CS159：程序设计
Python Programming

Report for “Emotion Catcher”

小组名称：HoldOn

小组成员：于心瞳 丁天艾 李佳纯

指导老师：鲍杨

时间：2020 年 6 月 10 日

1. 项目介绍

1.1 项目简介

使用电脑自带摄像头进行人脸识别，并通过分析面部各点的位置判断表情，进行实时情绪分析，并在屏幕上显示分析出的情绪。在情绪分析基础上添加拍照功能，将照片制作成漫画形式并根据表情配置相应文字(可以做成真人表情包)。

1.2 相关资源

在使用摄像头拍摄、处理照片的过程中需要使用到 OpenCV；在进行人脸识别并判断表情时需要使用到 dlib，通过使用 dlib 的 68 点模型分析面部各点的位置来判断表情。

2. 代码解释(括号内为报告撰写者)

2.1 人脸识别(于心瞳)

(1)使用 pip install 安装 OpenCV 库和 dlib 库，并分别导入，用于图片处理和人脸识别：

```
import cv2  # 图像处理的库 OpenCv
import dlib  # 人脸识别的库 dlib
```

(2)使用 dlib 库中的 get_frontal_face_detector()函数即特征提取器实现人脸检测，通过 shape_predictor("shape_predictor_68_face_landmarks.dat")函数使用官方训练好的人脸 68 点特征预测器，进行人脸面部轮廓特征提取。

接着使用 OpenCV 库中的 VideoCapture(0)函数打开摄像头，其中参数 0 表示打开电脑自带的摄像头，若接入外部摄像头，则换为外部摄像头：

```
# 使用特征提取器 get_frontal_face_detector
detector = dlib.get_frontal_face_detector()
# dlib 的68点模型，使用官方训练好的特征预测器
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
# 建cv2摄像头对象，参数0表示打开电脑自带的摄像头，如果换了外部摄像头，则自动切换到外部摄像头
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
```

(3)通过 isOpened()函数检测摄像头是否处于打开状态，使用 ret, frame = cap.read()把摄像头获取的图像信息保存到变量 frame 中，同时返回一个布尔值，若参数 ret 为 False 则代表没有成功读取图像信息，打印“Cannot receive frame”。

另外使用 imshow 函数打开人脸识别的窗口，使用 waitKey(1)函数确认每帧延时为 1ms，保持比较高的流畅度，并使用 detector()人脸检测器检测每一帧图像中的人脸。使用 putText 函数在窗口中添加如何拍照和退出的说明：

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Cannot receive frame")

    kk = cv2.waitKey(1)

# 人脸 / Faces
faces = detector(frame, 0)
```

```
# 字体
font = cv2.FONT_ITALIC
# 复制，用于照相
draw_img = frame.copy()
cv2.putText(frame, "Press 'q' to take a picture", (10, 20), font, 0.6, (0, 200, 0), 2)
cv2.putText(frame, "Press 'Esc' to end", (10, 50), font, 0.6, (0, 200, 0), 2)
cv2.imshow("emotion", frame)
```

(4)若检测到人脸，计算每帧人脸的宽度和高度，再通过 rectangle()函数用矩形框出人脸部分：

```
# 检测到人脸 / Face detected
if len(faces) != 0:
    # 矩形框 / Show the rectangle box of face
    for k, d in enumerate(faces):
        # 计算矩形大小
        # (x,y), (宽度width, 高度height)
        pos_start = tuple([d.left(), d.top()])
        pos_end = tuple([d.right(), d.bottom()])

        # 计算矩形框大小 / compute the size of rectangle box
        face_height = (d.bottom() - d.top())
        face_width = (d.right() - d.left())
        # 用黄色矩阵框出人脸，光的三原色Red(0,0,255), Green(0,255,0), Blue(255,0,0)
        #  rectangle(img, pt1, pt2, color)
        cv2.rectangle(frame, pos_start, pos_end, (0, 255, 255), 2)
```

若没有检测到人脸，则使用 putText 函数在窗口中添加“No Face”文本：

```
    else:
        # 没有检测到人脸
        cv2.putText(frame, "No Face", (20, 50), font, 0.6, (0, 0, 255), 1, cv2.LINE_AA)
```

(5)按 Esc 键退出，打印“goodbye”,用 break 结束本次循环：

```
    if kk == 27:
        print('goodbye')
        break
```

使用 cap.release()函数关闭摄像头，使用 destroyAllWindows()函数关闭人脸识别窗口：

```
# 释放摄像头 / Release camera and destroy all windows
cap.release()
cv2.destroyAllWindows()
```

2.2 68 点分析(李佳纯)

(1)表情设定

我们将表情分为：开心、厌恶、恐惧、惊讶、悲伤、愤怒、自然

happy, disgust, fear, sad, angry, surprise, nature = 1, 1, 1, 1, 1, 1, 1

(2)参数设定

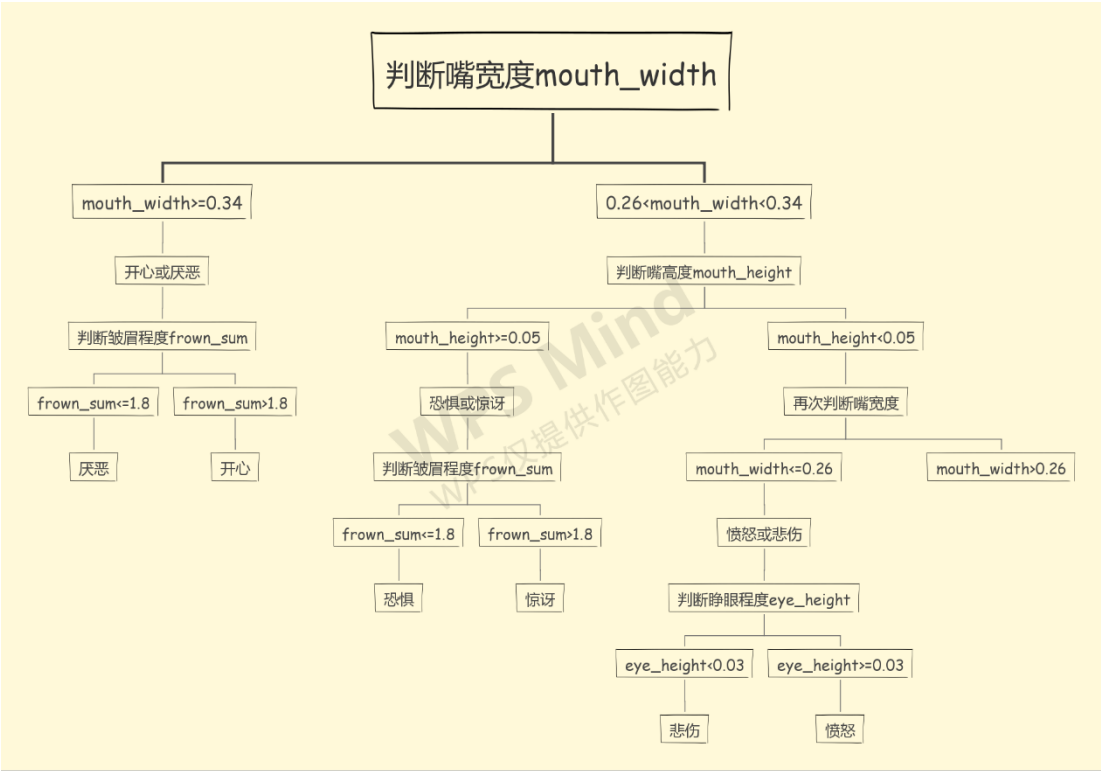
在识别人脸并标记 68 点之后，开始用过分析特定点的位置特征来分析表情。

我们初步决定，可以通过判断识别目标的嘴的相对宽度、嘴的相对高度、相对眉间距、相对睁眼程度、眉毛拟合成的直线斜率（这里的相对是指对于所识别人脸的长度或宽度）。

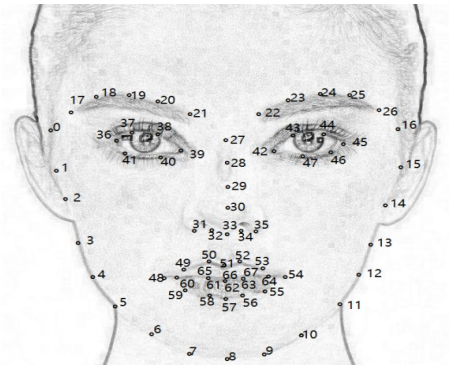
在设定参数之前，我们找到了不同表情的多张图片，并用如下代码对其进行相应的数据分析

```
import cv2
import dlib
import numpy as np
import pandas as pd
eye_height_list = []
mouth_height_list = []
mouth_wide_list = []
brow_k_list = []
frown_sum_list = []
for i in range(1, 6):
      path = f"picture/emotion/emotion{i}.jpg"      # 将emotion替换成相应情绪
      img = cv2.imread(path)
      detector = dlib.get_frontal_face_detector()
      predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
      dets = detector(img, 1) #获取人脸框位置信息
      for k, d in enumerate(dets):
            # 计算矩形大小
            pos_start = tuple([d.left(), d.top()])
            pos_end = tuple([d.right(), d.bottom()])
            # 计算矩形框大小 / compute the size of rectangle box
            face_height = (d.bottom() - d.top())
            face_width = (d.right() - d.left())
            # 用红色矩阵框出人脸
            cv2.rectangle(img, pos_start, pos_end, (0, 255, 255), 2)
            shape = predictor(img, d) # 寻找人脸的68个标定点
            for pt in shape.parts():
                  pt_pos = (pt.x, pt.y)
                  cv2.circle(img, pt_pos, 2, (0, 0, 255), 1)#img, center, radius, color, thickness
      eye_height = (shape.part(41).y - shape.part(37).y)/face_height
      mouth_width = (shape.part(54).x - shape.part(48).x) / face_width   # 嘴巴张开程度
      mouth_height = (shape.part(66).y - shape.part(62).y) / face_height # 嘴巴张开程度
      brow_sum = 0   # 高度之和
      frown_sum = 0   # 两边眉毛距离之和
      line_brow_x = []
      line_brow_y = []
      for j in range(17, 21):
            brow_sum += (shape.part(j).y - d.top()) + (shape.part(j + 5).y - d.top())
            frown_sum += shape.part(j + 5).x - shape.part(j).x
            line_brow_x.append(shape.part(j).x)
            line_brow_y.append(shape.part(j).y)
      # 计算眉毛的倾斜程度
      tempx = np.array(line_brow_x)
      tempy = np.array(line_brow_y)
      z1 = np.polyfit(tempx, tempy, 1)   # 拟合成一次直线
      brow_k = -round(z1[0], 3)
      frown_sum = frown_sum / face_width
      eye_height_list.append(eye_height)
      mouth_height_list.append(mouth_height)
      mouth_wide_list.append(mouth_width)
      frown_sum_list.append(frown_sum)
      brow_k_list.append(brow_k)
df2 = pd.DataFrame({'eye_height': eye_height_list,
                              'mouth_height':mouth_height_list,
                              'mouth_wide': mouth_wide_list,
                              'frown_sum': frown_sum_list,
                              'brow_k': brow_k_list})
print(df2)
```

根据所得的结果，并根据自己实际操作进行调整，我们对参数进行如下设定



我们需要分析的参数有识别目标的嘴的相对宽度、嘴的相对高度、相对眉间距、相对睁眼程度，在 dlib68 点模型中，上嘴唇中心为第 66 点，下嘴唇中心为第 62 点，嘴左角为 48 点，嘴右角为 54 点，左眼最高点为 37 点，左眼最低点为 41 点，左眉上有 17-21 五个点，右眉有 22-26 五个点。



(3)先用模型定义人脸上 68 个点，并显示出它们的位置

```
shape = predictor(frame, d)
for i in range(68):
    # circle(img, center, radius, color),
    cv2.circle(frame, (shape.part(i).x, shape.part(i).y), 2, (0, 255, 0), 1)
```

(4)下面计算各个参数

```
mouth_width = (shape.part(54).x - shape.part(48).x) / face_width  # 嘴巴张开宽度
mouth_height = (shape.part(66).y - shape.part(62).y) / face_height  # 嘴巴张开高度
eye_height = (shape.part(41).y - shape.part(37).y) / face_height  # 眼睛睁开程度
frown_sum = 0  # 两边眉毛距离之和
for j in range(17, 21):
    frown_sum += shape.part(j + 5).x - shape.part(j).x
frown_sum = frown_sum / face_width
```

(5)计算好各个参数后，我们开始根据参数对人脸进行表情分析，并用 cv2.putText 在人脸下方显示对应的表情。

```
if mouth_width >= 0.34:
    if frown_sum <= 1.8:
        cv2.putText(frame, "disgust", (d.left(), d.bottom() + 20), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2, 4)

    else:
        cv2.putText(frame, "happy", (d.left(), d.bottom() + 20), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2, 4)

elif mouth_height >= 0.05:
    if frown_sum <= 1.8:
        cv2.putText(frame, "fear", (d.left(), d.bottom() + 20), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2, 4)
    else:
        cv2.putText(frame, "surprise", (d.left(), d.bottom() + 20), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2, 4)

elif mouth_width <= 0.26:
    if eye_height < 0.03:
        cv2.putText(frame, "sad", (d.left(), d.bottom() + 20), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2, 4)

    else:
        cv2.putText(frame, "angry", (d.left(), d.bottom() + 20), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2, 4)

else:
    cv2.putText(frame, "nature", (d.left(), d.bottom() + 20), cv2.FONT_HERSHEY_SIMPLEX, 1,
                (0, 0, 255), 2, 4)
```

(6)之后，我们添加拍照功能，为了使代码简洁，我们定义了函数 screenshoot( )，用来照相、把原图保存至相应路径，并为图像处理做准备。

```
def screenshoot(emotion, num):
    name = f'{emotion}/picture{num}' + '.jpg'
    cv2.imwrite(name, draw_img)
    cartoonise(str(emotion), name, num)
    print(f"{emotion},{num}")
```

(7)定义函数后，我们在每次判断表情并显示的代码后面添加代码

```
if kk == ord('q'):
    screenshoot("emotion", emotion)
    emotion += 1
```

这样就实现了判断表情并拍照保存原图至指定路径。

2.3 图像处理(丁天艾)

在小组项目中我主要负责图像处理部分。

我们调用 OpenCV 进行图片处理。为了方便不同表情图片的处理，首先定义函数 cartoonise（）方便调用，它有三个变量，分析出的情绪、表情图片的名称和进行数据处理的变量。将照片卡通化的关键是减少图片色彩并强化边缘，我们通过 OpenCV 对照片进行处理。def cartoonise(emotion, picture_name, num):

首先是应用双边滤波器减少图片的色彩。双边滤波器（Bilateral Filter）是可以使影像平滑化的非线性滤波器，它考虑像素之间几何上的靠近程度和像素之间的色彩差异，能够将色彩相似区域变得平滑平坦但保留边缘清晰，适合将 RGB 图像转化为卡通图像。但是双边滤波器的计算速度比较慢，所以可以用重复的小的双边滤波器来代替只用一次大的双边滤波器计算。这里我们定义双边滤波的数目为 10。然后通过表情图片的名称这个参数读取相应的图像。为了对图片进行精确处理，我们首先利用高斯金字塔降低采样，来提高图像分辨率，但同时也会增大图片的尺寸。然后通过 for 循环实现用重复的小的双边滤波代替大的滤波，OpenCV 提供了 bilateralFilter()函数实现双边滤波操作，它的参数为 InputArray src: 输入的图像；int d: 表示在过滤过程中每个像素邻域的直径范围，如果这个值是非正数，则函数会从第五个参数 sigmaSpace 计算该值，我们将直径设置为 9；sigmaColor: 颜色空间过滤器的 sigma 值，这个参数的值越大，表明该像素邻域内有越宽广的颜色会被混合到一起，产生较大的半相等颜色区域；sigmaSpace: 坐标空间中滤波器的 sigma 值，如果参数值较大，则意味着颜色相近的较远的像素将相互影响，从而使更大的区域中足够相似的颜色获取相同的颜色，当 d>0 时，d 指定了邻域大小且与 sigmaSpace 无关，否则 d 正比于 sigmaSpace。结束之后我们再将采样图片提升到原始大小。

```
num_bilateral = 10# 定义双边滤波的数目
img_rgb = cv2.imread(picture_name)
# 用高斯金字塔降低取样
img_color = img_rgb
img_color = cv2.pyrDown(img_color)
# 重复使用小的双边滤波代替一个大的滤波
for _ in range(num_bilateral):
img_color = cv2.bilateralFilter(img_color, d=9,
                                sigmaColor=9,
                                sigmaSpace=7)
# 升采样图片到原始大小
img_color = cv2.pyrUp(img_color)
```

第二步是将彩色图像转换为灰度，应用中值滤波器减少图像中的图像噪点。在图像读取时，会出现噪声点，所以在进行边缘检测处理之前，需要首先对图像进行一定程度的降噪。中值滤波可以保留边缘，避免边缘模糊。而图像去噪是对单通道进行处理的，所以要先将图片转为灰度图像。利用颜色空间转换函数 cv2.cvtColor(p1,p2)，其中 p1 是需要转换的图片，p2 是转换成何种格式。cv2.COLOR_GRAY2RGB 将灰度图片转换成 RGB 格式，cv2.COLOR_BGR2GRAY 将 BGR 格式转换成灰度图片。然后用 medianBlur()函数实现中值滤波操作，第一个参数为需要处理的图片，第二个参数为当前的方框尺寸。

```
# 转换为灰度并使其产生中等的模糊
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
img_blur = cv2.medianBlur(img_gray, 3)
```

第三步是使用自适应阈值处理灰度图像并创建轮廓。在基本处理好色彩后，我们要利用自适应阈值强化图片边缘。自适应阈值是利用图像局部阈值替换全局阈值进行图像计算的一种方法，具体针对光影变化过大的图片或者范围内颜色差异不太明显的图片，它可以保证计算机能够通过判断和计算取得该图像区域的平均阈值进行迭代。因此它的优点是可以检测图像的每个小领域中最突出的特征，使其独立于图像的整体属性。我们利用adaptiveThreshold()函数实现自适应阈值操作，它的参数为 InputArray src：需要处理的图像；maxValue：预设满足条件的最大值；int adaptiveMethod：在一个邻域内计算阈值所采用的算法，它的两个取值分别为 ADAPTIVE_THRESH_MEAN_C 和 ADAPTIVE_THRESH_GAUSSIAN_C （ADAPTIVE_THRESH_MEAN_C 的计算方法是计算出领域的平均值再减去 C 的值，ADAPTIVE_THRESH_GAUSSIAN_C 的计算方法是计算出领域的高斯均值再减去 C 的值）；int thresholdType：阈值类型，它的两个取值分别为 THRESH_BINARY（二进制阈值） 和 THRESH_BINARY_INV（反二进制阈值）；int blockSize：像素的局部邻域大小；C：是一个偏移值调整量，用均值和高斯计算阈值后，再减或加这个值就是最终阈值。然后我们将图像转换回彩色。

```
#使用自适应阈值处理灰度图像创建轮廓
Img_edge = cv2.adaptiveThreshold(img_blur, 255,
                                 cv2.ADAPTIVE_THRESH_MEAN_C,
                                 cv2.THRESH_BINARY,
                                 blockSize=5, C=7)
#转换回色彩图像
img_edge = cv2.cvtColor(img_edge, cv2.COLOR_GRAY2RGB)
```

第四步是将处理好的彩色图像与创建的轮廓叠加。我们可以用 bitwise_and()函数对二进制数据进行“与”操作，将 img_color 和 img_edge 直接叠加，也可以利用 addWeighted()函数进行加权叠加，形成透明的效果。它的参数是：src1：第一张图像；alpha：第一张图像的权重，即透明度；src2：第二张图像；beta：第二张图像的权重；gamma：图 1 与图 2 作和后添加的数值，数值越大，叠加图片越白。

```
#叠加彩色图像和轮廓
#img_cartoon = cv2.bitwise_and(img_color, img_edge)
img_cartoon = cv2.addWeighted(img_color,0.8, img_edge,0.2,0)
```

最后对卡通图片进行命名并保存图像。

```
#命名并保存图像
name = 'cartoon/' + str(emotion) + str(num) + '.jpg'
cv2.imwrite(name, img_cartoon)
```

2.4 添加文字

将可以表达不同情绪的配字添加到列表中，并用 choice()函数随机选取，再用 putText()函数将配字添加到卡通图像中并保存。

首先 import random

```
def puttext(emotion, cartoon, name):
    happy_word = ["hahahaha", "Today is a good day!", "Aha"]
    angry_word = ["piss me off", "Don't bother me!"]
    disgust_word = ["disgusting", "hate you", "stay away from me", "Ugh"]
    sad_word = ["Life is so hard.", "TAT", "feel blue now..."]
    fear_word = ["It's' horrible!", "That sounds awful."]
    surprise_word = ["What a surprise!", "This shocked me", "Wow!", "Awesome", "can't believe it"]
    nature_word = ["just so so", "emmmm"]
    emotion_list = [happy_word, angry_word, disgust_word, sad_word, fear_word, surprise_word, nature_word]
    choice = ["happy", "angry", "disgust", "sad", "fear", "surprise", "nature"]
    location = choice.index(str(emotion))
    cv2.putText(cartoon, random.choice(emotion_list[location]), (d.left(), d.bottom() + 50), font, 1,
                (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)), 2)
    cv2.imwrite(str(name), cartoon)
```

3. 不足之处与优化设想

目前的 68 点分析主要基于人脸正对镜头的情况，当人脸对镜头的角度有所偏移时，可能难以准确地识别情绪。可以考虑再加入 if，else 结构增加人脸侧对镜头时的各项数据。

图像处理的部分，最终处理完成的照片“卡通”程度不够。可以修改参数设置或更换处理图像的程序。另外添加文字的位置也可以更灵活一些。

4. 总结

在本项目中我们将人脸识别与图像处理两项技术结合起来，满足了使用者根据自己的情绪制作个人表情包的需求。在搜索资料、阅读和学习数据库使用方法及大量代码的过程中，我们提高了编程能力及解决实际问题的能力，感受到了 python 这门语言与科学的巨大魅力。希望未来能通过学习更多相关知识对本项目的程序进行优化，并尝试用编程解决其他实际问题。感谢每位小组成员在本项目中的辛勤付出。感谢鲍杨老师给予的指导建议。