Final Report

组员分工

- 葛宇泽（19307130176）：GAC算法和Chan-Vese算法及对应报告部分
- 邵诣（19307130113）：Graph Cut算法及对应报告部分
- 邵彦骏（19307110036）：GUI设计及使用手册部分

方法简介

本学期学习到的图像分割算法如 OTSU、KMeans、GMM都是通过设置阈值，根据像素值与阈值的大小关系，将像素点分为不同的类别。这些分类方法都是基于图像的像素值分布（直方图），而忽略了图像本身的区域等结构信息（如图像中两个区域之间可能存在明显的分界线）。在本次项目中，我们实现了Geodesic Active Contour、Chan-Vese model两类使用level-set的分割算法，以及基于最大流最小割原理实现的Graph Cut算法。

将图像看作在连续域中是前两种算法的理论基础。前两种算法使用闭合曲线将图像分割为曲线内部和曲线外部两部分。从设置初始曲线开始，算法检测曲线附近的图像梯度或区域差异，并逐步更新曲线，使曲线沿着图像滑动，最终"吸附"在分割区域的边界上。另外，我们还可以根据图像的特点，手动设置初始的曲线的位置。这样，图像分割的结果能够更加符合我们的意愿。第三种算法定义了一种损失函数，利用最小割算法寻找二分类损失最小的分割方式，以此实现算法分割

Level-Set Method

数字图像是由一个个像素点离散存储的。接下来，我们将图像视为连续的对像，在连续域中设计和分析算法，而在实现算法时使用数值计算等离散的方法实现。

曲线的隐式表示

根据隐映照定理，$\Gamma = \left\{x \in \mathbb{R}^m \mid \phi(x) = 0 \in \mathbb{R}^{m-1}\right\}$ 可以确定$\mathbb{R}^m$ 中的曲线 C。

那么 $\Gamma = \left\{x \in \mathbb{R}^2 \mid \phi(x) = 0 \in \mathbb{R}\right\}$ 就确定了$\mathbb{R}^2$ 中的曲线。

对于闭合曲线，对图像域中的任意点 (x, y) 有

$$\phi(x,y) = \begin{cases} > 0 & \text{for } (x,y) \in \Omega^+ \\ = 0 & \text{for } (x,y) \in \Omega_0 \\ < 0 & \text{for } (x,y) \in \Omega^- \end{cases}$$

Ω^-: 曲线内部

Ω^+: 曲线外部

Ω_0: 曲线上

可以证明，曲线的法向量为：$\vec{N} = -\frac{\nabla\phi}{|\nabla\phi|}$

曲线的曲率为：$\kappa = \operatorname{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right)$

曲线的演化

我们希望，曲线是随时间变化的（受变化的力的影响），并最终能够吸附在我们想要分割的区域的边界上。

记$\mathbf{z}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$，那么曲线 C 可以表示为 $\phi(\mathbf{z}(t), t) = 0$

上式对z求偏导数，得到: $\nabla\phi \cdot \frac{\partial \mathbf{z}(t)}{\partial t} + \frac{\partial\phi}{\partial t} = 0$

图像域中的力场对曲线位置的影响可以表示为：$\frac{\partial \mathbf{z}(t)}{\partial t} = F\vec{N}$

作用在曲线 C 方向上的力不会改变曲线的位置。

上式化简得到 *level set equation*：$\frac{\partial\phi}{\partial t} = -F\|\nabla\phi\|$

在实际曲线更新中(时间离散)，使用下式进行迭代计算: (这里假设F与ϕ相关)\

$$\phi^{n+1} = \phi^n - \Delta t\left\{F\left(\phi^n\right)\|\nabla\phi^n\|\right\}$$

Geodesic active contour (GAC)

原理

这里，力场 F 基于曲线 C 的曲率和图像的边界。

设曲线 C 的参数形式为：$\mathbf{c}_s$，W随图像梯度的增大而减小，称为 edge-indication function 。

GAC希望最小化泛函

$$E(\mathbf{c}(s)) = \frac{\alpha}{2}\int_0^1 \left\|\mathbf{c}'(s)\right\|^2 ds + \lambda\int_0^1 W(\|\nabla f(\mathbf{c}(s))\|)^2 ds$$

即，在使曲线 C 长度尽可能小的同时，让曲线两侧图像的梯度尽可能大。

使用 level-set 方法求解，可以证明：

$$\frac{\partial\phi}{\partial t} = (c+\kappa)W\|\nabla\phi\| + \nabla\phi\cdot\nabla W$$

$$F = -\nabla\cdot\left(W\frac{\nabla\phi}{\|\nabla\phi\|}\right) - cW$$

将F代入上面的迭代方程即可求解ϕ。

实现

准备工作

- signed distance functions

  实际中，ϕ 使用 符号距离函数，其定义如下:

$$\phi(x,y)=\begin{cases}D(x,y)>0 & \text{if } (x,y)\in\Omega^+\\ 0 & \text{if } (x,y)\in\Omega_0\\ -D(x,y)>0 & \text{if } (x,y)\in\Omega^-\end{cases}$$

  式中 $D(x,y)$ 表示点 (x,y) 到曲线 C 上的点的最小距离.

  易见，这样的 ϕ 满足: $\|\nabla\phi(x,y)\|=1$

- edge-indication function

  使用: $g(x)=\frac{1}{1+\|\nabla G_\sigma I^*(x)\|^2}$ 作为edge-indication function

  式中，$I(x)$表示图像，G_σ表示高斯卷积核

  使用高斯核先进行卷积操作是为了减少噪声的影响。

  具体代码如下:

```
def create_g(img):
    G = guassion_kernel(3,3) # 高斯卷积核
    Gx, Gy = np.gradient(G)  # 梯度
    img2 = padding(img, G)   # 填充图像
    g = conv2d(img2, Gx)**2 + conv2d(img2, Gy)**2
    g = g[1:-1, 1:-1]
    return 1/(1+g)
```

  使用 $3\times3,\sigma=1$ 的高斯核进行操作。式中用到$\nabla(G_\sigma I^*(x))=\nabla G_\sigma I^*(x)$

  guassion_kernel, padding, conv2d 都是之前作业中实现过的函数，这里不再展示。

- 梯度计算

  这里，我在x,y两个方向上分别计算了三种梯度: $\phi(x+1)-\phi(x)$, $\phi(x)-\phi(x-1)$, $(\phi(x+1)-\phi(x-1))/2$

  代码如下: （仅展示x方向，y方向几乎相同）

```
def Dx_p(phi):
    A = shift_L(phi)
    return A - phi

def Dx_m(phi):
    B = shift_R(phi)
    return phi - B

def Dx_o(phi):
    A = shift_L(phi)
    B = shift_R(phi)
    return (A-B)/2
```

其中，函数 *shift_L* 表示将矩阵向左滑动一个列，最后一列复制倒数第二列。

相似的函数还有 *shift_R, shift_U, shift_D* ,都是将矩阵向某方向滑动一个单位，便于梯度的计算。

下面展示 *shift_L* 代码：（其余函数几乎相同）

```
def shift_L(phi):
    A = np.roll(phi, -1, axis=1)
    A[:, -1] = phi[:, -1]
    return A
```

另外，还需要计算二阶偏导数 *Dxx, Dxy, Dyy*,

根据公式：$\phi_{xx} = \phi(x+1,y) - 2\phi(x,y) + \phi(x-1,y)$
$\phi_{yy} = \phi(x,y+1) - 2\phi(x,y) + \phi(x,y-1)$
$\phi_{xy} = \frac{1}{4}[\phi(x+1,y+1) - \phi(x-1,y+1) - \phi(x+1,y-1) + \phi(x-1,y-1)]$
代码如下：

```
def Dxx(phi):
    A = shift_L(phi)
    B = shift_R(phi)
    return A + B - 2*phi

def Dyy(phi):
    C = shift_U(phi)
    D = shift_D(phi)
    return C  + D - 2*phi

def Dxy(phi):
    D_xy = (shift_R(shift_D(phi)) - shift_L(shift_D(phi)) +
shift_L(shift_U(phi)) - shift_R(shift_U(phi))) / 4
    return D_xy
```

- 曲率计算

$$\kappa = \operatorname{div}\left(\frac{\nabla\phi(x,y))}{\|\nabla\phi(x,y))\|}\right) = \nabla\cdot\left(\frac{\nabla\phi(x,y)}{\|\nabla\phi(x,y)\|}\right) = \frac{\phi_{xx}\phi_y^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{yy}\phi_x^2}{(\phi_x^2+\phi_y^2)^{3/2}}$$

```
def curvature(phi):
    # 计算梯度
    phi_x = Dx_o(phi)
    phi_y = Dy_o(phi)
    phi_xx = Dxx(phi)
    phi_yy = Dyy(phi)
    phi_xy = Dxy(phi)
    # 计算梯度的模长
    phi_cube = (phi_x * phi_x + phi_y * phi_y) ** 1.5
    kappa = (phi_xx * phi_y * phi_y - 2 * phi_y * phi_x * phi_xy +
phi_yy * phi_x * phi_x) /(phi_cube + (phi_cube < 1e-4))
    return kappa
```

lsm_evolution 迭代计算曲线演化

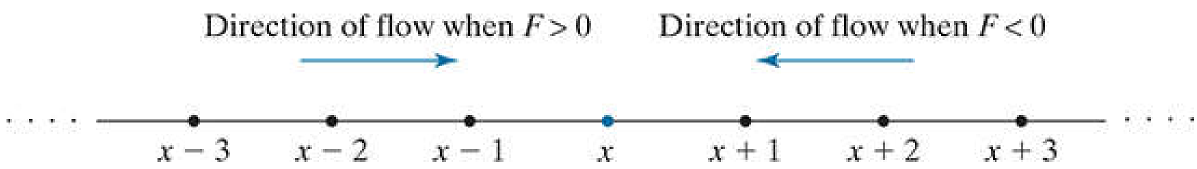
函数 *lsm_evolution* 计算了 多次迭代后ϕ的值，参数如下：

```
:param phi: 水平集函数 phi
:param a: -图像灰度值矩阵g（经过高斯核模糊）
:param u: g对 x 的偏导数
:param v: g对 y 的偏导数
:param b: 图像灰度值矩阵
:param iter: 迭代次数
:param alpha: 用于调整 CLF 条件
:return: 迭代后的水平集函数 phi
```

首先，需要考虑迭代方程在离散空间坐标下的计算。

- 空间离散情况下的迭代中的梯度

  考虑一维情况下的 *level set equation*：$\frac{\partial\phi}{\partial t}+F\frac{\partial\phi}{\partial x}=0$



假设 $\frac{\partial\phi}{\partial x}$ 非负 。如果$F>0$, 那么满足$\phi(x)=0$ 的x 将沿数轴正向移动。但此时刻我们并不知道$\phi(x+1)$, 故此时刻的导数只能使用$\phi(x)-\phi(x-1)$。

二维情况同理，我们需要根据不同的情况选择不同的梯度计算方法，这是前面计算梯度时实现了多种方法的原因。

记：

$$
\begin{aligned}
D_x^+ &= \phi(x+1,y)-\phi(x,y)\\
D_x^- &= \phi(x,y)-\phi(x-1,y)\\
D_y^+ &= \phi(x,y+1)-\phi(x,y)\\
D_y^- &= \phi(x,y)-\phi(x,y-1)
\end{aligned}
$$

$$
\|\nabla\phi\|^+ = \left(\left[\max\left(D_x^-,0\right)\right]^2+\left[\min\left(D_x^+,0\right)\right]^2+\left[\max\left(D_y^-,0\right)\right]^2+\left[\min\left(D_y^+,0\right)\right]^2\right)^{1/2}
$$

$$
\|\nabla\phi\|^- = \left(\left[\max\left(D_x^+,0\right)\right]^2+\left[\min\left(D_x^-,0\right)\right]^2+\left[\max\left(D_y^+,0\right)\right]^2+\left[\min\left(D_y^-,0\right)\right]^2\right)^{1/2}
$$

函数 *lsm_evolution* 是曲线演化函数，其中计算了以上梯度，我们选取了部分代码，如下：

```
phi_x_p = Dx_p(phi)   # Dx +
phi_x_m = Dx_m(phi)   # Dx -
phi_x_o = Dx_o(phi)   # Dx  [f(x+1)-f(x-1)]/2
phi_y_p = Dy_p(phi)   # Dy +
phi_y_m = Dy_m(phi)   # Dy -
phi_y_o = Dy_o(phi)   # Dy
```

```
T = a.copy() # 图像I
T[T>0] = 1
T[T<0] = -1

T_max_0 = np.where(T > 0, T, 0)
T_min_0 = np.where(T < 0, T, 0)
phiX_maxM_sq = np.where(phi_x_m > 0, phi_x_m, 0) ** 2   # max(Dx-,0)**2
phiX_minM_sq = np.where(phi_x_m < 0, phi_x_m, 0) ** 2   # min(Dx-,0)**2
phiX_maxP_sq = np.where(phi_x_p > 0, phi_x_p, 0) ** 2   # max(Dx+,0)**2
phiX_minP_sq = np.where(phi_x_p < 0, phi_x_p, 0) ** 2   # min(Dx+,0)**2

phiY_maxM_sq = np.where(phi_y_m > 0, phi_y_m, 0) ** 2   # 同上,对y
phiY_minM_sq = np.where(phi_y_m < 0, phi_y_m, 0) ** 2
phiY_maxP_sq = np.where(phi_y_p > 0, phi_y_p, 0) ** 2
phiY_minP_sq = np.where(phi_y_p < 0, phi_y_p, 0) ** 2

phi_x_sq = T_max_0 * (phiX_maxM_sq+phiX_minP_sq) - T_min_0 *
(phiX_minM_sq+phiX_maxP_sq)   # 计算 phi 的梯度
phi_y_sq = T_max_0 * (phiY_maxM_sq+phiY_minP_sq) - T_min_0 *
(phiY_minM_sq+phiY_maxP_sq)
phi_upwind = np.sqrt(phi_x_sq + phi_y_sq)
```

- 迭代公式的计算

根据以上讨论，迭代方程可以写为：

$$\phi^{n+1} = \phi^n - \Delta t \left\{ \max(F, 0) \|\nabla \phi^n\|^+ + \min(F, 0) \|\nabla \phi^n\|^- \right\}$$

将$F = -\nabla \cdot \left(W \frac{\nabla \phi}{\|\nabla \phi\|} \right) - cW$ 带入即可求解。

在具体实现中，我们细化了公式：（W和g相同）

$$\nabla \cdot \left(g \frac{\nabla \phi}{\|\nabla \phi\|} \right) = \left(\nabla g \cdot \frac{\nabla \phi}{\|\nabla \phi\|} + g \operatorname{div} \left(\frac{\nabla \phi}{\|\nabla \phi\|} \right) \right) = \frac{1}{\|\nabla \phi\|} ((g_x \phi_x + g_y \phi_y) + \kappa g)$$

这里的$\nabla \phi$ 同样使用了"逆风"形式的离散化梯度实现。

```
u_max0 = np.where(u > 0, u, 0) # gx, gy
u_min0 = np.where(u < 0, u, 0)
v_max0 = np.where(v > 0, v, 0)
v_min0 = np.where(v < 0, v, 0)
# 部分细化公式的实现
convection_upwind = u_max0 * phi_x_m + u_min0 * phi_x_p + v_max0 * phi_y_m
+ v_min0 * phi_y_p # 计算细化公式中的gx*phi_x + gy*phi_y
abs_grad_phi_central = np.sqrt(phi_x_o ** 2 + phi_y_o ** 2)
# 曲率
kappa = curvature(phi)
# 迭代
phi = phi + delta_t * (-a * abs_grad_phi_upwind - convection_upwind + b *
kappa * abs_grad_phi_central)
```

- CLF 条件

  前面提到，使用类似"逆风"形式来计算梯度，这说明梯度的传播速度比水平集的传播速度慢。另外，时间步长也会影响水平集传播的速度。水平集方程迭代得到稳定解的一个必要条件是 Courant-Friedreichs-Lewy (CLF) 条件：

  $$\Delta t < \frac{1}{\max |F|}$$

  同样，代码在函数 *lsm_evolution* 中，我选取了部分代码，如下：

  ```
  # CFL condition
  abs_H1 = np.abs(u + a * np.sqrt(phi_x_sq) / (abs_grad_phi_upwind +
  (abs_grad_phi_upwind == 0)))
  abs_H2 = np.abs(v + a * np.sqrt(phi_y_sq) / (abs_grad_phi_upwind +
  (abs_grad_phi_upwind == 0)))
  max_H1_H2 = np.max(abs_H1 + abs_H2 + 4*b)
  delta_t = alpha / (max_H1_H2 + (max_H1_H2 == 0))
  ```

Reinitialization

由于数值原因，水平集ϕ 在迭代过程中可能无法保证 signed distance function的性质，即 $\|\nabla\phi\| = 1$.因此，水平集函数需要周期性的被重新初始化为满足上述条件的函数。

$S(\tilde{\phi}) = \frac{\tilde{\phi}}{\sqrt{\tilde{\phi}^2+\epsilon^2}}$是 smoothed sign function, 可以用它来对ϕ 进行矫正：

$$\partial_\tau \phi^v + S(\tilde{\phi})\left(|\nabla\phi^v| - 1\right) = 0$$

理论推导比较复杂，我们直接用数值迭代的方法求解该方程，代码如下：

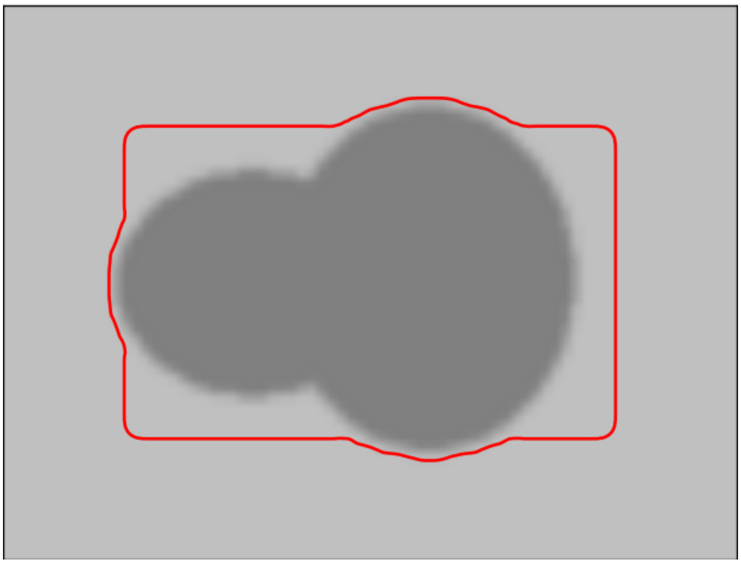
```
def reinitialization(phi, iter):
    S = phi/np.sqrt(phi**2 + 1)    # 计算smoothed sign function,
    for i in range(iter):
        phi_x_p = Dx_p(phi)    # 这里的大部分代码仍在计算"逆风"梯度，和上面展示的代
码几乎是相同的
        phi_x_m = Dx_m(phi)
        phi_y_p = Dy_p(phi)
        phi_y_m = Dy_m(phi)
        T = S.copy()
        T[T>0] = 1
        T[T<0] = -1
        T_max_0 = np.where(T>0, T, 0)
        T_min_0 = np.where(T<0, T, 0)
        phiX_maxM_sq = np.where(phi_x_m>0, phi_x_m, 0)**2
        phiX_minM_sq = np.where(phi_x_m < 0, phi_x_m, 0)**2
        phiX_maxP_sq = np.where(phi_x_p > 0, phi_x_p, 0)**2
        phiX_minP_sq = np.where(phi_x_p<0, phi_x_p, 0)**2
        temp1 = np.where(phiX_maxM_sq > phiX_minP_sq, phiX_maxM_sq,
phiX_minP_sq)
```

```
        temp2 = np.where(phiX_minM_sq > phiX_maxP_sq, phiX_minM_sq,
phiX_maxP_sq)

        phiY_maxM_sq = np.where(phi_y_m > 0, phi_y_m, 0) ** 2
        phiY_minM_sq = np.where(phi_y_m < 0, phi_y_m, 0) ** 2
        phiY_maxP_sq = np.where(phi_y_p > 0, phi_y_p, 0) ** 2
        phiY_minP_sq = np.where(phi_y_p < 0, phi_y_p, 0) ** 2
        temp3 = np.where(phiY_maxM_sq > phiY_minP_sq, phiY_maxM_sq,
phiY_minP_sq)
        temp4 = np.where(phiY_minM_sq > phiY_maxP_sq, phiY_minM_sq,
phiY_maxP_sq)

        phi_x_sq = T_max_0 * temp1 - T_min_0 * temp2
        phi_y_sq = T_max_0 * temp3 - T_min_0 * temp4
        abs_grad_phi = np.sqrt(phi_x_sq + phi_y_sq)
        # 这里使用了上面提到的CLF条件来选取delta_t
        abs_H1 = np.abs(S * np.sqrt(phi_x_sq) / (abs_grad_phi +
(abs_grad_phi == 0)))
        abs_H2 = np.abs(S * np.sqrt(phi_y_sq) / (abs_grad_phi +
(abs_grad_phi == 0)))
        max_H1_H2 = np.max(abs_H1 + abs_H2)

        delta_t = 1 / (max_H1_H2 + (max_H1_H2 == 0))
        # 迭代求解
        phi = phi + delta_t * (-S * abs_grad_phi + S)
    return phi
```

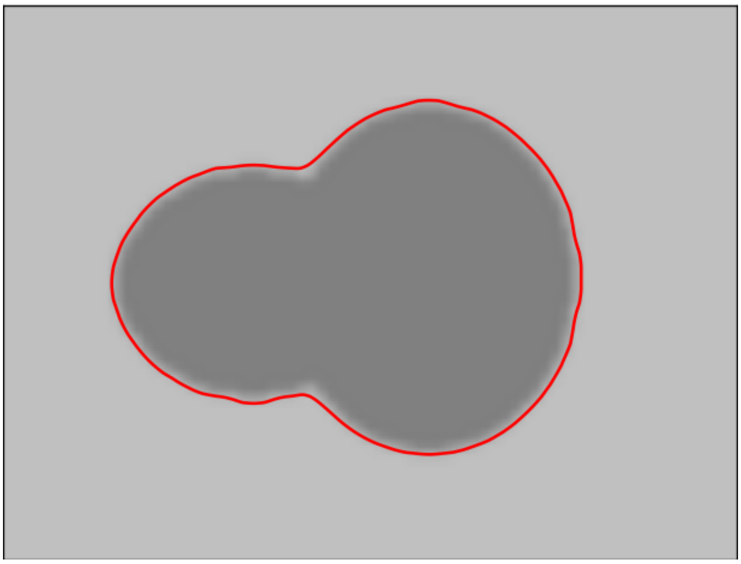
我们用如下图片并选择初始水平集：

结果如下：

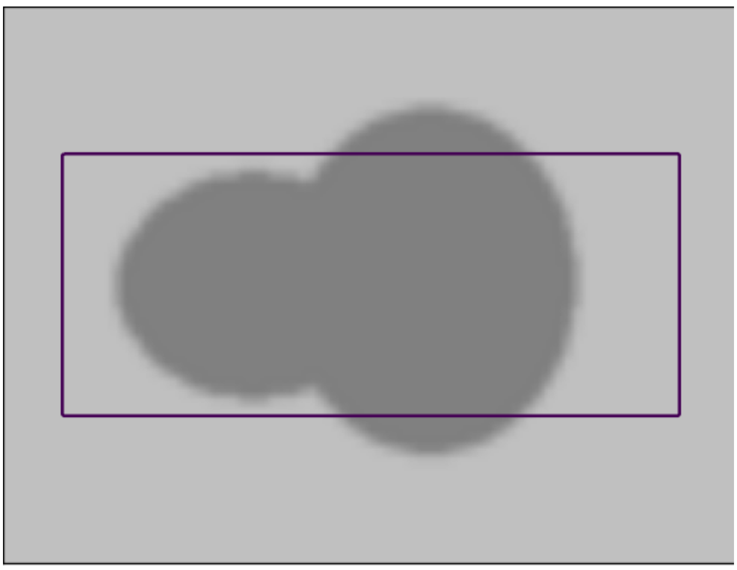
Iteration: 40

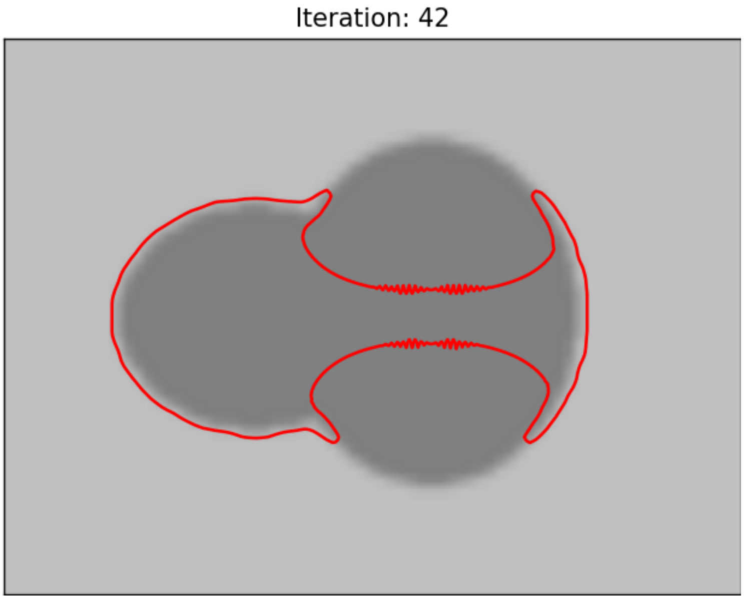


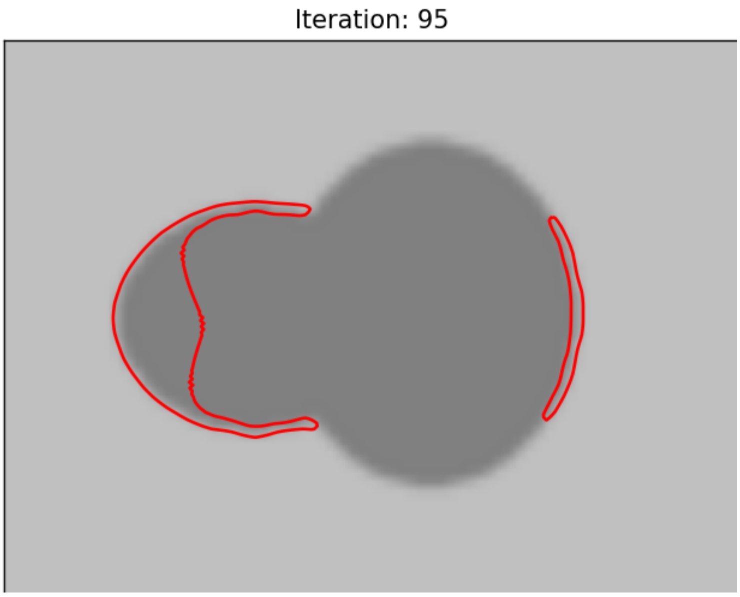
Iteration: 101



但是，如果初始水平集没能框住中心图像的全部，由于目标函数要求最小化曲线长度，会导致曲线最终收缩。







Chan-Vese model

原理

上述 Geodesic active contour 模型对图像的分割是基于图像的梯度，分割的结果受到图像边缘的质量的影响。

Chan-Vese是基于图像的区域进行分割。

首先定义：

分割曲线 $C = \Omega^0 = \{(x,y) \mid \phi(x,y) = 0\}$，并且在曲线内部有 $\phi > 0$，曲线外部有$\phi < 0$。

Heaviside function：$H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$

冲激函数：$\delta(z) = \frac{dH(z)}{dz}$

我们希望最小化以下泛函：

$$
\begin{aligned}
E(C, c_1, c_2) = & \mu \int_\Omega \delta(\phi(x,y)) \|\nabla \phi(x,y)\| dxdy \\
& + v \int_\Omega H(\phi(x,y)) dxdy \\
& + \lambda_1 \int_\Omega (f(x,y) - c_1)^2 H(\phi) dxdy \\
& + \lambda_2 \int_\Omega (f(x,y) - c_2)^2 (1 - H(\phi)) dxdy
\end{aligned}
$$

式中，第一项表示曲线 C 的长度，第二项表示曲线C围成的面积。

后两项假设图像的两个区域的灰度值为常数，并计算各个区域的灰度方差(假设均值为c)。

仅对c_1, c_2 求导，可求出其取值：

$$
\begin{aligned}
c_1 &= \frac{\int_\Omega f(x,y) H(\phi(x,y)) dxdy}{\int_\Omega H(\phi(x,y)) dxdy} \\
c_2 &= \frac{\int_\Omega f(x,y)[1 - H(\phi(x,y))] dxdy}{\int_\Omega [1 - H(\phi(x,y))] dxdy}
\end{aligned}
$$

可以求出：

$$
\frac{\partial \phi}{\partial t} = \delta(\phi)\left[\mu \nabla \cdot \left(\frac{\nabla \phi}{\|\nabla \phi\|}\right) - v - \lambda_1 (f - c_1)^2 + \lambda_2 (f - c_2)^2\right]
$$

故：

$$
F = -\left[\mu \nabla \cdot \left(\frac{\nabla \phi}{\|\nabla \phi\|}\right) - v - \lambda_1 (f - c_1)^2 + \lambda_2 (f - c_2)^2\right]
$$

将F 代入GAC中的迭代公式，即可求解 ϕ

实现

准备工作

CV算法中求梯度等方法都已在GAC算法中实现，所用的函数均为GAC算法中的函数。

cv_evolution 迭代计算曲线演化

函数*cv_evolution* 使用CV算法进行曲线演化，参数如下：

```
:param phi: 水平集函数phi
:param u: 图像灰度值矩阵
:param mu: 公式 E 中，曲线长度项前的系数
:param v: 公式 E 中，曲线围成面积项前的系数
:param lambda1: 公式 E 中，第一个方差项前的系数
:param lambda2: 公式 E 中，第二个方差项前的系数
:param delta_t: 步长
:param iter_num: 迭代次数
:return: 演化后的phi
```

- 计算$\nabla \cdot \left(\frac{\nabla \phi}{\|\nabla \phi\|}\right)$

```
phi_y, phi_x = np.gradient(phi)   # 计算两个方向的梯度
grad_norm = np.sqrt(phi_x**2 + phi_y**2)
phi_x = phi_x/(grad_norm + 1e-6)   # 归一化
phi_y = phi_y/(grad_norm + 1e-6)
Mxx, Nxx = np.gradient(phi_x)   # 计算散度
Nyy, Myy = np.gradient(phi_y)
divergence = Nxx + Nyy
```

- 计算c_1, c_2

```
c1 = np.mean(u[phi>0])
c2 = np.mean(u[phi<0])
```

- 计算 冲激函数

  使用$H(\phi) = \frac{1}{2}\left[1 + \frac{2}{\pi}\arctan\left(\frac{\phi}{\varepsilon}\right)\right]$ 近似 Heaviside function

  冲激函数为 $\delta(\phi) = \frac{dH(\phi)}{d\phi} = \frac{1}{\pi}\frac{\varepsilon}{\varepsilon^2+\phi^2}$

```
delta_phi = eps/(np.pi * (eps**2 + phi**2))
```

- ϕ 的迭代公式

```
phi_t = mu * divergence - v - lambda1 * (u - c1) **2 + lambda2 * (u -
c2) ** 2
phi = phi + delta_t * phi_t * delta_phi
```

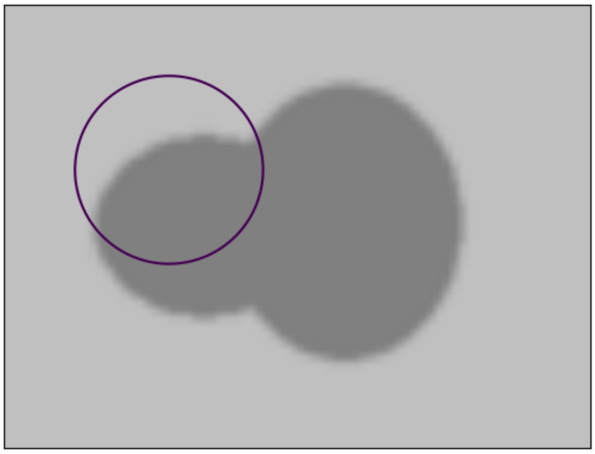
Reinitialization

同样，由于数值原因，水平集ϕ 在迭代过程中可能无法保证 signed distance function的性质，即$\|\nabla \phi\| = 1$

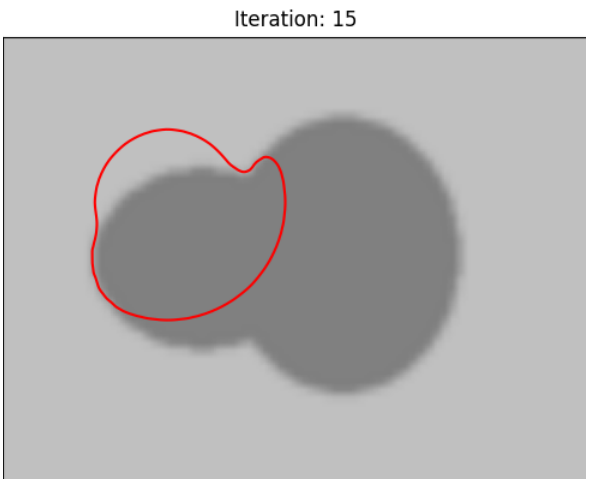
我们使用和GAC中同样的方法对水平集函数进行重新初始化，代码不再展示。

实验结果

我们使用如下图片，并选定初始水平集：



使用Chan-Vese算法，结果如下：



曲线逐渐滑动并包裹了中间的图形。

Graph Cut Method

问题描述

Graph Cut算法是由Yuri Y.Boykov和Marie-Pierre Jolly于2001年提出的一种基于图论的图分割算法。Graph Cut的目标是通过人工交互式分割技术，将图像分割成两个部分：前景(object，简写为obj)和背景(background，简写为bkg)。

在图论观点中，一张图可以被表征为许多像素点p的集合，我们将这个集合称为P。其中每两个相邻的像素点都会产生一个无序像素点对$\{p, q\}$，我们把这些无序像素点对形成的集合称为N。例如，在二维图像中，一个像素点有8个邻居节点，在三维图像中，一个像素点有26个邻居节点。

现在我们假设有一个二值的向量$A = (A_1, \ldots, A_p, \ldots, A_{|P|})$。其中$A_p$对应于图中某像素点的标签，用于标识该像素点属于前景或是背景，如此我们就可以通过A来定义任意一种分割方法。

损失函数设计

基于原论文，我们构建如下的损失函数来量化对A的分割的代价。在所有分割中，代价最小化的分割方式即被称为最小割，并作为Graph Cut的最终结果返回。

$$E(A) = \lambda \cdot R(A) + B(A)$$

其中

$$\begin{aligned} R(A) &= \sum_{p \in \mathcal{P}} R_p(A_p) \\ B(A) &= \sum_{\{p,q\} \in \mathcal{N}} B_{\{p,q\}} \cdot \delta(A_p, A_q) \end{aligned}$$

并有

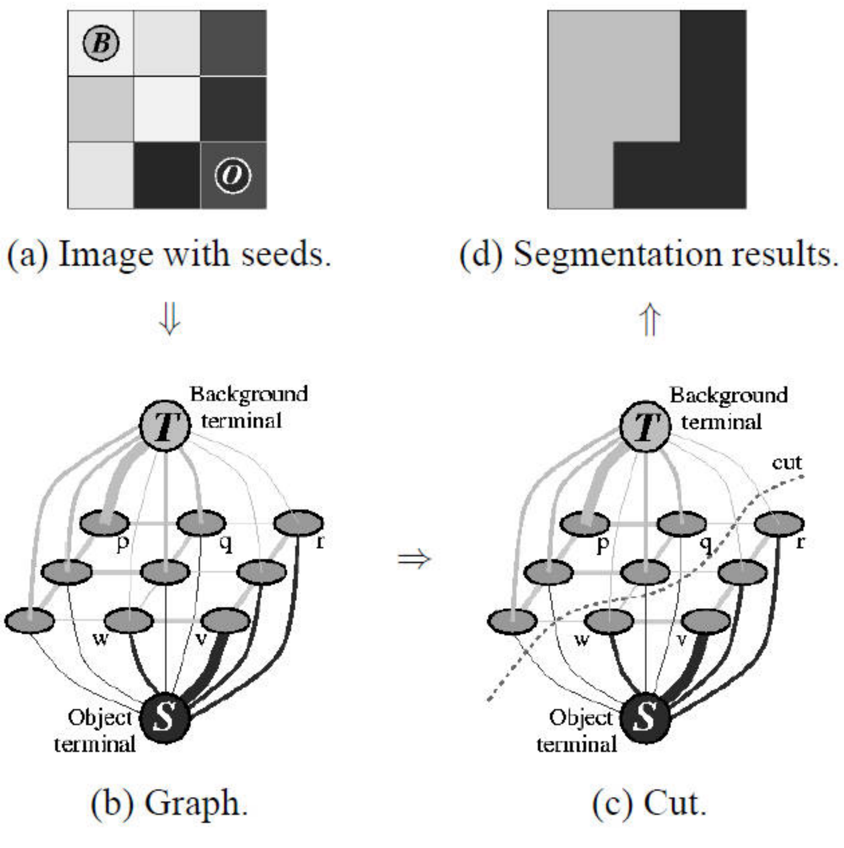
$$\delta(A_p, A_q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{otherwise} \end{cases}$$

由上述损失函数定义可知，损失函数$E(A)$主要分为两项：

- 区域项(region properties)$R(A)$：该项用于描述分割A中每一个像素点的损失总和。其中单个像素点p的损失为$R_p(A_p)$，又因为A_p只有两种取值，所以损失为$R_p(obj)$或$R_p(bkg)$，即像素点p被归为前景的损失或者被归为背景的损失。此外，我们还有$\lambda \geq 0$，系数λ主要用于调节区域项和边界项的比例关系。
- 边界项(boundary properties)$B(A)$：该项用于描述被分割开的边的损失总和，可以理解为将这些边割开所要付出的代价。每一对相邻像素$\{p, q\}$所连接成的边的cost是$B_{\{p,q\}}$。当p和q的差异(比如距离上的差异或者像素值上的差异)很小时，$B_{\{p,q\}}$很大；反之亦然。此外，$\delta(A_p, A_q)$是用于计算，达到将不同标签的像素点分开时才计算该边分割损失的作用。

最大流-最小割模型构建

我们可以将一幅图像理解为一个无向图$G =< V, E >$，其中，V是像素点的集合，E是无向边的集合。如下图所示：



以此为例，上图是一张3×3分辨率的图像，共有9个像素点。作者在原像素点的基础上引入了两个特殊节点，称为终端节点(terminals)。这两个终端节点分别为前景节点(图中S点)和背景节点(图中T点)。同样的，边也可被分为两种，像素点与像素点之间相连接的边称为n-link边，像素点与终端节点相连的边则称为t-link边。联系上一小节对损失函数的分析不难发现，t-link边的损失就对应$R_p(A_p)$，n-link边的损失则为$B_{\{p,q\}}$。

在进行分割操作时，分割路径上的n-link边会断开，其一是属于前景的像素点与T点连接的t-link边断开链接，其二是属于背景的像素点与S点连接的t-link边断开，由此，图像就自然地被分成两部分。这种分割方法的总损失可以用之前定义的损失函数进行计算，最后根据所有可能的分割方式，得到总损失最小的分割方式(最小割)，即等同于求损失函数的最小值。为快速求解惩罚函数的最小值，我们使用最大流-最小割的算法进行求解，此处从略。

区域项的计算

区域项的计算需要用到人工标注信息(seeds)，尔后分别按照前景seeds和背景seeds绘制两个灰度直方图。灰度直方图只需将原始图像转化得到灰度图像即可。

现在，假设有像素点p(非seeds)，其像素值为I_p，根据直方图，可以得到该像素点属于前景的概率为$Pr(I_p|O)$，即在前景灰度直方图中，像素值为I_p的像素点数目除以该直方图所有像素点的数目。同理，可以求得点p属于背景的概率为$Pr(I_p|B)$。对其取负对数，即可得到$R_p(A_p)$的计算公式：

$$R_p(\mathrm{obj}) = -\ln \Pr(I_p \mid \mathcal{O})$$
$$R_p(\mathrm{bkg}) = -\ln \Pr(I_p \mid \mathcal{B})$$

边界项的计算

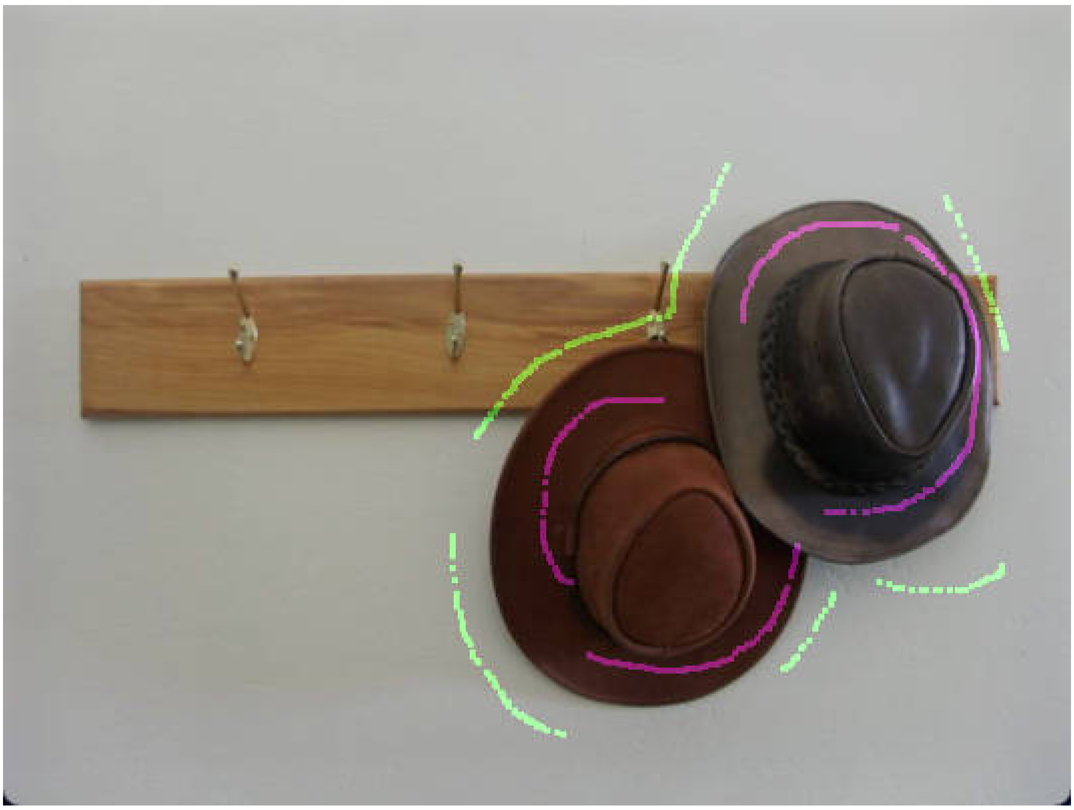
论文给出计算公式如下：

$$B_{\{p,q\}} \propto \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right) \cdot \frac{1}{\mathrm{dist}(p,q)}$$

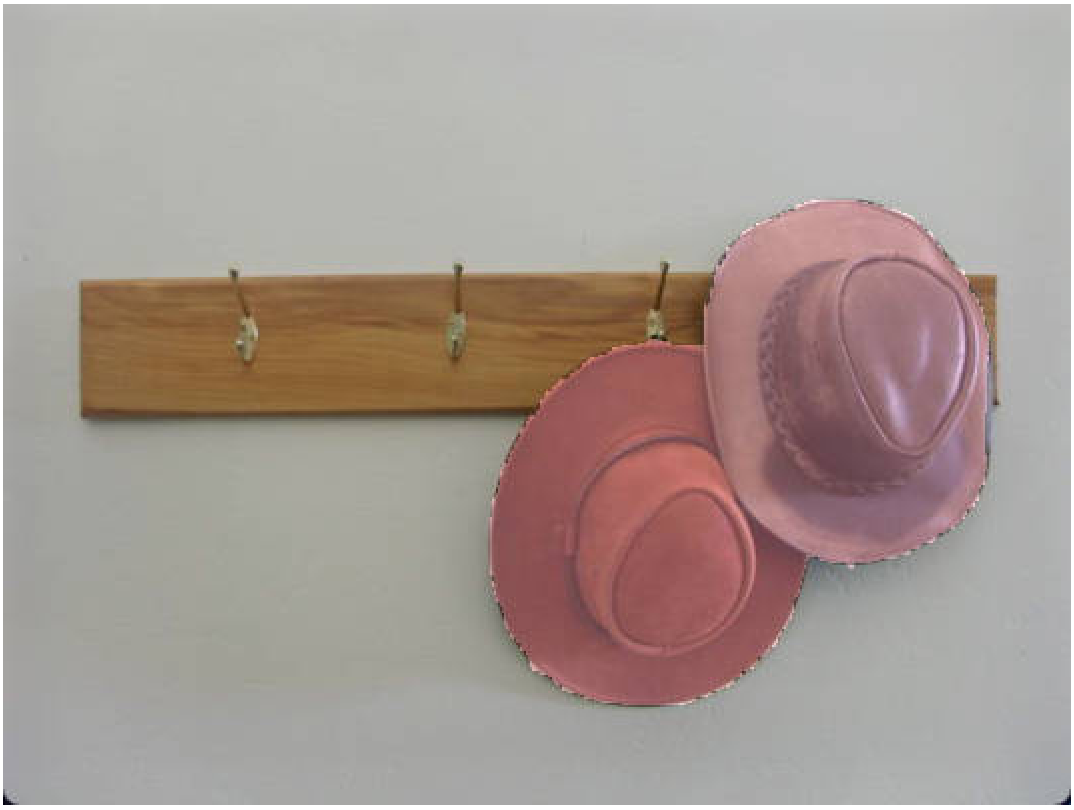
- 第一项主要用于衡量两个相邻像素点的像素值差异，是一个高斯分布，I_p和I_q是相邻像素$\{p,q\}$中p,q的灰度值，σ^2是方差。
- 第二项主要用于衡量两个相邻像素点的距离差异，即$dist(p,q)$，通常采用欧式距离进行计算。

实验结果

如图采样，粉色点为前景，绿色点为背景



算法前景遮罩效果如下



最终分割结果为

图像分割 (Image Segmentation)

开发环境

环境	版本
numpy	1.22.3
matplotlib	3.5.1
pillow	9.0.1
scipy	1.8.0
opencv-python	4.5.3.56
pyqt5	5.15.4
PyMaxFlow	1.2.13

代码结构

- cv.py：Chan-Vese算法的实现，您可以直接运行，在这里的运行速度较快
- gac.py：Geodesic Active Contour算法的实现，您可以直接运行，在这里的运行速度较快
- graph_cut.py：Graph Cut算法的实现
- gc_gui.py：Graph Cut算法的前端，您也可以直接运行
- gui.py：一个整合的前端应用，在这里可以访问三个任务，但执行速度不快
- plot_utils.py：涵盖了选点等操作

运行

考虑到运行速度，我们避免生成可执行文件(.exe)。您可以尝试以下格式运行文件。

```
python xxx.py   #xxx = {cv, gac, gc_gui, gui}
```

使用手册（详见视频展示）

首先安装必要的环境，部分环境等待时间较长

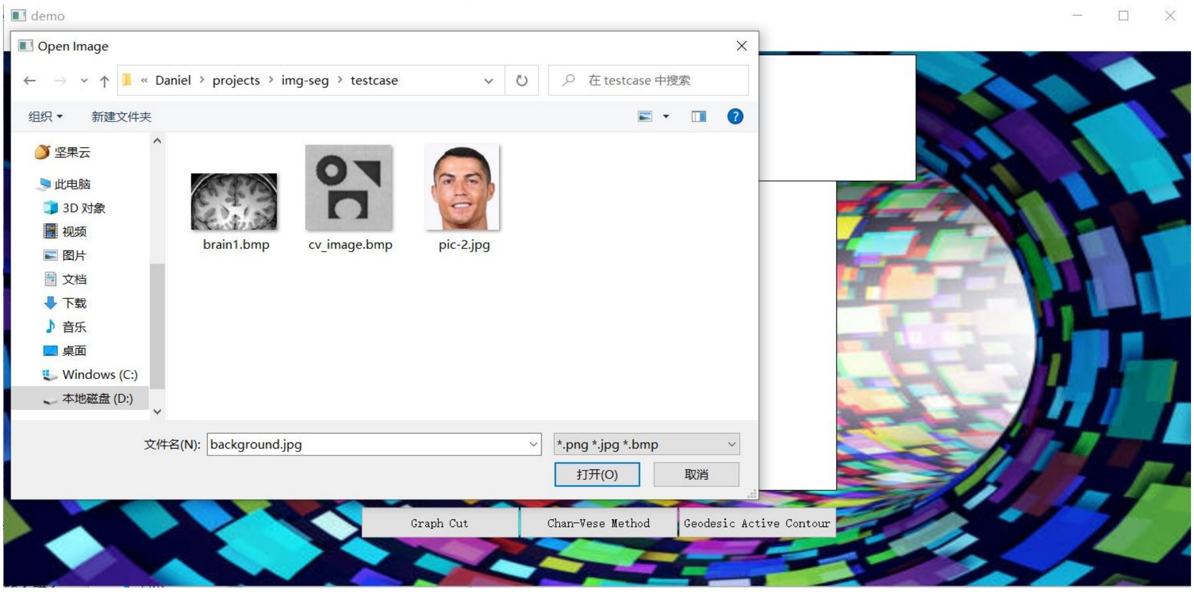
```
pip install -r requirements.txt
```

使用命令行运行GUI

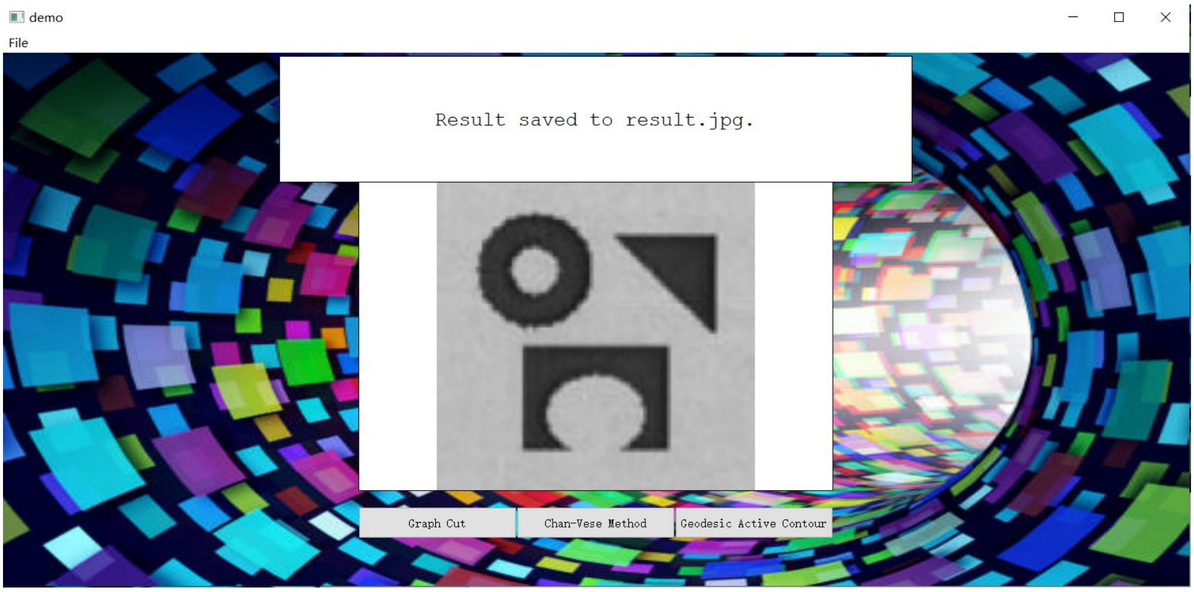
```
python gui.py
```

如左上角所示，您可以点击Load Image或者使用快捷键CTRL+O来读取一张照片

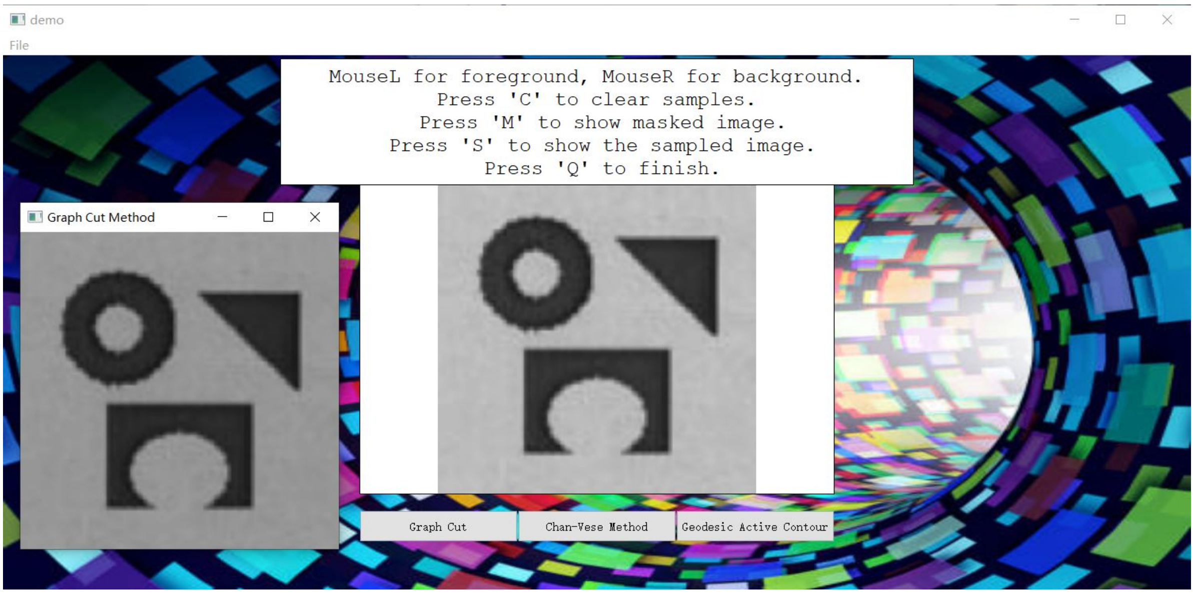
选取您想要的照片，./testcase文件夹中存放了一些可以使用的样例



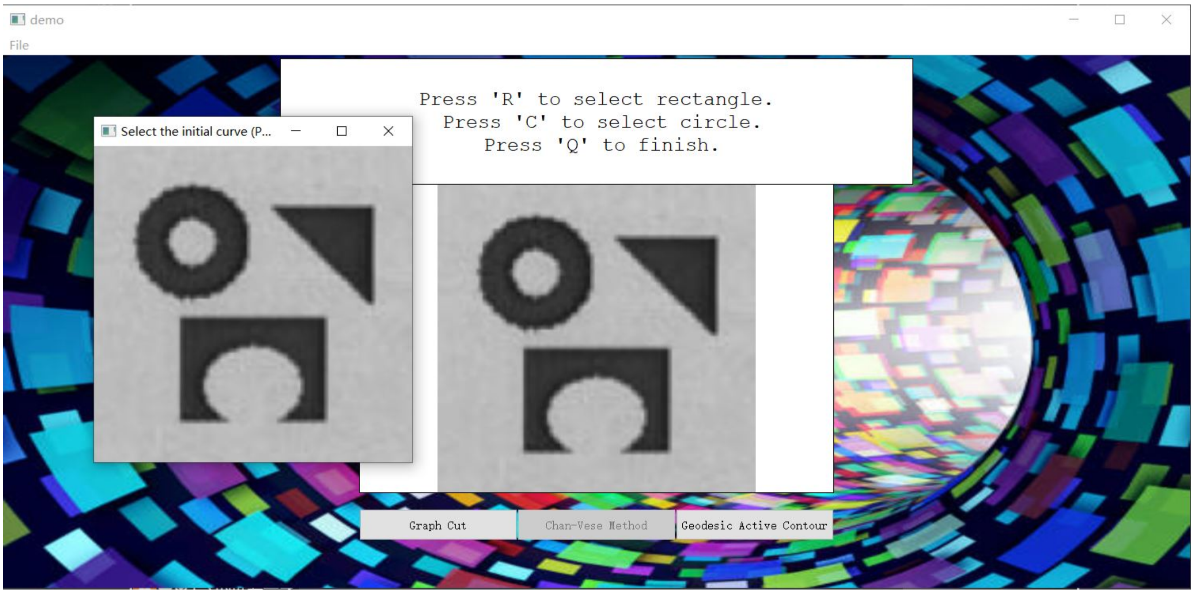
一旦中间的图片展示区有图片，您可以点击Save或者CTRL+S保存图片为./result.jpg



点击Graph Cut按钮，进入Graph Cut算法演示。请按照Msgbox中的指示操作，详见视频。



点击Chan-Vese Method，进入Chan-Vese算法演示。请按照Msgbox中的指示操作，详见视频。



点击Geodesic Active Contour，进入GAC算法演示。请按照Msgbox中的指示操作，详见视频。