBSCRIPTION\_KEY'],

vi\_location=CONFIG['LOCATION'],

vi\_account\_id=CONFIG['ACCOUNT\_ID']

)

video\_analysis.check\_access\_token()

video\_id = '30fa204ae0'

video\_analysis.get\_video\_info(video\_id)

info = video\_analysis.get\_video\_info(video\_id, video\_language='English')

if len(info['videos'][0]['insights']['faces'][0]['thumbnails']):

print("We found {} faces in this video.".format(str(len(info['videos'][0]['insights']['faces'][0]['thumbnails']))))

info['videos'][0]['insights']['faces'][0]['thumbnails']

images = []

img\_raw = []

img\_strs = []

for each\_thumb in info['videos'][0]['insights']['faces'][0]['thumbnails']:

if 'fileName' in each\_thumb and 'id' in each\_thumb:

file\_name = each\_thumb['fileName']

thumb\_id = each\_thumb['id']

img\_code = video\_analysis.get\_thumbnail\_from\_video\_indexer(video\_id, thumb\_id)

img\_strs.append(img\_code)

img\_stream = io.BytesIO(img\_code)

img\_raw.append(img\_stream)

img = Image.open(img\_stream)

images.append(img)

for img in images:

print(img.info)

plt.figure()

plt.imshow(img)

i = 1

for img in images:

print(type(img))

img.save('human-face' + str(i) + '.jpg')

i= i+ 1

thumbnail\_id = "40294a85-eb72-4085-84c4-081e094ac866"

img\_code = video\_analysis.get\_thumbnail\_from\_video\_indexer(video\_id, thumbnail\_id)

print(img\_code)

img\_code = video\_analysis.get\_thumbnail\_from\_video\_indexer(video\_id, thumbnail\_id)

img\_stream = io.BytesIO(img\_code)

img = Image.open(img\_stream)

imshow(img)

keyframes = []

for shot in info["videos"][0]["insights"]["shots"]:

for keyframe in shot["keyFrames"]:

keyframes.append(keyframe["instances"][0]['thumbnailId'])

for keyframe in keyframes:

img\_str = video\_analysis.get\_thumbnail\_from\_video\_indexer(video\_id, keyframe)

info['summarizedInsights']['sentiments']

info['summarizedInsights']['emotions']

FACE\_KEY = "72c95a765bb54e47a54fb8f33f290252"

FACE\_ENDPOINT = "https://face-boarding.cognitiveservices.azure.com"

face\_client = FaceClient(FACE\_ENDPOINT, CognitiveServicesCredentials(FACE\_KEY))

face\_client.api\_version

print("API VERSION")

print(face\_client.api\_version)

PERSON\_GROUP\_ID = str(uuid.uuid4())

person\_group\_name = 'person-manfred'

def build\_person\_group(client, person\_group\_id, pgp\_name):

print('Create and build a person group...')

# Create empty Person Group. Person Group ID must be lower case, alphanumeric, and/or with '-', '\_'.

print('Person group ID:', person\_group\_id)

# client.person\_group.create(person\_group\_id = person\_group\_id, name=person\_group\_id)

client.person\_group.create(person\_group\_id = person\_group\_id, name=pgp\_name)

# Create a person group person.

human\_person = client.person\_group\_person.create(person\_group\_id, pgp\_name)

# Find all jpeg human images in working directory.

human\_face\_images = [file for file in glob.glob('\*.jpg') if file.startswith("human-face")]

# Add images to a Person object

for image\_p in human\_face\_images:

with open(image\_p, 'rb') as w:

client.person\_group\_person.add\_face\_from\_stream(person\_group\_id, human\_person.person\_id, w)

# Train the person group, after a Person object with many images were added to it.

client.person\_group.train(person\_group\_id)

# Wait for training to finish.

while (True):

training\_status = client.person\_group.get\_training\_status(person\_group\_id)

print("Training status: {}.".format(training\_status.status))

if (training\_status.status is TrainingStatusType.succeeded):

break

elif (training\_status.status is TrainingStatusType.failed):

client.person\_group.delete(person\_group\_id=PERSON\_GROUP\_ID)

sys.exit('Training the person group has failed.')

time.sleep(5)

build\_person\_group(face\_client, PERSON\_GROUP\_ID, person\_group\_name)