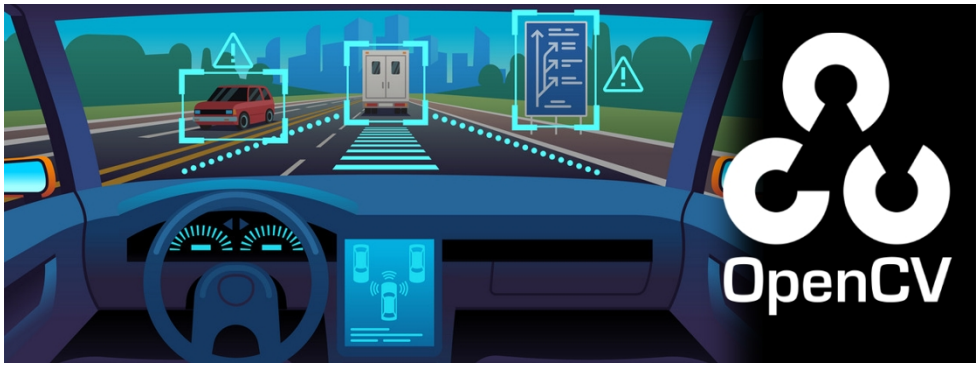
实验：ROI 掩码前后景分离

本实验需要在桌面交互式环境下操作



实验概要

在前面的练习中，我们看到了 GrabCut 应用非常容易，但是结果需要一些改进。我们可以调用 OpenCV 的 `cv2.grabCut` 函数，进行交互式前景提取。

GradCut 算法的具体实施过程如下:

1. 将前景所在的区域大致用一个矩形框标注出来，此时矩形框标出的只是前景的大致位置。矩形外的所有区域是背景。矩形框内的东西是未知的，其中既包含前景有包含背景。
2. 根据矩形框外部的 **确定背景** 数据来区分矩形框区域内的前景和背景。
3. 使用一个高斯混合模型（GMM）对前景和背景建模。根据我们的输入，GMM 会学习并创建新的像素分布。对那些分类未知的像素（可能是前景也可能是背景），根据他们与已知分类（如背景）的像素关系来进行分类。
4. 这样就会根据像素的分布创建一幅图。图中的节点就是像素点。除了像素点做节点之外还有两个节点：前景节点和背景节点。所有的前景像素都和前景节点相连。所有的背景像素都和背景节点相连。每个像素连接到前景节点或背景节点的边的权重由像素是前景或背景的概率决定。
5. 图中的每个像素除了与前景节点或背景节点相连外，彼此之间也存在连接。两个像素之间的权重由边的信息或者两个像素的相似性来决定。如果两个像素的颜色有很大的不同，那么它们之间的边的权重就会很小。
6. 使用 `mincut` 算法对上面的图进行分割。它会根据最低成本方程将图像分为前景节点和背景节点。成本方程就是被剪掉的所有边的权重之和。在裁剪之后，所有连接到前景节点的像素被认为是前景，所有连接到背景节点的像素被认为是背景。
7. 不断重复上述过程，直到分类收敛为止。

实验目标

在本实验中，我们首先使用一个矩形的 ROI 掩码，用 GrabCut 进行分割，然后在掩码（掩膜）上添加一些最后的润色，以获得更好的结果。

1. 导入库

导入 OpenCV 模块和 numpy 模块。

In [1]:

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

2. 加载图片

调用 cv2.imread 函数加载图片。

In [2]:

```
# 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")
```

3. 图片备份

调用 copy 进行原图像备份。

In [3]:

```
imgCopy = img.copy()
```

4. 创建掩码

创建一个与输入图像大小相同的掩码，初始化为 0 ，使用无符号八位整数，如下面的代码所示:

In [4]:

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

5. 创建两个临时数组

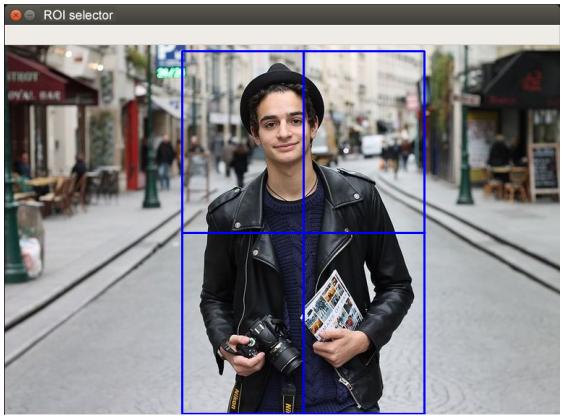
主要是为后面实现交互式前景提供参数，两个数组为算法内部的使用数组，只需要创建大小为(1,65)的numpy.float64数组。

In [5]:

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

6. 选择 ROI

调用 cv2.selectROI 函数手动选择 ROI 区域，如下图所示操作（选择后按Esc退出）：



In [6]:

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

7. 获取 ROI 信息

通过 `cv2.selectROI` 函数返回一个四元组，四元组包括左上角点坐标和矩形框的长宽：

In [7]:

```
# 获取通过你选取的ROI返回的四元组信息
x,y,w,h = rect
```

8. 绘制矩形框

调用 `cv2.rectangle` 来绘制矩形框。函数如下：

```
img = rectangle(img, pt1, pt2, color[, thickness[, lineType[, shift]]])
```

参数说明

- **img**：在其上面绘制图形的载体
- **pt1**：矩形顶点坐标
- **pt2**： `pt1` 对角顶点的坐标
- **color**：绘制形状的颜色，通常使用RGB模型表示颜色
- **thickness**：线条的粗细。默认为 `1` ，如果设置为 `-1` ，则表示填充图形
- **lineType**：线条类型，默认为8连续类型
- **shift**：数据精度。

In [8]:

```
# 绘制矩形
cv2.rectangle(imgCopy, (x, y), (x+w, y+h), (0, 0, 255), 3)
```

Out[8]:

```
array([[[ 45,  55,  32],
        [ 39,  48,  27],
        [ 32,  41,  28],
        ...,
        [130, 166, 176],
        [129, 165, 175],
        [128, 164, 174]],

       [[ 44,  54,  31],
        [ 41,  50,  30],
        [ 37,  46,  33],
        ...,
        [131, 168, 176],
        [131, 166, 176],
        [129, 166, 174]],

       [[ 44,  53,  32],
        [ 43,  52,  32],
        [ 43,  52,  39],
        ...,
        [132, 168, 174],
        [133, 166, 175],
        [130, 166, 172]],

       ...,

       [[174, 170, 169],
        [174, 170, 169],
        [174, 170, 169],
        ...,
        [178, 175, 171],
        [180, 177, 173],
        [181, 178, 174]],

       [[173, 169, 168],
        [173, 169, 168],
        [172, 168, 167],
        ...,
        [175, 172, 168],
        [178, 175, 171],
        [179, 176, 172]],

       [[172, 168, 167],
        [171, 167, 166],
        [171, 167, 166],
        ...,
        [172, 169, 165],
        [174, 171, 167],
        [176, 173, 169]]], dtype=uint8)
```

9. 保存 ROI

调用 `cv2.imwrite` 函数保存 ROI 图像，保存下来的 ROI 图像如下所示：

In [9]:

```
cv2.imwrite("roi_06.png",imgCopy)
```

Out[9]:

True

10. 交互式前景提取

在 OpenCV 中，交互式前景提取函数如下所示：

```
mask, bgdModel, fgdModel = grabCut(img, mask, rect, bgdModel, fgdModel, iterCount[, mode])
```

参数说明

- **img**：输入图像，要求是8为3通道
- **mask**：掩码图像，要求是8位3通道。该参数用于确定前景区域、背景区域和不确定区域，有四种形式：
    1. *cv2.GCD_BGD*，表示确定是背景，也可用数字 `0` 表示
    2. *cv2.GCD_FGD*，表示确定是前景，也可用数字 `1` 表示
    3. *cv2.GCD_PR_BGD*，表示可能的背景，也可用数字 `2` 表示
    4. *cv2.GCD_PR_FGD*，表示可能的前景。也可用数字 `3` 表示
- **rect**：用于限定需要进行分割的图像范围，只有该矩形窗口内的图像部分才被处理
- **bgdModel**, **fgdModel**： 算法使用的两个大小为 `(1,65)` 的临时数组，数据类型为 `numpy.float64`
- **iterCount**：表示迭代的次数
- **mode**：迭代模式。主要模式如下：
    1. *cv2.GC_INIT_WITH_RECT* 模式，意味着由 `rect` 参数指定你正在初始化的 GrabCut 算法使用矩形掩码。
    2. *cv2.GC_INIT_WITH_MASK* 模式，意味着您正在用 `mask` 输入参数指定的通用掩码（基本上是一个二值图像）初始化 GrabCut 算法。掩膜具有与输入图像相同的形状。

In [10]:

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

Out[10]:

```
(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([[1.23192700e-01, 3.18637659e-01, 1.96289221e-01, 2.35519254e-01,
         1.26361165e-01, 3.44902577e+01, 3.69610041e+01, 3.69301244e+01,
         1.84001808e+02, 1.80953502e+02, 1.78534658e+02, 1.41443622e+02,
         1.46522644e+02, 1.49078415e+02, 2.04837494e+02, 2.04216585e+02,
         2.03838450e+02, 6.71243777e+01, 7.53839805e+01, 8.49315097e+01,
         2.11096206e+02, 2.22108532e+02, 1.54466088e+02, 2.22108532e+02,
         2.51722605e+02, 1.74972039e+02, 1.54466088e+02, 1.74972039e+02,
         2.28209367e+02, 6.41711088e+01, 5.76277168e+01, 5.36055005e+01,
         5.76277168e+01, 5.43045783e+01, 4.90501823e+01, 5.36055005e+01,
         4.90501823e+01, 4.71016786e+01, 9.71375534e+02, 9.37395448e+02,
         8.93455945e+02, 9.37395448e+02, 9.75942270e+02, 9.71611807e+02,
         8.93455945e+02, 9.71611807e+02, 1.04727037e+03, 1.13580036e+02,
         1.07413753e+02, 1.04735879e+02, 1.07413753e+02, 1.29543158e+02,
         1.41393823e+02, 1.04735879e+02, 1.41393823e+02, 1.66294668e+02,
         7.73836152e+02, 7.26048610e+02, 2.57319004e+02, 7.26048610e+02,
         8.89120048e+02, 4.05269369e+02, 2.57319004e+02, 4.05269369e+02,
         5.87086385e+02]]),
 array([[4.60587648e-01, 2.48088882e-02, 1.30910208e-01, 2.96587955e-01,
         8.71053007e-02, 2.41991904e+01, 2.07176569e+01, 1.94869865e+01,
         2.24741379e+02, 2.16839523e+02, 2.13417772e+02, 8.78812835e+01,
         9.43212131e+01, 1.10769102e+02, 5.58869536e+01, 4.31049848e+01,
         4.14762592e+01, 1.47204482e+02, 1.64524427e+02, 2.05026945e+02,
         9.78340463e+01, 7.88978640e+01, 7.23352151e+01, 7.88978640e+01,
         7.56747055e+01, 7.05825741e+01, 7.23352151e+01, 7.05825741e+01,
         6.84012296e+01, 6.44021091e+02, 6.41853723e+02, 6.65087708e+02,
         6.41853723e+02, 6.93983531e+02, 7.41686406e+02, 6.65087708e+02,
         7.41686406e+02, 9.11895758e+02, 8.89552177e+02, 7.21806146e+02,
         4.14067847e+02, 7.21806146e+02, 7.17734791e+02, 6.25353909e+02,
         4.14067847e+02, 6.25353909e+02, 9.11508851e+02, 3.37167791e+02,
         2.80256340e+02, 2.27761701e+02, 2.80256340e+02, 3.28697986e+02,
         2.87580057e+02, 2.27761701e+02, 2.87580057e+02, 2.62923869e+02,
         9.09930738e+02, 8.13577352e+02, 7.96400559e+02, 8.13577352e+02,
         7.69560660e+02, 7.44716416e+02, 7.96400559e+02, 7.44716416e+02,
         7.84678448e+02]]))
```

11. 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 [11]:

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

12. 显示和保存

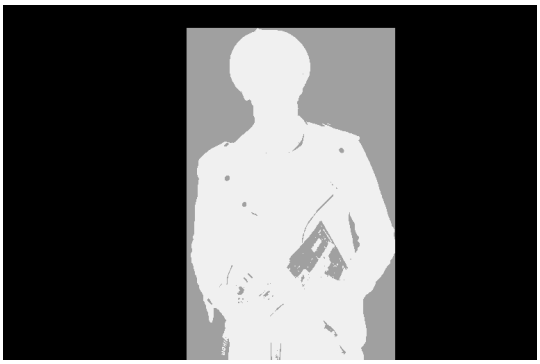
调用 `cv2.imshow` 显示掩码和提取的背景，并调用 `cv2.imwrite` 进行保存。

In [12]:

```
cv2.imshow("Mask",mask*80)
cv2.imshow("Mask2",mask2*255)
cv2.imwrite("mask_06.png",mask*80)
cv2.imwrite("mask2_06.png",mask2*255)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

请注意，我们将掩码乘以 `80` ，因为掩膜中的像素值仅从 `0` 到 `3` (包括 `3`)，因此不会直接可见。同样，我们用 `mask2` 乘以 `255` 。掩膜图像如下所示：

- **mask.png**：其中，较亮的区域对应已确认的前景像素；灰色区域对应可能的背景像素；黑色像素保证属于背景。



- **mask2.png**：将确认背景和可能的背景像素全部归类为背景后的掩膜图像。

13. 保留前景

由于掩码的背景为黑色，像素值为 `0` ，将掩码图像与原始图像相乘，原图像中的背景区域像素值与掩码的 `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 [13]:

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

14. 备份前景和背景

In [14]:

```
img_mask = img.copy()

# 类似上面的步骤，使非0像素完全可见
mask2 = mask2*255
mask_copy = mask2.copy()
```

15. 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

使用 OpenCV 提供的 `Sketcher` 类来处理鼠标。使用这个类，在光标的帮助下修改掩膜 (即：前景和背景)。

In [15]:

```
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])

    # onMouse函数用于鼠标处理
    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()
```

使用此流程，我们将能够动态地看到结果中的变化。前面的函数 `__init__` ，用于初始化 Sketcher 类对象。

16. 利用 Sketcher 类创建草图

`lambda` 函数的参数意味着在图像中绘制的任何蓝色像素 `(255,0,0)` 将在掩膜中显示为白色像素 `(255)` 。

In [16]:

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

17. 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

根据给出的绘制结果，进行交互式前景提取。

In [17]:

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

    # 按ESC退出
    if ch == 27:
        print("exiting...")
        cv2.imwrite("img_mask_grabcut_06.png",img_mask)
        cv2.imwrite("mask_grabcut_06.png",mask2)
        break

    # 按 r 重新生成草图
    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()

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

    # 按 f 变换成前景
    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()
```

```
performing grabcut...
switching bank to foreground...
exiting...
```

- 键盘按 r 时，两张图同时重置

- 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。

- 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。

完成上述微调后，我们将得到一个完整的前后景分离图像。

18. 关闭所有窗口

经过前景和背景校正后得到的最后的掩码图像，你应该得到一个类似与下图的结果：

使用 `cv2.destroyAllWindows` 关闭窗口，结束实验。

In [18]:

```
cv2.destroyAllWindows()
```

实验小结

在本实验中，我们首先使用一个矩形 ROI 执行一个粗糙的 GrabCut。然后在 Sketcher 类的帮助下修正了掩码。分别通过前景校正和背景校正两部分实现前后景分离微调。然后，GrabCut 再次使用得到的掩模来获得最终的前景图像。一旦你成功提取了前景图像，我们便能够将背景替换为任意的其他图片，最终实现在很多照片修图应用中常见的背景修改功能。

练习题

- **尝试使用在本实验与前面的实验中学习到的 GrabCut 前后景分离技术，对本实验中的人像照片进行背景替换。**