ECE 6560 Final Project: Image Segmentation based on Active Contour

Sikai Zhao

April 2020

Contents

1 Introduction

Active contour model, also called snakes, is a framework in computer vision introduced by Michael Kass, Andrew Witkin and Demetri Terzopoulos for delineating an object outline from a possibly noisy 2D image [1]. The active contour model is widely used in many fields of computer vision and image processing, such as segmentation, object tracking, shape recognition, edge detection and stereo matching.

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments to simplify or change the representation of an image into something that is more meaningful and easier to analyze [2]. Hence, the applications of image segmentation is of vital importance to object detection, medical imaging or recognition.

The target of this project is to implement Image Segmentation based on the Active Contour model, which detects the contour of the object in the image. In this way, we can segment the whole image into the meaningful object part and the background part, providing more meaningful information than the original image. For example, the self-driving vehicles need information about the objects in the direction of travel and nearby, such as the pedestrians; the medical imaging devices need to extract the meaningful part of the CT scans, such as the organs; the face recognition algorithm needs to first obtain the face from the image collected.

Boundary-Based Active Contour is used to construct and derive the Partial Differential Equation (PDE). Images of pedestrian, face and lung CT scan, namely the three examples of applications listed above, are used to test and analyze the implementation and performance.

2 Mathematical problem

The basic idea of active contour is to formulate an energy for a curve, and then to derive a PDE that minimizes it using a gradient descent flow. To perform the image segmentation in this project, Boundary-Based Active Contours is used, which is also known as Geodesic Active Contours [3]. It employs ideas from Euclidean curve shortening evolution.

Based on this idea, a flow that forces a curve to wrap around an object needs to be devised. We can use a potential function and obtain the energy based on the integral of it over the curve, i.e.

$$E[C] = \int_C \phi \, ds. \tag{1}$$

where the C denotes the curve, s is the parameter of arc length and ϕ is the potential function. We need to choose ϕ that is small at the desirable locations,

namely the boundary of the object, and large elsewhere.

One of the typical potential function, ϕ, is:

$$\phi = \frac{1}{1 + (||\nabla \mathbf{I}||)^2}. \tag{2}$$

where $\mathbf{I}$ represents the image and $\nabla \mathbf{I}$ is the gradient of the image. This function become very small when near the boundary where the gradient is extremely large, and become large when far from the boundary. And we will also test different potential functions such as $\phi = \frac{1}{1+||\nabla \mathbf{I}||}$, and compare the performance of difference potential functions.

3 Deriving and formulating of the PDE

3.1 Gradient descend

First, take the derivative of equation(1):

$$\begin{aligned}
\frac{\partial}{\partial t} E[C] &= \frac{\partial}{\partial t} \int_C \phi \, ds \\
&= \frac{\partial}{\partial t} \int_0^1 \phi(C(p,t)) \, ||C_p|| \, dp \\
&= \int_0^1 (\phi(C(p,t)) \, ||C_p||)_t \, dp \\
&= \int_0^1 [(\phi(C(p,t)))_t \, ||C_p|| + \phi(C(p,t)) \, ||C_p||_t] \, dp
\end{aligned} \tag{3}$$

The $||C_p||_t$ part can be derived as follow:

$$\begin{aligned}
||C_p||_t &= (\sqrt{C_p \cdot C_p})_t \\
&= \frac{C_{pt} \cdot C_p}{\sqrt{C_p \cdot C_p}} \\
&= \frac{C_{pt} \cdot C_p}{||C_p||}
\end{aligned} \tag{4}$$

Because t represents time and p represents the parameterization, which are independent, so we have $C_{pt} = C_{tp}$. And we also have $C_p = C_s \cdot ||C_p||$. The

equation (3) is transfered as follow:

$$\begin{aligned}
\frac{\partial}{\partial t}E[C] &= \int_0^1 [(\nabla\phi \cdot C_t \, ||C_p|| + \phi \frac{C_{pt} \cdot C_p}{||C_p||}] \, dp \\
&= \int_0^1 [(\nabla\phi \cdot C_t \, ||C_p|| + \phi \frac{C_{tp} \cdot C_p}{||C_p||}] \, dp \\
&= \int_0^1 [(\nabla\phi \cdot C_t \, ||C_p|| + \phi C_{tp} \cdot C_s] \, dp \\
&= \int_0^1 [(\nabla\phi \cdot C_t \, ||C_p|| + \phi C_{ts} \cdot C_s \, ||C_p||] \, dp \\
&= \int_0^1 [(\nabla\phi \cdot C_t + \phi C_{ts} \cdot C_s] \, ||C_p|| \, dp
\end{aligned} \tag{5}$$

Then we can obtain the integral over s:

$$\begin{aligned}
\frac{\partial}{\partial t}E[C] &= \int_C [(\nabla\phi \cdot C_t + \phi C_{ts} \cdot C_s] \, ds \\
&= \int_C [(\nabla\phi \cdot C_t + (\phi C_s) \cdot C_{ts}] \, ds
\end{aligned} \tag{6}$$

The unit tangent $T = \frac{C_p}{||C_p||} = C_s$ and $T_s = -\kappa N$, which is an outward normal, then $(\phi T)_s = \phi_s T + \phi T_s$. And $\phi_s T = \nabla\phi \cdot C_s = \nabla\phi \, T$. Implementing integral by part, we obtain:

$$\begin{aligned}
\frac{\partial}{\partial t}E[C] &= \int_C [(C_t \cdot \nabla\phi - C_t \cdot (\phi C_s)_s] \, ds \\
&= \int_C [C_t \cdot \nabla\phi - C_t \cdot T \, (\nabla\phi \cdot \, T) + C_t \cdot \phi \cdot \kappa \, N] \, ds
\end{aligned} \tag{7}$$

The $\nabla\phi \cdot T$ is the norm of the tangential vector, so $T \, (\nabla\phi \cdot T)$ is the tangential vector. Therefore, $C_t \cdot \nabla\phi - C_t \cdot T \, (\nabla\phi \cdot T) = C_t \cdot N \, (\nabla\phi \cdot N)$ is just the normal vector. So the equation (7) can transform to:

$$\begin{aligned}
\frac{\partial}{\partial t}E[C] &= \int_C C_t \cdot N \, (\nabla\phi \cdot N) + C_t \cdot \phi \cdot \kappa N \, ds \\
&= \int_C C_t \, (\phi \, \kappa + \nabla\phi \cdot N) \, N \, ds \\
&= \langle C_t, \, (\phi \, \kappa + \nabla\phi \cdot N) N \rangle
\end{aligned} \tag{8}$$

To obtain the evolution of the curve, gradient descend is needed, i.e

$$C_t = -\nabla_C E \tag{9}$$

Gradient of a function is the spread vector that the inner production of it and every direction is the same as the derivative. So we have:

$$\frac{\partial}{\partial t}E[C] = \langle C_t, \nabla_C \, E \rangle \tag{10}$$

and,

$$\nabla_C E = (\phi\,\kappa + \nabla\phi \cdot N)N \tag{11}$$

Therefore, the gradient descend flow is:

$$\begin{aligned} C_t &= -\phi\kappa N - (\nabla\phi \cdot N)N \\ &= \beta N,\, where\, \beta = -(\phi\,\kappa + \nabla\phi \cdot N) \end{aligned} \tag{12}$$

Equation (12) is the gradient descend flow and the evolution of the curve can be obtained according to it. The first term of the equation, $-\phi\kappa\, N$, which has relationship with the curvature, is close to zero when the curve is close to the actual contour, and it has the effect of smoothing the curve and slowing down the diffusion. And the second term, $-(\nabla\phi \cdot N))$, is a directional derivative becoming large when near the actual contour and close to zero when far away, it decreases the ϕ.

Besides these two terms, we can add another inflationary term $-\alpha\phi N$ according to the lectures and notes, which should be an erosion flow [4], where $\alpha \in \mathbb{R}$, is a scalar value having no relationship with the curvature. This term is on the normal direction and weighted by the ϕ. When the curve is initialized far from the actual contour, the curvature κ can be small, and the $\nabla\phi$ is also small because the local image is flat. So intuitively, we need the added term to drive the curve to the contour so that the curve will not stop at the initial position itself. And the ϕ in this term will also adjust this term, getting larger when it's far away and smaller when close to the object contour. This term is usually from the external source

Finally the gradient descend flow is constructed as three term, i.e:

$$C_t = -\phi\kappa N - (\nabla\phi \cdot N)N - \alpha\phi\, N \tag{13}$$

3.2 Level set method

If we implement the active contour based on the topology changes of the curve, there are so much work to do and conditions to consider. If the curve become longer, we need interpolation, and when the curves intersect or shocks emerge, it will be harder to deal with. Therefore, the level set method, which can be used. This implicit representation of a moving front is its ability to naturally handle changes in topology without directly dealing with. [5]:

$$\frac{\partial}{\partial t}\psi(x,t) = \frac{\partial}{\partial t}\psi(C(p,t),t) \quad = \nabla\psi \cdot C_t + \psi_t = 0 \tag{14}$$

Then we obtain:

$$\begin{aligned} \psi_t &= -C_t \cdot \nabla\psi \\ &= (\phi\kappa + \nabla\phi \cdot N + \alpha\phi)\nabla\psi \cdot N \end{aligned} \tag{15}$$

For a curve, the normal can be written as:

$$N = \frac{\nabla\psi}{||\nabla\psi||} \tag{16}$$

And also, the curvature κ of a curve can be seen as the divergence of the normal vectors. Because when moving along a curve, if the local curvature is large, then the change of the direction of the normal vectors is quick and vice versa.

$$\kappa = \nabla(\frac{\nabla\psi}{||\nabla\psi||}) \tag{17}$$

Therefore we can obtain the level set version of the equation:

$$\begin{aligned}
\psi_t &= (\phi\kappa + \nabla\phi \cdot N + \alpha\phi)\nabla\psi \cdot N \\
&= (\phi\kappa + \nabla\phi \cdot \frac{\nabla\psi}{||\nabla\psi||} + \alpha\phi)\nabla\psi \cdot \frac{\nabla\psi}{||\nabla\psi||} \\
&= (\phi\nabla(\frac{\nabla\psi}{||\nabla\psi||}) + \nabla\phi \cdot \frac{\nabla\psi}{||\nabla\psi||} + \alpha\phi)\,||\nabla\psi|| \\
&= \nabla\phi \cdot \nabla\psi + \phi\,||\nabla\psi|| \cdot \nabla(\frac{\nabla\psi}{||\nabla\psi||}) + \alpha\phi\,||\nabla\psi||
\end{aligned} \tag{18}$$

4 Discretization and Implementation of the PDE

The PDE we derived is shown as equation 13 and equation 18 (level set representation). To implement it as a computer program or software, the discretization should be performed. We need to choose discretization methods for the three terms of the PDE, which can be seen as a set of three component PDEs:

$$\psi_t = \nabla\phi \cdot \nabla\psi + \phi\,||\nabla\psi|| \cdot \nabla(\frac{\nabla\psi}{||\nabla\psi||}) + \alpha\phi\,||\nabla\psi|| \tag{19}$$

4.1 Advection term

The first term $\nabla\phi \cdot \nabla\psi$ is a transport equation.

$$\begin{aligned}
\psi_t &= \nabla\phi \cdot \nabla\psi \\
&= \phi_x\psi_x + \phi_y\psi_y
\end{aligned} \tag{20}$$

Equation (20) can be used to discrete this term when implementing. It is a 2D version of transport equation, we can deal with it based on the combined ϕ_x and ϕ_y of 1D version. Therefore, we start from 1D version, namely,

$$\psi_t = \phi\,\psi_x. \tag{21}$$

First, if central difference is used:

$$\frac{\psi(x, t+\Delta t) - \psi(x,t)}{\Delta t} = \phi\frac{\psi(x+\Delta x, t) - \psi(x-\Delta x, t)}{2\Delta x} \tag{22}$$

$$\psi(x,t+\Delta t)=\psi(x,t)+\phi\Delta t\frac{\psi(x+\Delta x,t)-\psi(x-\Delta x,t)}{2\Delta x} \tag{23}$$

Take the Discrete Time Fourier Transform(DTFT),

$$\begin{aligned}\Psi(\omega,t+\Delta t)&=\Psi(\omega,t)+\frac{\phi\Delta t}{2\,\Delta x}(e^{j\omega\Delta x}-e^{-j\omega\Delta x})\,\Psi(\omega,t)\\&=\Psi(\omega,t)[1+\frac{\phi\Delta t}{\Delta x}sin(\Delta x\,\omega)]\\&=\alpha(\omega)\Psi(\omega,t)\end{aligned} \tag{24}$$

Then $\left|1+\frac{\phi\Delta t}{\Delta x}sin(\Delta x\,\omega)\leq 1\right|$ cannot be satisfied all the time. Therefore, it is impossible to keep the factor $\alpha(\omega)$ always less than 1 for any $\phi,\Delta t$ and Δx. We cannot use central difference.

Second, if forward difference is used:

$$\frac{\psi(x,t+\Delta t)-\psi(x,t)}{\Delta t}=\phi\frac{\psi(x+\Delta x,t)-\psi(x,t)}{\Delta x} \tag{25}$$

$$\psi(x,t+\Delta t)=\psi(x,t)+\phi\Delta t\frac{\psi(x+\Delta x,t)-\psi(x,t)}{\Delta x} \tag{26}$$

Take DTFT:

$$\begin{aligned}\Psi(\omega,t+\Delta t)&=\Psi(\omega,t)+\frac{\phi\Delta t}{\Delta x}(e^{j\omega\Delta x}-1)\,\Psi(\omega,t)\\&=\Psi(\omega,t)[1+\frac{\phi\Delta t}{\Delta x}\,(e^{j\omega\Delta x}-1)]\\&=\alpha(\omega)\Psi(\omega,t)\end{aligned} \tag{27}$$

The magnitude of factor should obey $|\alpha(\omega)|\leq 1$, namely:

$$\begin{aligned}\left|1+\frac{\phi\Delta t}{\Delta x}\,(e^{j\omega\Delta x}-1)\right|&=\left|1-\frac{\phi\Delta t}{\Delta x}+\frac{\phi\Delta t}{\Delta x}[cos(\omega\Delta x)+j\,sin(\omega\Delta x)]\right|\\&=\left|1-\frac{\phi\Delta t}{\Delta x}(1-cos(\omega\Delta x))+j\,\frac{\phi\Delta t}{\Delta x}sin(\omega\Delta x)\right|\\&=[1-\frac{\phi\Delta t}{\Delta x}(1-cos(\omega\Delta x))]^2+(\frac{\phi\Delta t}{\Delta x}sin(\omega\Delta x))^2\\&=1-\frac{2\phi\Delta t}{\Delta x}(1-cos(\omega\Delta x))(1-\frac{\phi\Delta t}{\Delta x})\leq 1\end{aligned} \tag{28}$$

In this situation, $1-cos(\omega\Delta x)\geq 1$ is always true, so if $\phi>0$ then we need $1-\frac{\phi\Delta t}{\Delta x}\geq 0$; and if $\phi<0$, then we need $1-\frac{\phi\Delta t}{\Delta x}\leq 0$, which is impossible. Therefore, if $\phi>0$, the CFL condition is

$$\phi\Delta t\leq\Delta x \tag{29}$$

Third, if backward difference is used:

$$\frac{\psi(x,t+\Delta t)-\psi(x,t)}{\Delta t}=\phi\frac{\psi(x,t)-\psi(x-\Delta x,t)}{\Delta x} \tag{30}$$

$$\psi(x,t+\Delta t)=\psi(x,t)+\phi\Delta t\frac{\psi(x,t)-\psi(x-\Delta x,t)}{\Delta x} \tag{31}$$

Take DTFT:

$$\begin{aligned}\Psi(\omega,t+\Delta t)&=\Psi(\omega,t)+\frac{\phi\Delta t}{\Delta x}(1-e^{-j\omega\Delta x})\,\Psi(\omega,t)\\&=\Psi(\omega,t)[1+\frac{\phi\Delta t}{\Delta x}\,(1-e^{-j\omega\Delta x})]\\&=\alpha(\omega)\Psi(\omega,t)\end{aligned} \tag{32}$$

The magnitude of amplification factor should obey $|\alpha(\omega)|\leq 1$, namely:

$$\begin{aligned}\left|1+\frac{\phi\Delta t}{\Delta x}\,(1-e^{-j\omega\Delta x})\right|&=\left|1+\frac{\phi\Delta t}{\Delta x}-\frac{\phi\Delta t}{\Delta x}[cos(\omega\Delta x)-j\,sin(\omega\Delta x)]\right|\\&=\left|1+\frac{\phi\Delta t}{\Delta x}(1-cos(\omega\Delta x))-j\,\frac{\phi\Delta t}{\Delta x}sin(\omega\Delta x)\right|\\&=[1+\frac{\phi\Delta t}{\Delta x}(1-cos(\omega\Delta x))]^2+(\frac{\phi\Delta t}{\Delta x}sin(\omega\Delta x))^2\\&=1+\frac{2\phi\Delta t}{\Delta x}(1-cos(\omega\Delta x))(1+\frac{\phi\Delta t}{\Delta x})\leq 1\end{aligned} \tag{33}$$

In this situation, $1-cos(\omega\Delta x)\geq 1$ is always true, so if $\phi>0$ then we need $1+\frac{\phi\Delta t}{\Delta x}\leq 0$, which is impossible; and if $\phi<0$, then we need $1+\frac{\phi\Delta t}{\Delta x}\geq 0$. Therefore, if $\phi>0$, the CFL condition is

$$\phi\Delta t\geq-\Delta x \tag{34}$$

Therefore, we can use forward difference if $\phi>0$ and backward difference if $\phi<0$, and the overall CFL condition should be:

$$|\phi|\,\Delta t\leq\Delta x \tag{35}$$

The CFL for ψ_y is the same:

$$|\phi|\,\Delta t\leq\Delta y \tag{36}$$

After analyzing the 1D version of ψ, we can combine the x and y dimension together with a new combined speed ϕ to get the $\psi_t=\phi_x\psi_x+\phi_y\psi_y$, assuming $\Delta x=\Delta y$, then the CFL condition for 2D version is:

$$\sqrt{\phi_x^2+\phi_y^2}\,\Delta t<\Delta x \tag{37}$$

4.2 Diffusion term

For the second term, $\psi_t = \phi\,||\nabla\psi|| \cdot \nabla(\frac{\nabla\psi}{||\nabla\psi||})$ is a geometric heat equation. Assuming ξ is the direction "along the edge", and Ω is the direction "cross the edge".

$$\begin{aligned}
&\psi_t = \phi\,||\nabla\psi|| \cdot \nabla(\frac{\nabla\psi}{||\nabla\psi||}) = \phi\,||\nabla\psi|| \cdot [(\frac{\psi_x}{\sqrt{\psi_x^2+\psi_y^2}})_x + (\frac{\psi_y}{\sqrt{\psi_x^2+\psi_y^2}})_y] \\
&= \phi\frac{\psi_{xx}(\psi_x^2+\psi_y^2) - (\psi_x^2\psi_{xx} + \psi_x\psi_y\psi_{yx}) + \psi_{yy}(\psi_x^2+\psi_y^2) - (\psi_x\psi_y\psi_{xy} + \psi_y^2\psi_{yy})}{\psi_x^2+\psi_y^2} \\
&= \phi\frac{\psi_x^2\psi_{yy} - 2\psi_x\psi_y\psi_{xy} + \psi_y^2\psi_{xx}}{\psi_x^2+\psi_y^2} \\
&= \phi[\Delta\psi - (\frac{\nabla\psi}{||\nabla\psi||})^T(\nabla^2\psi)(\frac{\nabla\psi}{||\nabla\psi||})] \\
&= \phi(\Delta\psi - \psi_{\Omega\Omega}) \\
&= \phi\,\psi_{\xi\xi}
\end{aligned} \tag{38}$$

Hence, the second term is equal to 1D diffusion equation in the direction along with the edge. We can choose central difference method to discrete 1D version and obtain the CFL condition:

$$\begin{aligned}
\psi_t &= \phi\,\psi_{xx} \\
\frac{\psi(x,t+\Delta t) - \psi(x,t)}{\Delta t} &= \phi\frac{\psi(x+\Delta x,t) - 2\psi(x,t) + \psi(x-\Delta x,t)}{\Delta x^2} \\
\psi(x,t+\Delta t) &= \psi(x,t) + \phi\Delta t\frac{\psi(x+\Delta x,t) - 2\psi(x,t) + \psi(x-\Delta x,t)}{\Delta x^2}
\end{aligned} \tag{39}$$

Take DTFT of the above equation:

$$\begin{aligned}
\Psi(\omega,t+\Delta t) &= \Psi(\omega,t) + \frac{\phi\Delta t}{\Delta x^2}[e^{j\omega\Delta x}\Psi(\omega,t) - 2\Psi(\omega,t) + e^{-j\omega\Delta x}\Psi(\omega,t)] \\
&= [1 - \frac{2\phi\Delta t}{\Delta x^2} + \frac{2\phi\Delta t}{\Delta x^2}(\frac{e^{j\omega\Delta x} + e^{-j\omega\Delta x}}{2})]\,\Psi(\omega,t) \\
&= [1 - \frac{2\phi\Delta t}{\Delta x^2} + \frac{2\phi\Delta t}{\Delta x^2}cos(\omega\Delta x)]\,\Psi(\omega,t) \\
&= (1 - \frac{2\phi\Delta t}{\Delta x^2}[1 - cos(\omega\Delta x)])\,\Psi(\omega,t) \\
&= \alpha(\omega)\,\Psi(\omega,t)
\end{aligned} \tag{40}$$

The magnitude of amplification factor should obey $|\alpha(\omega)| \leq 1$, namely:

$$
\begin{aligned}
\alpha(\omega)^2 &= \left|1 - \frac{2\phi\Delta t}{\Delta x^2}[1 - cos(\omega\Delta x)]\right|^2 \\
&= 1 - \frac{4\phi\Delta t}{\Delta x^2}[1 - cos(\omega\Delta x)] + \frac{4\phi^2\Delta t^2}{\Delta x^4}[1 - cos(\omega\Delta x)]^2 \leq 1 \qquad (41) \\
&\frac{4\phi^2\Delta t^2}{\Delta x^4}[1 - cos(\omega\Delta x)]^2 \leq \frac{4\phi\Delta t}{\Delta x^2}[1 - cos(\omega\Delta x)] \\
&\phi\Delta t[1 - cos(\omega\Delta x)] \leq \Delta x^2 \qquad (42)
\end{aligned}
$$

To keep the above inequality always true, the maximum value of the left hand side should be used. Because $1 - cos(\omega\Delta x) \in [0, 2]$, then $2\phi\Delta t \leq \Delta x^2$. And also, we choose $\Delta x = \Delta y$. Finally we obtain the CFL condition:

$$\phi\Delta t \leq \frac{\Delta x^2}{2} \qquad (43)$$

4.3 Dilation/erosion term

For the third term, $\psi_t = \alpha\phi\,||\nabla\psi|| = \alpha\phi\sqrt{\psi_x^2 + \psi_y^2}$ is a nonlinear transport equation. If we initialize the curve outside the object, then it should be erosion term. To discrete it, a 2D version Entropy Upwind Difference should be used.

Similarly, the 2D Entropy Upwind is the extension of the 1D version, which is

$$
\begin{aligned}
\psi_t &= \alpha\phi\,|\psi_x| \\
&= \alpha\phi sign(\psi_x)\psi_x \\
&= \begin{cases} \alpha\phi\psi_x,\ \psi_x \geq 0 \\ -\alpha\phi\psi_x,\ \psi_x < 0 \end{cases}
\end{aligned} \qquad (44)
$$

D_x^+ and D_x^- are used to denote the forward and backward differences respectively. According to the lectures, we should choose the scheme that "reinforce" the result of the scheme. For example, $\alpha\phi > 0$ and if $D_x^+\psi$ is positive, then the coefficient is also positive. For the positive coefficient, we also need to use forward difference to discrete the linear transport equation. This is what we called "reinforce". A table of the sign of the results from the two schemes can be drawn to show the choice of scheme. For $\alpha\phi > 0$:

D_x^+	D_x^-	Choice
+	+	D_x^+
-	-	D_x^-
+	-	$\sqrt{D_x^{+\,2} + D_x^{-\,2}}$
-	+	0

Table 1: choice of difference scheme(coefficient ¿ 0)

The result in table 1 can be concluded into the entropy upwind difference as follow:

$$\psi_t = \alpha\phi\sqrt{max^2(D_x^+\psi, 0) + min^2(D_x^-\psi, 0)},\ \alpha\phi \geq 0 \tag{45}$$

If $\alpha\phi < 0$, the first two lines and last two lines in table1 are flipped and the table and equation are as follow:

D_x^+	D_x^-	Choice
+	+	D_x^-
-	-	D_x^+
+	-	0
-	+	$\sqrt{D_x^{+2} + D_x^{-2}}$

Table 2: choice of difference scheme(coefficient ¿ 0)

$$\psi_t = \alpha\phi\sqrt{max^2(D_x^-\psi, 0) + min^2(D_x^+\psi, 0)},\ \alpha\phi < 0 \tag{46}$$

Then the CFL condition is the same as the linear transport equation, which is:

$$|\alpha|\,\phi\Delta t \leq \Delta x \tag{47}$$

Therefore, the 2D version can be obtained by combining the x and y part of the 1D version, which is:

$$\psi_t = \alpha\phi\sqrt{max^2(D_x^+\psi, 0) + min^2(D_x^-\psi, 0) + max^2(D_y^+\psi, 0) + min^2(D_x^-\psi, 0)},\ \alpha\phi \geq 0 \tag{48}$$

And the CFL condition for the 2D version($\Delta x = \Delta y$) should be:

$$\sqrt{2}\,|\alpha|\,\phi\Delta t \leq \Delta x \tag{49}$$

4.4 Extension and reinitialization

The gradient flow and level set equations we used are as follow:

$$\begin{aligned} C_t &= -\phi\kappa N - (\nabla\phi \cdot N)N - \alpha\phi\, N \\ \psi_t &= \nabla\phi \cdot \nabla\psi + \phi\,||\nabla\psi|| \cdot \nabla(\frac{\nabla\psi}{||\nabla\psi||}) + \alpha\phi\,||\nabla\psi|| \end{aligned} \tag{50}$$

According to the lectures, there is a problem of the difference of the external field ϕ in the C_t and ψ_t. When we evolve C_t and calculate the ϕ for the equation, we only care about the ϕ at the curve. But for the ψ_t, the value of ϕ all over the whole image are needed to consider. This is because there might be overlappings and even though the level set equations do not interfere with each other, there may be unpredictable changes of ϕ because of the image data is

unpredictable.

One method to solve this problem is to calculate the extension of $\hat{\phi}$ to make sure only the values of ψ on the curve influence the evolution of the C_t [4].

$$\psi_t = \hat{\nabla\phi} \cdot \nabla\psi + \hat{\phi}\,||\nabla\psi|| \cdot \nabla(\frac{\nabla\psi}{||\nabla\psi||}) + \alpha\hat{\phi}\,||\nabla\psi|| \tag{51}$$

The extension of ϕ is:

$$\hat{\phi} = \begin{cases} \phi(x), where\, \psi(x) : x \in zero\, level\, set \\ \phi(y), where\, \psi(y) = 0\, and\ ||x-y||\ is\, minimum \end{cases} \tag{52}$$

Namely,

$$\hat{\phi(x)} = \begin{cases} \phi(C), if\, x \in C \\ \nabla\phi \cdot \nabla\psi,\ if\, x \notin C \end{cases} \tag{53}$$

And the $\hat{\nabla\phi}$ needs to be calculated seperately, therefore we need to extends three equations. The method to run the three PDEs and finding the minimum distance is very time-consuming and results in an expensive and inconvenient algorithm.

$$\begin{aligned} \hat{\phi_t} &= \nabla\hat{\phi} \cdot \nabla\psi \\ \hat{\phi_{xt}} &= \nabla\hat{\phi_x} \cdot \nabla\psi \\ \hat{\phi_{yt}} &= \nabla\hat{\phi_y} \cdot \nabla\psi \end{aligned} \tag{54}$$

After searching for solution, a method was found. Reference [6] gives a concise method called reinitialization to reinitialize the level set field to enforce the level set to sign distance function and does not use extension. By enforcing the level set to sign distance function, we can prevent the level set form becoming ill-conditioned. And also the wrong changes and overlappings can be corrected. So we assume doing reinitialization periodically can have the same effect as the extension.

The reinitialization PDE given in [6] is as follow:

$$\begin{aligned} \psi_t &= S(\psi)(1 - |\nabla\psi|) \\ &= S(\psi) - S(\psi)\,|\nabla\psi|) \end{aligned} \tag{55}$$

Where

$$S(\psi) = \frac{\psi}{\sqrt{\psi^2 + 1}} \tag{56}$$

The sign function will get 0 inside the contour, 1 outside and 0 on the contour. Then we need to discrete the equation $\psi_t = -S(\psi)\,|\nabla\psi|$, which is also a nonlinear transport equation like the erosion term in 4.3. So we will use the entropy upwind difference again:

$$\psi_t = \begin{cases} -S(\psi)\sqrt{max^2(D_x^+\psi,0)+min^2(D_x^-\psi,0)+max^2(D_y^+\psi,0)+min^2(D_x^-\psi,0)}, & S(\psi)\le 0 \\ -S(\psi)\sqrt{min^2(D_x^+\psi,0)+max^2(D_x^-\psi,0)+min^2(D_y^+\psi,0)+max^2(D_x^-\psi,0)}, & S(\psi)\ge 0 \end{cases} \tag{57}$$

And the CFL condition($\Delta x = \Delta y$) should be:

$$\sqrt{2}\,|S(\psi)|\,\Delta t \le \Delta x \tag{58}$$

Namely,

$$\sqrt{2}\Delta t \le \Delta x \tag{59}$$

5 Experiments

In this section, the experiments are performed to test and verify the implementation of the segmentation algorithm based on active contour. The algorithm is implemented using Python(3.7.3) based on opencv-python [7], matplotlib and numpy libraries. Testing environment is PyCharm. Source code is attached in the appendix. Three images of *face*, *organ* and *pedestrian* are chosen to represent the three example applications of image segmentation. The face image will be used to present the trends and others will be used for comparison.

5.1 Metrics

To present and compare the performance, some quantitative metrics are used, including *energy*, *error* and *number of iterations*.

Energy

We should use energy to show the trend of decreasing of energy while updating level set. The gradient descent flow of the project is $C_t = -\phi\kappa N - (\nabla\phi \cdot N)N - \alpha\phi\, N$. Then the energy is obtained from:

$$E(c) = \int_c \phi ds + \alpha \iint_{c\ in} \phi dxdy \tag{60}$$

Error

To evaluate the performance of the algorithm, we need to show that the error decreases with the process of the calculation. The way the error is calculated here is to compare the current contour with the actual contour of the object. First the real object is extracted and leave the original position white as the reference contour. Then, for every iteration in the program, the current ψ is changed to 0-1 format and compare the sum of 1s with the reference and calculate the ratio as error. The original images and references are as follow:

Figure 1: face image

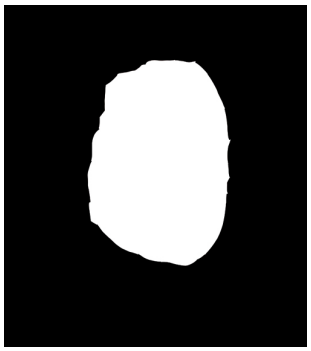

Figure 2: face reference

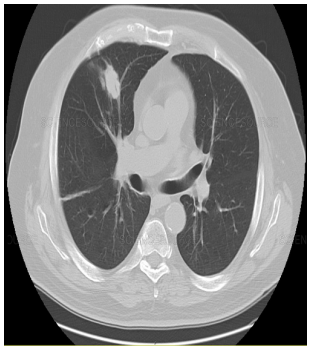

Figure 3: lung scan image

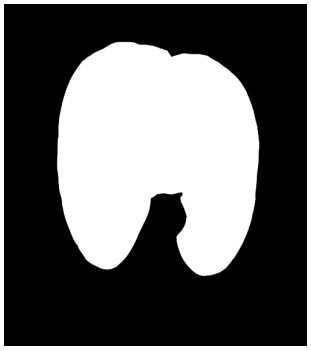

Figure 4: lung scan reference

Figure 5: pedestrian image

Figure 6: Pedestrian reference

Number of iterations

Number of iterations represents the efficiency of the algorithm and needs to be considered.

5.2 Experiments results

Initial level set and segementation result

The initial contour is selected as a circle that includes the actual object and outside it. We transfer the RGB image to gray image and implement the PDE based algorithm and obtain the segmentation result:

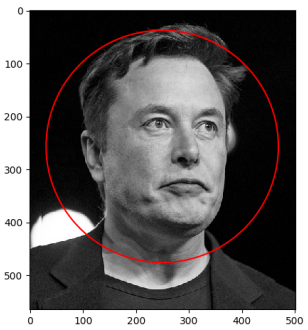

Figure 7: Initialization

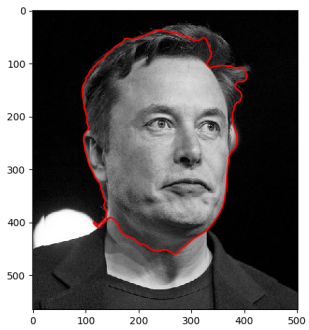

Figure 8: Result of segementation

Energy decreasing

Use the face image as an example to show the trend of energy with iteration:

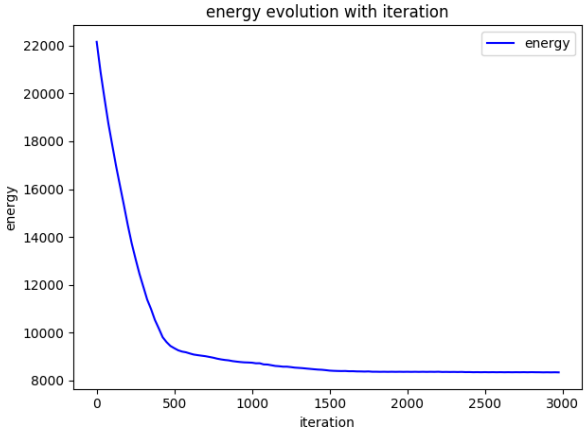

Figure 9: Energy evolution with iteration of face.jpg

As we can see, with the iteration going on, the energy is decreasing and then reach a stable state. The decreasing is fast at the first 500 iterations and then slows down and finally maintains constant.

Error decreasing

Use the face image as an example to show the trend of error with iteration:

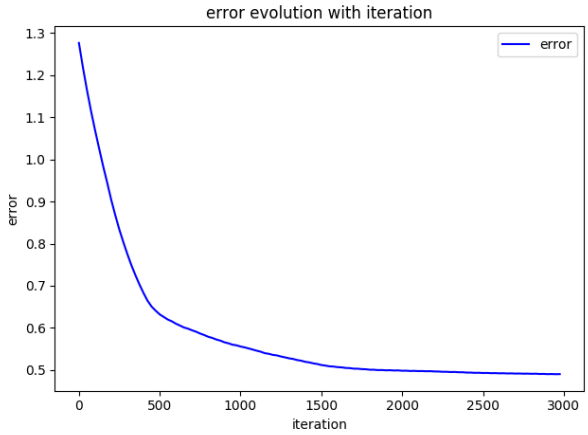

Figure 10: Error evolution with iteration of face.jpg

As is shown, the error is also decreasing within the iteration process. Similarly to the trend of energy evolution, the error curve is also going down fast at first and then slows down to a stable state.

Comparison of potential functions

In addition to the potential function we used:

$$\phi = \frac{1}{1 + (||\nabla \mathbf{I}||)^2} \tag{61}$$

Another one is also used to do experiment:

$$\phi = \frac{1}{1 + ||\nabla \mathbf{I}||} \tag{62}$$

Use the face image as an example to show the difference:

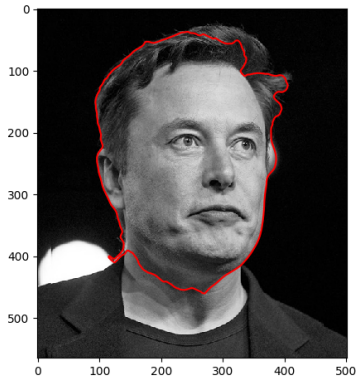

Figure 11: $\phi = \frac{1}{1+(||\nabla \mathbf{I}||)^2}$

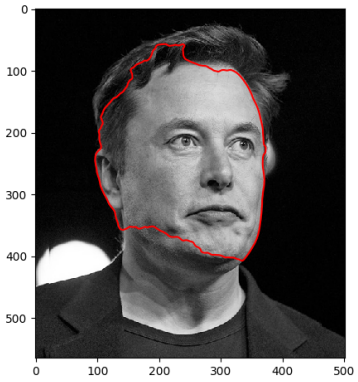

Figure 12: $\phi = \frac{1}{1+||\nabla \mathbf{I}||}$

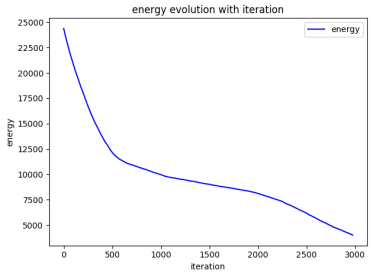

Figure 13: energy evolution of $\phi = \frac{1}{1+||\nabla \mathbf{I}||^2}$

Figure 14: error evolution of $\phi = \frac{1}{1+||\nabla \mathbf{I}||}$

Figure 12 shows the result for $\phi = \frac{1}{1+||\nabla \mathbf{I}||^2}$, which is not so good as the default potential function we used $\phi = \frac{1}{1+||\nabla \mathbf{I}||}$. It has some contour inside the actual contour and loses some information. During testing, I also found that the new potential function takes more time to reach stable state. If we set the number of iteration to 3000 for both of them to compare with the original one, we can see that there is no final stable state shown and the error becomes negative value after 2000 iterations. Therefore, we believe the $\phi = \frac{1}{1+||\nabla \mathbf{I}||}$ is more suitable for our PDEs.

Comparison of different erosion term

The third term we considered in previous section is the erosion term, which helps the curve evolve to the actual contour. In this part, different coefficients of this term, α, is tested using the face image as an example to show the differences. We fix the number of iterations to 3000 and compare the results of segementation:

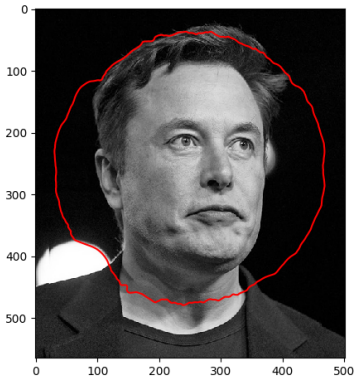

Figure 15: $\alpha = 0$

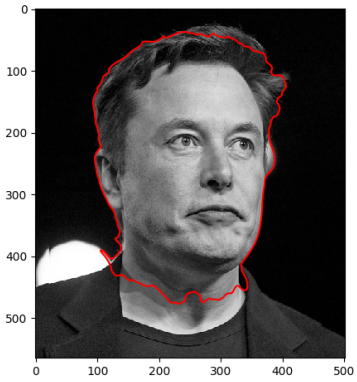

Figure 16: $\alpha = 0.2$

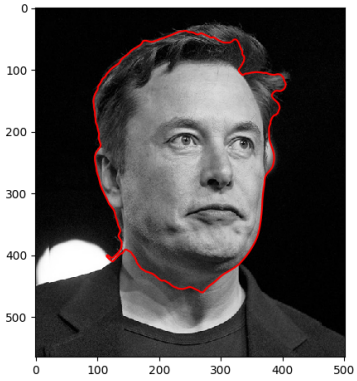

Figure 17: $\alpha = 0.4$

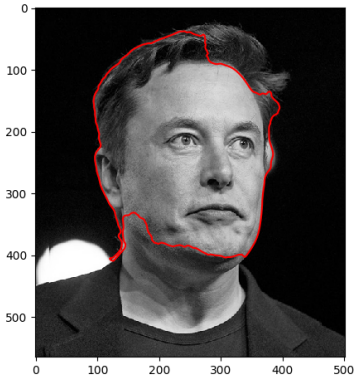

Figure 18: $\alpha = 0.6$

Figures 15 - 18 shows the different results for $\alpha \in \{0, 0.2, 0.4, 0.6\}$. when $\alpha = 0$, the curve evolves slowly and only stays near to the initial position and find the local edges. With the α goes up to 0.2 and 0.4, the performance becomes better and has more meaningful results. The result for $\alpha = 0.4$ is closer to the actual meaningful part of the object. But if we continue to increase the value of α, the result contour will get inside to the actual contour and results in a not so good result. Therefore, the value of 0.4 is the most suitable our PDEs.

Influence of reinitialization

In this part, we test the influence of whether reinitialization is implemented or not to verify the assumption that reintialization can solve the problem of level

set methods. And also we analyze what step length should be used to implement.

First, If we do not use reinitialization to the level set method, the result should be not desirable because of the overlapping and accumulative errors. The results for 100, 300, 500 and 1000 iterations are shown:

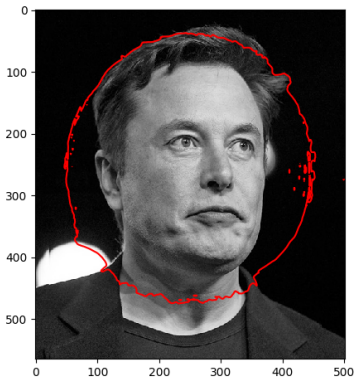

Figure 19: number of iteration = 100

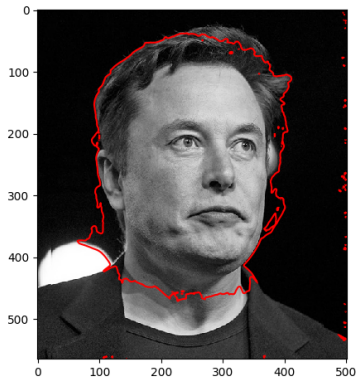

Figure 20: number of iteration = 300

Figure 21: number of iteration = 500

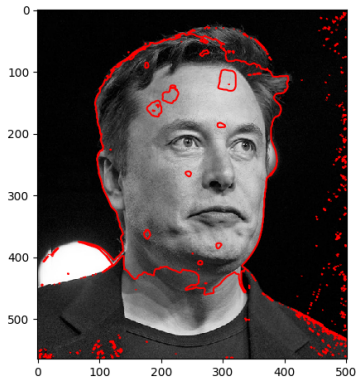

Figure 22: number of iteration = 1000

The results show that without reinitialization, the curve is influenced by local edges and there are some smaller incorrect contours existing. The results cannot represent meaningful information of segmentation.

Furthermore, if we use reinitialization, the interval between each two times of reinitialization is also needed to be considered. We test the step length of 5, 10 and 15 iterations, i.e. to reinitialize the ψ every step length times of iterations:

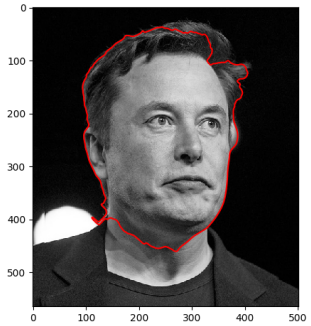

Figure 23: step = 5

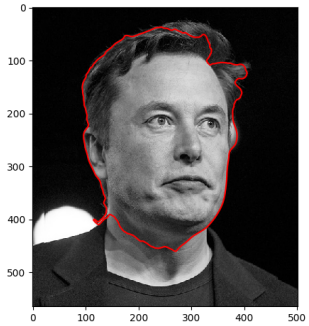

Figure 24: step = 10

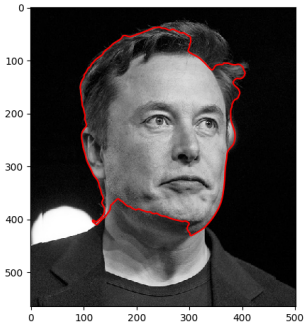

Figure 25: step = 15

The results of step = 5 and step = 10 seems similar, except for some details along the curve. The result of 10 is more smooth than that of 5. As for the result for step = 15, the curve goes inside the actual contour and loses some information. Therefore we cannot choose a very large step length for reinitialization and the very small one is also not so suitable. So the step = 10 fits our PDEs.

Comparing the results for different kinds of images

In addition to the image of a face we used, two another images are also tested, the CT scan of a lung and an image of a pedestrian on the road. The three images has different features and also shows different results:

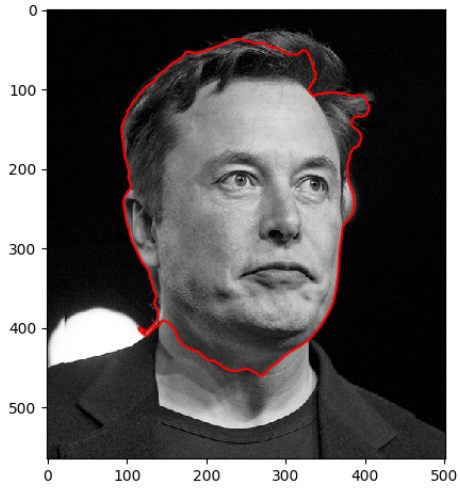

Figure 26: Segementation result of face

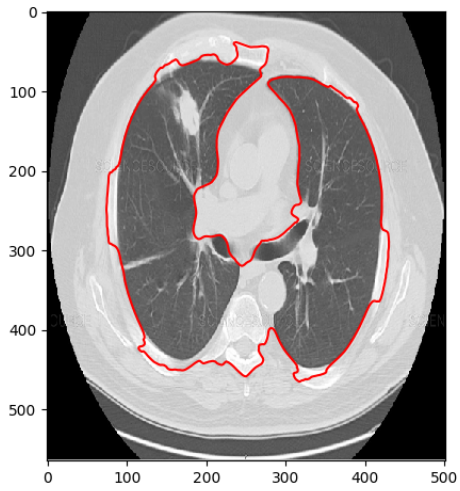

Figure 27: Segementaion result of lung scan

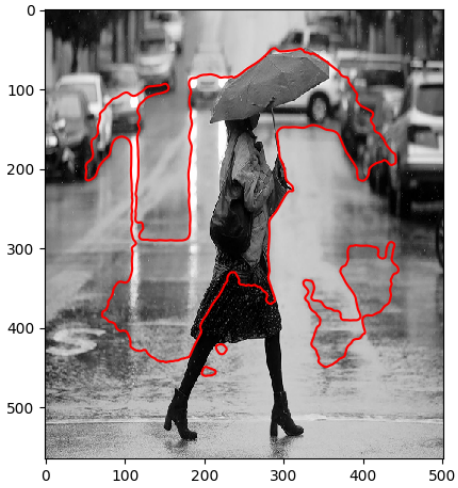

Figure 28: Segementation result of pedestrian

Generally, we can see that Figure 26 (face) has the best performance, next is Figure 27 (lung scan) and the Figure 28 (pedestrian) has the worst performance.

And specificly, the boundary of the face image is almost detected and information loss of the segmentation is little. There is some wrong edges next to the highlight at the regions near (400, 100) and (100, 350) but the overall performance is good.

The lung scan image also shows pretty good results except for some incorrectly detected nearby tissue of the lung. The result contour of region between the two lungs shows very good performance, which is smooth and exactly along the interface.

But the results for pedestrian image are not so good as expected. There are lots of wrong edges and the curve cannot show the actual contour of the pedestrian. There are also some other incorrect contours that are around some incorrect region in the image.

Comparaing the different results of different images, we can see the overall performance of our PDEs and program. For the images whose contents have very strong contrast, the program works well. For example, the face image has very strong contrast, the background is nearly black and the number of local edges is small. So the active contour can evolve correctly and reach the actual contour. However, the lung scan has less contrast than the face image, where the tissue or other organs near the lungs also have lots of boundaries that influence the potential function. Therefore there are some false positive results, although the most part is correct due to the strong contrast with the gray background.

But for the pedestrian image, there are lots of objects in the image besides the pedestrian. So the evolution of the curve can be easily influenced and the curve is prone to stay at local edges. So the result is not so good.

What's more, local highlight is also an influence to the results. In the face image, there is a light near (400, 100) and in the pedestrian image, there are some lights of vehicles. Although these lights are not physical objects in the images, all cause local high intensity and influence the evolution of the curve.

Therefore, the current algorithm and code of this PDE based active contour work well with the restrictions that the image has strong contrast and little local high intensity. If we use a highly contrast image to test, the result will be excellent, such as the moon in Figure 30, but might not be so useful as the face, CT scan or pedestrian.

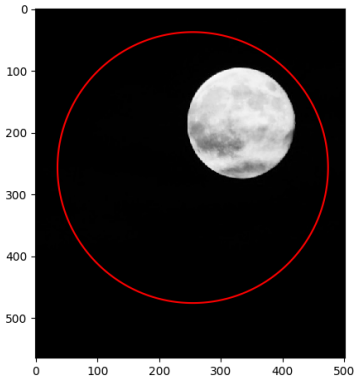

Figure 29: Initial state

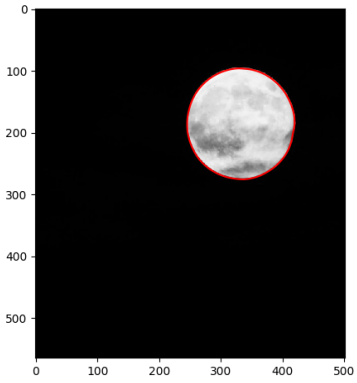

Figure 30: Segmentation result

6 Discussion

In this project, an active contour based image segmentation algorithm is proposed and implemented using PDE methods. The final results are acceptable, such as Figure 26, but not perfect. To obtain best performance, there should be restrictions on images that the image should have strong contrast and little local high intensity, such as the moon image in Figure 29 and 30. This is a weakness of the PDEs. To solve this problem, we can design more complicated algorithms to remove the local highlights or wrong edges before run the level set PDEs. An intuitive way is to detect the parts with high gradient and maintain the largest part unchanged and deal with other smaller parts. This can work but also with an assumption that the intended object should be the main part of the image. Some advanced design of the level set methods and PDEs [5] can also improve the performance, which should be future direction of this project.

From the experiments in previous section, we analyzed the coefficients and some variables of the PDEs. And we can see that the potential functions have influence on the performances and results, so choosing a suitable potential function is important for the design of PDEs. Also, the coefficient of the erosion term, α, has influence on the results, and it should not be zero because the curve will evolve incorrectly without erosion term. And also this coefficient should not be to large because this curve may evolve deep inside the actual contour. As for the reinitialization, without it the level set will be ill-conditioned and overlapping, causing the curve not correct and influenced by local edges. And the interval between two reinitialization also has influence on results. We cannot choose a too large step length. For future work, we can further analyze the influence of the coefficients or variables and also introduce and analyze more influencing factors and find their relationship or balance. Also, currently we just transfer the RGB images to gray images, maybe next step we can also consider the colors and try to deal with RGB images directly.

The three applications that I considered at the very beginning of the image segmentaion are face recognition, medical imaging and self-driving. Therefore I tested three images of face, CT scan and pedestrian. But in the practice, the designer of algorithm or devices also needs to consider the running time of the program, because most of the applications needs responsive services. For example, the pedestrian detection of the self-driving vehicles requires very low processing time, which is a big question of safety. And also most of the face recognition devices or technologies are customer services, such as for the smart phone, for the entrances, so they also require a fast response. Although some back-end services for face recognition do not require fast response, they are in minority and slow program also influences efficiency. However, the medical imaging should be a good application because the medical imaging hardly ever consider the very fast response, but the high accuracy, which is of vital importance to the diagnosis. The current running time of this program is in tens of minutes, the best case can reach several minutes but it requires the image to be suitable. So another future work is to improve the efficiency of the algorithm and optimize the code to meet the requirements of responsive applications; and improve the accuracy to fulfill the requirements of medical applications.

In this project, the whole process of defining problem, "translating" problem to mathematical problem, deriving PDEs, discretization and implementation and experiments are experienced. It is a good opportunity to construct, derive and discrete the PDEs and implement in code and debug it. This help understand the materials from lectures, which are the basis of the projects and also learn some new knowledge, such as the reinitialization and opencv-python library.

To sum up, the PDE method of this project can yield some meaningful results and good performance and fair runing time. But it still has large space to improve. The future work is designing advanced methods to deal with not

perfect images, analyzing more influence factors and their relationship and optimizing the running time and accuracy. The work should improve the project significantly.

References

[1] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[2] L. G. Shapiro and G. C. Stockman, *Computer vision*. Prentice Hall, 2001.

[3] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International journal of computer vision*, vol. 22, no. 1, pp. 61–79, 1997.

[4] B. student, "2011 lecture notes," *Lecture notes*, vol. 1, no. 1, pp. 43–47, 2011.

[5] F. Gibou, R. Fedkiw, and S. Osher, "A review of level-set methods and some recent applications," *Journal of Computational Physics*, vol. 353, pp. 82–109, 2018.

[6] M. Sussman, P. Smereka, S. Osher *et al.*, "A level set approach for computing solutions to incompressible two-phase flow," 1994.

[7] J. Howse, *OpenCV computer vision with python*. Packt Publishing Ltd, 2013.

A Appendix: Python code

```
# Active_contour.py
import cv2 as cv
import numpy as np
from levelset import *
from gradient_descent import *

# Image
I0 = cv.imread("face.jpg")  # Image read
I1 = cv.cvtColor(I0, cv.COLOR_RGB2GRAY)  # Image transfer from RGB
                                    to Gray
size = I1.shape  # Image size
sigma = 1.2  # Scale parameter in Gauss
I = cv.GaussianBlur(I1.astype(np.float32), (7, 7), sigma)  # Gauss
                                    filter
a = 0.4  # Coefficient in the erosion term

# Gredient
Iy, Ix = np.gradient(I)  # Gradient, x and y directions
f = Ix ** 2 + Iy ** 2  # Default potential function
# f = np.sqrt(Ix ** 2 + Iy ** 2)  # Another potential function
potential = [1]/([1] + f)  # Potential function, phi = 1/1+||nabla(
                                    phi)||^2
```

```
grad_y, grad_x = np.gradient(potential)  # Gradient of phi
phi = potential
grad_phi_x = grad_x
grad_phi_y = grad_y

# Reference
reference = cv.imread("face_ref.jpg")  # The reference contour of
                                       the actual object
reference = cv.cvtColor(reference, cv.COLOR_RGB2GRAY)  # Image
                                       transfer from RGB to Gray
ret, reference = cv.threshold(reference, 175, 255, cv.THRESH_BINARY
                                       )
reference = reference/255

# Initial level set
psi = init_levelset(size)  # Init level set
image_psi(I1, psi)  # Plot the initial

# Energy
energy_array = []
diffusion_energy_gradient = []
advevtion_energy_gradient = []
erosion_energy_gradient = []
energy_gradient = []

# Error
psi_r = cv.threshold(psi, 0.5, 255, cv.THRESH_BINARY)#/[255]
psi_reverse = psi_r[1]/255
psi_bw = 1 - psi_reverse
current_error_mat = psi_bw - reference
current_error = np.sum(current_error_mat)
norm = np.sum(reference)
error = []

# Iteration
cur = 1
n = 25
flag = True
while flag:
    if cur % 10 == 0:
        psi = reinitialization(psi, 10)

    psi, diffusion_energy_grad, advection_energy_grad,
                                        erosion_energy_grad =
                                        gradient_descent(psi, phi,
                                        grad_phi_x, grad_phi_y, a)
    if cur % n == 0:
        # print(psi)
        diffusion_energy_gradient.append(diffusion_energy_grad)
        advevtion_energy_gradient.append(advection_energy_grad)
        erosion_energy_gradient.append(erosion_energy_grad)
        energy_gradient.append(diffusion_energy_grad+
                                            advection_energy_grad+
                                            erosion_energy_grad)

        energy = compute_energy(psi, phi, a)
        energy_array.append(energy)
```

```
        psi_r = cv.threshold(psi, 0.5, 255, cv.THRESH_BINARY)
        psi_reverse = psi_r[1]/255
        psi_bw = 1 - psi_reverse
        current_error_mat = psi_bw - reference
        current_error = np.sum(current_error_mat)/norm
        error.append(current_error)

        # # Stable condition
        if cur/n > 3 and abs(energy_gradient[int(cur / n) - 1] +
                                        energy_gradient[int(cur /
                                         n) - 2]
               - energy_gradient[int(cur / n) - 3] -
                                                energy_gradient[
                                                int(cur / n) - 4])
                                                 < 0.1:
            flag = False
        # iteration number condition
        # if cur == 3000:
        #     flag = False
    cur += 1

# Plot the trend
def plot_trend(mat, name):
    size = len(mat)
    x =[i for i in range(size)]
    it = [i * 25 for i in x]
    plt.figure()
    plt.plot(it, mat, 'b', label=name)
    plt.title(name + ' evolution with iteration')
    plt.xlabel('iteration')
    plt.ylabel(name)
    plt.legend()
    plt.show()

image_psi(I1, psi)
plot_trend(error, 'error')
cv.waitKey(0)


# Gradient_descent.py
from differences import *
import copy
import cv2 as cv

# Graient descent
def gradient_descent(psi, phi, grad_phi_X, grad_phi_Y, a):
    psi_x = Dx_central(psi)
    psi_y = Dy_central(psi)
    psi_xy = Dx_central(psi_y)
    psi_xx = Dx_forward(Dx_backward(psi))
    psi_yy = Dy_forward(Dy_backward(psi))  # Partial derivativea
    # Diffusion term
    diffusion = phi * (psi_yy * psi_x ** 2 - 2 * psi_x * psi_y *
                                      psi_xy + psi_xx * psi_y ** 2)
                                      /(psi_x ** 2 + psi_y ** 2)
    psi_temp = copy.copy(psi)
```

```
    psi_temp[psi < 0.5] = 1
    psi_temp[psi > -0.5] = 0
    diffusion_mat = psi_temp * diffusion / (np.sqrt(psi_x ** 2 +
                                    psi_y ** 2))
    diffusion_energy = np.sum(diffusion_mat)

    # Advection term
    advection = upwind_difference(grad_phi_X, grad_phi_Y, psi)
    advection_mat = psi_temp * advection / (np.sqrt(psi_x ** 2 +
                                    psi_y ** 2))
    advection_energy = np.sum(advection_mat)

    # Erosion term
    erosion = a * entropy_upwind(phi, psi)
    erosion_mat = psi_temp * erosion / (np.sqrt(psi_x ** 2 + psi_y
                                    ** 2))
    erosion_energy = np.sum(erosion_mat)
    delta_t = CFL(phi, grad_phi_X, grad_phi_Y, a)
    grad_energy = advection + diffusion + erosion

    # Update psi
    psi_new = psi + delta_t * grad_energy
    return psi_new, diffusion_energy, advection_energy,
                                    erosion_energy

# Computing energy
def compute_energy(psi, phi, a):
    psi_temp_1 = copy.copy(psi)
    psi_temp_2 = copy.copy(psi)
    psi_temp_1[psi < 0.5] = 1
    psi_temp_1[psi >= 0.5] = 0
    psi_temp_2[psi > -0.5] = 1
    psi_temp_2[psi <= -0.5] = 0
    psi_temp = psi_temp_1 * psi_temp_2
    psi_temp_in = copy.copy(psi)
    psi_temp_in[psi < 0] = 1
    psi_temp_in[psi >= 0] = 0
    energy_mat = psi_temp * phi + a * psi_temp_in * phi
    energy = np.sum(energy_mat)
    return energy


# Levelset.py
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from differences import *
# np.set_printoptions(threshold=np.inf)

# Initialization
def init_levelset(size):
    x = np.linspace(1, size[1], size[1])
    y = np.linspace(1, size[0], size[0])
    X, Y = np.meshgrid(x, y)
    # Plot a circle
    r = min(size[0], size[1])/2
    radius = r
```

```
    center = (size[0]/2 - 25, size[1]/2)
    psi0 = np.sqrt((X - [center[1]]) ** 2 + (Y - [center[0]]) ** 2)
                                        - radius * 0.875
    return psi0

# Reinitialization
def reinitialization(psi, it):
    e = 1
    sign_function = psi / np.sqrt(psi ** 2 + e ** 2)  # Sign
                                       distance function
    psi_new = psi

    # Update psi
    for i in range(it):
        psi_delta = entropy_upwind(sign_function, psi_new, True)  #
                                            Delta t
        delta_t = 1/np.sqrt(2)
        psi_new = psi_new + delta_t * (sign_function - psi_delta)
                                            # iterate
    return psi_new

# Plot image with psi
def image_psi(img, psi):
    plt.imshow(img, cmap='Greys_r')
    plt.contour(psi, 0, colors='red')
    plt.show()


# Differences.py
import numpy as np
import copy

# Central difference of x
def Dx_central(m):
    y0, x0 = m.shape
    f = np.pad(m, ((0, 0), (1, 1)), 'edge')
    y, x = f.shape
    fx = np.zeros([y0, x0])
    fx[:, 0:x0] = 0.5 * (f[:, 2:x] - f[:, 0:x-2])
    return fx

# Central difference of y
def Dy_central(my):
    y0, x0 = my.shape
    f = np.pad(my, ((1, 1), (0, 0)), 'edge')
    y, x = f.shape
    fy = np.zeros([y0, x0])
    fy[0:y0, :] = 0.5 * (f[2:y, :] - f[0:y-2, :])
    return fy


# Forward difference of x
def Dx_forward(m):
    y0, x0 = m.shape
    f = np.pad(m, ((0, 0), (0, 1)), 'edge')
    y, x = f.shape
    fx = np.zeros([y0, x0])
```

```
    fx[:, 0:x0] = f[:, 1:x] - f[:, 0:x-1]
    return fx

# Backward difference of x
def Dx_backward(m):
    y0, x0 = m.shape
    f = np.pad(m, ((0, 0), (1, 0)), 'edge')
    y, x = f.shape
    fx = np.zeros([y0, x0])
    fx[:, 0:x0] = f[:, 1:x] - f[:, 0:x-1]
    return fx

# Forward difference of y
def Dy_forward(m):
    y0, x0 = m.shape
    f = np.pad(m, ((0, 1), (0, 0)), 'edge')
    y, x = f.shape
    fy = np.zeros([y0, x0])
    fy[0:y0, :] = f[1:y, :] - f[0:y-1, :]
    return fy

# Backward difference of y
def Dy_backward(m):
    y0, x0 = m.shape
    f = np.pad(m, ((1, 0), (0, 0)), 'edge')
    y, x = f.shape
    fy = np.zeros([y0, x0])
    fy[0:y0, :] = f[1:y, :] - f[0:y-1, :]
    return fy

# Upwind difference
def upwind_difference(v1, v2, m):
    v1_temp_1 = copy.copy(v1)
    v1_temp_2 = copy.copy(v1)
    v2_temp_1 = copy.copy(v2)
    v2_temp_2 = copy.copy(v2)
    v1_temp_1[v1 < 0] = 0
    v1_temp_2[v1 > 0] = 0
    v2_temp_1[v2 < 0] = 0
    v2_temp_2[v2 > 0] = 0
    x = v1_temp_1 * Dx_forward(m) + v1_temp_2 * Dx_backward(m)
    y = v2_temp_1 * Dy_forward(m) + v2_temp_2 * Dy_backward(m)
    r = x + y
    return r

# entropy upwind differnce
def entropy_upwind(phi, m, p=None):
    Dx_f = Dx_forward(m)
    Dx_b = Dx_backward(m)
    Dy_f = Dy_forward(m)
    Dy_b = Dy_backward(m)
    Dx_f_temp = copy.copy(Dx_f)
    Dx_b_temp = copy.copy(Dx_b)
    Dy_f_temp = copy.copy(Dy_f)
    Dy_b_temp = copy.copy(Dy_b)
    Dx_f_temp_1 = copy.copy(Dx_f)
    Dx_b_temp_1 = copy.copy(Dx_b)
```

```
    Dy_f_temp_1 = copy.copy(Dy_f)
    Dy_b_temp_1 = copy.copy(Dy_b)
    Dx_f_temp[Dx_f < 0] = 0
    Dx_b_temp[Dx_b > 0] = 0
    Dy_f_temp[Dy_f < 0] = 0
    Dy_b_temp[Dy_b > 0] = 0
    Dx_f_temp_1[Dx_f > 0] = 0
    Dx_b_temp_1[Dx_b < 0] = 0
    Dy_f_temp_1[Dy_f > 0] = 0
    Dy_b_temp_1[Dy_b < 0] = 0
    phi_temp = copy.copy(phi)
    phi_temp_1 = copy.copy(phi)
    phi_temp[phi < 0] = 0
    phi_temp_1[phi > 0] = 0
    output = phi_temp * (np.sqrt(Dx_f_temp_1 ** 2 + Dx_b_temp_1 **
                                 2 + Dy_f_temp_1 ** 2 +
                                 Dy_b_temp_1 ** 2)) \
             + phi_temp_1 * (np.sqrt(Dx_f_temp ** 2 + Dx_b_temp **
                                     2 + Dy_f_temp ** 2 +
                                      Dy_b_temp ** 2))
    return output

# CFL condition calculation
def CFL(phi, grad_phi_x, grad_phi_y, a):
    delta_t_diffusion = 0.5/np.max(phi)
    delta_t_advection = 1/np.max(np.sqrt(grad_phi_x ** 2 +
                                 grad_phi_y ** 2))
    delta_t_erosion = 1/(np.sqrt(2)*a*np.max(phi))
    delta_t = min(delta_t_advection, delta_t_diffusion,
                                 delta_t_erosion)
    return delta_t
```