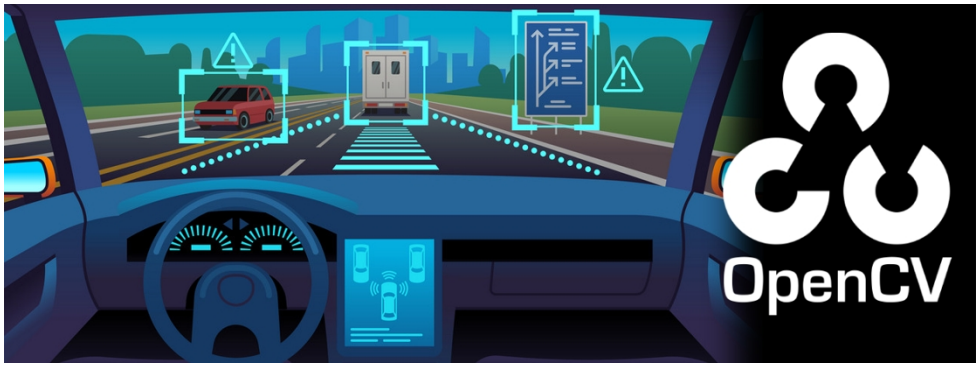
案例：人脸图像皮肤分割

本案例需要在桌面交互式环境下操作



案例概要

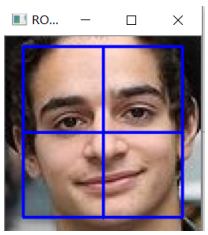
目前为止，我们已经学会了如何识别脸部和眼睛，但我们如何剪掉皮肤区域呢？在本案例中，你将使用 GrabCut 技术实现人脸图像的皮肤分割。

案例目标

在本案例中，我们将首先创建一个正面人脸级联分类器来检测人脸。使用 GrabCut 将人脸皮肤作为前景，将图像中的其他元素作为背景，实现人脸图像皮肤分割，您还可以使用前面的实验作为参考，使用鼠标修改蒙版。

案例详细操作说明

1. 导入 OpenCV 和 NumPy 模块。
2. 添加 `Sketcher` 鼠标通用类代码，因为我们将使用它来修改 GrabCut 获得的蒙版。
3. 使用 `cv2.imread` 函数读取输入图像， `grabcut.jpg` 。数据路径为： `./data/grabcut.jpg` ，注意：请以数据实际存储路径为准。
4. 使用 `cv2.cvtColor` 函数，将 BGR 图像转换为灰度模式。
5. 使用 `cv2.CascadeClassifier` 函数加载 Haar 级联，用于正面人脸检测。Haar 级联模型存储路径为： `./data/haarcascade_frontalface_default.xml` ，注意：请以数据实际存储路径为准。
6. 使用级联分类器 `detectMultiScale` 函数，检测图像中的人脸使用。选择参数值，检测正确的人脸。
7. 使用上一步中获得的边界框，从图像中裁剪人脸。
8. 调整图像尺寸为原图的两倍。使用 `cv2.resize` 函数，我们现在将处理这个调整尺寸后的图像。
9. 你还可以使用 ROI 进一步选取人脸中的特定皮肤区域，如下图：



10. 使用 `cv2.grabCut` 函数获取掩码。样本结果应该类似于下图：

11. 将确认的背景和可能的背景像素划入为背景像素。下图显示了示例结果:

12. 将新蒙版与图像相乘，获得前景。
13. 通过鼠标交互式微调，获得一个修改后的蒙版和最后的皮肤区域。下图显示了蒙版的示例效果：

14. 分割后的皮肤效果示例如下:

案例小结

在本案例中，你实现了对人脸图像进行皮肤分割。在美妆类增强现实应用中，经常需要对用户头发 - 面部皮肤区域进行分割，通过替换各种美妆效果增强用户体验。另一方面，在临床医学应用上，恶性黑色素瘤在疾病的早期并不明显，很难通过肉眼观察来诊断。皮肤镜检查是目前早期诊断恶性黑色素瘤的最佳方法，皮肤镜手术也是通过解剖消除恶性黑色素瘤威胁的常用方法。然而，当医生使用皮肤镜图像进行诊断和治疗时，人工检查皮肤镜图像通常非常耗时和主观。因此，计算机辅助诊断 (CAD) 方法是必要的，可以大大提高诊断的效率和准确性。目前，在医学图像分割中，UNET 是最流行的网络架构，已被广泛用于各种医学成像模式以及许多分割任务。

案例答案

1. 导入库

导入 OpenCV 模块和 numpy 模块。

In [1]:

```
import cv2
import numpy as np
```

2. 定义鼠标通用类

OpenCV 中对于鼠标的动作有如下：

- *EVENT_MOUSEMOVE 0*：# 滑动
- *EVENT_LBUTTONDOWN 1*：# 左键点击
- *EVENT_RBUTTONDOWN 2*：# 右键点击
- *EVENT_MBUTTONDOWN 3*：# 中键点击
- *EVENT_LBUTTONUP 4*：# 左键放开
- *EVENT_RBUTTONUP 5*：# 右键放开
- *EVENT_MBUTTONUP 6*：# 中键放开
- *EVENT_LBUTTONDBLCLK 7*：# 左键双击
- *EVENT_RBUTTONDBLCLK 8*：# 右键双击
- *EVENT_MBUTTONDBLCLK 9*：# 中键双击

在鼠标操作函数中，当满足鼠标左键点击然后进行拖拽的话，就会调用 `cv2.line` 函数进行线的绘制。

```
img = cv2.line(img, pt1, pt2, color[, thickness[, lineType[, shift]]])
```

参数说明

- **img**：背景图
- **pt1**：直线起点坐标
- **pt2**：直线终点坐标
- **color**：当前绘画的颜色。如在 BGR 模式下，传递 `(255,0,0)` 表示蓝色画笔。灰度图下，只需要传递亮度值即可。
- **thickness**：画笔的粗细，线宽。若是 `-1` 表示画封闭图像，如填充的圆。默认值是 `1`
- **lineType**：线条的类型

In [2]:

```
class Sketcher:
    def __init__(self, windowname, dests, colors_func):
        self.prev_pt = None
        self.windowname = windowname
        self.dests = dests
        self.colors_func = colors_func
        self.dirty = False
        self.show()
        cv2.setMouseCallback(self.windowname, self.on_mouse)

    def show(self):
        cv2.imshow(self.windowname, self.dests[0])
        cv2.imshow(self.windowname + ": mask", self.dests[1])

    # 鼠标操作函数
    def on_mouse(self, event, x, y, flags, param):
        pt = (x, y)
        if event == cv2.EVENT_LBUTTONDOWN:
            self.prev_pt = pt
        elif event == cv2.EVENT_LBUTTONUP:
            self.prev_pt = None
        if self.prev_pt and flags & cv2.EVENT_FLAG_LBUTTON:
            for dst, color in zip(self.dests, self.colors_func()):
                cv2.line(dst, self.prev_pt, pt, color, 5)
            self.dirty = True
            self.prev_pt = pt
            self.show()
```

3. 加载图片

调用 cv2.imread 函数进行图片的加载。

In [3]:

```
# 设置输入输出路径
import os
base_path = os.environ.get("BASE_PATH",'../data/')
data_path = os.path.join(base_path + "lab5/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)


img = cv2.imread("./data/grabcut.jpg")
```

4. 灰度图转换

调用 cv2.cvtColor 将加载图片转换为灰度图。

In [4]:

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

5. 加载级联分类器

调用 cv2.CascadeClassifier 加载级联分类器。加载函数如下：

```
<CascadeClassifier object> = cv2.CascadeClassifier(filename)
```

其中， filename 是分类器的路径和名称。

Hear 级联分类器多达几十种，且随着版本的更新还在不断增加。提供了对多种对象的检测功能，部分级联分类器如下表所示：

XML文件名	级联分类器类型
harrcascade_eye.xml	眼睛检测
harrcascade_eye_tree_eyeglasses.xml	眼镜检测
harrcascade_mcs_nose.xml	鼻子检测
harrcascade_mcs_mouth.xml	嘴巴检测
harrcascade_smile.xml	微笑检测
harrcascade_pedestrians.xml	行人检测
harrcascade_frontalface.xml	正面人脸检测
harrcascade_profileface.xml	人脸检测
harrcascade_silverware.xml	金属检测

In [5]:

```
haarCascadePath = "./data/haarcascade_frontalface_default.xml"
haarCascade = cv2.CascadeClassifier(haarCascadePath)
```

6. 人脸检测

使用级联分类器检测灰度图的人脸。当分类器加载完成，调用 cv2.CascadeClassifier.detectMultiScale() 函数，可以检测出图片中的所有人脸。

函数形式如下：

```
detectedObjects = cv2.CascadeClassifier.detectMultiScale(image,
                                                         [scaleFactor,minNeighbors, flags,
minSize, maxSize])
```

唯一必需的是 image ，该图像是要在其中检测面部的灰度输入图像。可选参数如下：

- scaleFactor： 决定每次迭代将图像调整大小的程度。这有助于检测输入图像中存在的不同大小的面部。
- minNeighbors： 用于确定检测目标时必须考虑的最小邻居的数量，默认为 3 。表明被检测出来的目标的旁边至少还需要有多少个类似的目标，其作用是避免模型将背景中的人脸也作为识别的对象。
- flag： 在新的分类器上面已经没有作用，主要是用于告诉分类器跳过平滑（无边缘区域）。
- minSize 和 maxSize： 指定要检测的脸部的最小和最大尺寸，超出此范围的人脸都会被忽略。

In [6]:

```
detectedFaces = haarCascade.detectMultiScale(gray, 1.2, 9)
```

7. 选出第一个人脸

从检测的所有人脸中，选择其中第一个人脸，并使用切片的方式将人脸提取出来。

In [7]:

```
x,y,w,h = detectedFaces[0]
img = img[y:y+h, x:x+w]
imgCopy = img.copy()
```

8. 参数设置

设置两个参数，宽和高。参数值为截取的人脸图片的两倍，方便后面的尺寸变换。

In [8]:

```
scale_percent = 200
# 新的宽
width = int(img.shape[1] * scale_percent/100)
# 新的高
height = int(img.shape[0] * scale_percent/100)
# 新的维度
dim = (width, height)
```

9. 尺寸变换

利用上面设置的参数，调用 `cv2.resize` 函数将截取的人脸图像进行变换。

```
dst = resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
```

参数说明

- **src**：输入图片
- **dst**：输出图片
- **dsize**：输出图片的尺寸
- **fx, fy**：沿 x 轴，y 轴的缩放系数
- **interpolation**：插入方式。插值方式有多种

1. *INTER_NEAREST*：最近邻插值
2. *INTER_LINEAR*：双线性插值（默认设置）
3. *INTER_AREA*：使用像素区域关系进行重采样。
4. *INTER_CUBIC*： `4x4` 像素邻域的双三次插值
5. *INTER_LANCZOS4*： `8x8` 像素邻域的 Lanczos 插值

In [9]:

```
img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
```

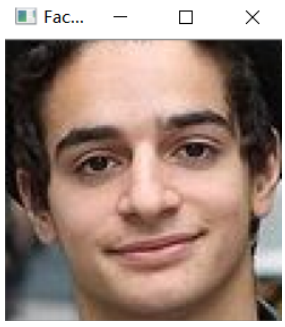
10. 显示图片（按Esc退出）

In [10]:

```
cv2.imshow("Face Detected", img)
cv2.waitKey(0)
```

Out[10]:

27



11. 创建掩码

创建一个和变换的人脸图像尺寸一致的掩码，掩码像素值都使用 `0` 填充。

In [11]:

```
mask = np.zeros(img.shape[:2], np.uint8)
```

12. 设置临时参数

主要是为后面实现交互式前景提供参数，两个数组为算法内部的使用数组，只需要创建大小为 (1,65) 的 numpy.float64 数组。

In [12]:

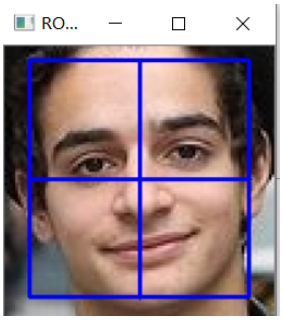
```
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
```

13. 选择感兴趣区域

调用 cv2.selectROI 手动选择感兴趣区域（选择后按Esc退出）

In [13]:

```
rect = cv2.selectROI(img)
```



14. 输出 ROI 信息

ROI 信息包括左上角坐标，矩形框的长和宽。

In [14]:

```
x,y,w,h = rect
```

15. 绘制矩形框

调用 cv2.rectangle 函数，根据左上角坐标和右下角坐标绘制出矩形框。

In [15]:

```
cv2.rectangle(imgCopy, (x, y), (x+w, y+h), (0, 0, 255), 3);
```

16. 交互式前景提取

调用 cv2.grabCut 函数进行交互式前景提取。

In [16]:

```
cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
```

Out[16]:

```
(array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=uint8),
 array([[ 2.57493941e-01,  1.80790000e-01,  1.81470301e-01,
          2.02219482e-01,  1.78026277e-01,  1.72404227e+01,
          1.96545575e+01,  2.54933950e+01,  1.50207432e+02,
          1.65018109e+02,  1.89200376e+02,  9.33926898e+01,
          1.03339503e+02,  1.18244611e+02,  4.72363331e+01,
          5.66770395e+01,  7.45971405e+01,  1.85688799e+02,
          2.02213757e+02,  2.41427991e+02,  8.04177583e+01,
          7.59588779e+01,  6.82803198e+01,  7.59588779e+01,
          8.01864819e+01,  7.96565691e+01,  6.82803198e+01,
          7.96565691e+01,  1.13182585e+02,  1.31377447e+03,
          8.86851606e+02,  2.83196441e+02,  8.86851606e+02,
          6.51708750e+02,  3.21028827e+02,  2.83196441e+02,
          3.21028827e+02,  4.61284402e+02,  3.51613836e+02,
          1.83526943e+02, -3.72303112e+01,  1.83526943e+02,
          1.70216275e+02,  1.66297460e+02, -3.72303112e+01,
          1.66297460e+02,  5.51151506e+02,  1.78747343e+02,
          1.67717201e+02,  1.58889154e+02,  1.67717201e+02,
          2.23086613e+02,  2.90576368e+02,  1.58889154e+02,
          2.90576368e+02,  5.03059740e+02,  1.96426440e+02,
          1.38990094e+02,  9.38690406e+01,  1.38990094e+02,
          1.08367731e+02,  7.70788029e+01,  9.38690406e+01,
          7.70788029e+01,  6.43699640e+01]]),
 array([[2.57379662e-01, 1.33630858e-01, 3.17245776e-01, 5.99298693e-02,
         2.31813835e-01, 1.69185782e+02, 1.85296507e+02, 2.30317810e+02,
         6.13129771e+01, 7.44160305e+01, 1.13183206e+02, 1.29693730e+02,
         1.47863344e+02, 1.90833601e+02, 3.01531915e+01, 3.86680851e+01,
         6.20851064e+01, 9.60858086e+01, 1.12115237e+02, 1.48904290e+02,
         2.58387085e+02, 2.22416552e+02, 1.93985544e+02, 2.22416552e+02,
         1.99650751e+02, 1.68666976e+02, 1.93985544e+02, 1.68666976e+02,
         1.63430580e+02, 1.71298992e+02, 1.75854525e+02, 1.24158310e+02,
         1.75854525e+02, 2.09281117e+02, 1.34879888e+02, 1.24158310e+02,
         1.34879888e+02, 1.86565672e+02, 2.02103144e+02, 1.82505574e+02,
         1.94741642e+02, 1.82505574e+02, 1.96962033e+02, 1.80941087e+02,
         1.94741642e+02, 1.80941087e+02, 2.14886299e+02, 2.22921213e+02,
         2.11629570e+02, 1.75701856e+02, 2.11629570e+02, 2.26511109e+02,
         2.14606971e+02, 1.75701856e+02, 2.14606971e+02, 2.71303395e+02,
         9.45740450e+01, 8.38407718e+01, 6.95783447e+01, 8.38407718e+01,
         1.01038151e+02, 8.58457377e+01, 6.95783447e+01, 8.58457377e+01,
         1.46689960e+02]]))
```

17. 背景提取

调用 `np.where` 函数将所有确定背景和疑似背景设置为 `0`，其余的设置为 `1`。`np.where` 函数形式如下：

```
where(condition, [x, y])
```

满足条件 (condition)，输出 `x`，不满足输出 `y`。

In [17]:

```
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
```

18. 显示和保存

调用 cv2.imshow 显示掩码和提取的背景，并调用 cv2.imwrite 进行保存。

In [18]:

```
cv2.imshow("Mask",mask*80)
cv2.imshow("Mask2",mask2*255)
cv2.imwrite(result_path+"mask_07.png",mask*80)
cv2.imwrite(result_path+"mask2_07.png",mask2*255)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Mask

Mask2

19. 保留前景

由于掩码为黑色，像素值为 0 ，将蒙版图像与原始图像相乘，原图像中的背景区域像素值与蒙版的 0 相乘，全部归零。

于是，图像运算结果便只保留了前景图像，而背景变成了黑色，即像素值为 0 。

我们使用以下代码实现:

In [19]:

```
img = img*mask2[:,:,np.newaxis]
```

20. 备份前景和背景

In [20]:

```
img_mask = img.copy()
mask2 = mask2*255
mask_copy = mask2.copy()
```

21. 利用 Sketcher 类创建草图

In [21]:

```
sketch = Sketcher('image', [img_mask, mask2], lambda : ((255,0,0), 255))
```

22. 鼠标操作

创建一个无限的 while 循环，因为我们想要修改掩码和结果。当按住鼠标左键在 mask 上进行滑动时， mask2 会跟着进行直线的绘制。

其中 OpenCV 设置了多个按键进行操作转换。

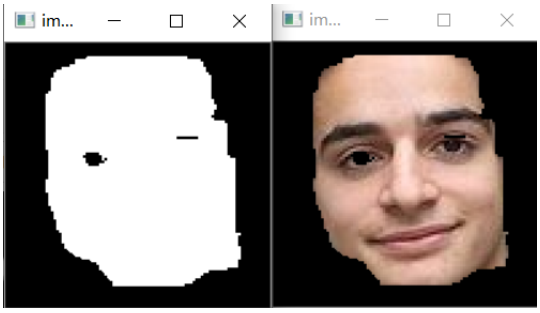
- **ESC**：退出
- **r**：重新生成草图
- **b**：切换为背景
- **f**：切换为前景
- **根据给出的绘制进行交互式前景提取**

In [22]:

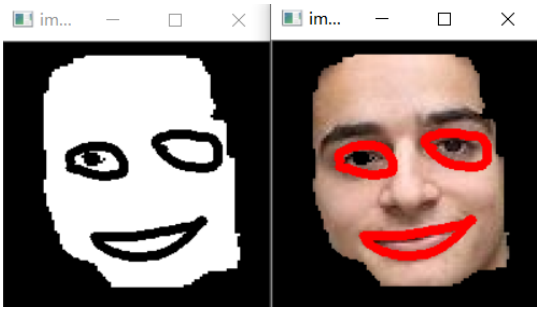
```
while True:
    ch = cv2.waitKey()

    # 按ESC退出
    if ch == 27:
        print("exiting...")
        cv2.imwrite(result_path+"img_mask_grabcut_07.png",img_mask)
        cv2.imwrite(result_path+"mask_grabcut_07.png",mask2)
        break

    # 重置
    elif ch == ord('r'):
        print("resetting...")
        img_mask = img.copy()
        mask2 = mask_copy.copy()
        sketch = Sketcher('image', [img_mask, mask2], lambda : ((255,0,0), 255))
        sketch.show()

    # 切换到背景
    elif ch == ord('b'):
        print("drawing background...")
        sketch = Sketcher('image', [img_mask, mask2], lambda : ((0,0,255), 0))
        sketch.show()

    # 切换到前景
    elif ch == ord('f'):
        print("drawing foreground...")
        sketch = Sketcher('image', [img_mask, mask2], lambda : ((255,0,0), 255))
        sketch.show()
    else:
        print("performing grabcut...")
        mask2 = mask2//255
        cv2.grabCut(img,mask2,None,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_MASK)
        mask2 = np.where((mask2==2)|(mask2==0),0,1).astype('uint8')
        img_mask = img*mask2[:,:,np.newaxis]
        mask2 = mask2*255
        print("switching bank to foreground...")
        sketch = Sketcher('image', [img_mask, mask2], lambda : ((255,0,0), 255))
        sketch.show()
```

```
exiting...
```

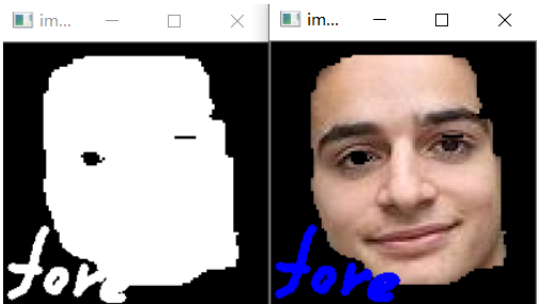
- 键盘按 r 时，两张图同时重置



- 按键按 b 时，切换到背景，背景校正用红色表示。标记区域包括真正属于背景但被 GrabCut 标记为前景的区域。这些区域将在掩膜中标记为黑色。此时在一张图的背景操作， mask 也会跟着显示出来，而在前景是无效的。



- 按键按 `f` 时，切换到前景，前景区域校正用蓝色标记，用它覆盖前景区域掩膜中出现的黑色点。蓝色标记的区域现在将成为蒙版中前景的一部分，这意味着它们现在将在蒙版中变成白色，此时在一张图的前景操作， `mask` 也会跟着显示出来，而在背景是无效的。



23. 关闭所有窗口

In [23]:

```
cv2.destroyAllWindows()
```