实验：GrabCut 人物分割与背景替换

实验概要

图像分割基本概念

图像分割 是形成 GrabCut 的基础。通俗而言，图像分割任务包括了根据像素所属的类别，将其划分为不同的分类目标。我们从最简单的开始，要找的分类目标只是背景和前景。我们要通过确定图像的哪个区域是背景的一部分，哪个区域是前景的一部分来分割图像。

然而，您应该了解到：在深度学习中， 图像分割 是一个非常大的课题，它包括基于类的图像分割，有时基于语义分割等。目前，我们可以认为 GrabCut 是一种简单的背景去除技术，尽管在技术上它也可以被认为是一种只考虑了两个主要类：背景和前景的图像分割技术。另外，还需要注意的是，GrabCut 使用的不只两个（前景、背景），而是四个标签（肯定前景、可能前景、肯定背景、可能背景）。但是基本的思想将保持不变，即：从图像中移除背景。

譬如下面的图片，如果我们把焦点放在前景和背景上，我们可以说背景是一个人正在运动的公园，而前景是由照片中的人物组成的。

考虑到这一点，如果我们继续进行图像分割，我们将得到以下图像:

请注意，图像中只有前景。整个背景已经被移除，并替换为白色像素（或者说是白色背景）。您应该很熟悉可以在哪些地方使用这种技术 —— 也就是我们常说的 抠图 。对于 Photoshop 用户，类似的技术在照片编辑中被广泛使用，另外，电影中使用绿幕的想法也是基于类似的概念。

GrabCut 的技术价值

GrabCut 是互动的技术前景提取（用户可以交互式地提取前景），由 Carsten Rother, Vladimir Kolmogorov, 和 Andrew Blake 他们的论文《GrabCut ——使用迭代图分割的交互式前景提取 (http://pages.cs.wisc.edu/~dyer/cs534-fall11/papers/grabcut-rother.pdf)》中提出的。该算法使用了颜色、对比度和用户输入来获得令人满意的效果。通过构建在算法之上的应用程序的数量，我们可以从侧面理解该算法的强大功能。乍一看，这似乎是一个手工操作的过程，但事实上，手工操作只是为了最后完成结果或进一步改进它们。同时，根据 Fakrul Islam Tushar 的论文《HSV 颜色空间的 GrabCut 皮肤病变自动分割 (https://arxiv.org/ftp/arxiv/papers/1810/1810.00871.pdf)》这是一个很好的展示了 GrabCut 能力的例子。使用 HSV 颜色空间代替通常的 BGR 颜色空间，并使用 GrabCut 的迭代程序成功地分割了皮肤病变。正如论文所提到的。

> 这是迈向皮肤癌数字化自动诊断的第一步。

GrabCut 算法实现

为了实现 GrabCut ， 我们将使用 `cv2.grabCut()` 函数。该函数的语法如下:

```
outputmask, bgModel, fgModel = cv2.grabCut(image, mask, rect, bgModel, fgModel, iterCount, mode)
```

参数说明:

- `image` ： 进行 GrabCut 操作的输入图像。
- `iterCount` ： 运行的 GrabCut 的迭代次数。
- `bgModel, fgModel` ： 算法使用的两个大小为(1,65)的临时数组。

另外，GrabCut 有四种可能的模式（完整模式说明参考 (https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#gaf8b5832ba85e59fc7a98a2afd034e558)），但我们只关注两种重要的模式，即 `mask` 和 `rect` ——

- `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。
- `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。蒙版具有与输入图像相同的形状。

该函数返回三个值。在这三个值中，对我们来说唯一重要的是 `outputmask` ，它是一个灰度图像，与作为输入的图像大小相同，但只有以下四个可能的像素值:

- `Pixel value 0 / cv2.GC_BGD` ： 意味着像素肯定属于背景。
- `Pixel value 1 / cv2.GC_FGD` ： 意味着像素肯定属于前景。
- `Pixel value 2 / cv2.GC_PR_BGD` ： 意味着像素可能属于背景。
- `Pixel value 3 / cv2.GC_PR_FGD` ： 意味着像素可能属于前景。

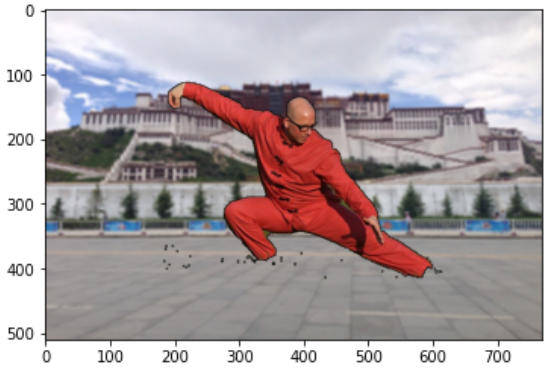
我们将广泛使用的另一个重要函数是 OpenCV 的 `cv2.selectROI` 函数，在使用 `cv2.GC_INIT_WITH_RECT` 模式中选择感兴趣的矩形区域，这个函数的用法非常简单。您可以直接使用鼠标选择左上角，然后拖动光标选择矩形。一旦完成，你可以按空格键得到 ROI。

需要注意的是： 与 `cv2.imshow` 类似，直接通过鼠标选择 ROI 区域在当前云计算 Jupyter Notebook 实验环境中并不适用（会造成 Kernel 崩溃），它仅能在您本地的笔记本上进行实验的时候使用。我们会使用折中的方案来满足实验操作。—— 然而，您仍然可以把 `cv2.selectROI` / `cv2.imshow` 相关的代码在本地 OpenCV 环境中实现。

实验目标

在本实验中，我们将进行前景提取，从输入图像中分割出人物。我们将使用 `cv2.GC_INIT_WITH_MASK` 模式实现这个需求，要处理的图片如下所示：

通过前景提取从从输入图像中分割出人物后，我们就可以进一步将原图像中的背景进行替换。最终实验效果如下图所示：



1. 导入依赖库

本实验中，我们需要导入 CV2 与 Numpy 两个库。

同时，为了保证图像能在 Notebook 中正常显示，不能直接使用 OpenCV 自带的 `cv2.imshow` 命令，需要再导入matplotlib 库。

In [1]:

```
import cv2                          # 导入OpenCV
import numpy as np                  # 导入NumPy
import matplotlib.pyplot as plt     # 导入matplotlib

# 魔法指令，使图像直接在Notebook中显示
%matplotlib inline
```

2. 加载图像并拷贝副本

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)

# 读取图像
# 指定输入图像的路径
img = cv2.imread("./data/person.jpg")
```

In [3]:

```
# 拷贝原始图像的副本作为备份
imgCopy = img.copy()
```

3. 创建掩码 (掩膜 / 蒙版)

创建与原始图像相同大小（宽度和高度）的掩码蒙版，`[:2]` 与原始图像 `img` 长宽一致。

In [4]:

```
# 创建掩码蒙版
mask = np.zeros(img.shape[:2], np.uint8)
```

使用 NumPy 的 `np.zeros` 将掩码初始化为 `0`，即创建两个全 `0` 填充的数组，您应该还记得在 OpenCV 中，图像就是数组。

In [5]:

```
# 创建临时数组
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
```

- **bgdModel**：背景模型，用于后面的 GrabCut 运算。如果为 `null`，函数内部会自动创建一个 `bgdModel`；`bgdModel` 必须是单通道浮点型（CV_32FC1）图像，且行数只能为 `1`，列数只能为 `13x5`；
- **fgdModel**：前景模型，用于后面的 GrabCut 运算。如果为 `null`，函数内部会自动创建一个 `fgdModel`；`fgdModel` 必须是单通道浮点型（CV_32FC1）图像，且行数只能为 `1`，列数只能为 `13x5`；

4. 选择图像 ROI 区域

需要注意的是： 与 `cv2.imshow` 类似，直接通过鼠标选择 ROI 区域在当前云计算 Jupyter Notebook 实验环境中并不适用（会造成 Kernel 崩溃），它仅能在您本地的笔记本上进行实验的时候使用。我们会使用折中的方案来满足实验操作。—— 然而，您仍然可以把 `cv2.selectROI` / `cv2.imshow` 相关的代码在本地 OpenCV 环境中实现。

In [6]:

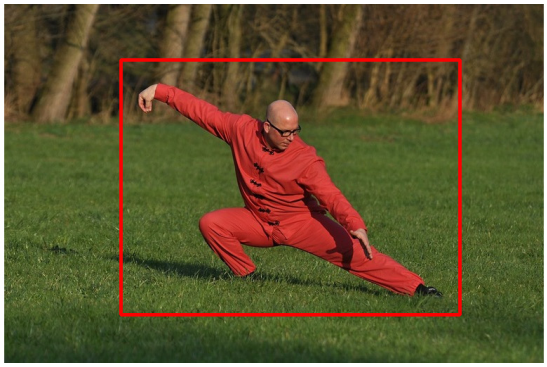
```
# 以下代码只能在本地执行
# 捕获用户在弹出窗口的图像中画出的矩形边框
# 自动将边框参数提取出来，通过rect变量传递给 x，y，w，h
# 之后，在下一步中被赋值后的x，y，w，h代入cv2.rectangle在图像上画出边框
# 另外，由于我们在cv2.selectROI函数中采用cv2.GC_INIT_WITH_RECT模式
# 变量`rect`将把x，y，w，h参数作为蒙版的位置与尺寸之传递给cv2.selectROI函数

# 选择 ROI
# rect = cv2.selectROI(img)

# 获取矩形边框参数
# x,y,w,h = rect
```

在这里，我们使用手工指定 x， y， w， h 四个参数的折中办法来规避选择图像 ROI 无法在 Notebook 中运行的问题。首先，使用matplotlib显示原始图像：

In [7]:

```
# 将图像转换为RGB
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB);
# 指定显示转换后的图像
imgplot = plt.imshow(img)
# 指定图像标题
plt.title('Original image')
# 显示图像
plt.show()
```

由于 matplotlib 的输出图带有坐标轴，根据坐标轴，我们便能获得如下图所示 —— 本应用鼠标在图像上选定的 ROI 区域 x，y，w，h 参数值。



In [8]:

```
x = 180          #（ROI原点，即左上角X坐标）
y = 100          #（ROI原点，即左上角Y坐标）
w = 620-180      #（ROI右上角X坐标-ROI原点X坐标=ROI区域的宽：w）
h = 420-100      #（ROI左下角Y坐标-ROI原点Y坐标=ROS区域的高：h）
```

手工指定 x， y， w， h 后，反过来将对应的值赋给 rect 如此一来， rect 便能作为输入参数应用到 cv2.grabCut 函数中。

In [9]:

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

5. 绘制 ROI 区域并显示

In [10]:

```
# 使用红色线条，在原始图像拷贝副本 `imgCopy` 上绘制 ROI 区域的矩形边框：
cv2.rectangle(imgCopy, (x, y), (x+w, y+h), (0, 0, 255), 3)
```

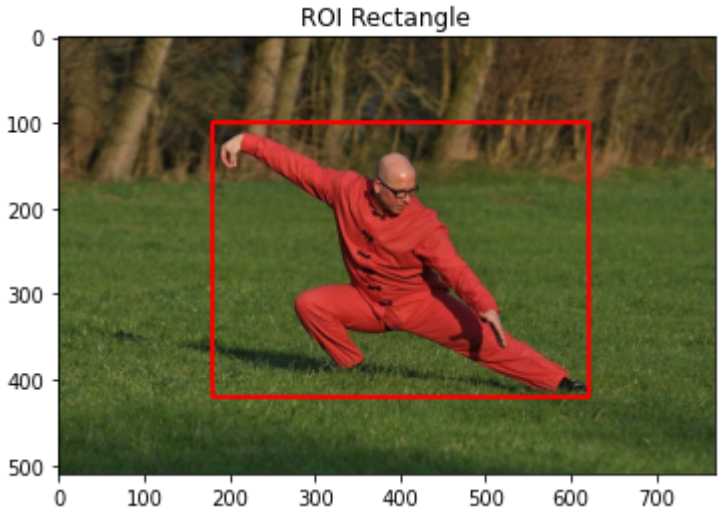
Out[10]:

```
array([[[34, 60, 67],
        [31, 57, 64],
        [29, 54, 64],
        ...,
        [30, 56, 63],
        [32, 55, 63],
        [32, 55, 63]],

       [[34, 60, 67],
        [34, 60, 67],
        [33, 58, 68],
        ...,
        [28, 54, 61],
        [32, 54, 65],
        [32, 55, 63]],

       [[36, 61, 71],
        [35, 60, 70],
        [33, 58, 68],
        ...,
        [28, 53, 63],
        [33, 55, 67],
        [34, 56, 67]],

       ...,

       [[37, 55, 38],
        [38, 56, 39],
        [35, 54, 37],
        ...,
        [54, 99, 82],
        [51, 95, 78],
        [46, 90, 73]],

       [[36, 55, 36],
        [37, 56, 37],
        [35, 55, 36],
        ...,
        [53, 98, 82],
        [50, 95, 79],
        [46, 91, 75]],

       [[35, 54, 35],
        [37, 56, 37],
        [35, 55, 36],
        ...,
        [50, 97, 81],
        [48, 93, 77],
        [43, 88, 72]]], dtype=uint8)
```

保存图像并显示，确认根据手工指定的 x， y， w， h 参数画出来的 ROI 区域与我们用鼠标划出的区域一致。（可能需要经过几次参数调整）

In [11]:

```
# 保存图像到本地存储，您可以随时在本地存储查看
cv2.imwrite(result_path+"roi-exe03.png",imgCopy)

# 将图像转换为RGB
img = cv2.cvtColor(imgCopy,cv2.COLOR_BGR2RGB);
# 指定显示转换后的图像
imgplot = plt.imshow(img)
# 指定图像标题
plt.title('ROI Rectangle')
# 显示图像
plt.show()
```



6. 执行 GrabCut 算法

In [12]:

```
# 执行grabcut操作
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)
```

Out[12]:

```
(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([[ 3.94398365e-01,  2.34201740e-01,  2.16355544e-02,
          2.05112925e-01,  1.44651416e-01,  8.04444163e+01,
          9.88648887e+01,  4.41578624e+01,  5.41538855e+01,
          5.45493128e+01,  2.99870463e+01,  2.55000000e+02,
          0.00000000e+00,  0.00000000e+00,  7.40202955e+01,
          7.54091636e+01,  3.60946801e+01,  1.07171312e+02,
          9.77337578e+01,  5.58164866e+01,  4.57624068e+01,
          4.00712586e+01,  3.37297465e+01,  4.00712586e+01,
          4.14971907e+01,  3.20160659e+01,  3.37297465e+01,
          3.20160659e+01,  3.58975363e+01,  1.73502050e+02,
          4.72461156e+01,  4.75241449e+01,  4.72461156e+01,
          1.27798403e+02,  5.07051762e+01,  4.75241449e+01,
          5.07051762e+01,  5.71828025e+01,  9.99999999e-03,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          1.00000000e-02,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  1.00000000e-02,  1.31145396e+02,
         -4.61903107e+01,  4.20735333e+01, -4.61903107e+01,
          7.07575612e+01,  4.09073477e+00,  4.20735333e+01,
          4.09073477e+00,  4.66306370e+01,  2.51220061e+02,
          1.10950404e+02,  1.36297818e+02,  1.10950404e+02,
          3.11566721e+02,  2.01858734e+02,  1.36297818e+02,
          2.01858734e+02,  1.77178970e+02]]),
 array([[8.38846708e-02, 6.42513611e-01, 2.59734870e-01, 0.00000000e+00,
         1.38668490e-02, 1.54269208e+02, 8.79534342e+01, 7.15334692e+01,
         1.96926894e+02, 6.03645794e+01, 5.43231629e+01, 9.49621205e+01,
         1.93928001e+01, 1.68690666e+01, 2.55000000e+02, 0.00000000e+00,
         0.00000000e+00, 7.40651408e+01, 5.44014085e+01, 3.43785211e+01,
         6.89668214e+02, 5.55793386e+02, 5.12034209e+02, 5.55793386e+02,
         9.73210870e+02, 8.21712851e+02, 5.12034209e+02, 8.21712851e+02,
         7.76323385e+02, 2.94063658e+02, 1.92964110e+02, 1.90885811e+02,
         1.92964110e+02, 4.27082030e+02, 3.64078625e+02, 1.90885811e+02,
         3.64078625e+02, 3.27099039e+02, 1.37984131e+03, 2.74083871e+02,
         2.39576956e+02, 2.74083871e+02, 1.37849562e+02, 1.22671884e+02,
         2.39576956e+02, 1.22671884e+02, 1.40613837e+02, 1.00000000e-02,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e-02,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e-02,
         3.31758081e+02, 3.54435120e+02, 3.11623230e+02, 3.54435120e+02,
         5.43779012e+02, 3.40177284e+02, 3.11623230e+02, 3.40177284e+02,
         3.57354961e+02]]))
```

根据 GrabCut 算法的返回值，对掩码进行选择，mask 只能取以下四种值：

- GCD_BGD（= **0**），背景
- GCD_FGD（= **1**），前景
- GCD_PR_BGD（= **2**），可能的背景
- GCD_PR_FGD（= **3**），可能的前景

下面的代码中，将所有确认的（ 0 ）和可能的背景（ 2 ）像素设置为 0 ，包括在背景分类中:

In [13]:

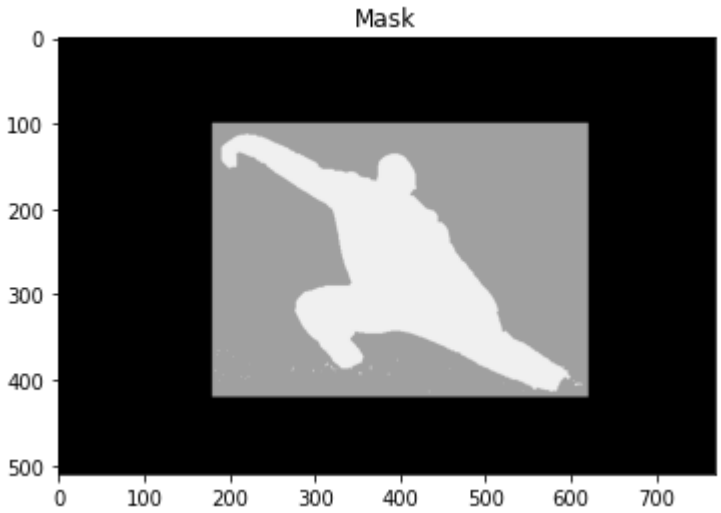
```
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
```

使用以下代码，保存并显示使用 GrabCut 操作获得的掩码 mask ，其中 ——

- 较亮的区域对应于确认的前景像素
- 灰色区域对应于可能的背景像素
- 黑色像素保证属于背景

In [14]:

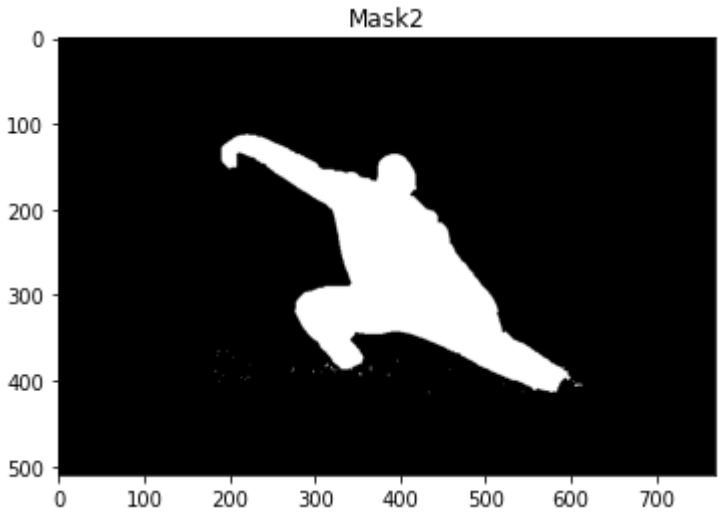
```
# cv2.imshow("Mask",mask*80)
# cv2.imshow("Mask2",mask2*255)
# cv2.waitKey(0)

# 将掩码图像保存在本地存储
cv2.imwrite(result_path+"mask-exe03.png",mask*80)

# 将图像转换为RGB
Mask = cv2.cvtColor(mask*80,cv2.COLOR_BGR2RGB);
# 指定显示转换后的图像
imgplot = plt.imshow(Mask)
# 指定图像标题
plt.title('Mask')
# 显示图像
plt.show()
```



使用以下代码，保存并显示将确认的背景像素和可能的背景像素结合后得到的掩码: mask2 ，白色区域与前景相对应。

In [15]:

```
# 将掩码图像保存在本地存储
cv2.imwrite(result_path+"mask2-exe03.png",mask2*255)

# 将图像转换为RGB
Mask2 = cv2.cvtColor(mask2*255,cv2.COLOR_BGR2RGB);
# 指定显示转换后的图像
imgplot = plt.imshow(Mask2)
# 指定图像标题
plt.title('Mask2')
# 显示图像
plt.show()
```



7. 保留前景并显示

由于蒙版的背景为黑色，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:

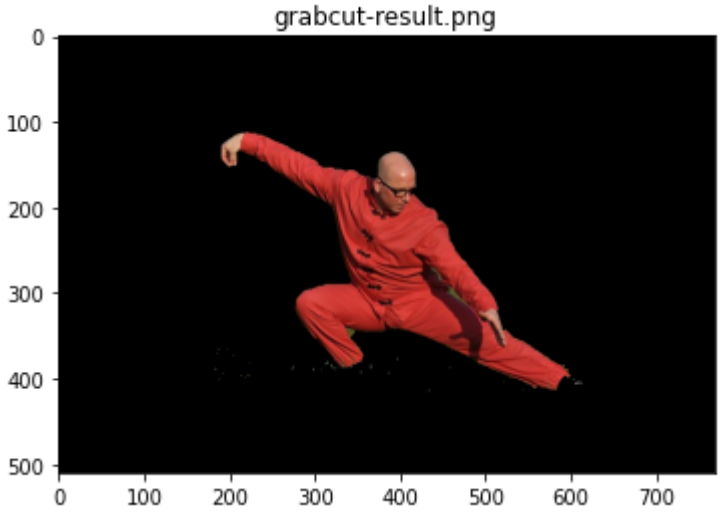
In [16]:

```
# 将蒙版图像与原始图像相乘，并引入一个新的坐标轴
img = img*mask2[:,:,np.newaxis]
```

最后，我们可以保存分割后的人物图像，并在 Jupyter Notebook 中显示结果。

In [17]:

```
# 将图像转换为RGB
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB);
# 将分割后的图像保存在本地存储，方便随时查看
cv2.imwrite(result_path+"grabcut-result-exe03.png",img)
# cv2.imshow("Image",img)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像转换为RGB
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB);
# 指定显示转换后的图像
imgplot = plt.imshow(img)
# 指定图像标题
plt.title('grabcut-result.png')
# 显示图像
plt.show()
```



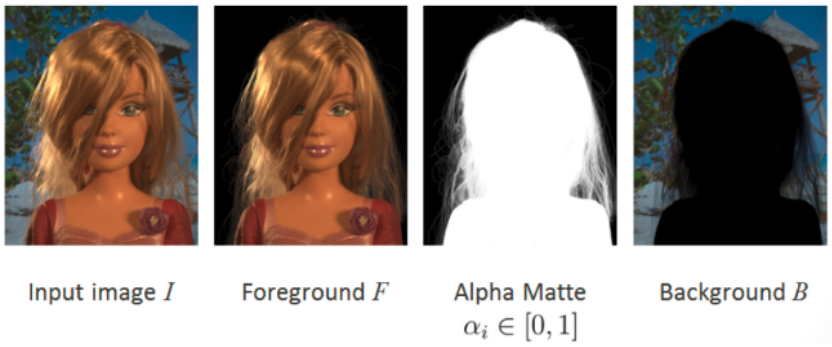
从结果看到，人物的鞋子仍然在背景中，另外，还有一些额外的草被检测到作为前景的一部分。

您可以尝试选择不同的 ROI，看看得到的结果是否有任何不同。

8. 背景替换

我们将人物从图像中分割出来的最终目标，是希望将人物放置到其他背景图片上面，这已经是很多数字图像处理软件的必备功能：

采用 Alpha 通道融合 (Alpha Blending) 是常见的实现方式之一，Alpha 融合是将一张前景透明图像覆盖到一张背景图片上的过程。在 RGB 色彩空间三个通道的基础上，还可以加上一个 `A` 通道，也叫 `alpha` 通道，表示透明度。透明图往往是四通道的图像，如：PNG 透明图，而且是可分离的。其中，透明 mask 图也被叫作 `alpha mask` 或 `alpha matte` 。如下图所示：



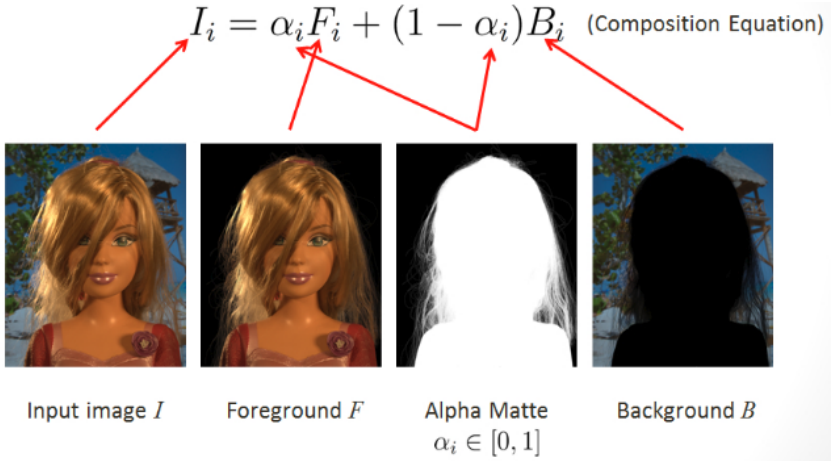
对于每个像素，

- 前景图像颜色 $F = [R_F, G_F, B_F, \alpha]$ 或 $F = [\alpha R_F, \alpha G_F, \alpha B_F]$；
- 背景图像颜色 $B = (1-\alpha)[R_B, (1-\alpha)G_B, (1-\alpha)B_B]]$。

数学问题可以转化为：

$$I_i = F_i + (1 - \alpha_i)\, B_i$$

可以认为图像是由前景和背景合成，在不同的区域， `alpha` 值控制前景和背景占得比例不一样，其取值范围 `[0, 1]` 。



对于图像的每个像素，需要采用 `alpha mask` 将前景图像 F 和背景图像 B 进行组合，其中：$0 \leq \alpha \leq 1$

1. 当 $\alpha = 0$ 时，输出像素值为背景图像
2. 当 $\alpha = 1$ 时，输出像素值为前景图像
3. 当 $0 < \alpha < 1$ 时，输出像素值是背景图像和前景图像的融合，实际场景中， `alpha mask` 边界处的像素值往往在 `[0, 1]` 区间

8.1 将前景图片转换回 BGR

由于之前我们为了使图像在 Matplotlib 中显示，将它转换为 RGB 格式，我们需要将它转换回 BGR 在 OpenCV 中运算。

In [18]:

```
# 将前景图片转换回 BGR
img = cv2.cvtColor(img,cv2.COLOR_RGB2BGR)

# 提取原图片尺寸
h, w, ch = img.shape
```

8.2 导入背景图片并调整尺寸

根据从原图片提取到的高、宽、通道数尺寸，对即将要替换的新背景图片进行尺寸调整。同时，

In [19]:

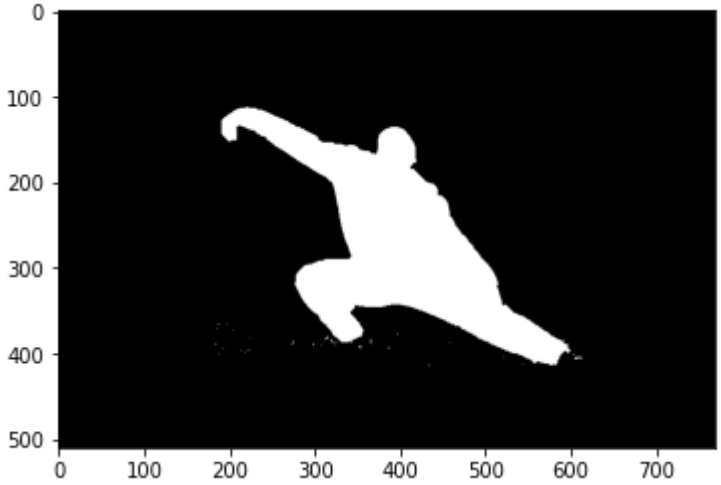
```
# 加载准备替换的新背景图片
background = cv2.imread("./data/tibet_new.jpg")

# 根据原图片调整新背景图片的尺寸
background = cv2.resize(background, (w, h))

# 增加高斯模糊效果
background = cv2.GaussianBlur(background, (5, 5), sigmaX=15)
```

8.3 掩码预处理

在执行 Alpha blending 融合前，我们需要对 `alpha mask` 执行一定的预处理。在本实验中，由于我们此前已经执行了 GrabCut 前后景分离，因此，我们可以直接使用 `mask2` 作为 `alpha mask`：

In [20]:

```
# 提取前景和可能的前景区域
mask2 = np.where((mask==1) + (mask==3), 255, 0).astype('uint8')

# 输出 alpha mask
plt.imshow(cv2.cvtColor(mask2, cv2.COLOR_BGR2RGB))
plt.show()
```



↑ 根据 GrabCut 算法的返回值，对掩码进行选择，mask 只能取以下四种值：

- GCD_BGD（= **0**），背景
- GCD_FGD（= **1**），前景
- GCD_PR_BGD（= **2**），可能的背景
- GCD_PR_FGD（= **3**），可能的前景

In [21]:

```
# 对 alpha mask 进行膨胀，使白色的前景物体颜色面积变大
# 生成 3x3 的矩形核进行滑动
se = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
# 使用矩形核对掩码执行膨胀操作
cv2.dilate(mask2, se, mask2)

# 对 alpha mask 执行高斯模糊，主要为了避免在两幅图之间的融合边缘变得突兀
mask2 = cv2.GaussianBlur(mask2, (5, 5), sigmaX=5)

# 归一化 alpha mask，保证 alpha mask 在 [0,1]之间
mask2 = mask2/255.0

# 补全 alpha 通道维度
alpha =  mask2[..., None]
```

8.4 图片融合

现在我们已经有了前景图片、 alpha mask 、以及新的替换背景图片，可以根据 Alpha Blending 数据公式融合图片：$I = \alpha F + (1 - \alpha)B$。

注意：这里要将前景图像与背景图像转为 float32 数据类型，代表 32 位图像，是在 RGB 图像基础上增加了 α 通道，代表还存在 256 级透明度。

In [22]:

```
# 转为 float32 进行融合运算
result = alpha* (img.astype(np.float32)) + \
    (1 - alpha) * (background.astype(np.float32))

# 完成计算后再转换为 unit8
result = result.astype(np.uint8)
```

保存并显示融合后的新图片：

实验小结

在本实验中，您了解了图像分割基本概念，GrabCut 的技术应用价值，通过选择 ROI，将人物形象从图像中分割出来，了解 GrabCut 如何实现前后景的分割。最后，你还通过 Alpha Blending 图像融合技术，实现了背景替换。

In [23]:

```
cv2.imwrite(result_path+"grabcut-background.png",result)
plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.show()
```