计算机科学技术学院

人工智能课程设计报告

题目：基于 MTCNN 的人脸识别

专业：计算机科学与技术

班级：20 计科 06

学号：2020202673

姓名：王中校

日期：2023/4/8

一、 项目设计目标及要求

伴随着人工智能技术的发展，机器学习特别是深度学习技术变得越来越热门， 本课程设计围绕人脸识别相关技术，设置多个模型的设计与实现，涉及人脸检测、人脸匹配、人脸关键点定位等基础研究技术，最终搭建一个人脸识别智能系统，并结合具体的应用场景完成整个项目的实战应用。 此项目设计实现 AI 技术落地应用，实现工程性与趣味性的结合，不仅可以锻炼学生工程项目搭建的能力，同时适用于培养学生的学术研究能力，最终使学生达到人工智能专业本科毕业生的基本要求。

二、 人脸识别问题概述

1. 概述

人脸识别，特指利用分析比较人脸视觉特征信息进行身份鉴别的计算机技术。 广义的人脸识别实际包括构建人脸识别系统的一系列相关技术，包括人脸图像采集、人脸定位、人脸识别预处理、身份确认以及身份查找等；而狭义的人脸识别 特指通过人脸进行身份确认或者身份查找的技术或系统。人脸识别是一项热门的计算机技术研究领域，它属于生物特征识别技术，是对生物体(一般特指人)本身的生物特征来区分生物体个体。生物特征识别技术所研究的生物特征包括脸、指纹、手掌纹、虹膜、视网 膜、声音(语音)、体形、个人习惯(例如敲击键盘的力度和频率、签字)等，相应的识别技术就有人脸识别、 指纹识别、掌纹识别、虹膜识别、视网膜识别、语音识别（用语音识别可以进行身份识别，也可以进行语音内容的识别，只有前者属于生物特征识别技术）、体形识别、键盘敲击识别、签字识别等。虽然人脸识别有很多其他识别无法比拟的优点，但是它本身也存在许多困难。人脸识别被认为是生物特征识别领域甚至人工智能领域最困难的研究课题之一。人脸识别的困难主要是人脸作为生物特征的特点所带来的。人脸在视觉上的特点是：

1.不同个体之间的区别不大，所有的人脸的结构都相似，甚至人脸器官的结构外形都很相似。这样的特点对于利用人脸进行定位是有利的，但是对于利用人脸区分人类个体是不利的。

2.人脸的外形很不稳定，人可以通过脸部的变化产生很多表情，而在不同观察角度，人脸的视觉图像也相差很大，另外，人脸识别还受光照条件(例如白天和夜晚，室内和室外等)、人脸的很多遮盖物 (例如口罩、墨镜、头发、胡须等)、 年龄、拍摄的姿态角度等多方面因素的影响。

2. 人脸识别基础模块

设计了三个 python 文件，分别用于人脸获取、模型训练和人脸识别。

三、 具体实现

1. 人脸抓取.py

首先导入所需要的库文件

```
import os
import cv2
import torch
from facenet_pytorch import MTCNN
```

定义人脸获取函数

```
def get_face(name, num):
```

参数 name 为获取人脸的名称，必须为英文，num 为要获取的人脸图片数量，这里我们设置为 500.

对于每一次运行，都会判断当前工作区中是否有以 name 命名的文件夹，如果不存在，那么我们创建名为 name 的文件夹。

使用 MTCNN 对当前的人脸进行检测，返回人脸的五个坐标，分别为两只眼睛、鼻子和两个嘴角在图片中的坐标 mtcnn = MTCNN(device=device, keep_all=True)

然后利用 opencv 创建视频窗口，cv2.namedWindow(windows_name)，然后使用 cv2.VideoCapture()函数使用电脑的摄像头进行拍摄。

接着就可以进行人脸图片的获取工作。使用 while 循环，首先使用 cap.read()读取一帧图像，cap.read()有两个返回值，分别为 success 和 image，如果读取到一帧图像，success 会返回 true，否则返回 false；如果读取图像成功便会返回图像 image。使用 mtcnn.detect(image)检测人脸位置，detect()有两个返回值，分别为人脸的五个特征值和人脸检测率，如果检测成功，boxes 不为空，接着判断 prob 的值，如果低于我们设定的阈值，使用 continue 转入下一次循环，如果成功，将当前读取到的图像保存为图片，

```
# 将当前人脸保存为图片
face_image_name = f'faces/{name}/{name}_{count}.jpg'
```

然后对于当前图片进行处理，使用偏移量圈出人脸在获取图片中的位置

```
face_image = image[y1 - 10:y2 + 10, x1 - 10: x2 + 10]
cv2.imwrite(face_image_name, face_image, [int(cv2.IMWRITE_PNG_COMPRESSION), 9])

# 框出人脸位置
cv2.rectangle(image, (x1 - 10, y1 - 10), (x2 + 10, y2 + 10), color=(0, 255, 0), thickness=2)
cv2.putText(image, f'num:{count}', (x1, y1 - 30), cv2.FONT_ITALIC, 1, (255, 0, 255), 4)
```

2. 模型训练.py

首先导入所需的库文件

```
from torchvision import datasets
import numpy as np
import pandas as pd
import os
from sklearn.neighbors import KNeighborsClassifier as KNN
import joblib
```

使用 mtcnn 检测人脸位置

```
mtcnn = MTCNN(
    image_size=160, margin=0, min_face_size=20,
    thresholds=[0.6, 0.7, 0.7], factor=0.709, post_process=True,
    device=device
)
```

使用 InceptionResnetV1 函数生成人脸 512 维特征向量，用于后续的模型训练，

```
resnet = InceptionResnetV1(pretrained='vggface2').eval().to(device)  # 使用 InceptionResnetV1 卷积神经网络  在 vggface2 人脸数据集上进行训练
```

定义 collate_fn 用于数据整理，返回传入数据的标签，用于后续的模型训练

```
def collate_fn(x):  # 数据整理 获取样本
    return x[0]
```

定义 train 函数用于模型训练，定义 path 用于说明训练图像路径，使用

```
loader = DataLoader(dataset, collate_fn=collate_fn, num_workers=0,
batch_size=1)
```

加载路径下的图像并打印输出。

将读取到的图像传入到神经网络中 aligned = torch.stack(aligned).to(device)，使用 cpu 进行特征提取 embeddings = resnet(aligned).detach().cpu()。将 embeddings 转换为 DataFrame 格式， a = pd.DataFrame(np.array(embeddings), index=names)， 对 a 进行特征提取 a.reset_index(inplace=True)，对 a 添加特征列 a.columns = ['label'] + [f'v{i}' for i in range(512)]，最后返回 a 即为我们所需。

下面进行模型训练，使用 train 函数训练 faces 文件夹下的文件，然后将返回值拼接到 face_data 变量中，具体实现为

```
face_data = pd.concat([
    train('faces')
])
```

将 face_data 特征写入 face_feature.csv 文件，用于后续人脸识别的特征匹配，实现代码为

```
face_data.to_csv('face_feature.csv', index=False, encoding='utf8')  # 保存为
csv 特征文件
```

训练 KNN 模型，用于后续人脸识别。

```
x = face_data.drop(columns=['label'])
y = face_data['label']
knn = KNN(n_neighbors=5)
knn.fit(x.values, y)
```

将训练好的模型参数保存，如果已经存在模型，则删除旧模型并保存新的模型。

```
if os.path.exists('knn_model.pkl'):
    os.remove('knn_model.pkl')  # 删除旧的模型

joblib.dump(knn, 'knn_model.pkl')  # 保存新的模型
print('导出模型')
```

最后是我们生成的两个文件

3. 人脸识别

首先导入所需的库文件

```
import cv2
import joblib
import numpy as np
import pandas as pd
import torch
from facenet_pytorch import MTCNN, InceptionResnetV1
from PIL import Image, ImageDraw, ImageFont
```

定义人脸识别器，我们使用类实现。

首先进行初始化，加载数据，我们选择读取训练好的人脸特征数据，在第二步中我们生成了人脸特征 csv 文件，这里我们对他进行读取

```
self.data = pd.read_csv(face_feature_path)
self.x = self.data.drop(columns=['label'])
self.y = self.data['label']
```

然后加载训练好的 KNN 分类器模型，self.knn_model = joblib.load(knn_model_path)。

根据特征向量识别人脸，使用欧氏距离，如果距离大于 1 则认为识别失败，这里与 KNN 模型功能重复，只是想要计算一个最小距离，略微影响识别性能。

```
    def _recognize(self, v):
        dis = np.sqrt(sum((v[0] - self.x.iloc[0]) ** 2))
        name = self.y[0]

        for i in range(1, self.x.shape[0]):
            temp_dis = np.sqrt(sum((v[0] - self.x.iloc[i]) ** 2))
            if temp_dis < dis:
                dis = temp_dis
                name = self.y[i]

        return name, dis
```

然后实现人脸识别主函数，首先是用 mtcnn 检测人脸位置，mtcnn = MTCNN(device=self.device, keep_all=True)，然后生成人脸 512 维特征向量，resnet = InceptionResnetV1(pretrained='vggface2').eval().to(self.device)。

接着开始人脸识别的准备工作，我们使用 opencv 调用摄像头识别人脸，首先初始化视频窗口，windows_name = 'face'，然后调用本机摄像头，

```
cv2.namedWindow(windows_name)
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
```

我们从摄像头读取一帧图像，读取成功后，我们将图像转换为 PIL 格式，

```
img_PIL = Image.fromarray(image)
draw = ImageDraw.Draw(img_PIL)
```

然后检测人脸位置,获得人脸框坐标和人脸概率，boxes, probs = mtcnn.detect(image)，prob 会获取当前图像检测为人脸的概率，设置人脸检测阈值，这里我们设置为 0.9，当概率低于阈值时，我们重新获取图像并进行计算。

设置两个坐标，(x1,y1)和(x2,y2)，

```
x1, y1, x2, y2 = [int(p) for p in box]
```

我们框出人脸位置

```
draw.rectangle((x1, y1, x2, y2), outline=(0, 255, 0), width=2)
```

然后导出人脸图像

```
face = mtcnn.extract(image, [box], None).to(self.device)
```

然后生成 512 维特征向量

```
embeddings = resnet(face).detach().cpu().numpy()
```

然后进行 KNN 预测

```
name_knn = self.knn_model.predict(embeddings)
```

我们获得预测姓名和距离

```
_, dis = self._recognize(embeddings)
```

如果距离过大则认为识别失败

```
if dis > 0.9:
    draw.rectangle((x1, y1, x2, y2), outline=(0, 0, 255), width=2)
    draw.text((x1, y1 - 40), f'未知', font=self.font, fill=(0, 0, 255))
else:
    # 框出人脸位置并写上名字
    draw.rectangle((x1, y1, x2, y2), outline=(0, 255, 0), width=2)
    draw.text((x1, y1 - 40), f'{name_knn[0]}({round(dis, 2)})', font=self.font, fill=(0, 255, 0))
```

然后显示处理后的图片

```
cv2.imshow(windows_name, np.array(img_PIL))
```

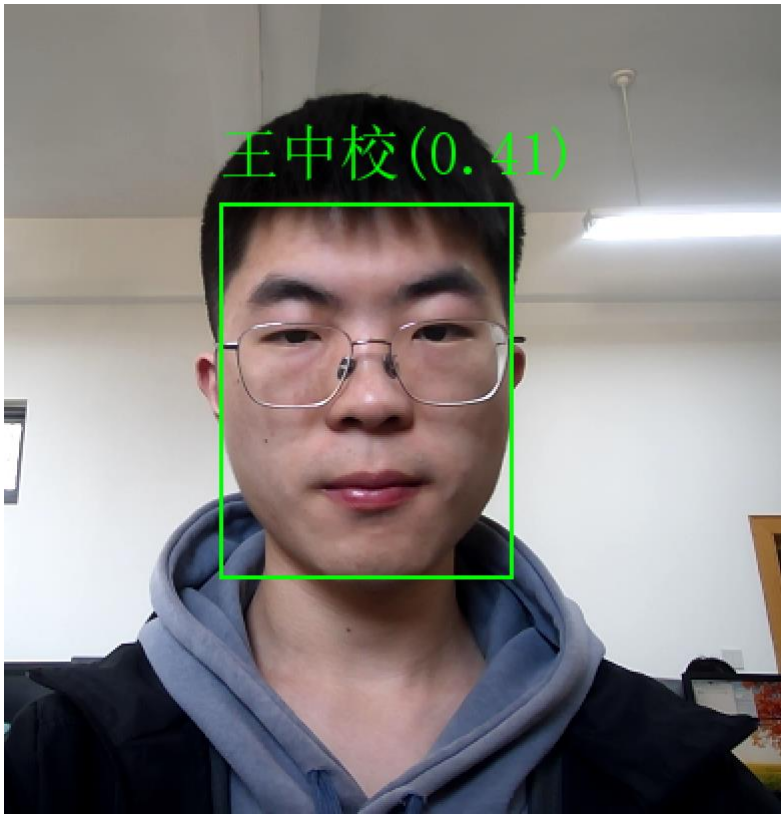
最后通过主函数启动功能

```
if __name__ == '__main__':
    try:
        fr = FaceRecognizer()
        fr.start_recognize()
    except Exception as e:
```

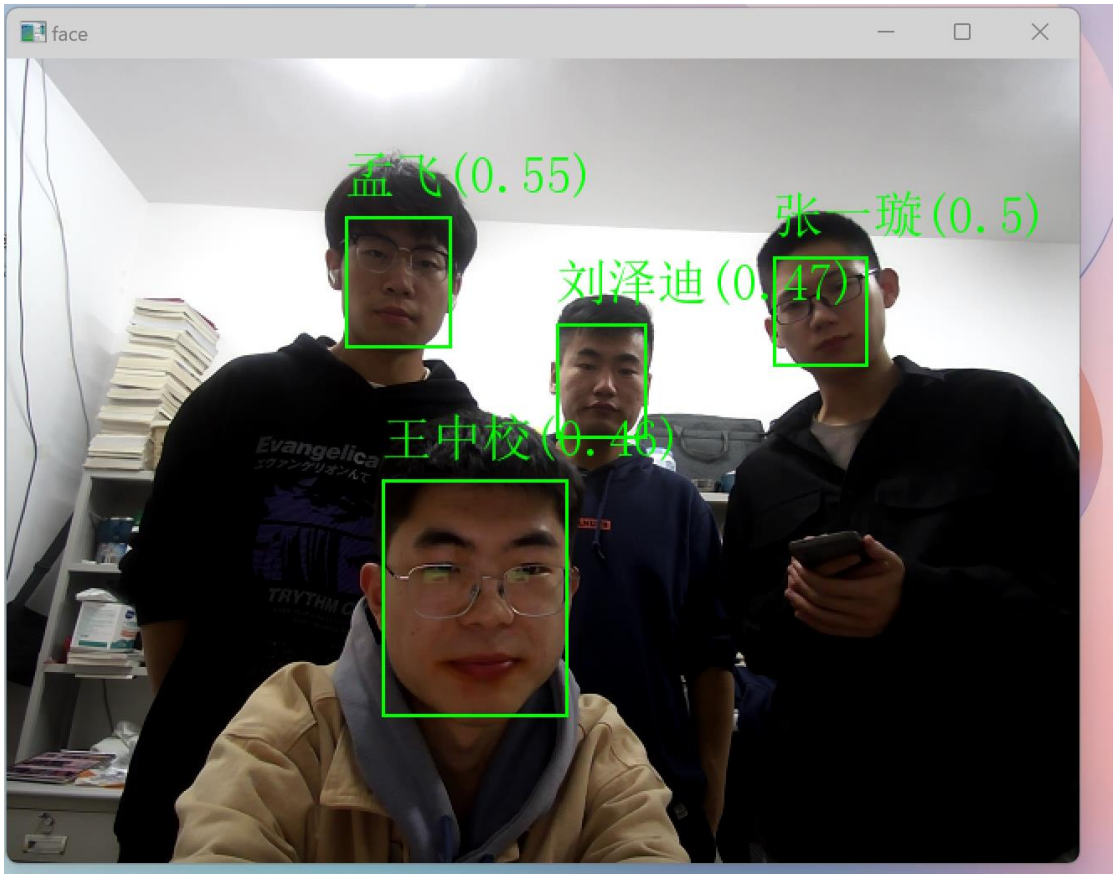
```
print(e)
```

四、 识别效果

当识别一个人时：



当识别多人时：



我们发现识别效果还是不错的。

附录代码：

人脸抓取.py

```
import os
import cv2
import torch
from facenet_pytorch import MTCNN

# 选择设备
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print('Running on device: {}'.format(device))
```

```
def get_face(name, num):
    if not os.path.exists(f'faces/{name}'):
        os.makedirs(f'faces/{name}')
    # mtcnn 检测人脸位置
    mtcnn = MTCNN(device=device, keep_all=True)

    # 初始化视频窗口
    windows_name = 'face'
    cv2.namedWindow(windows_name)
    cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

    count = 0
    while True:
        # 从摄像头读取一帧图像
        success, image = cap.read()
        if not success:
            break

        # 检测人脸位置
        boxes, probs = mtcnn.detect(image)
        if boxes is not None:
            for box, prob in zip(boxes, probs):
                # 设置人脸检测阈值
                if prob < 0.9:
                    continue

                x1, y1, x2, y2 = [int(p) for p in box]

                # 将当前人脸保存为图片
                face_image_name = f'faces/{name}/{name}_{count}.jpg'
                count += 1
                if count > num:
                    break
```

```
                print(face_image_name)
                face_image = image[y1 - 10:y2 + 10, x1 - 10: x2 + 10]
                cv2.imwrite(face_image_name, face_image, [int(cv2.IMWRITE_PNG_COMPRESSION), 9])

                # 框出人脸位置
                cv2.rectangle(image, (x1 - 10, y1 - 10), (x2 + 10, y2 + 10), color=(0, 255, 0), thickness=2)
                cv2.putText(image, f'num:{count}', (x1, y1 - 30), cv2.FONT_ITALIC, 1, (255, 0, 255), 4)

            # 显示处理后的图片
            cv2.imshow(windows_name, image)

            if count > num:
                break
            # 保持窗口
            key = cv2.waitKey(1)
            if key & 0xff == 27:
                break

        # 释放设备资源，销毁窗口
        cap.release()
        cv2.destroyAllWindows()


if __name__ == '__main__':
    get_face('wangzhongxiao', 500)
```

模型训练.py

```
from facenet_pytorch import MTCNN, InceptionResnetV1
import torch
from torch.utils.data import DataLoader
```

```
from torchvision import datasets
import numpy as np
import pandas as pd
import os
from sklearn.neighbors import KNeighborsClassifier as KNN
import joblib

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print('Running on device: {}'.format(device))

# mtcnn 检测人脸位置
mtcnn = MTCNN(
    image_size=160, margin=0, min_face_size=20,
    thresholds=[0.6, 0.7, 0.7], factor=0.709, post_process=True,
    device=device
)

# 生成人脸 512 维特征向量
resnet = InceptionResnetV1(pretrained='vggface2').eval().to(device)  # 使用 InceptionResnetV1 卷积神经网络 在 vggface2 人脸数据集上进行训练


def collate_fn(x):  # 数据整理 获取样本
    return x[0]


def train(path):  # 训练图像路径
    dataset = datasets.ImageFolder(path)
    dataset.idx_to_class = {i: c for c, i in dataset.class_to_idx.items()}
    loader = DataLoader(dataset, collate_fn=collate_fn, num_workers=0, batch_size=1)

    print(dataset.idx_to_class)

    aligned = []
    names = []
```

```
        i = 0
        for x, y in loader:
            try:
                x_aligned, prob = mtcnn(x, return_prob=True)  # mtcnn 返回 5 个关键点，分别是左眼，右眼，鼻子，左嘴角，右嘴角
                if x_aligned is not None:  # 有返回值
                    print(f'batch {i}')
                    i += 1
                    aligned.append(x_aligned)
                    names.append(dataset.idx_to_class[y])
            except Exception as e:
                print(e)

        aligned = torch.stack(aligned).to(device)
        embeddings = resnet(aligned).detach().cpu()

        a = pd.DataFrame(np.array(embeddings), index=names)
        a.reset_index(inplace=True)
        a.columns = ['label'] + [f'v{i}' for i in range(512)]

        return a


    if __name__ == '__main__':

        face_data = pd.concat([
            train('faces')
        ])

        face_data.to_csv('face_feature.csv', index=False, encoding='utf8')  # 保存为 csv 特征文件

        # 训练 KNN 模型
        x = face_data.drop(columns=['label'])
        y = face_data['label']
```

```
        knn = KNN(n_neighbors=5)
        knn.fit(x.values, y)

        if os.path.exists('knn_model.pkl'):
            os.remove('knn_model.pkl')  # 删除旧的模型

        joblib.dump(knn, 'knn_model.pkl')  # 保存新的模型
        print('导出模型')
```

人脸识别.py

```
import cv2
import joblib
import numpy as np
import pandas as pd
import torch
from facenet_pytorch import MTCNN, InceptionResnetV1
from PIL import Image, ImageDraw, ImageFont


# 人脸识别器
class FaceRecognizer:
    # 初始化，加载数据
    def __init__(self, knn_model_path='knn_model.pkl', face_feature_path='face_feature.csv'):
        # 选择设备
        self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
        print('Running on device: {}'.format(self.device))

        # 读取训练好的人脸特征数据
        self.data = pd.read_csv(face_feature_path)
        self.x = self.data.drop(columns=['label'])
        self.y = self.data['label']

        # 加载训练好的 KNN 分类器模型
```

```
        self.knn_model = joblib.load(knn_model_path)

        # 字体文件，用于在图片上正确显示中文
        self.font = ImageFont.truetype('simsun.ttc', size=30)

# 根据特征向量识别人脸，使用欧氏距离，如果距离大于 1 则认为识别失败
# 这里与 KNN 模型功能重复，只是想要计算一个最小距离，略微影响识别性能
def _recognize(self, v):
        dis = np.sqrt(sum((v[0] - self.x.iloc[0]) ** 2))
        name = self.y[0]

        for i in range(1, self.x.shape[0]):
            temp_dis = np.sqrt(sum((v[0] - self.x.iloc[i]) ** 2))
            if temp_dis < dis:
                dis = temp_dis
                name = self.y[i]

        return name, dis

# 人脸识别主函数
def start_recognize(self):
        # mtcnn 检测人脸位置
        mtcnn = MTCNN(device=self.device, keep_all=True)
        # 用于生成人脸 512 维特征向量
        resnet = InceptionResnetV1(pretrained='vggface2').eval().to(self.device)

        # 初始化视频窗口
        windows_name = 'face'

        cv2.namedWindow(windows_name)
        cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

        while True:
            # 从摄像头读取一帧图像
            success, image = cap.read()
```

```
if not success:
        break

img_PIL = Image.fromarray(image)
draw = ImageDraw.Draw(img_PIL)

# 检测人脸位置,获得人脸框坐标和人脸概率
boxes, probs = mtcnn.detect(image)
if boxes is not None:
        for box, prob in zip(boxes, probs):
                # 设置人脸检测阈值
                if prob < 0.9:
                        continue

                x1, y1, x2, y2 = [int(p) for p in box]
                # 框出人脸位置
                draw.rectangle((x1, y1, x2, y2), outline=(0, 255, 0), width=2)

                # 导出人脸图像
                face = mtcnn.extract(image, [box], None).to(self.device)
                # 生成 512 维特征向量
                embeddings = resnet(face).detach().cpu().numpy()
                # KNN 预测
                name_knn = self.knn_model.predict(embeddings)

                # 获得预测姓名和距离
                _, dis = self._recognize(embeddings)
                # 如果距离过大则认为识别失败
                if dis > 0.9:
                        draw.rectangle((x1, y1, x2, y2), outline=(0, 0, 255), width=2)
                        draw.text((x1, y1 - 40), f'未知', font=self.font, fill=(0, 0, 255))
                else:
                        # 框出人脸位置并写上名字
                        draw.rectangle((x1, y1, x2, y2), outline=(0, 255, 0), width=2)
```

```
                                draw.text((x1, y1 - 40), f'{name_knn[0]}({round(dis, 2)})',
font=self.font, fill=(0, 255, 0))

                    # 显示处理后的图片
                    cv2.imshow(windows_name, np.array(img_PIL))

                    # 保持窗口
                    key = cv2.waitKey(1)
                    # ESC 键退出
                    if key & 0xff == 27:
                        break

                # 释放设备资源，销毁窗口
                cap.release()
                cv2.destroyAllWindows()


    if __name__ == '__main__':
        try:
            fr = FaceRecognizer()
            fr.start_recognize()
        except Exception as e:
            print(e)
```