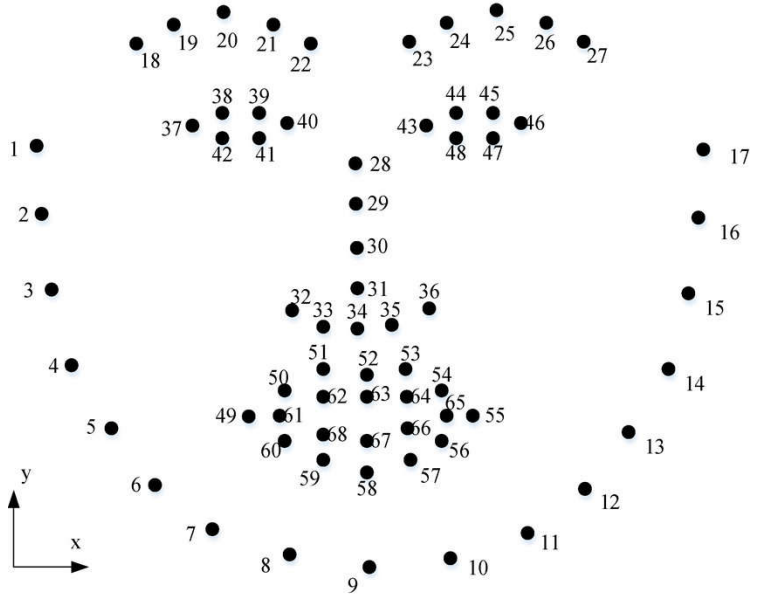
Python编程与人工智能实践

应用篇：基于Dlib的人脸与关键点检测

于泓
鲁东大学
信息与电气工程学院
2021.4.11

Dlib

- Dlib 也是一种比较常用的人脸检测工具。 Dlib 本身是一个用 C++编写的机器学习和信号处理相关的开源工具包，里面包含了很多人工智能相关理论的实现方法。人脸检测与识别是其中的一个子功能。利用其提供的预训练模型，除了可以检测人脸位置外，还可以对人脸中的 68 个标志点（landmarks）进行检测。



Dlib 安装

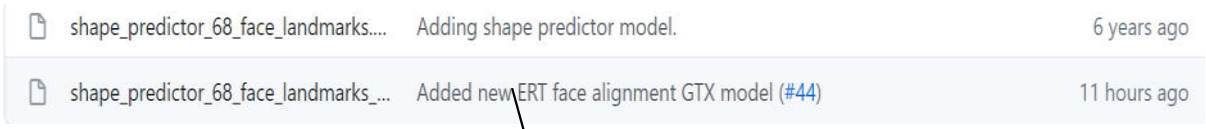
由于Dlib是C++函数库，因此在 window10 环境下安装时需要先安装Visual stdio 2019 (安装免费的社区版即可)，用来对C++代码进行编译。

然后依次安装下列python包：

```
pip install scipy
pip install scikit-image
pip install cmake
pip install dlib
```

安装时会进行自动编译
因此时间稍长。

安装完毕后需要下载预训练模型：
https://github.com/davisking/dlib-models



下载完毕后是一个压缩文件
需要进行解压

```python
import cv2
import dlib
import numpy as np

if __name__ == "__main__":
    # 创建人脸检测器
    det_face = dlib.get_frontal_face_detector()

    # 加载标志点检测器
    det_landmark = dlib.shape_predictor("shape_predictor_68_face_landmarks_GTX.dat")  # 68点

    # 打开摄像头
    cap = cv2.VideoCapture(0)

    t_left_eye = cv2.imread("left-eye.bmp")
    t_right_eye = cv2.imread("right-eye.bmp")
    center_eye = cv2.imread("center-eye.bmp")
    img_glasses = cv2.imread("glasses.bmp")
    img_hat1 = cv2.imread("hat1.bmp")
    img_hat2 = cv2.imread("hat2.bmp")

    # 显示脸部框与 68个关键点
    flag_base = 1

    # 1:眼睛 2: 卡通眼  3: 中心眼
    flag_eyes = 0

    # 1:  帽子1  2: 帽子2
    flag_hat = 0
```

人脸检测器

关键点检测器

```
while True:
    # 读取一帧图像
    success, img = cap.read()

    # 转换为灰度
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 检测人脸区域
    face_rects = det_face(gray, 0)

    for ret in face_rects:
        # 标志点检测
        landmarks = det_landmark(gray, ret)

        parts = landmarks.parts()

        if flag_base==1:
            # 画出人脸区域
            cv2.rectangle(img, (ret.left(),ret.top()), (ret.right(),ret.bottom()), (255, 0, ), 3)
            # 画出 关键点
            for part in landmarks.parts():
                pt = (part.x,part.y)
                cv2.circle(img, pt, 2, (0,0,255),-1)
```

检测人脸区域
返回 rect 列表

利用内置方法
获取人脸位置

在一个
人脸区域
内计算
关键点

得到关键点坐标

```
if flag_eyes ==1:
    img = add_glasses(img,img_glasses,parts)
elif flag_eyes ==2:
    img = add_cartoon_eye(img,t_left_eye,t_right_eye,parts)
elif flag_eyes ==3:
    img = add_cnter_eye(img,center_eye,parts)

if flag_hat ==1:
    img = add_hat(img,img_hat1,parts)
elif flag_hat ==2:
    img= add_hat(img,img_hat2,parts)
```

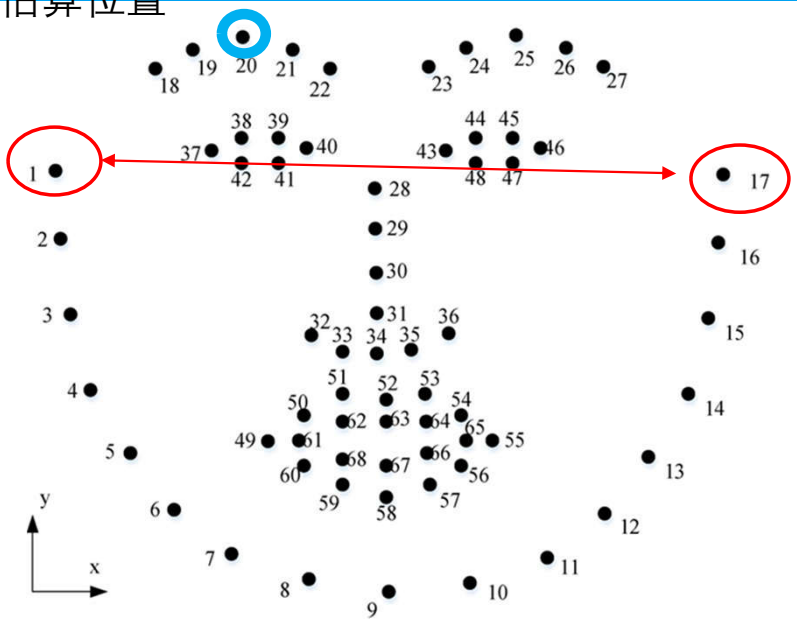
按键切换不同的显示方式

```
# 显示检测结果
cv2.imshow("Face",img)

key = cv2.waitKey(1) & 0xFF

# 按q退出
if key == ord('q'):
    break

# 按 b 切换基本显示
if key == ord('b'):
    key  = 0
    if flag_base==1:
        flag_base =0
    else:
        flag_base = 1


# 按e切换眼睛的显示方式
if key == ord('e'):
    key  = 0
    if flag_eyes==3:
        flag_eyes = 0
    else:
        flag_eyes = flag_eyes+1
        flag_base = 0
```

```
# 按h 切换帽子
if key == ord('h'):
    key  = 0
    if flag_hat==2:
        flag_hat = 0
    else:
        flag_hat = flag_hat+1
```

添加帽子

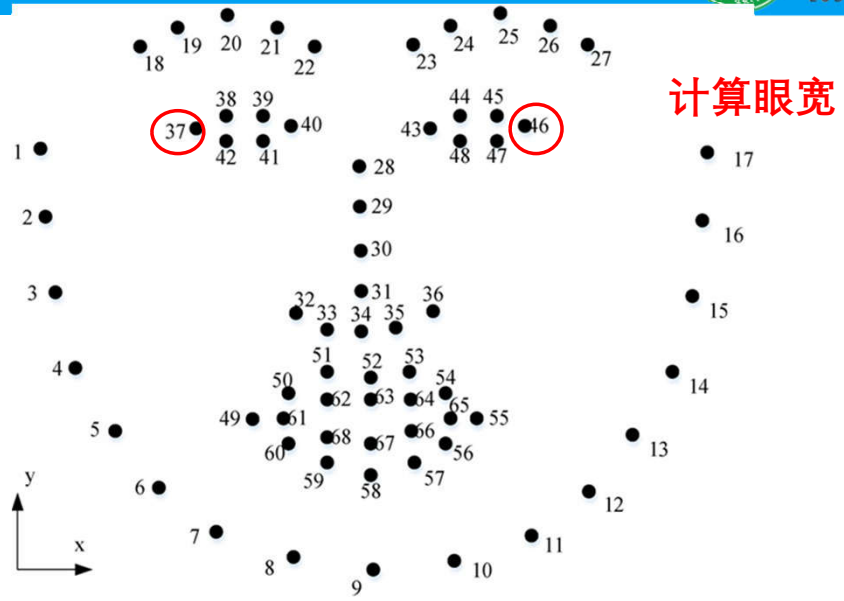
估算位置



计算脸宽

```
def add_hat(img,img_hat,parts):


    # 获取帽子图像大小
    w_hat = np.shape(img_hat)[1]
    h_hat = np.shape(img_hat)[0]

    # 计算脸的宽度
    face_w = int(parts[16].x - parts[0].x)

    # 计算缩放尺度
    scale = face_w/w_hat

    # 帽子图像缩放
    resize_hat = cv2.resize(img_hat,(int(w_hat*scale*(1.2)),int(h_hat*scale*(1.2))))

    # 计算帽子图像的起始位置(左上坐标)
    pos_hat = (parts[0].x-int(face_w*0.1), max(0,int(parts[19].y-resize_hat.shape[0])))

    # 图像叠加
    img_out = img_overlayer(img,resize_hat,pos_hat,bk_fg=255)
    return img_out
```

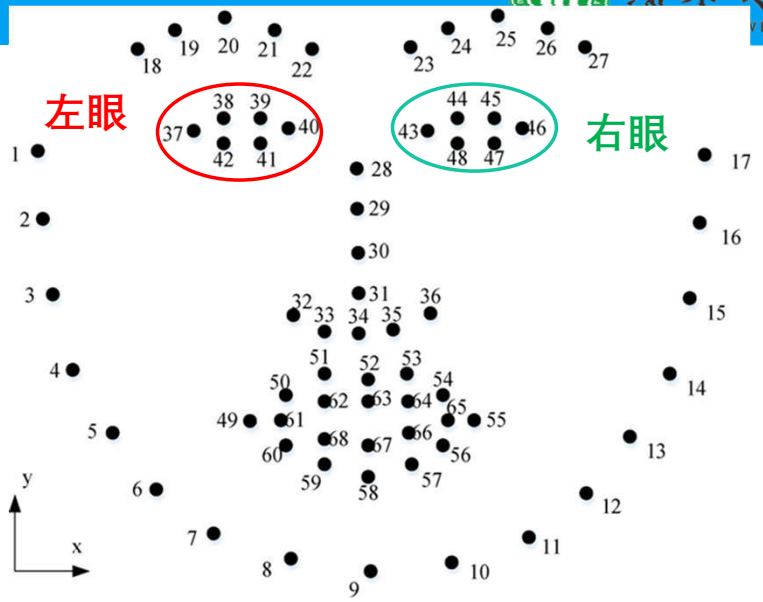
添加眼镜




```
def add_glasses(img,img_glasses,parts):

    # 获取眼镜图像大小
    w_glass = np.shape(img_glasses)[1]
    h_glass = np.shape(img_glasses)[0]

    # 计算缩放尺度
    scale = np.abs(parts[36].x-5 -parts[45].x-5)/w_glass

    # 眼镜图像缩放
    resize_glasses = cv2.resize(img_glasses,(int(w_glass*scale),int(h_glass*scale)))

    # 计算眼镜图像的起始位置(左上坐标)
    pos_glass = (parts[36].x-5,parts[36].y-int(h_glass*scale/2.0))

    # 图像叠加
    img_out = img_overlayer(img,resize_glasses,pos_glass,bk_fg=255)
    return img_out
```

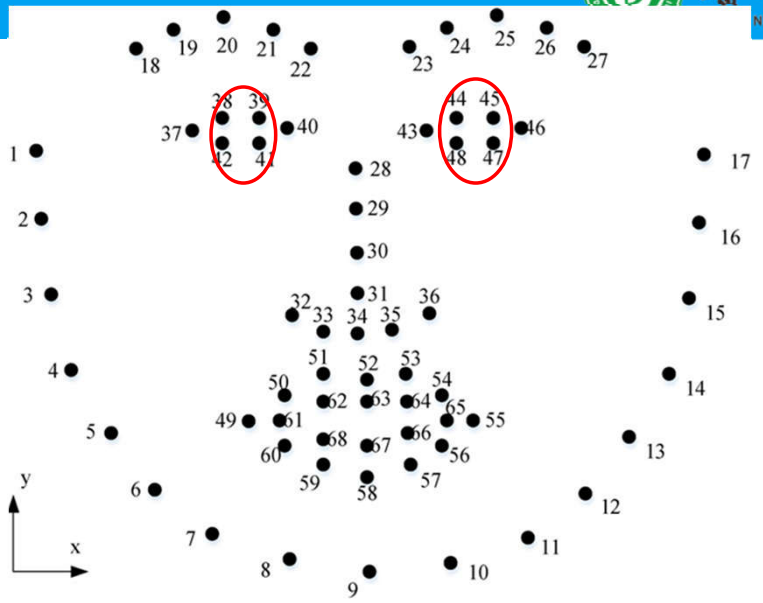
左眼

右眼



添加卡通眼

```
def add_cartoon_eye(img,img_left_eye,img_right_eye,parts):
    # 计算左眼的区域
    pos_left = parts[36].x-3
    pos_up = min(parts[37].y,parts[38].y)-3
    pos_right = parts[39].x+3
    pos_down = max(parts[40].y,parts[41].y)+3

    img_left_eye_overlayer = cv2.resize(img_left_eye,(pos_right-pos_left,pos_down-pos_up))
    img = img_overlayer(img,img_left_eye_overlayer,(pos_left,pos_up),bk_fg=0)
    # 计算右眼的区域
    pos_left = parts[42].x-3
    pos_up = min(parts[43].y,parts[44].y)-3
    pos_right = parts[45].x+3
    pos_down = max(parts[46].y,parts[47].y)+3

    img_right_eye_overlayer = cv2.resize(img_right_eye,(pos_right-pos_left,pos_down-pos_up))

    img = img_overlayer(img,img_right_eye_overlayer,(pos_left,pos_up),bk_fg=0)
    return img
```



添加中心眼

```
def add_cnter_eye(img,center_eye,parts):
    # 计算左眼的区域
    pos_left = min(parts[37].x,parts[41].x)-3
    pos_right = max(parts[38].x,parts[40].x)+3
    scale = np.abs((pos_right-pos_left))/center_eye.shape[1]

    img_center_eye_overlayer = cv2.resize(center_eye,(int(center_eye.shape[0]*scale),int(center_eye.shape[1]*scale)
    img = img_overlayer(img,img_center_eye_overlayer,(pos_left,parts[37].y-3),bk_fg=255)

    # 计算右眼的区域
    pos_left = min(parts[43].x,parts[47].x)-3
    pos_right = max(parts[44].x,parts[46].x)+3
    scale = np.abs((pos_right-pos_left))/center_eye.shape[1]

    img_center_eye_overlayer = cv2.resize(center_eye,(int(center_eye.shape[0]*scale),int(center_eye.shape[1]*scale)
    img = img_overlayer(img,img_center_eye_overlayer,(pos_left,parts[43].y-3),bk_fg=255)
    return img
```

其他

- 根据眼睛的夹角，计算是否闭眼
- 根据嘴巴的张合判断是否打哈欠