

# 实战项目 3：人脸考勤机

## 人脸检测方法概述

Haar cascade + opencv

HOG + Dlib

CNN + Dlib

SSD

MTCNN

各种检测方法对比

视频中的人脸检测

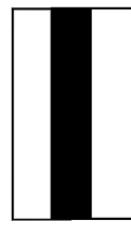
## Haar 特征

Haar 特征是由 Paul Viola 和 Michael Jones 在其 2001 年的论文中提出的，用于快速的面部检测。Haar 特征是一种在图像处理中用于对象识别的特征集，它基于图像的局部区域内像素强度的差异。

### Haar 特征



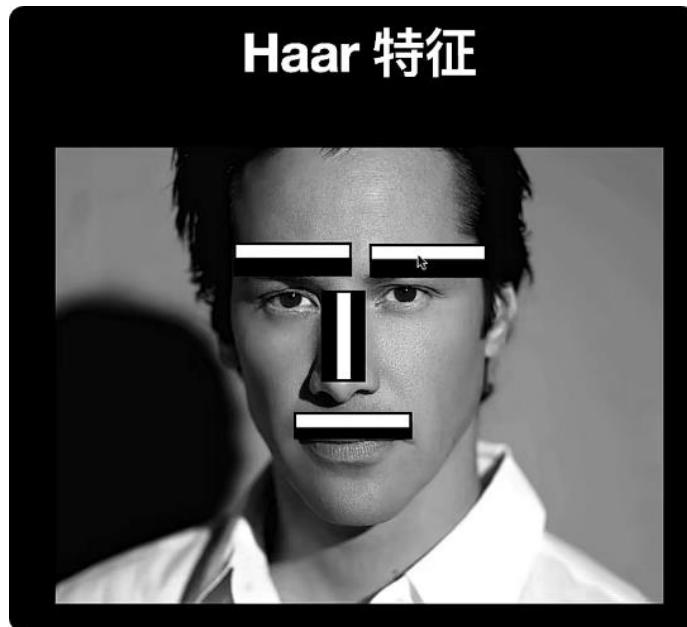
边缘特征检测



线状特征检测

Haar 特征由边缘特征、线特征、矩形特征和中心-周围特征四种基本类型组成，这些特征由连接的矩形区域组成，每个区域内像素的和计算出来，然后从相邻矩形区域的和中减去，得到的结果就是 Haar 特征的值。这些矩形特征通常被应用在灰度图像上。

例如，一个简单的 Haar 特征可能是由两个并排的矩形组成的，一个矩形覆盖图像的某个区域，另一个矩形覆盖紧邻的相同大小的区域。如果第一个矩形内的像素平均强度明显不同于第二个矩形内的像素平均强度，那么这个特征的值就会很大，表明这一区域可能包含了图像的某种边缘或者颜色变化。



在面部检测中，Haar 特征可以用来识别面部的不同部分，比如眼睛周围（眼睛区域通常比脸颊颜色要深）或者鼻梁两侧的边缘。将很多这样的特征组合起来，再通过一个机器学习模型（如 AdaBoost）进行训练，就可以用来快速有效地检测图像中的人脸。

## Haar 特征

算法会比较实际特征与理想特征接近程度

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

理想特征

0.1	0.2	0.6	0.8
0.2	0.3	0.8	0.6
0.2	0.1	0.6	0.8
0.2	0.1	0.8	0.9

实际特征

1. 计算黑色像素平均值；
2. 计算白色像素平均值；
3. 计算黑白像素平均值差值；

$$\Delta = dark - white = \frac{1}{n} \sum_{dark}^n P(x) - \frac{1}{n} \sum_{white}^n P(x)$$

△ 理想特征 = 1

△ 实际特征 =  $0.74 - 0.18 = 0.56$

越接近1，说明越可能找到这个特征！

## haar 代码实现

# 步骤

# 1、读取包含人脸的图片

# 2. 使用 haar 模型识别人脸

# 3. 将识别结果用矩形框画出来

```
# 导入相关包

import cv2

import numpy as np

import matplotlib.pyplot as plt

# %matplotlib inline

plt.rcParams['figure.dpi'] = 200
```

# 读取图片

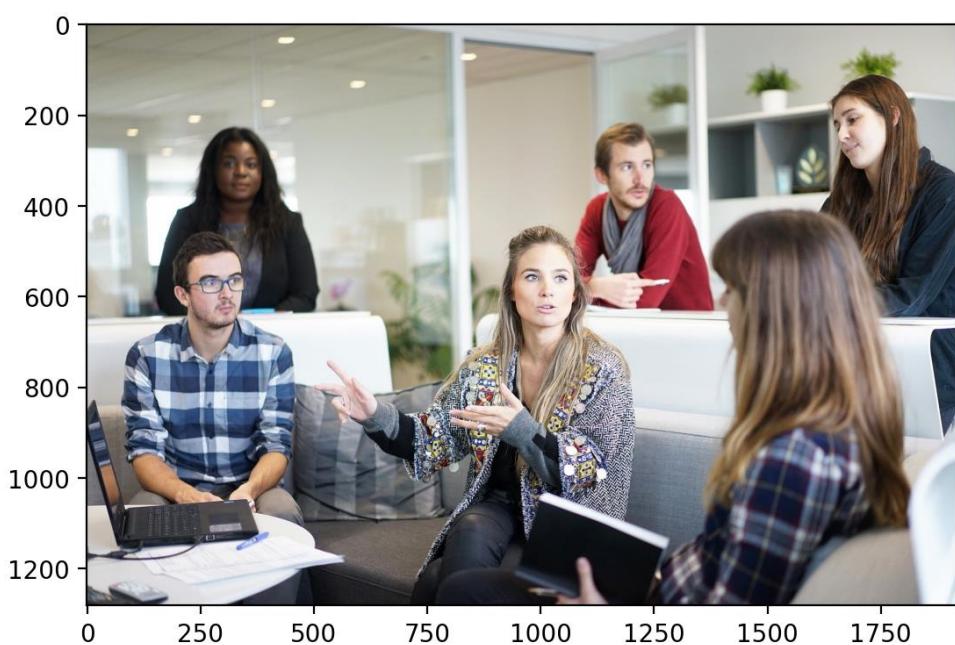
```
img = cv2.imread('./images/faces1.jpg')
```

# 查看大小

```
img.shape

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

><matplotlib.image.AxesImage at 0x25e1264fffd0>
```



# 构造 haar 检测器（权重文件，GitHub 上有除了人脸的，还有全身的等等）

```
face_detector = cv2.CascadeClassifier('./cascades/haarcascade_frontalface_default.xml')
```

# 转为灰度图

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

# 检测结果

```
detections = face_detector.detectMultiScale(img_gray)
```

# 打印

```
detections
```

```
>array([[ 284,  263,   113,   113],  
       [1148,  260,   129,   129],  
       [ 928,  488,   172,   172],  
       [1637,  158,   142,   142],  
       [ 229,  509,   142,   142],  
       [ 888,  396,   226,   226],  
       [ 103,  784,     56,     56]])
```

```
detections.shape
```

```
>(7, 4)
```

# 解析检测结果

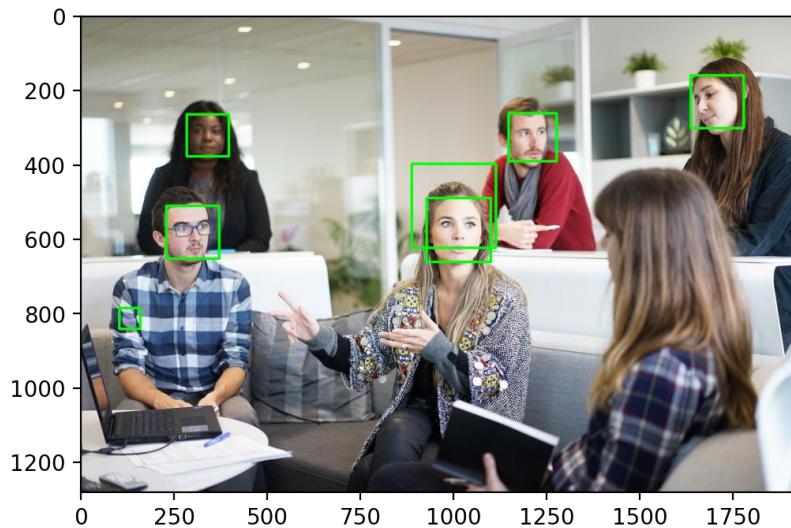
```
for (x,y,w,h) in detections:
```

```
    cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 5)
```

# 显示绘制结果

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
><matplotlib.image.AxesImage at 0x1ef0121d310>
```



# 调节参数，优化结果

# scaleFactor: 调整图片尺寸

# minNeighbors: 候选人脸数量

```
img = cv2.imread('./images/faces1.jpg')

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

detections = face_detector.detectMultiScale(img_gray, scaleFactor=1.3)

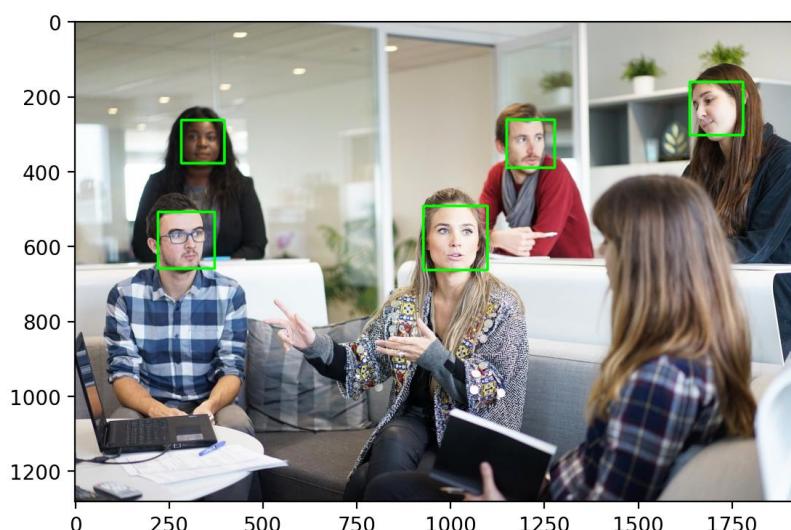
# 解析检测结果

for (x,y,w,h) in detections:

    cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 5)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

><matplotlib.image.AxesImage at 0x25e127224c0>
```



```
# 调节参数解释
```

# **scaleFactor**: 这个参数指定在图像尺寸减小时，图像缩放的比例。它用于在图像的不同大小上检测物体，因为可能不同的图像尺寸会包含不同大小的物体。**scaleFactor** 越大，检测过程中图像的缩放就越快，这可能会快速减少计算时间，但也可能错过一些较小的物体。通常，**scaleFactor** 的值介于 1.01（最慢，最多尺寸变化）到 1.1 之间。

# **minNeighbors**: 这个参数指定每个候选矩形需要保留的邻居个数。这个参数会影响检测到的物体的质量。较小的值会导致更多的检测结果，但可能会增加误检的数量。相对的，较大的值会得到更高质量的检测结果，但可能会错过一些检测。

```
# minSize: 最小人脸尺寸（元组）
```

```
# maxSize: 最大人脸尺寸（元组）
```

```
img = cv2.imread('./images/faces2.jpg')

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

detections =

face_detector.detectMultiScale(img_gray, scaleFactor=1.2, minNeighbors=7, minSize=(10, 10), maxSize=(100, 100))

# 解析检测结果

for (x, y, w, h) in detections:

    print(w, h)

    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 5)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

> 46 46

47 47

47 47

49 49

49 49

45 45

51 51

47 47
```

```
52 52  
51 51  
51 51  
[16]:  
<matplotlib.image.AxesImage at 0x25e12fdb880>
```



## HOG 代码实现

# 导入相关包

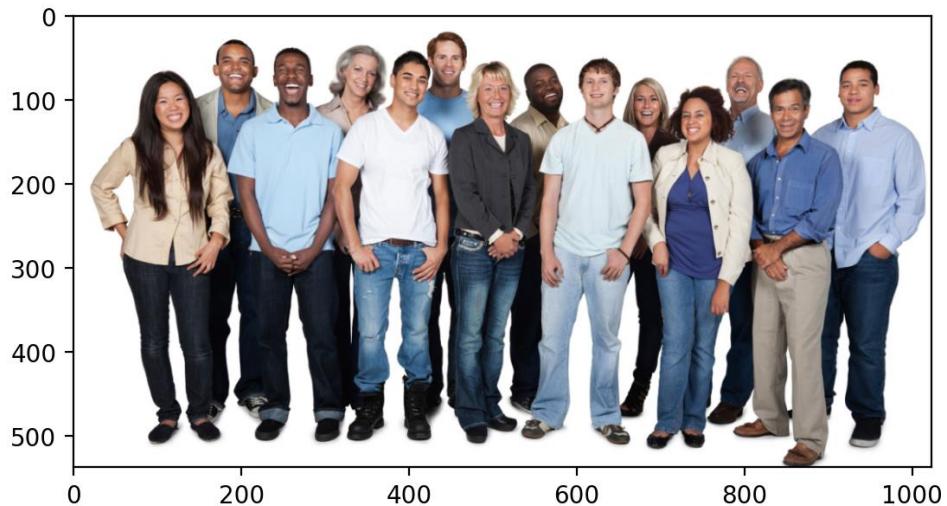
```
import cv2  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
# %matplotlib inline  
  
plt.rcParams['figure.dpi'] = 200
```

# 读取照片

```
img = cv2.imread('./images/faces2.jpg')
```

# 显示照片

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  
><matplotlib.image.AxesImage at 0x178ce9d3700>
```



# 安装 DLIB，你可能需要在 conda 终端中输入命令 “`conda install -c conda-forge dlib`” 来安装 `dlib` 库。

```
import dlib

# 构造 HOG 人脸检测器

hog_face_detetor = dlib.get_frontal_face_detector()

# 检测人脸

# scale 类似 haar 的 scaleFactor

detections  = hog_face_detetor(img,1)

# 打印一下

detections

>rectangles[[ (717, 103) (760, 146)], [(665, 90) (701, 126)], [(429, 38) (465, 74)], [(909, 70) (952, 113)], [(828, 98) (871, 142)], [(605, 70) (641, 106)], [(777, 62) (813, 98)], [(485, 78) (521, 114)], [(386, 60) (429, 103)], [(170, 41) (213, 84)], [(93, 89) (136, 132)], [(237, 50) (280, 94)], [(323, 50) (367, 94)], [(544, 65) (588, 108)]]


type(detections)

>_dlib_pybind11.rectangles


len(detections)

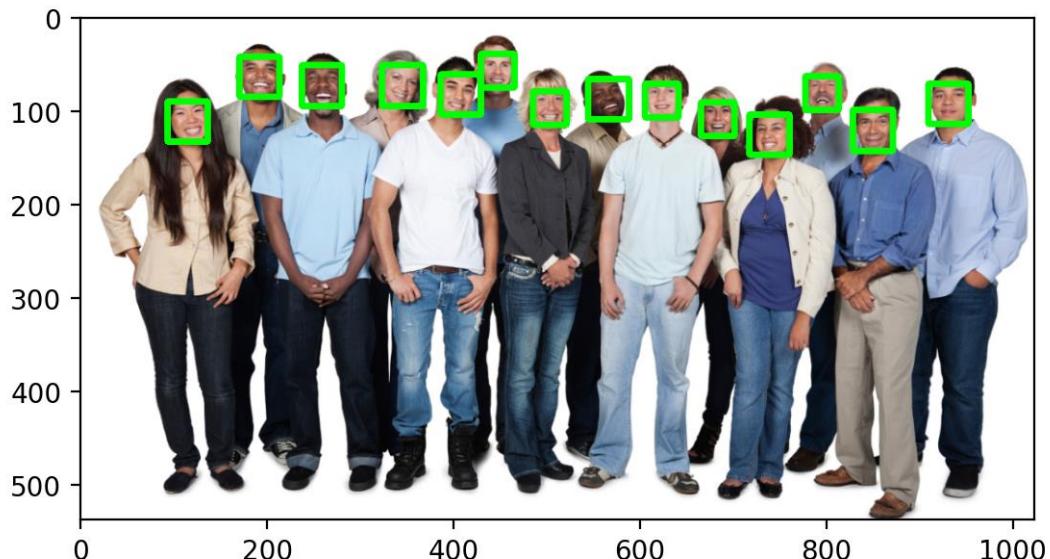
>14
```

### # 解析矩形结果

```
for face in detections:  
    x = face.left()  
    y = face.top()  
    r = face.right()  
    b = face.bottom()  
  
    cv2.rectangle(img, (x,y), (r,b), (0,255,0), 5)
```

### # 显示照片

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  
  
<matplotlib.image.AxesImage at 0x178cef65970>
```



## 卷积神经网络 CNN 方法

### # 导入相关包

```
import cv2  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
# %matplotlib inline  
  
plt.rcParams['figure.dpi'] = 200
```

# 读取照片

```
img = cv2.imread('./images/faces2.jpg')
```

# 显示照片

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

# 安装 DLIB

```
import dlib
```

# 构造 CNN 人脸检测器

```
cnn_face_detector =
```

```
dlib.cnn_face_detection_model_v1('./weights/mmod_human_face_detector.dat')
```

# 检测人脸

```
detections = cnn_face_detector(img, 1)
```

# 解析矩形结果

```
for face in detections:
```

```
    x = face.rect.left()
```

```
    y = face.rect.top()
```

```
    r = face.rect.right()
```

```
    b = face.rect.bottom()
```

#置信度

```
c = face.confidence
```

```
print(c)
```

```
cv2.rectangle(img, (x,y), (r,b), (0,255,0), 5)
```

```
>1.1351913213729858
```

```
1.1195666790008545
```

```
1.099207878112793
```

```
1.08876371383667
```

```
1.0814990997314453
```

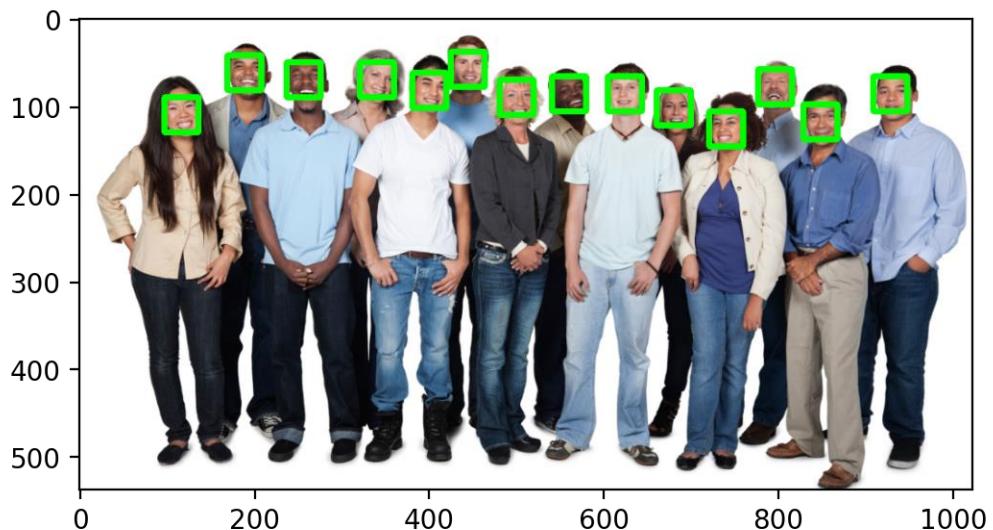
```
1.0712707042694092
```

```
1.069579005241394
```

```
1.0681637525558472  
1.0630664825439453  
1.0594005584716797  
1.0533159971237183  
1.0519158840179443  
1.039461612701416  
1.0381401777267456
```

# 显示照片

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  
<matplotlib.image.AxesImage at 0x1f1918b17c0>
```



## SSD 代码实现

# 导入包

```
import cv2  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
plt.rcParams['figure.dpi']=200
```

# 读取照片

```
img = cv2.imread('./images/faces2.jpg')
```

# 展示

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

# deploy.prototxt.txt:

[https://github.com/opencv/opencv/tree/master/samples/dnn/face\\_detector](https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector)

# res10\_300x300\_ssd\_iter\_140000.caffemodel:

[https://github.com/Shiva486/facial\\_recognition/blob/master/res10\\_300x300\\_ssd\\_iter\\_140000.caffemodel](https://github.com/Shiva486/facial_recognition/blob/master/res10_300x300_ssd_iter_140000.caffemodel)

# 加载模型

```
face_detector =  
cv2.dnn.readNetFromCaffe('./weights/deploy.prototxt','./weights/res10_300x300_ssd_iter  
_140000.caffemodel')
```

# 原图尺寸

```
img_height = img.shape[0]  
img_width = img.shape[1]
```

# 缩放至模型输入尺寸

```
img_resize = cv2.resize(img, (500,300))
```

# 图像转为 blob

```
img_blob = cv2.dnn.blobFromImage(img_resize,1.0,(500,300),(104.0, 177.0, 123.0))
```

# 输入

```
face_detector.setInput(img_blob)
```

# 推理

```
detections = face_detector.forward()  
  
detections  
  
>array([[[[0.          , 1.          , 0.9757996 , ... , 0.06375282,  
        0.21002546, 0.16640264],  
        [0.          , 1.          , 0.96749574, ... , 0.1006325 ,  
        0.4138369 , 0.20273851],  
        [0.          , 1.          , 0.9486636 , ... , 0.1493349 ,  
        0.3544941 , 0.1961784 ]]] )
```

```
0.13494208, 0.24810775],  
...,  
[0., 0., 0., ..., 0.,  
0., 0.],  
[0., 0., 0., ..., 0.,  
0., 0.],  
[0., 0., 0., ..., 0.,  
0., 0.]]], dtype=float32)
```

# 查看大小 (200 是人脸的数量)

```
detections.shape  
>(1, 1, 200, 7)
```

# 查看检测人脸数量

```
num_of_detections = detections.shape[2]  
num_of_detections  
>200
```

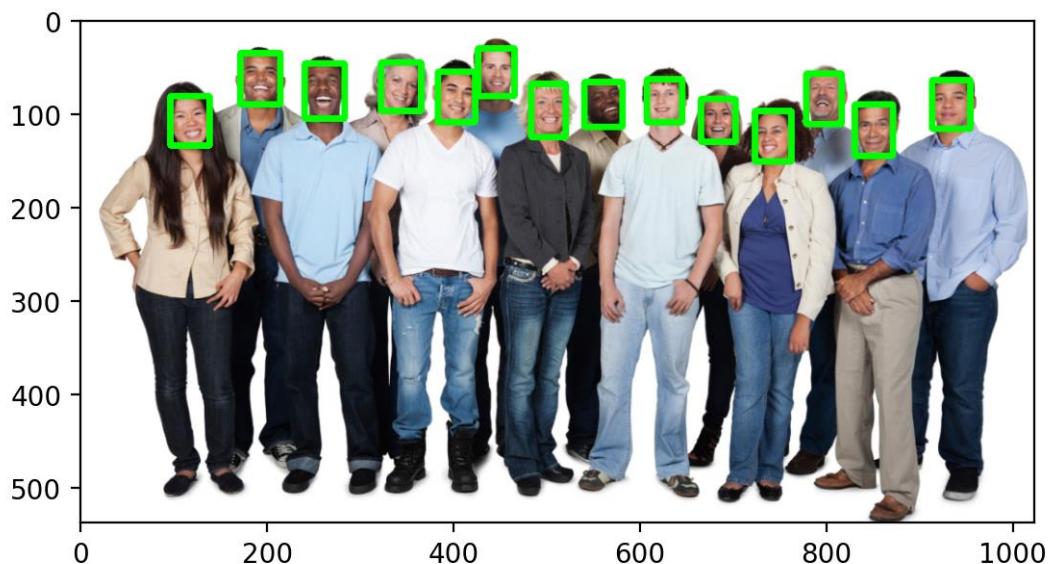
# 原图复制，一会绘制用

```
img_copy = img.copy()  
  
for index in range(num_of_detections):  
  
    # 置信度  
  
    detection_confidence = detections[0, 0, index, 2]  
  
    # 挑选置信度  
  
    if detection_confidence > 0.15:  
  
        # 位置  
  
        locations = detections[0, 0, index, 3:7] *  
  
        np.array([img_width, img_height, img_width, img_height])  
  
        # 打印  
  
        print(detection_confidence * 100)  
  
        lx, ly, rx, ry = locations.astype('int')
```

```
# 绘制
cv2.rectangle(img_copy, (lx,ly), (rx,ry), (0,255,0),5)

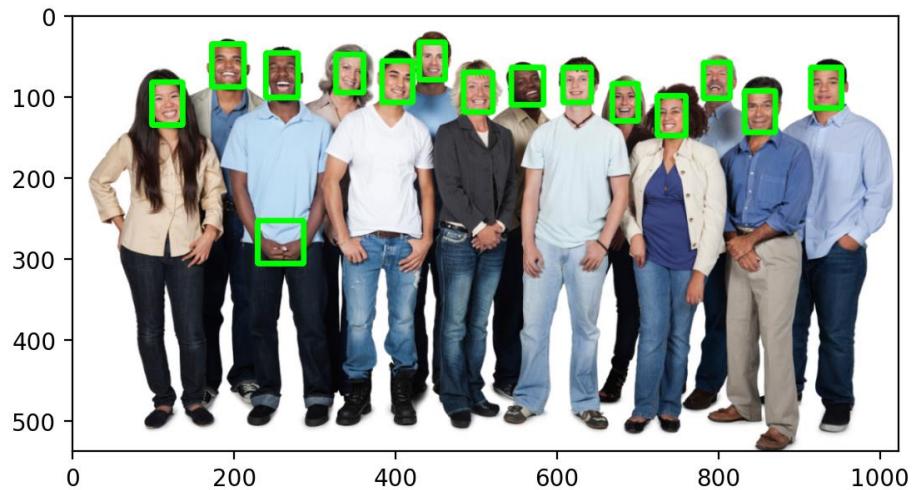
# 展示
plt.imshow(cv2.cvtColor(img_copy, cv2.COLOR_BGR2RGB))

>97.57995009422302
96.74956202507019
94.86633539199829
93.22233200073242
90.09220600128174
86.59671545028687
85.67284941673279
82.45850801467896
79.41350936889648
68.80926489830017
67.29023456573486
62.12405562400818
40.834400057792664
20.884333550930023
<matplotlib.image.AxesImage at 0x151d7d79910>
```



## MTCNN 代码实现

```
#安装 MTCNN: conda install -c conda-forge mtcnn  
#安装 Tensorflow: conda install -c conda-forge tensorflow  
  
# 导入包  
  
import cv2  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
plt.rcParams['figure.dpi']=200  
  
# 读取照片  
  
img = cv2.imread('./images/faces2.jpg')  
  
# MTCNN 需要 RGB 通道顺序  
  
img_cvt = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
  
# 展示  
  
plt.imshow(img_cvt)  
  
# 导入 MTCNN  
  
from mtcnn.mtcnn import MTCNN  
  
# 加载模型  
  
face_detetor = MTCNN()  
  
# 检测人脸  
  
detections = face_detetor.detect_faces(img_cvt)  
  
for face in detections:  
  
    (x, y, w, h) = face['box']  
  
    cv2.rectangle(img_cvt, (x, y), (x + w, y + h), (0,255,0), 5)  
  
plt.imshow(img_cvt)
```



# 读取照片

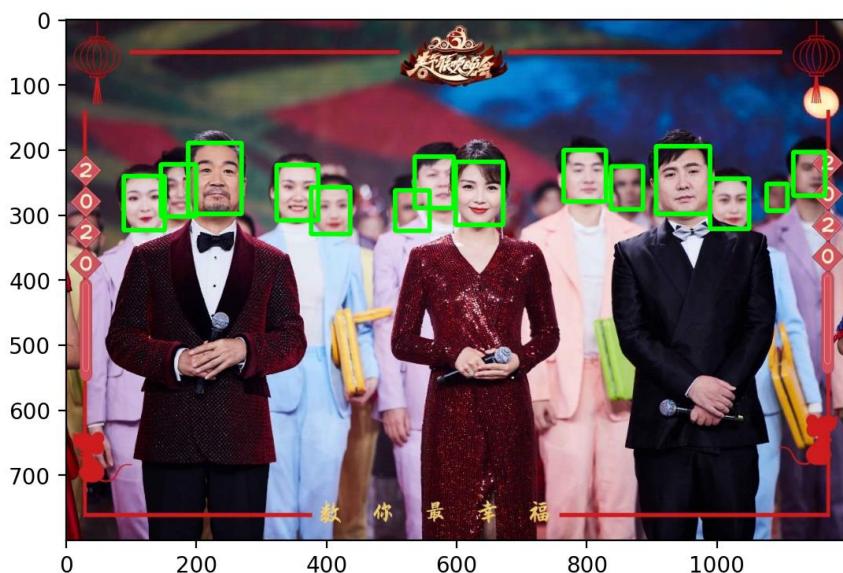
```
img = cv2.imread('./images/test.jpg')  
  
img_cvt = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

# 展示

```
plt.imshow(img_cvt)
```

# 检测人脸

```
detections = face_detetor.detect_faces(img_cvt)  
  
for face in detections:  
  
    (x, y, w, h) = face['box']  
  
    cv2.rectangle(img_cvt, (x, y), (x + w, y + h), (0, 255, 0), 5)  
  
plt.imshow(img_cvt)
```



## 几种算法的对比

	优势	劣势
<b>Haar cascade</b>	速度最快 轻量 (~400KB) 适合算力较小设备	准确度低 偶尔误报 无旋转不变性
<b>HOG + Dlib</b>	比Haar准确率高	速度比Haar低 计算量大 无旋转不变性 Dlib兼容性问题
<b>SSD (opencv DNN)</b>	比Haar 和hog准确率高 使用深度学习 大小一般 (~10MB)	低光照图片准确率低 受肤色影响
<b>CNN (MMOD Dlib)</b>	最准确 误报率低 轻量 (~1MB)	相对于其他方法慢 计算量大 Dlib兼容性问题

- 无旋转不变性：意思是人脸颠倒或者旋转，侧脸的情况下，可能就无法获得识别准确的结果。

## 视频流检测人脸

```
"""
视频流检测人脸：

1、构造 haar 人脸检测器
2、获取视频流
3、检测每一帧画面
4、画人脸框并显示
"""

import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
import time

video = cv2.VideoCapture(0)

#系统启动时间

start_time = time.time()

face_detector =

cv2.CascadeClassifier('./face_detection/cascades/haarcascade_frontalface_default.xml')

while True:

    ret, frame = video.read()

    frame = cv2.flip(frame, 1)

    img_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #计算帧率

    current_time = time.time()

    fps = int(1/(current_time - start_time))

    start_time = current_time

    #设置显示帧率

    font = cv2.FONT_HERSHEY_PLAIN

    frame = cv2.putText(img = frame, text = str(fps), org = (20, 50), fontFace = font,

fontScale = 3, color = (0, 255, 0), thickness = 3, lineType = cv2.LINE_AA)

detections = face_detector.detectMultiScale(img_gray, scaleFactor=1.2, minNeighbors=2)

for(x, y, w, h) in detections:

    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 5)

    cv2.imshow("MengDu Demo", frame)

if cv2.waitKey(10) & 0xFF == ord("q"):

    break

video.release()

cv2.destroyAllWindows()
```

## 经典人脸识别的方法

### EigenFaces (特征脸, 1991 年)

**Eigenfaces** 是一种面部识别的方法，它基于主成分分析（Principal Component Analysis, PCA）来简化和识别面部图像。这种方法由 Sirovich 和 Kirby 在 1987 年提出，并由 Turk 和 Pentland 在 1991 年进一步发展。

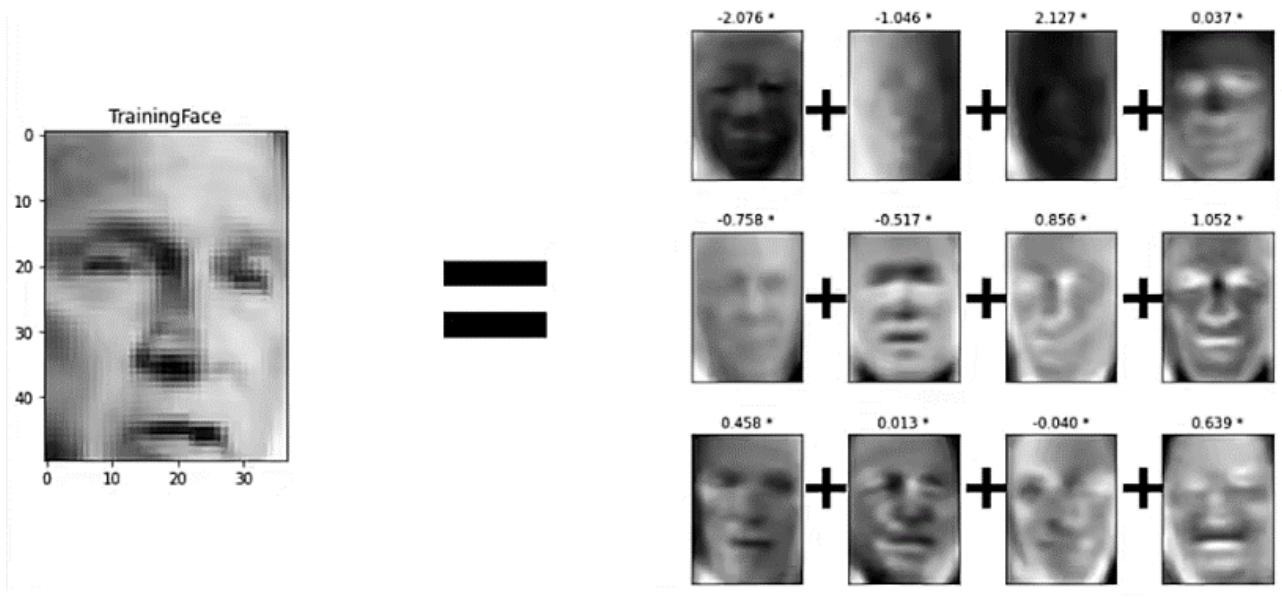
**Eigenfaces** 的方法就像是在创造一个特别的眼镜，戴上它后，我们不看人脸的全部细节，只关注几个最显著的特点来辨认人。这样我们就不会被一些不那么重要的信息（比如背景光线或者微小的表情变化）分心。

这个过程可以想象成以下几步：

1. 整理相册：我们先把所有的脸都剪裁成一样的大小和形状，让它们都能统一比较。
2. 找出平均脸：我们计算出一个“平均脸”——这是相册里所有人脸的平均样子。它代表了这个相册中的“普通”脸孔。
3. 注意脸部差异：之后，我们看每个人脸与这个“平均脸”的区别。这些区别帮助我们注意到每个人独特的特点。
4. 提取关键特征：然后我们挑出几种最能代表脸部差异的特征——这就是所谓的“特征脸”。它们可能包括诸如眼睛大小、脸型、鼻子高度等等。
5. 描述每个脸：通过这些“特征脸”，我们可以用几个数字来描述每个人的脸。就好比用一串密码来代表每个人的外貌。
6. 辨认新脸：最后，当有一个新的人脸出现时，我们也用同样的方式来描述它，并且和我们的相册里的“密码”进行比较，来看看它和哪个最接近，从而识别出这个人是谁。

用这种方法，我们不需要记住每个人脸的所有细节，只要通过几个关键特征就可以快速找到相似的脸。

**Eigenfaces** 方法是一种相对简单且计算高效的面部识别技术，它能够捕捉面部表情、发型、光照等因素的变化。然而，**Eigenfaces** 对姿态和遮挡敏感，如果面部图像的变化不仅仅是因为表情或光照，这种方法的有效性可能会受到限制。尽管如此，**Eigenfaces** 为后来的许多高级面部识别算法奠定了基础。



## FisherFaces (1997 年)

LDA (Linear Discriminant Analysis, 线性判别分析)

如果说 Eigenfaces 是用一种特殊的眼镜来找出人脸中最显著的特征，那么 FisherFaces 就是对这副眼镜进行了升级，让它更擅长区分不同人的脸，即使这些脸在某些特征上看起来非常相似。

想象你有两个朋友，他们看起来非常相似，常常被人混淆。如果我们还用 Eigenfaces 的方法，我们可能只会注意到他们共同的、显著的特征，比如他们都有着非常大的眼睛和宽阔的额头，而忽略了能区分他们的细微特征，比如眼睛的形状和额头的高度。这就是 Eigenfaces 方法的局限性。

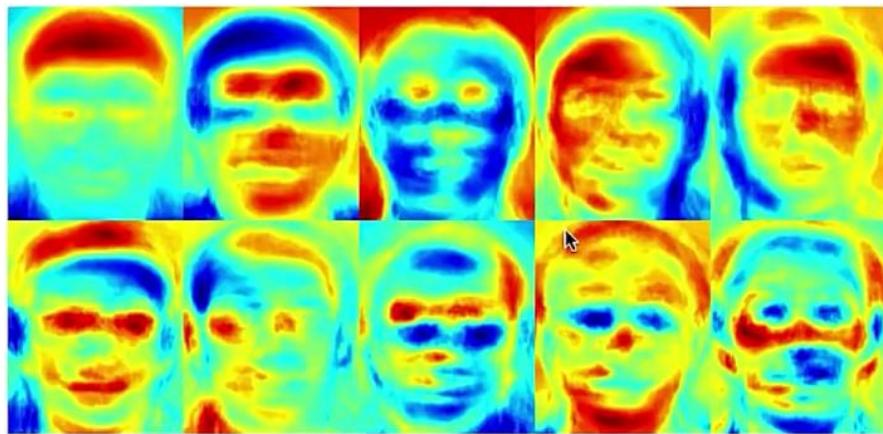
FisherFaces 方法试图改进这一点。它不仅关注人脸的显著特征，还尝试找出那些能够最好地区分人脸的特征。我们可以这样理解：

- 1. 聚焦差异：**FisherFaces 方法首先寻找哪些特征能最好地将人群分开，比如分辨双胞胎，你可能会更关注他们的鼻子而不是他们的眼睛，因为那是他们相异之处。
- 2. 强调类别间差异：**它尝试让不同人之间的差异尽可能明显，即使他们有很多相似的地方。
- 3. 减少同类别内差异：**同时，FisherFaces 方法还尽量减少同一个人在不同照片中的变化对识别的影响，比如表情变化或光线不同。

通过这种方式，FisherFaces 为每个人的脸创建了一个更加专业和精确的“描述密码”，即

使在很多人看起来都有相似特征的情况下，也能更好地区分他们。

简而言之，如果 Eigenfaces 是关于找到脸上最明显的特征，那么 FisherFaces 则是关于找到最能区分不同人脸的特征，即使这些特征不是那么一眼就能看出来的。



## LBPH (Local Binary Pattern Histogram, 1994 年)

LBPH 是局部二值模式直方图 (Local Binary Patterns Histogram) 的缩写，这是一种用于面部识别的纹理描述算法。简单来说，LBPH 方法通过比较每个像素与其周围像素的大小，为面部图像创建一个描述符，从而捕捉图像的局部纹理特征。这种方法对光照变化、面部表情变化以及一定程度的遮挡都相对鲁棒（指某个系统、模型或算法能够在面对错误、变化和意外情况时仍然能够正常工作的能力。简单来说，鲁棒性强意味着某样东西很“坚固”或“健壮”，不容易因为外界的干扰或内部的错误而出现问题）。

下面是 LBPH 的工作原理的简化说明：

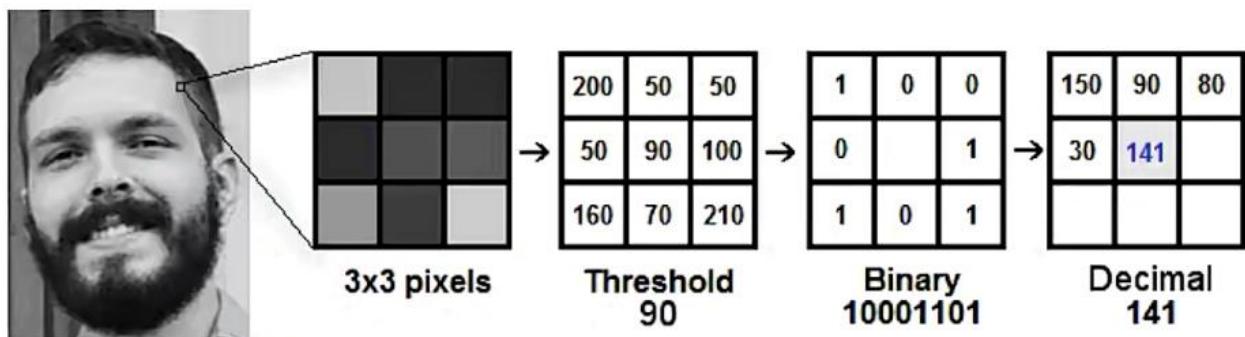
1. 分析局部区域：算法会遍历面部图像中的每个像素，并将该像素与它周围的像素进行比较。
2. 创建二值模式：对于每个像素，算法会检查它的邻居。如果邻居像素的值大于中心像素的值，则邻居被标记为 1，否则为 0。这样，对于中心像素的每个邻居，你都会得到一个 0 或 1 的值。
3. 构建二进制序列：这些 0 和 1 的值按照一定的顺序排列，形成一个二进制数。例如，一个像素周围的八个邻居可以生成一个 8 位的二进制数（如 11100101）。
4. 转换为十进制：将这个二进制数转换成十进制数，这个十进制数就是中心像素的局部二值模式。

5. 计算直方图：通过计算图像中每一个小区域的局部二值模式的直方图，即统计每个十进制数出现的频率，我们可以得到图像的纹理描述。

6. 构建全局描述符：将这些局部直方图连接起来，形成一个全局特征直方图，这个直方图就是整个面部图像的纹理描述符。

7. 面部识别：当需要识别一个新的面部图像时，可以提取它的 **LBPH** 特征，并与数据库中已知的特征进行比较，找到最匹配的特征，从而完成识别。

**LBPH** 方法的优点在于它的简单性和对环境变化的鲁棒性，尤其是对于光照条件的变化。此外，它的计算效率高，容易在不同的系统和设备上实现。因此，**LBPH** 在实际应用中是一个非常流行的面部识别算法。



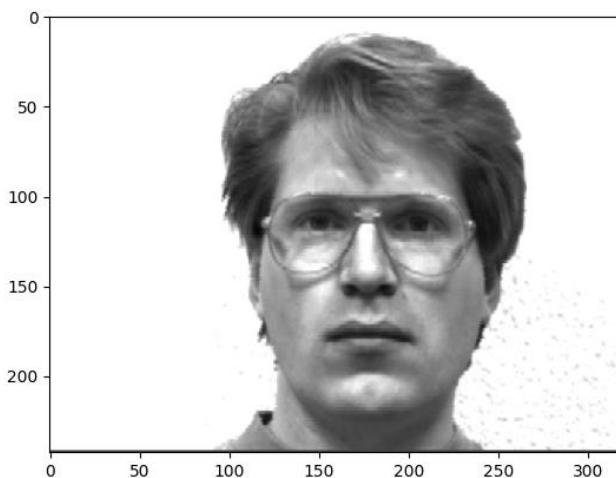
## 代码实现

```
# 步骤
# 1、图片数据预处理
# 2、加载模型
# 3、训练模型
# 4、预测图片
# 5、评估测试数据集
# 6、保存模型
# 7、调用加载模型
# 导入包
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
import dlib  
%matplotlib inline  
  
# 随机选一张图片  
  
img_path = './yalefaces/train/subject01.glasses.gif'  
  
# 读取 GIF 格式图片  
  
cap = cv2.VideoCapture(img_path)  
  
ret,img = cap.read()  
  
img.shape  
  
>(243, 320, 3)
```

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  
  
><matplotlib.image.AxesImage at 0x1eb64c3a2b0>
```



```
# 图片预处理  
  
# img_list: numpy 格式图片  
  
# label_list: numpy 格式的 label  
  
# cls.train(img_list,np.array(label_list))  
  
# 为了减少运算，提高速度，将人脸区域用人脸检测器提取出来  
  
# 构造 hog 人脸检测器（将人脸的区域提取出来）  
  
hog_face_detector = dlib.get_frontal_face_detector()  
  
def getFaceImgLabel(fileName):hog_face_detector = dlib.get_frontal_face_detector()
```

```
def getFaceImgLabel(fileName):
    # 读取图片
    cap = cv2.VideoCapture(fileName)
    ret, img = cap.read()

    # 转为灰度图
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 检测人脸
    detections = hog_face_detector(img, 1)

    # 判断是否有人脸
    if len(detections) > 0:
        # 获取人脸区域坐标
        x = detections[0].left()
        y = detections[0].top()
        r = detections[0].right()
        b = detections[0].bottom()

        # 截取人脸
        img_crop = img[y:b, x:r]

        # 缩放解决冲突
        img_crop = cv2.resize(img_crop, (120, 120))

        # 获取人脸 labelid
        label_id = int(fileName.split('/')[-1].split('.')[0].split('subject')[-1])

        # 返回值
        return img_crop, label_id
    else:
        return None, -1
```

```
img_path = './yalefaces/train/subject01.glasses.gif'
```

# 测试一张图片

```
img,label = getFaceImgLabel(img_path)
```

```
label
```

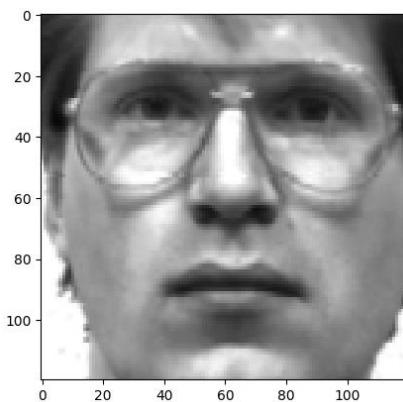
```
>1
```

```
img.shape
```

```
>(120, 120)
```

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_GRAY2RGB))
```

```
><matplotlib.image.AxesImage at 0x1eb64eaec40>
```



# 遍历 train 文件夹，对所有图片同样处理

# 拼接成大的 list

```
import glob

file_list = glob.glob('./yalefaces/train/*')

# 构造两个空列表

img_list = []

label_list = []

for train_file in file_list:

    # 获取每一张图片的对应信息

    img,label = getFaceImgLabel(train_file)
```

```
#过滤数据

if label != -1:

    img_list.append(img)

    label_list.append(label)
```

# 查看 label\_list 大小

```
len(label_list)
```

```
>132
```

# 查看 img\_list 大小

```
len(img_list)
```

```
>132
```

# 构造分类器

```
face_cls = cv2.face.LBPHFaceRecognizer_create()

# cv2.face.EigenFaceRecognizer_create()

# cv2.face.FisherFaceRecognizer_create()
```

# 训练

```
face_cls.train(img_list,np.array(label_list))
```

# 预测一张图片

```
test_file = './yalefaces/test/subject03.glasses.gif'

img,label = getFaceImgLabel(test_file)

#过滤数据

if label != -1:

    predict_id,distance = face_cls.predict(img)

    print(predict_id)

>3
```

# 评估模型

```
file_list =glob.glob('./yalefaces/test/*')

true_list = []

predict_list = []
```

```

for test_file in file_list:

    # 获取每一张图片的对应信息

    img,label = getFaceImgLabel(test_file)

    #过滤数据

    if label != -1:

        predict_id,distance = face_cls.predict(img)

        predict_list.append(predict_id)

        true_list.append(label)

```

# 查看准确率（需要安装 sklearn: conda install scikit-learn）

```

from sklearn.metrics import accuracy_score

accuracy_score(true_list,predict_list)

>0.7241379310344828

```

# 获取融合矩阵

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(true_list,predict_list)
```

```

cm

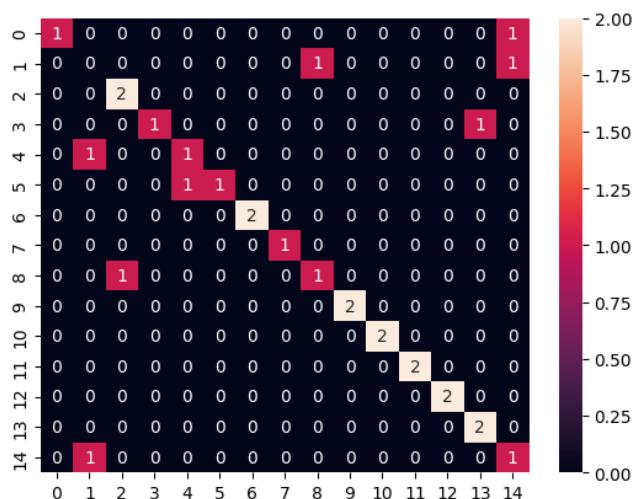
>array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0],

```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0],  
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]], dtype=int64)
```

# 可视化（需要安装 seaborn: conda install -c anaconda seaborn）

```
import seaborn  
  
seaborn.heatmap(cm, annot=True)  
  
><Axes: >
```



# 保存模型

```
face_cls.save('./weights/LBPH.yml')
```

# 调用模型

```
new_cls = cv2.face.LBPHFaceRecognizer_create()  
  
new_cls.read('./weights/LBPH.yml')
```

# 预测一张图片

```
test_file = './yalefaces/test/subject03.glasses.gif'  
  
img,label = getFaceImgLabel(test_file)  
  
# 过滤数据  
  
if label != -1:  
  
    predict_id,distance = new_cls.predict(img)
```

```
print(predict_id)  
>3
```

## 基于残差网络（ResNet）的人脸识别方法

残差网络（Residual Network，简称 ResNet）是一种深度学习架构，于 2015 年由微软研究院的 Kaiming He 等人提出。它通过使用所谓的“残差学习”来解决深度神经网络训练过程中遇到的退化问题（即随着网络深度的增加，网络性能开始饱和甚至下降的现象）。

传统的深度网络通过堆叠层来学习特征，但训练非常深的网络时会遇到两个主要问题：梯度消失和梯度爆炸，这会导致网络难以训练。此外，理论上虽然更深的网络能够学习更复杂的特征表示，但实际上增加网络层数往往会导致性能下降，这是由于退化问题所致。

残差网络通过引入了残差块（residual block）的概念来解决这些问题。残差块允许输入直接跳过一些层传递到更深层的网络中，这种结构被称为“跳跃连接”（skip connection）或“快捷连接”（shortcut connection）。具体来说，残差块的输出是该块内层的输出（学习到的残差）和块输入的总和。

想象一下你有一堆乐高积木，你要建造一个非常高的塔。正常情况下，你可能会一层层地搭建它。但是如果你搭得越高，越容易出错，而且每增加一层都变得更困难。

残差网络（ResNet）就像是在这个乐高塔中间加入了一些魔法的捷径，这些捷径允许你直接从较低的层跳到更高的层，而不是一层层地堆叠上去。这样做好处是，就算你的塔（网络）建得非常高，通过这些捷径，搭建起来就不会那么困难了。

在实际的网络中，这些“捷径”帮助网络不会忘记它在搭建的初期学到的东西，即使它已经搭建得非常复杂和高了。这就是为什么残差网络可以建得很深，但仍然能有效学习和改进其性能的原因。

ResNet 的这种设计允许训练非常深的神经网络，有助于网络学习更复杂的特征表示，从而在多项视觉识别任务上取得了突破性的性能提升。ResNet 的变种，如 ResNet50、ResNet101 和 ResNet152（数字代表层的数量），已经广泛应用于图像分类、目标检测和语义分割等领域。

# 步骤

# 1、图片数据预处理

# 2、加载模型  
# 3、提取图片的特征描述符  
# 4、预测图片：找到欧氏距离最近的特征描述符  
# 5、评估测试数据集  
# 导入包

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

import dlib

# %matplotlib inline

plt.rcParams['figure.dpi'] = 200
```

# 获取人脸的 68 个关键点

# 人脸检测模型

```
hog_face_detector = dlib.get_frontal_face_detector()
```

# 关键点 检测模型

```
shape_detector = dlib.shape_predictor('./weights/shape_predictor_68_face_landmarks.dat')
```

# 读取一张测试图片

```
img = cv2.imread('./images/faces2.jpg')
```

# 检测人脸

```
detections = hog_face_detector(img, 1)

for face in detections:

    # 人脸框坐标

    l, t, r, b = face.left(), face.top(), face.right(), face.bottom()

    # 获取 68 个关键点

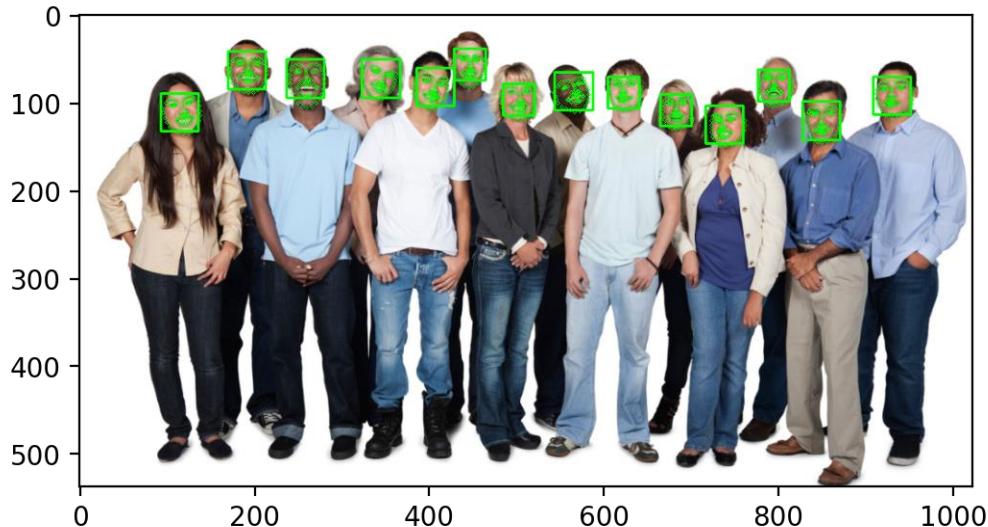
    points = shape_detector(img, face)
```

# 绘制关键点

```
for point in points.parts():

    cv2.circle(img, (point.x, point.y), 2, (0, 255, 0), 1)
```

```
# 绘制矩形框  
cv2.rectangle(img, (l,t), (r,b), (0,255,0), 2)  
  
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  
  
> <matplotlib.image.AxesImage at 0x2ef2b283a90>
```



## # 面部特征描述符

```
# 人脸检测模型  
hog_face_detector = dlib.get_frontal_face_detector()  
  
# 关键点 检测模型  
shape_detector = dlib.shape_predictor('./weights/shape_predictor_68_face_landmarks.dat')  
  
# resnet 模型  
face_descriptor_extractor =  
  
dlib.face_recognition_model_v1('./weights/dlib_face_recognition_resnet_model_v1.dat')
```

## # 提取单张图片的特征描述符, label

```
def getFaceFeatLabel(fileName):  
  
    # 获取人脸 labelid  
    label_id = int(fileName.split('/')[-1].split('.')[0].split('subject')[-1])  
  
    # 读取图片  
    cap = cv2.VideoCapture(fileName)  
    ret, img = cap.read()  
  
    # 转为 RGB
```

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#人脸检测

detections = hog_face_detector(img, 1)

face_descriptor = None

for face in detections:

    # 获取关键点

    points = shape_detector(img, face)

    # 获取特征描述符

    face_descriptor = face_descriptor_extractor.compute_face_descriptor(img, points)

    # 转为 numpy 格式的数组

    face_descriptor = [f for f in face_descriptor]

    face_descriptor = np.asarray(face_descriptor, dtype=np.float64)

    face_descriptor = np.reshape(face_descriptor, (1, -1))

    return label_id, face_descriptor
```

# 测试一张图片

```
id1,fd1 = getFaceFeatLabel('./yalefaces/train/subject01.leftlight.gif')
```

```
fd1.shape
> (1, 128)
```

# 对 train 文件夹进行处理

```
import glob

file_list = glob.glob('./yalefaces/train/*')

# 构造两个空列表

label_list = []
```

```
feature_list = None

name_list = {}

index= 0

for train_file in file_list:

    # 获取每一张图片的对应信息

    label,feat = getFaceFeatLabel(train_file)

    #过滤数据

    if feat is not None:

        #文件名列表

        name_list[index] = train_file

        #label 列表

        label_list.append(label)

        # 特征列表

        if feature_list is None:

            feature_list = feat

        else:

            feature_list = np.concatenate((feature_list,feat),axis=0)

        index +=1
```

```
len(label_list)
```

```
>132
```

```
feature_list.shape
```

```
>(132, 128)
```

```
len(name_list)
>132
```

```
name_list
>{0: './yalefaces/train\\subject01.glasses.gif',
1: './yalefaces/train\\subject01.leftlight.gif',
2: './yalefaces/train\\subject01.noglasses.gif',
3: './yalefaces/train\\subject01.normal.gif',
4: './yalefaces/train\\subject01.sad.gif',
...}
```

```
feature_list[100]
>array([-0.04002353,  0.1270756 ,  0.0712276 , -0.00709001, -0.06835698,
       -0.05430241, -0.01340284, -0.03349683,  0.21279033, -0.02416467,
       0.23374976, -0.02222661, -0.24263582,  0.01428987, -0.11278771,
       0.17095208, -0.07398933, -0.06234868, -0.1701569 , -0.12964757,
       0.00911354,  0.12395205,  0.00404465, -0.04684508, -0.1459292 ,
       -0.23717026, -0.11186991, -0.06274647,  0.16915584, -0.06881971,
       0.05980719,  0.09719942, -0.17893974, -0.07927143,  0.10608829,
       0.00437263, -0.04056147, -0.06083017,  0.25581253,  0.15630315,
       -0.13259435,  0.10182938, -0.00568939,  0.34255689,  0.12212187,
       0.003549 ,  0.06189087, -0.10344566,  0.04815118, -0.216341 ,
       0.04030668,  0.18290767,  0.06320614,  0.08376811,  0.15554932,
       -0.17170465,  0.01758712,  0.18572752, -0.24583155,  0.13962844,
       0.10026111,  0.00446303, -0.11024437, -0.07532805,  0.07948148,
       0.05880733, -0.10819434, -0.1403235 ,  0.26048198, -0.12129794,
       0.06681733,  0.1600437 , -0.09273494, -0.15797757, -0.22543158,
       0.08716235,  0.4466396 ,  0.16455799, -0.15942498, -0.01008424,
```

```

0.00993463, -0.02114565, 0.04819157, -0.01039814, -0.11212274,
-0.06056442, -0.00080675, 0.03041239, 0.158804, 0.04165056,
-0.03373896, 0.24092659, 0.01626588, -0.069848, -0.01487511,
0.00900624, -0.0908756, -0.11121298, -0.09889594, -0.03088601,
0.02986297, -0.18065356, -0.01414195, 0.09142634, -0.15214048,
0.13194977, 0.01288422, -0.06848911, -0.07442028, 0.09353367,
-0.09857555, 0.06099321, 0.23246332, -0.24601872, 0.27148089,
0.18788871, -0.01543176, 0.0421946, 0.07114153, 0.06067063,
-0.00306126, -0.02650717, -0.06876908, -0.1984171, -0.06596848,
-0.08234069, 0.01143895, -0.00508888])

```

# 计算距离（判断是否是同一个人，距离越小，越可能是同一个人）

```
np.linalg.norm((feature_list[100]-feature_list[100]))  
>0.0
```

# 计算距离

```
np.linalg.norm((feature_list[100]-feature_list[101]))  
>0.2683925464620862
```

# 计算距离

```
np.linalg.norm((feature_list[100]-feature_list[112]))  
>0.8267784171824537
```

# 计算距离

```
np.linalg.norm((feature_list[100]-feature_list[96]))  
  
>0.384012342713569
```

```
# 计算一个特征描述符与所有特征的距离
```

```
np.linalg.norm((feature_list[0]-feature_list),axis=1)

>array([0.           , 0.48992029, 0.37211911, 0.35581028, 0.51543908,
       0.48151447, 0.54761613, 0.49566379, 0.6950431 , 0.77329649,
       0.73359842, 0.73359842, 0.73421626, 0.73696717, 0.73482733,
       0.74683863, 0.76282718, 0.85345605, 0.76441795, 0.80798221,
```

```

0.80798221, 0.77109683, 0.78844059, 0.80426717, 0.76877739,
0.78156313, 0.94512002, 0.89450371, 0.91552612, 0.92289976,
0.90163334, 0.92874091, 0.90163334, 0.90006939, 0.8960838 ,
0.7656308 , 0.73200037, 0.7655796 , 0.74759713, 0.73759675,
0.75155068, 0.77416643, 0.75906315, 0.83163383, 0.83925553,
0.86775178, 0.86775178, 0.89771323, 0.86838225, 0.88262811,
0.82586691, 0.89110904, 0.88370862, 0.76782481, 0.90608703,
0.90608703, 0.89297016, 0.90314584, 0.88692007, 0.89899775,
0.8934966 , 0.88344023, 0.88311846, 0.85766729, 0.84975744,
0.93234985, 0.85751841, 0.87528151, 0.83665292, 0.87857221,
0.83686733, 0.7253908 , 0.82191077, 0.78971114, 0.81990323,
0.81990323, 0.79070875, 0.79465351, 0.7997662 , 0.74267065,
0.85230701, 0.87127568, 0.83334428, 0.8165271 , 0.84331366,
0.82608128, 0.81389648, 0.85623805, 0.84103467, 0.90428896,
0.79680514, 0.77829374, 0.78373383, 0.79026002, 0.74887283,
0.77050444, 0.78605625, 0.76262044, 0.80356235, 0.82377045,
0.78390882, 0.79702586, 0.7939804 , 0.79998088, 0.76707686,
0.89655258, 0.75343013, 0.77418599, 0.8418776 , 0.89640526,
0.75625989, 0.82625187, 0.77509894, 0.80697659, 0.91916941,
0.86093399, 0.87199567, 0.91029299, 0.88791261, 0.88136855,
0.88991523, 0.88505397, 0.93438128, 0.81318609, 0.7887356 ,
0.80851345, 0.87765133, 0.82736927, 0.82647742, 0.76562843,
0.81180428, 0.81094532])

```

# 计算一个特征描述符与所有特征的距离（排除自己）

```

np.linalg.norm((feature_list[0]-feature_list[1:]),axis=1)

>array([0.48992029, 0.37211911, 0.35581028, 0.51543908, 0.48151447,
       0.54761613, 0.49566379, 0.6950431 , 0.77329649, 0.73359842,
       0.73359842, 0.73421626, 0.73696717, 0.73482733, 0.74683863,

```

```

0.76282718, 0.85345605, 0.76441795, 0.80798221, 0.80798221,
0.77109683, 0.78844059, 0.80426717, 0.76877739, 0.78156313,
0.94512002, 0.89450371, 0.91552612, 0.92289976, 0.90163334,
0.92874091, 0.90163334, 0.90006939, 0.8960838 , 0.7656308 ,
0.73200037, 0.7655796 , 0.74759713, 0.73759675, 0.75155068,
0.77416643, 0.75906315, 0.83163383, 0.83925553, 0.86775178,
0.86775178, 0.89771323, 0.86838225, 0.88262811, 0.82586691,
0.89110904, 0.88370862, 0.76782481, 0.90608703, 0.90608703,
0.89297016, 0.90314584, 0.88692007, 0.89899775, 0.8934966 ,
0.88344023, 0.88311846, 0.85766729, 0.84975744, 0.93234985,
0.85751841, 0.87528151, 0.83665292, 0.87857221, 0.83686733,
0.7253908 , 0.82191077, 0.78971114, 0.81990323, 0.81990323,
0.79070875, 0.79465351, 0.7997662 , 0.74267065, 0.85230701,
0.87127568, 0.83334428, 0.8165271 , 0.84331366, 0.82608128,
0.81389648, 0.85623805, 0.84103467, 0.90428896, 0.79680514,
0.77829374, 0.78373383, 0.79026002, 0.74887283, 0.77050444,
0.78605625, 0.76262044, 0.80356235, 0.82377045, 0.78390882,
0.79702586, 0.7939804 , 0.79998088, 0.76707686, 0.89655258,
0.75343013, 0.77418599, 0.8418776 , 0.89640526, 0.75625989,
0.82625187, 0.77509894, 0.80697659, 0.91916941, 0.86093399,
0.87199567, 0.91029299, 0.88791261, 0.88136855, 0.88991523,
0.88505397, 0.93438128, 0.81318609, 0.7887356 , 0.80851345,
0.87765133, 0.82736927, 0.82647742, 0.76562843, 0.81180428,
0.81094532])

```

## # 寻找最小值索引

```

np.argmin(np.linalg.norm((feature_list[0]-feature_list[1:]),axis=1))

>2

```

```
np.linalg.norm((feature_list[0]-feature_list[1:]),axis=1)[2]  
>0.35581028298402767
```

```
name_list[1+2]  
> './yalefaces/train\\subject01.normal.gif'
```

```
np.linalg.norm((feature_list[0]-feature_list[3]))  
>0.35581028298402767
```

## # 评估测试数据集

```
file_list = glob.glob('./yalefaces/test/*')  
# 构造两个空列表  
predict_list = []  
label_list= []  
# 距离阈值  
threshold = 0.5  
  
for test_file in file_list:  
    # 获取每一张图片的对应信息  
    label,feat = getFaceFeatLabel(test_file)  
  
    # 读取图片  
    cap = cv2.VideoCapture(test_file)  
    ret,img = cap.read()  
  
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)  
  
    #过滤数据  
    if feat is not None:
```

```
# 计算距离

distances = np.linalg.norm((feat-feature_list),axis=1)

min_index = np.argmin(distances)

min_distance = distances[min_index]

if min_distance < threshold:
    # 同一人

    predict_id = int(name_list[min_index].split('/')[-
1].split('.')[0].split('subject')[-1])

else:
    predict_id = -1

predict_list.append(predict_id)

label_list.append(label)

cv2.putText(img,'True:'+str(label),(10,30),cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,255,0))

cv2.putText(img,'Pred:'+str(predict_id),(10,50),cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,255,0))

cv2.putText(img,'Dist:'+str(min_distance),(10,70),cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,255,
0))

# 显示

plt.figure()

plt.imshow(img)
```



# 公式评估

```
from sklearn.metrics import accuracy_score  
  
accuracy_score(label_list,predict_list)  
  
>1.0
```

## 人脸考勤机代码实现

```
"""  
人脸考勤  
  
人脸注册：将人脸特征存进 feature.csv  
  
人脸识别：将检测的人脸特征与 csv 中人脸特征作比较，如果比中的把考勤记录写入 attendance.csv  
"""  
  
# 导入包  
  
import cv2  
  
import numpy as np  
  
import dlib  
  
import time  
  
import csv
```

```
# 人脸注册方法

def faceRegister(label_id=1, name='enpei', count=3, interval=3):
    """
    label_id: 人脸 ID
    Name: 人脸姓名
    count: 采集数量
    interval: 采集间隔时间
    """

# 检测人脸

# 获取 68 个关键点

# 获取特征描述符

cap = cv2.VideoCapture(0)

# 获取长宽

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# 构造人脸检测器

hog_face_detector = dlib.get_frontal_face_detector()

# 关键点检测器

shape_detector =
    dlib.shape_predictor('./weights/shape_predictor_68_face_landmarks.dat')

# 特征描述符

face_descriptor_extractor =
    dlib.face_recognition_model_v1('./weights/dlib_face_recognition_resnet_model_v1.dat')
```

```
# 开始时间
start_time = time.time()

# 执行次数
collect_count = 0

# CSV Writer
f = open('./data/feature.csv', 'a', newline="")
csv_writer = csv.writer(f)

while True:
    ret, frame = cap.read()

    # 缩放
    frame = cv2.resize(frame, (width//2, height//2))

    # 镜像
    frame = cv2.flip(frame, 1)

    # 转为灰度图
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # 检测人脸
    detections = hog_face_detector(frame, 1)

    # 遍历人脸
    for face in detections:

        # 人脸框坐标
        l, t, r, b = face.left(), face.top(), face.right(), face.bottom()

        # 获取人脸关键点
        points = shape_detector(frame, face)

        for point in points.parts():

            cv2.circle(frame, (point.x, point.y), 2, (0, 255, 0), -1)
```

```
# 矩形人脸框
cv2.rectangle(frame, (l,t), (r,b), (0,255,0),2)

# 采集:
if collect_count < count:

    # 获取当前时间
    now = time.time()

    # 时间间隔
    if now -start_time > interval:

        # 获取特征描述符
        face_descriptor =
face_descriptor_extractor.compute_face_descriptor(frame,points)

        # 转为列表
        face_descriptor = [f for f in face_descriptor]

        # 写入 CSV 文件
        line = [label_id, name, face_descriptor]
        csv_writer.writerow(line)

        collect_count +=1

        start_time = now

        print("采集次数: {collect_count}".format(collect_count= collect_count))

    else:
        pass

else:
    # 采集完毕
    print('采集完毕')

    return

# 显示画面
cv2.imshow('Face attendance',frame)

# 退出条件
```

```
if cv2.waitKey(10) & 0xFF == ord('q'):  
    break  
  
f.close()  
  
cap.release()  
  
cv2.destroyAllWindows()  
  
# 获取并组装 CSV 文件中特征  
  
def getFeatureList():  
    # 构造列表  
  
    label_list = []  
  
    name_list = []  
  
    feature_list = None  
  
  
    with open('./data/feature.csv', 'r') as f:  
        csv_reader = csv.reader(f)  
  
        for line in csv_reader:  
            label_id = line[0]  
  
            name = line[1]  
  
            label_list.append(label_id)  
  
            name_list.append(name)  
  
            # string 转为 list  
  
            face_descriptor = eval(line[2])  
  
            #  
  
            face_descriptor = np.asarray(face_descriptor, dtype=np.float64)  
  
            face_descriptor = np.reshape(face_descriptor, (1, -1))  
  
  
            if feature_list is None:
```

```
feature_list = face_descriptor

else:

    feature_list = np.concatenate((feature_list, face_descriptor), axis=0)

return label_list, name_list, feature_list

# 人脸识别

# 1、实时获取视频流中人脸的特征描述符

# 2、将它与库里特征做距离判断

# 3、找到预测的 ID、NAME

# 4、考勤记录存进 csv 文件：第一次识别到存入或者隔一段时间存

def faceRecognizer(threshold = 0.5):

    cap = cv2.VideoCapture(0)

    # 获取长宽

    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # 构造人脸检测器

    hog_face_detector = dlib.get_frontal_face_detector()

    # 关键点检测器

    shape_detector =

        dlib.shape_predictor('./weights/shape_predictor_68_face_landmarks.dat')

    # 特征描述符

    face_descriptor_extractor =

        dlib.face_recognition_model_v1('./weights/dlib_face_recognition_resnet_model_v1.dat')
```

```
# 读取特征
label_list,name_list,feature_list = getFeatureList()

# 字典记录人脸识别记录
recog_record = {}

# CSV 写入
f = open('./data/attendance.csv','a',newline="")
csv_writer = csv.writer(f)

# 帧率信息
fps_time = time.time()

while True:
    ret,frame = cap.read()

    # 缩放
    frame = cv2.resize(frame,(width//2,height//2))

    # 镜像
    frame = cv2.flip(frame,1)

    # 检测人脸
    detections = hog_face_detector(frame,1)

    # 遍历人脸
    for face in detections:

        # 人脸框坐标
        l,t,r,b = face.left(),face.top(),face.right(),face.bottom()

        # 获取人脸关键点
```

```
points = shape_detector(frame,face)

# 矩形人脸框

cv2.rectangle(frame,(l,t),(r,b),(0,255,0),2)

# 获取特征描述符

face_descriptor =

face_descriptor_extractor.compute_face_descriptor(frame,points)

# 转为列表

face_descriptor = [f for f in face_descriptor]

# 计算与库的距离

face_descriptor = np.asarray(face_descriptor,dtype=np.float64)

distances = np.linalg.norm((face_descriptor-feature_list),axis=1)

# 最短距离索引

min_index = np.argmin(distances)

# 最短距离

min_distance = distances[min_index]

if min_distance < threshold:

    predict_id = label_list[min_index]

    predict_name = name_list[min_index]

    cv2.putText(frame,predict_name + 

str(round(min_distance,2)),(l,b+40),cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,255,0),1)

    now = time.time()

    need_insert = False

    # 判断是否识别过

    if predict_name in recog_record:

        # 存过

        # 隔一段时间再存

        if now - recog_record[predict_name] > 3:

            # 超过阈值时间，再存一次
```

```
need_insert =True

recog_record[predict_name] = now

else:

    # 还没到时间

    pass

need_insert =False

else:

    # 没有存过

    recog_record[predict_name] = now

    # 存入 csv 文件

    need_insert =True


if need_insert :

    time_local = time.localtime(recog_record[predict_name])

    # 转换格式

    time_str = time.strftime("%Y-%m-%d %H:%M:%S",time_local)

    line = [predict_id,predict_name,min_distance,time_str]

    csv_writer.writerow(line)

    print('{time}: 写入成功:{name}'.format(name =predict_name,time = time_str))

else:

    print('未识别')


# 计算帧率

now = time.time()

fps = 1/(now - fps_time)

fps_time = now

cv2.putText(frame,"FPS:
```

```
" + str(round(fps, 2)), (20, 40), cv2.FONT_HERSHEY_COMPLEX_SMALL, 2, (0, 255, 0), 1)

# 显示画面
cv2.imshow('Face attendance', frame)

# 退出条件
if cv2.waitKey(10) & 0xFF == ord('q'):
    break

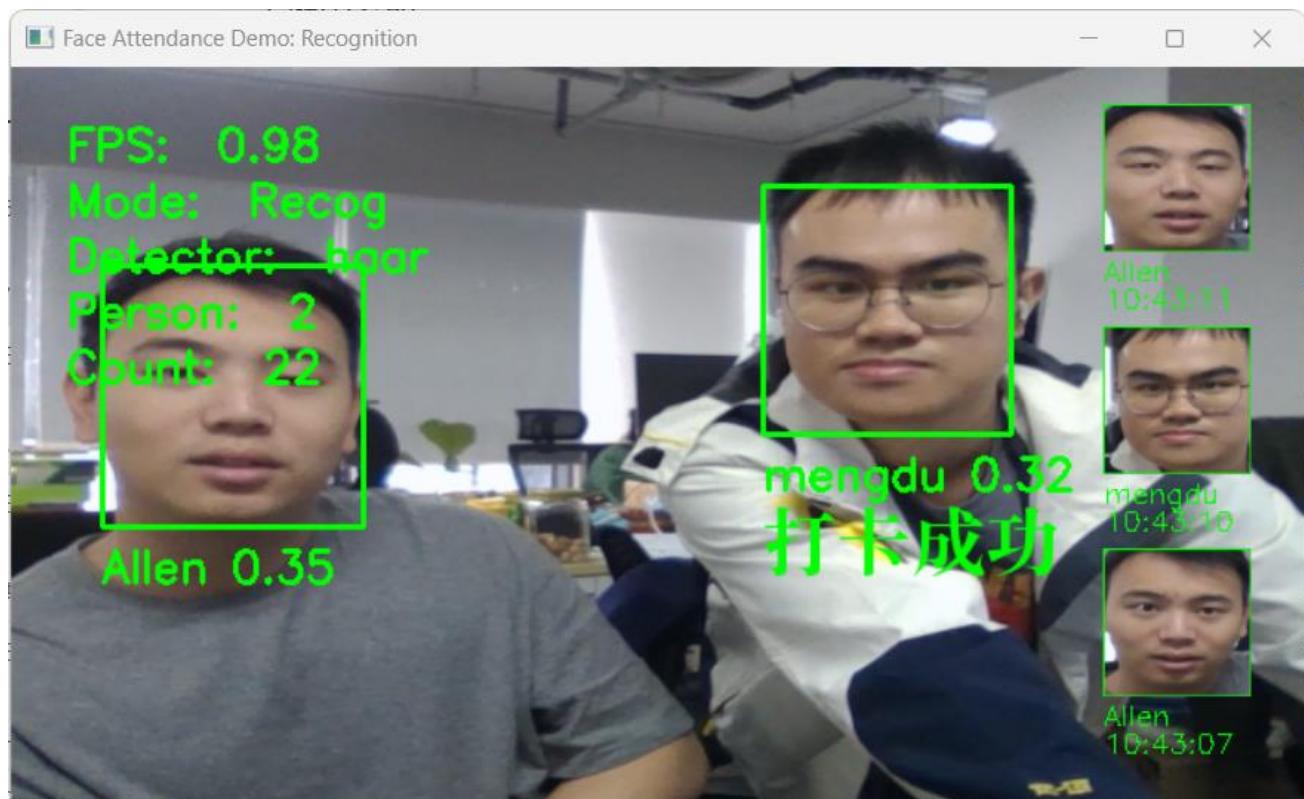
f.close()

cap.release()

cv2.destroyAllWindows()

# faceRegister(label_id=1, name='enpei', count=3, interval=3)

# faceRecognizer(threshold = 0.5)
```



By 黎孟度