图像处理实验报告

姓名：吴阳诚

学号：210810507

院系：理学院

专业：数据科学与大数据技术

班级：5 班

目录

实验一、CV 模型实验

目的和意义

本实验采用 C++语言，通过基础代码完成对 CV 模型的复现。

通过对 CV 模型的复现，可以体会能量泛函在图像分割上的应用，同时理解能量泛函的求解算法。

本实验探究了初始 Phi 的 margin 设置，以及训练过程的 Max_Iter, delta_t 设置对于 CV 模型的影响，并得到一定的基础结论。

模型介绍与说明

CV 模型是一个基于区域的活动轮廓模型，相比其他活动轮廓模型，CV 模型具有如下几个优点。

CV 模型是基于 Mumford-Shah 分割技巧和水平集方法，而不是基于边界函数使得演变曲线停在想要的边缘上。而且即使初始图像含有噪声，也不需要光滑初始图像，边缘的位置仍可以很好地被检测和保留。CV 模型可以检测不是由梯度定义的边缘或者是非常光滑的边缘，而对于这些边缘，经典的活动轮廓模型是不适用的。最后 CV 模型可以仅从一条初始曲线自动地检测内部轮廓，而且初始曲线的位置不必环绕待检测的物体，可以在图像的任意位置。这是 CV 模型一个非常重要的优点。

但是 CV 模型主要适用于具有同质区域的图像，对于不均匀强度背景的图像则效果较差。在 CV 模型里，常量 c1 和 c2 用来近似区域 inside(C) 和 outside(C) 内的图像强度，很显然，这只是一种粗糙的近似，而不包含图像的局部强度信息，因此 CV 模型不适用于图像强度不均匀的场景。

模型的数学原理

CV 的能量泛函模型如下：

$$
\begin{aligned}
F^{cv}(c_1, c_2, C) = \lambda_1 \int_{inside} |u_0(x,y) - c_1|^2 dx\,dy \\
+ \lambda_2 \int_{outside} |u_0(x,y) - c_2|^2 dx\,dy \\
+ \nu |C| .
\end{aligned}
$$

其中 $\nu, \lambda_1, \lambda_2$ 为参数

前两项为数据拟合项，最后一项为长度项。

引入水平集函数$\phi(x,y) = 0$表示边界 C ($\phi < 0{:}\, outside; \phi > 0{:}\, inside$)，引入 Heaviside

函数作为边界内外的示性函数。这样，曲线内外就能进行统一表达。另外，进行正规化近似表示，得到正规化的能量函数表达：

$$F_\varepsilon^{CV}(c_1, c_2, \phi) = \lambda_1 \int_\Omega |u_0(x,y) - c_1|^2 H_\varepsilon(\phi(x,y))\,dxdy$$
$$+ \lambda_2 \int_\Omega |u_0(x,y) - c_2|^2 [1 - H_\varepsilon(\phi(x,y))]\,dxdy$$
$$+ \nu \int_\Omega |\nabla H_\varepsilon(\phi(x,y))|\,dxdy$$

其中 $H_\varepsilon(z) = \frac{1}{2}\left[1 + \frac{2}{\pi}\arctan\left(\frac{z}{\varepsilon}\right)\right] \quad (\varepsilon > 0)$

$$\delta_\varepsilon(z) = H_\varepsilon'(z) = \frac{1}{\pi}\frac{\varepsilon}{\varepsilon^2 + z^2}$$

之后采用标准的梯度下降法，采用交替极小化格式，求解c_1, c_2, ϕ：

$$c_1(\phi) = \frac{\int_\Omega u_0(x,y) H_\varepsilon(\phi(x,y))\,dxdy}{\int_\Omega H_\varepsilon(\phi(x,y))\,dxdy}$$

$$c_2(\phi) = \frac{\int_\Omega u_0(x,y) \overset{1 - H_\varepsilon(\phi(x,y))}{H_\varepsilon(\phi(x,y))}\,dxdy}{\int_\Omega [1 - H_\varepsilon(\phi(x,y))]\,dxdy}$$

ϕ的梯度下降流方程：

$$\frac{\partial \phi}{\partial t} = -\frac{\partial F}{\partial \phi} = \delta_\varepsilon(\phi)\left[\nu\, \mathrm{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) - \lambda_1 (u_0 - c_1)^2 + \lambda_2 (u_0 - c_2)^2\right]$$

实验数据说明

本实验采用的图片是自选的人物图片（单通道）。共有 4 张，分别是同样的内容放缩得到的不同尺寸（像素）图片。尺寸分别为：（从大到小）396*534, 118*160; 59*80; 29*40。

程序设计流程

(程序语言为 C++)依照数学原理进行，进行离散化处理，得到程序。代码见附录。

对于单一图片：

设置 myCV 的 class，成员包含了 Max_Iter（最大迭代次数）, delta_t（每次 Phi 更新的步长），Ph, C1, C2i 等 其，并中包含了对单一图片进行 CV 泛函计算的全部方法。

图像读取为 Img（单通道），格式为 CV_8U。

Phi 初始化为与 Img 大小一致的 Mat，格式为 CV_32F，且设置 margin 参数，表示初始的 Phi 中矩形边界与图像边界的距离。于是初始矩形边界外部为-1，内部为+1，矩形本身为 0.

对于单个图像，使用 fit()函数可以完成对 Phi，C1,C2 的极小化。具体方法就是 CV 能量泛函的极小化过程的代码实现。其中与 Phi 计算相关的 div（散度矩阵）等均采用 CV_32F 格式，用于精确计算。

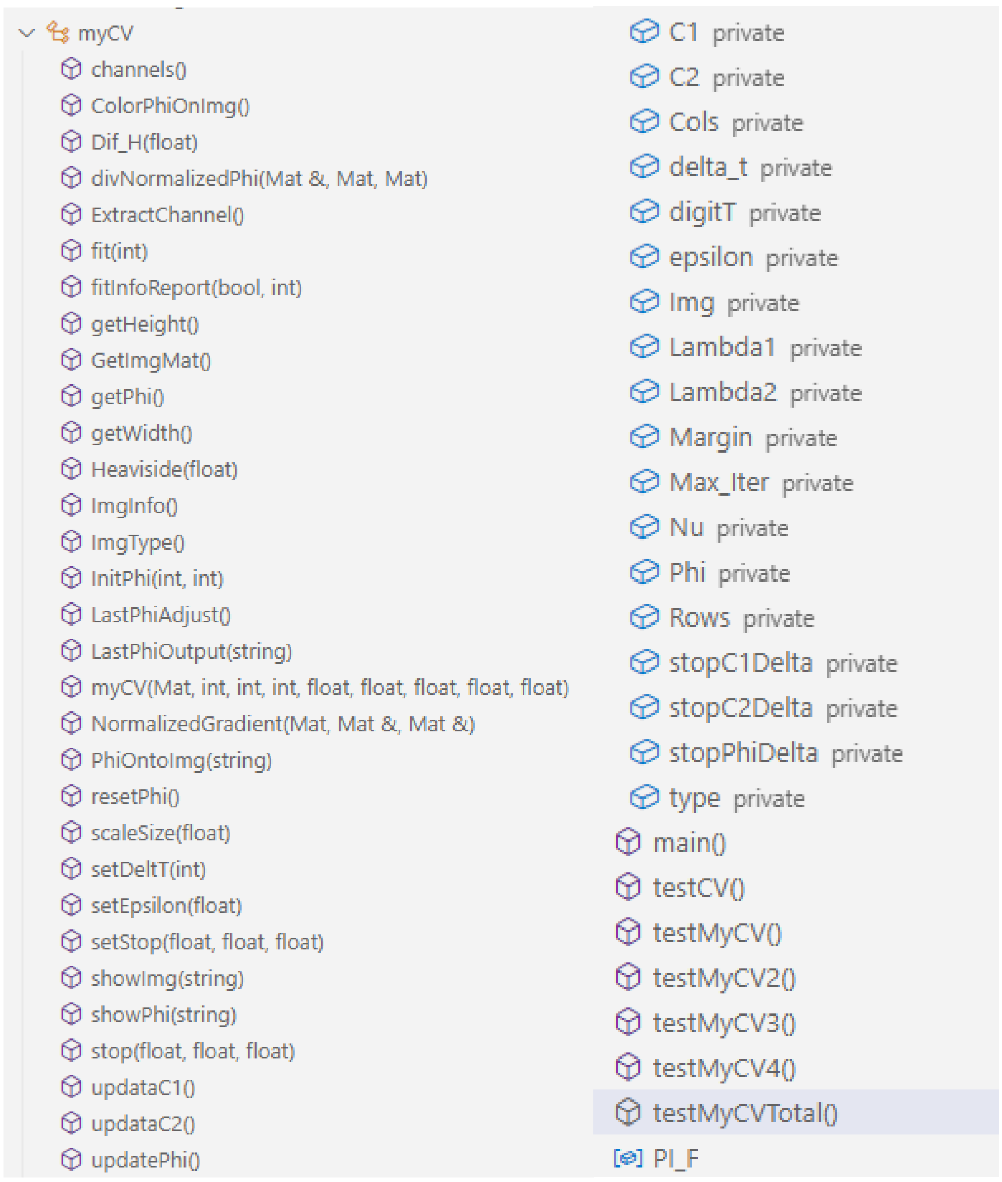
Ph，C1, C2i 的迭代过程通过 Max_Iter 控制终止。实际上,也尝试设置基于 delta 变化的终止条件，但是效果不好。

而最终的 Phi 则进行“示性处理”：负数为 outside 部分，取为 0，正数为 inside 部分，取为 255，再把 Phi 转化为 CV_8U 格式，即 inside 为白色，outside 为黑色。

保存最终 Phi 的结果到图片对应的输出文件夹中。

批量图片处理与参数网格搜索：

本实验处理了同一内容不同大小的一系列图片，并且希望考察初始 Phi 的 margin 设置，以及训练过程的 Max_Iter, delta_t 设置对于 CV 模型的影响。所以设计了 testMyCVTotal()函数，其中使用 vector 及迭代器等，完成了批量的图像处理，以及对于相依参数的 gridSearch（网格搜索）（很粗糙的测试）。其中 Max_Iter 固定为 10000.



```
myCV
  channels()
  ColorPhiOnImg()
  Dif_H(float)
  divNormalizedPhi(Mat &, Mat, Mat)
  ExtractChannel()
  fit(int)
  fitInfoReport(bool, int)
  getHeight()
  GetImgMat()
  getPhi()
  getWidth()
  Heaviside(float)
  ImgInfo()
  ImgType()
  InitPhi(int, int)
  LastPhiAdjust()
  LastPhiOutput(string)
  myCV(Mat, int, int, int, float, float, float, float, float)
  NormalizedGradient(Mat, Mat &, Mat &)
  PhiOntoImg(string)
  resetPhi()
  scaleSize(float)
  setDeltT(int)
  setEpsilon(float)
  setStop(float, float, float)
  showImg(string)
  showPhi(string)
  stop(float, float, float)
  updataC1()
  updataC2()
  updatePhi()
  C1  private
  C2  private
  Cols  private
  delta_t  private
  digitT  private
  epsilon  private
  Img  private
  Lambda1  private
  Lambda2  private
  Margin  private
  Max_Iter  private
  Nu  private
  Phi  private
  Rows  private
  stopC1Delta  private
  stopC2Delta  private
  stopPhiDelta  private
  type  private
main()
testCV()
testMyCV()
testMyCV2()
testMyCV3()
testMyCV4()
testMyCVTotal()
PI_F
```

实验结果

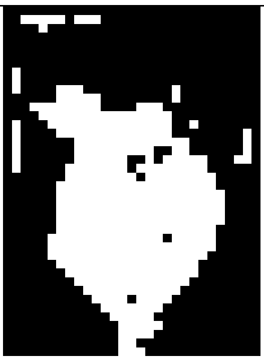
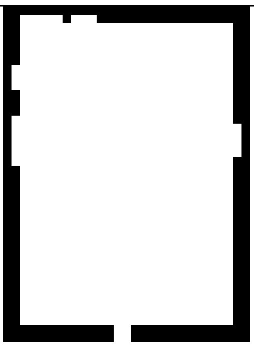
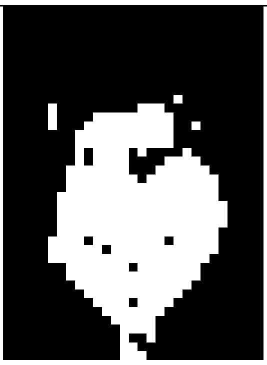
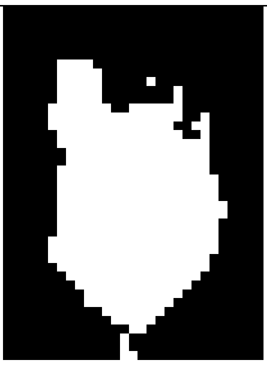
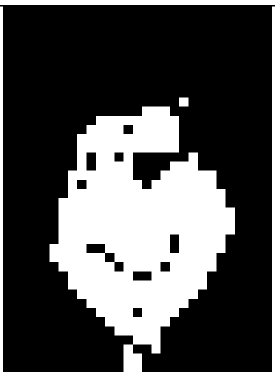
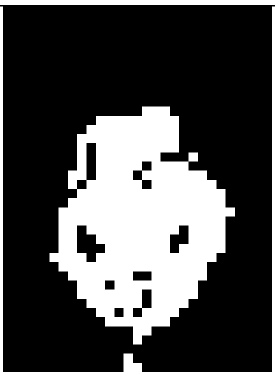
Mar 表示初始的 Phi 中矩形边界与图像边界的距离，delta_t 表示每次 Phi 更新的步长。所有结果都是 Max_Iter=10000 次循环得到。

结果图片为 Phi “示性化” 处理的展示，内部为白色，外部为黑色。

本实验结果强调不同 Margin 设置以及不同 delta_t 设置对于 CV 分割效果的影响。

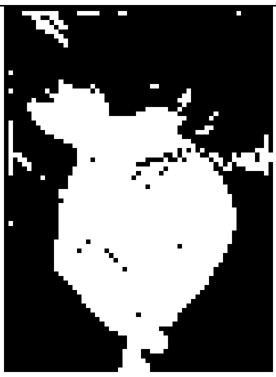
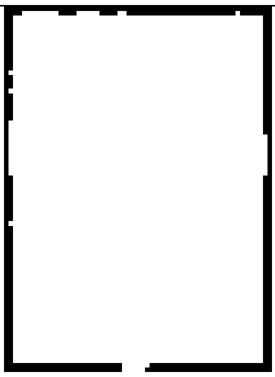
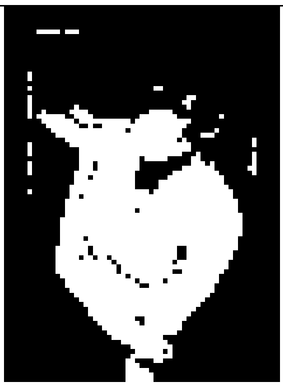
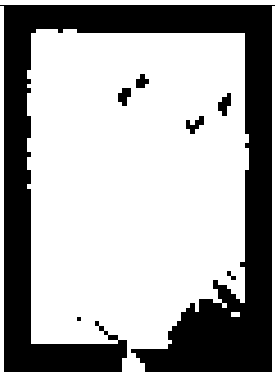
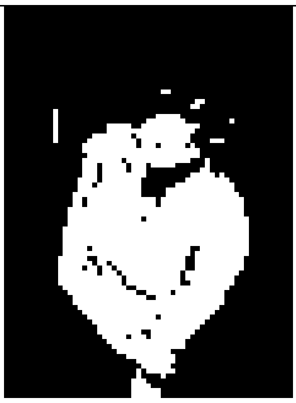
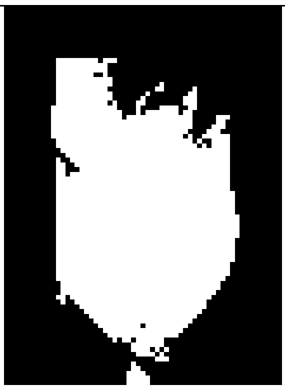
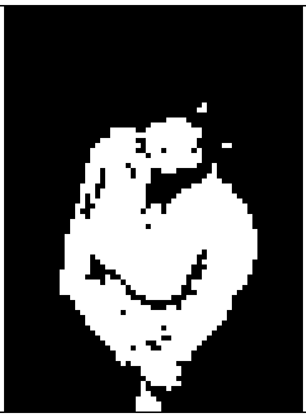
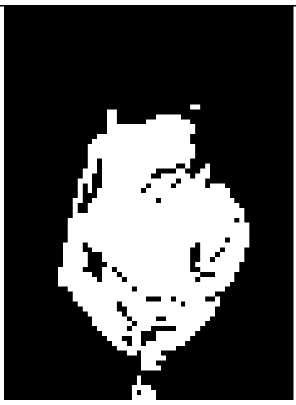
Size4: 29*40

Orginal:

Size4 **29*40**	Delta_t: 0.1	Delta_t: 0.01
Mar:1		
Mar:5		
Mar: 10		

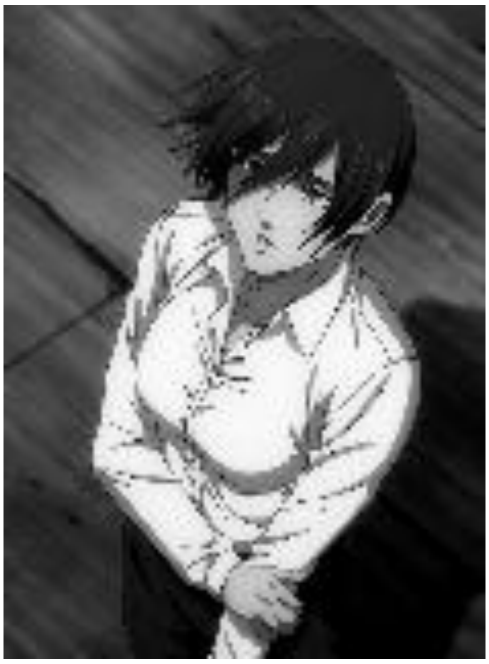
Size3: 59*80

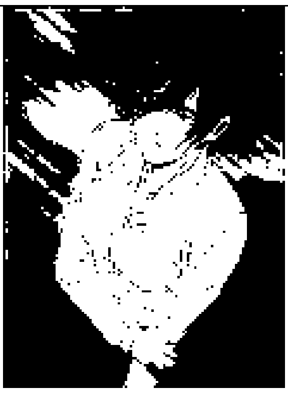
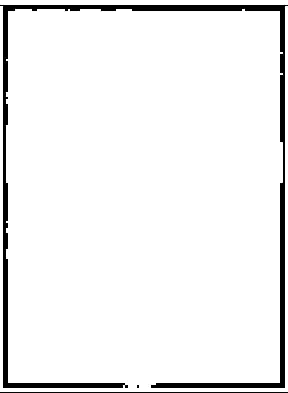
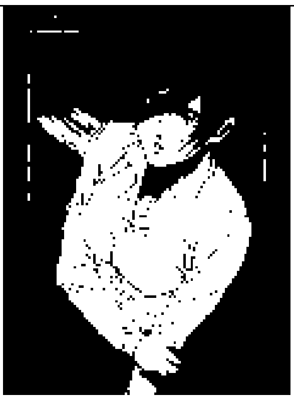
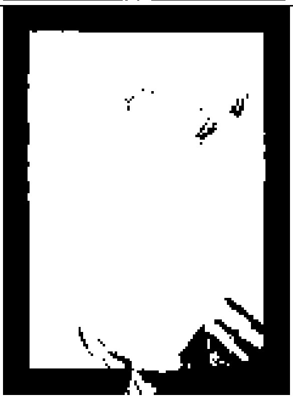
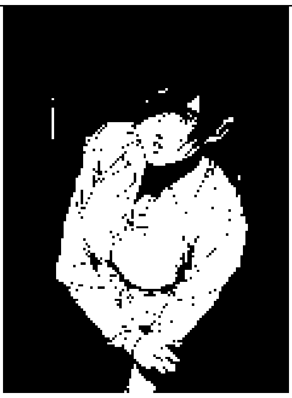
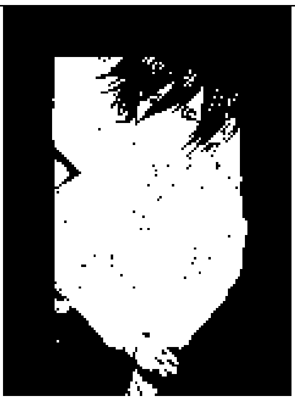
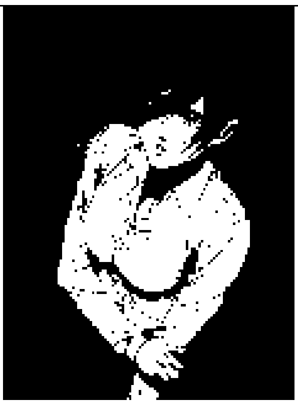
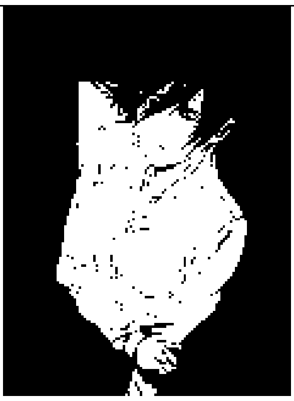
Original:

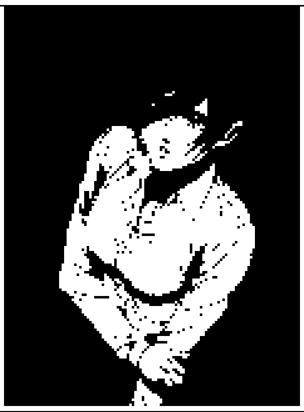
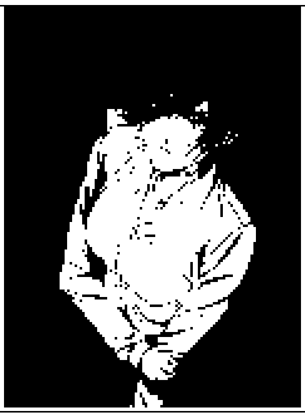
Size3 59*80	Delta_t: 0.1	Delta_t: 0.01
Mar:1		
Mar:5		
Mar: 10		
Mar: 20		

Size2: 118*160

Original:

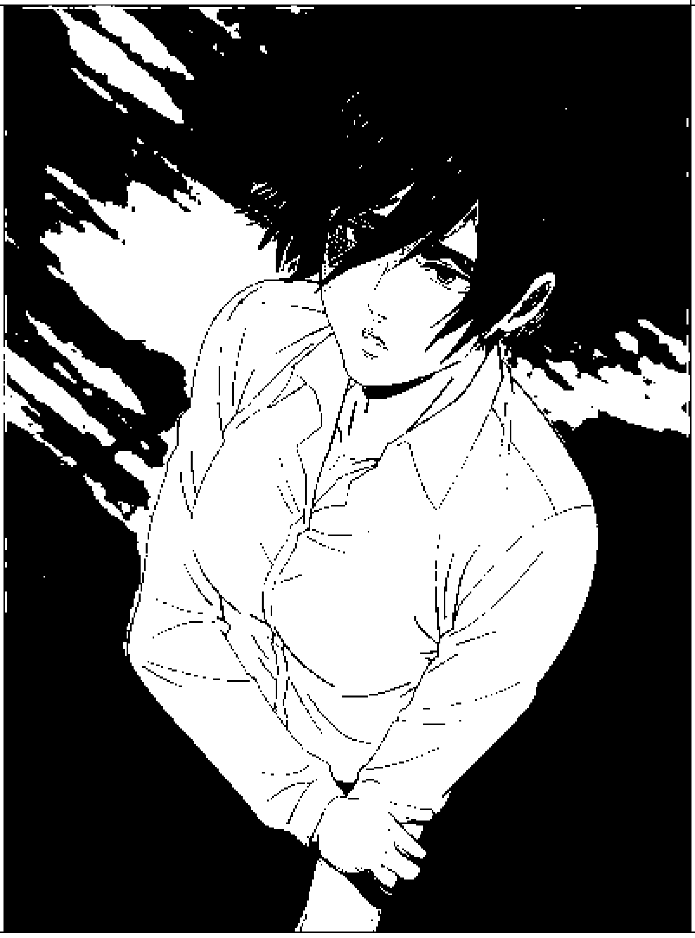
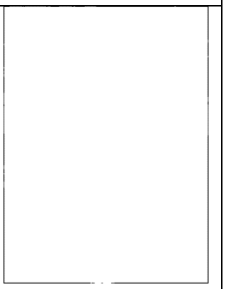


Size2: 118*160	Delta_t: 0.1	Delta_t: 0.01
Mar:1		
Mar:10		
Mar: 20		
Mar: 30		

Mar: 40		

Size1: 96*534

Original:

Size1: **96*534**	Delta_t: 0.1	Delta_t: 0.01
Mar:1		

Mar:10		
Mar: 30		

Mar: 50		
Mar: 100		

结论与讨论

1. 可以看到，对于图像强度相对均匀的图片，CV 模型也有较好的分割效果。本次实验自选的一系列图片，其实强度并不是很均匀，不过 CV 的效果依然不错。

2. CV 模型初始 Phi 的 margin（矩形 Phi 的边缘到图像的边缘的距离）的设置对于图像的分割效果有一定影响。CV 模型中 margin 很小时（如 1）就能自动完成内部轮廓的搜索，这与课本知识一致。同时，本实验中可以注意到，它似乎更容易搜索基于 Phi 矩形内部信息的轮廓，初始 Phi 的矩形边界外部的轮廓似乎不容易体现。

如此看来，初始化 Phi 可以直接取很小的 margin，从而自动完成内部所有轮廓的搜索。不过，如果想强调内部对象的轮廓，就可以取较大的 margin，从而识别出更单纯的内部对象的轮廓。（参见 Size1 结果中 Mar 的设置带来的结果变化）

3. 本实验在 CV 模型时，尝试设置基于每次变化的 stop 条件，也就是如果变化很小，就直接终止。但是这样的 stop 条件似乎很难设置，应为 Phi 的更新其实每次都是很小的，如果想直接设置 stop 条件，很容易很快地提前终止，而完全没有达到分割效果。所以，之后直接设置为 Max_Iter。

4. 本实验设置 Max_Iter 为 10000 次，而 Phi 梯度流中的 delta_t（每次 Phi 变化的步长）设置要与之匹配，才能确保 Phi 充分演变完毕。从实验结果的对比很容易看到，10000 次对应的 delta_t 取为 0.1 较合适，0.01 就过小了，Phi 离演变完全还很远。也就是说，如果取 delta_t 为 0.01， 需要设置更大 Max_Iter。

对于一般化的其他图片，Max_Iter 和 delta_t 的合理设置还有待进一步探究。

实验二、RSF 模型实验

目的和意义

本实验采用 C++语言，对经典的 RSF 模型通过基础代码进行复现。本实验能够加深对于 RSF 这一基于局部信息的图像分割模型的理解，体会该算法与 CV 模型的差异。

模型介绍与说明

不同于 CV 模型用两个常量 c1 和 c2 来近似轮廓线 C 两侧的区域 inside(C) 和 outside(C) 内的图像强度。RSF 模型用两个拟合函数$f_1(x)$和$f_2(x)$来拟合 C 两侧的区域内的图像强度。注意到f_1和 f_2 的值是随着中心点 x 的变化而变化的。f_1和 f_2的这种空间变化性质使得 RSF 模型从本质上区别于 CV 模型，这种空间变化性质来源于空间变化的核函数 K_σ 的局部化性质。RSF 模型的区域可伸缩性也来源于核函数K_σ 的尺度参数 σ， 可以控制局部区域的大小，从小的邻域到整个定义域，这样就可以在一个可控制尺度的区域内充分利用图像的强度信息用于引导活动轮廓线的移动。

RSF 模型通过使用核函数K_σ充分利用图像的局部强度信息，因此该模型可以分割具有图像强度不均匀性质的图像，而且对于一些具有弱边界的物体如血管等的分割有很好的效果。

但是 RSF 模型仅仅利用图像的局部信息可能会导致能量泛函的局部极小，因此 RSF 模型的分割结果会更加依赖于轮廓线的初始化。此外，因为 RSF 模型是非凸的，这也是导致局部极小解存在的一个原因。

模型的数学原理

RSF 的能量泛函如下：

引入高斯核函数:

$$K_\sigma(\vec{u}) = \frac{1}{2\pi\sigma^2} e^{-|\vec{u}|^2/2\sigma^2}$$

$\Omega_1 = \text{inside}(C)$, $\Omega_2 = \text{outside}(C)$

$$\mathcal{F}^{RSF}(C, f_1(\vec{x}), f_2(\vec{x})) = \sum_{i=1}^{2} \lambda_i \int_\Omega \left(\int_\Omega K(\vec{x}-\vec{y}) \left| u_0(\vec{y}) - f_i(\vec{x}) \right|^2 d\vec{y} \right) d\vec{x}$$

$$+ \nu |C|$$

($\lambda_1, \lambda_2 > 0$; $f_1(\vec{x})$ $f_2(\vec{x})$ 近似 $\vec{x}$ 邻域的图像强度)

对该泛函模型，类似地，引入水平集函数进行统一表示，同时添加一项水平级正则项（用来保持水平集函数的正规性，保证计算精确性，同时避免重新初始化水平集函数的过程），最后进行光滑化，得到如下光滑的 RSF 水平集格式的能量泛函：

$$\mathcal{F}^{RSF}(\phi, f_1, f_2) = \mathcal{E}_\varepsilon^{RSF}(\phi, f_1, f_2) + \nu \mathcal{L}_\varepsilon(\phi) + \mu \mathcal{P}(\phi)$$

$$= \sum_{i=1}^{2} \lambda_i \int_\Omega \left(\int_\Omega K_\sigma(\vec{x}-\vec{y}) \left| u_0(\vec{y}) - f_i(\vec{x}) \right|^2 M_i^\varepsilon(\phi(\vec{y})) d\vec{y} \right) d\vec{x}$$

$$+ \nu \int_\Omega \left| \nabla H_\varepsilon(\phi(\vec{x})) \right| d\vec{x}$$

$$+ \int_\Omega \frac{1}{2} \left(|\nabla \phi(\vec{x})| - 1 \right)^2 d\vec{x}$$

$$= \int_\Omega \Big[\sum_{i=1}^{2} \lambda_i \Big(\int_\Omega K_\sigma(\vec{x}-\vec{y}) \left| u_0(\vec{y}) - f_i(\vec{x}) \right|^2 M_i^\varepsilon(\phi(\vec{y})) d\vec{y}$$

$$+ \nu \left| \nabla H_\varepsilon(\phi(\vec{x})) \right| + \frac{1}{2} \left(|\nabla \phi(\vec{x})| - 1 \right)^2 \Big] d\vec{x}$$

其中 $M_1^\varepsilon(\phi) = H_\varepsilon(\phi)$， $M_2^\varepsilon(\phi) = 1 - H_\varepsilon(\phi)$

为了极小化该能量泛函，采用标准的梯度下降法，交替极小化格式，求解f_1, f_2, ϕ如下：

$$f_1(\vec{x}) = \frac{K_\sigma(\vec{x}) * \left[M_1^\varepsilon(\phi(\vec{x})) u_0(\vec{x}) \right]}{K_\sigma(\vec{x}) * M_1^\varepsilon(\phi(\vec{x}))}$$

$$f_2(\vec{x}) = \frac{K_\sigma(\vec{x}) * \left[M_2^\varepsilon(\phi(\vec{x})) u_0(\vec{x}) \right]}{K_\sigma(\vec{x}) * M_2^\varepsilon(\phi(\vec{x}))}$$

$$\frac{\partial \phi}{\partial t} = -\delta_\varepsilon(\phi)(\lambda_1 e_1 - \lambda_2 e_2) + \nu \delta_\varepsilon(\phi) \operatorname{div}\left(\frac{\nabla \phi}{|\nabla \phi|} \right) + \mu \left(\nabla^2 \phi - \operatorname{div}\left(\frac{\nabla \phi}{|\nabla \phi|} \right) \right)$$

其中 $e_1(\vec{x}) = \int_\Omega K_\sigma(\vec{y}-\vec{x}) \left| u_0(\vec{x}) - f_1(\vec{y}) \right|^2 d\vec{y}$

$$e_2(\vec{x}) = \int_\Omega K_\sigma(\vec{y}-\vec{x}) \left| u_0(\vec{x}) - f_2(\vec{y}) \right|^2 d\vec{y}$$

程序设计流程

依照数学过程，离散化处理，转化为程序。代码见附录。

实验结果

结论与讨论

可以注意到，对于图像强度不均匀的场景，RSF 的分割效果较 CV 有较大的提升。不过，RSF 也暴露出对初始形状相当敏感的问题，调节初始参数是一件困难的事情。

附录 代码

实验一 CV C++ opencv

```
// to reproduct CV algorithm
// the original picture may be CV_8U, what if I do not change it,
directly use a CV_32F Phi on it? the calculation of Phi should can
"include" the CV_8U type.
// PHi needs negative pixel, but there is no need to change the
original picture!
// so we can try just same size between Img and Phi, but Phi with
different type CV_32F for calculation, and when it comes to show Phi on
Img, then make some convertion.
// so Img CV_8U, while PHI and NGX, NGY, div :  CV_32F (for calcuation)
// maybe use CV_64F (double) for higher accuracy?

// let us try the experiment picture
// stoppage is not suitable! each time just tiny moves, so only use
max_iter!!
// adjust last Phi,to distinguish negative(as 0, outside) and
positive(255, inside)
// max_iter may be 1e4, 1e5; 1e5 too large and slower, 1e4 is preferred
// for 1e4 max_iter, delt_t can not be too tiny like 0.01, 0.1 is
suitable

// use vector for grid search of parameters!
// make testMyCVTotal() to systematically handle those pictures and use
grid search!

// for different pictures, may use Threads !! to parallelly handle them!
to try...

    // from the result, it seeems that Phi is easier to check edges
inside the original Phi?
#include <filesystem>
#include <string>
#include <vector>
#include <math.h>
#include <cmath>
#include <iostream>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\opencv.hpp>
#include "MatAt.hpp" // my hpp to uniformly get Mat_at
#include "progressbar.hpp" // to show progress

using namespace std;
using namespace cv;
namespace fs = std::filesystem; // rename

const float  PI_F=3.14159265358979f;
```

"MatAt.hpp"

```
// to give an automatica mat_at() without typename
// FMat_at return float of data !
//
#ifndef MATAT_HPP_
#define MATAT_HPP_

#include <opencv2/core.hpp>
#include <iostream>
using namespace std;

// Function for accessing pixel values of CV_8U Mat
float FMat_at(cv::Mat& img, int row, int col) {
    if(img.type()==0){ // 8U
        return img.at<uchar>(row, col);
    }else if(img.type()==5){ // 32F
        return img.at<float>(row, col);
    }else{ // many other situation
        cout << " type error" << endl;
        exit(0);
    }
}

#endif
```

"progressbar.hpp"

```
#ifndef _PROGRESSBAR_HPP
#define _PROGRESSBAR_HPP

#include <iostream>
using namespace std;

void ProgressShow(int loopNum, int i, int SectionNum=10){ // inside
loop, get index i(0~loopNum-1)
            int section = loopNum/SectionNum;

            // at start
            if(i == 0){
                cout <<  "progress: 0%" ;
                return;
            }

            if( (i+1)%section == 0){
                cout << "\r";  // return to the head
                cout << "progress: " << (i+1)/section*100
/SectionNum<<"%"; // those int!! should *100 first, or get 0!
            }
}

#endif
```

class MyCV

```
class myCV
{
    private:
        Mat Img;
        int Rows; // height of Img
        int Cols; // width of Img
        int type; // type of Img, 0:CV_8U, 5:CV_32F,   should make a
type dict!
        int digitT = 2;
        float delta_t = pow(0.1, digitT); // for gradient decline of
Phi
        float epsilon = 1e-6;
        // float sigma = 25.0;
        float C1;
        float C2;
        float Lambda1;
        float Lambda2;
        float Nu;
        Mat Phi;   // 2D
        int Margin;
        int Max_Iter;

        // stop delta
        float stopC1Delta=1e-6;
        float stopC2Delta=1e-6;
        float stopPhiDelta = 1e-6;

    public:
        myCV(Mat img = Mat(), int margin =10, int max_iter=100, int
thickness = 1, float c1=1, float c2=1, float lambda1=1, float lambda2=1,
float nu=1){
            // c1, c2, Phi to update
            Img = img;
            type = Img.type();
            Rows = Img.rows;
            Cols = Img.cols;
            if(Img.channels()>1){
                ExtractChannel();
                cout << "multichannel, have already extracted one
channel picture." << endl;
                exit(0);
            }
            // no change on Img, just set Phi!

            C1 = c1; // inside  first 0?
            C2 = c2; // outside first 255?
            Lambda1 = lambda1;
            Lambda2 = lambda2;
            Nu = nu;

            Margin = margin;
            InitPhi(Margin, thickness); // Init Phi

            Max_Iter = max_iter;
        }
```

// initPhi()

```
        void InitPhi(int margin, int thickness){ // yes!  but actually
the 0 in middle is not neccessary
            Phi = cv::Mat::ones(Img.size(), CV_32F);  // set Phi CV_32F
for calculation
            Phi = -1*Phi;  // outside edge as -1, just white

            // first Rect, get 0 inside
            cv::Rect PhiInner1(margin,margin,Phi.size().width-2*margin,
Phi.size().height-2*margin); // choose inner ROI
            Phi(PhiInner1) = 0*Phi(PhiInner1);  // edge as 0

            // second Rect, get -1 inside the edge, thickness of the
edge is 1
            int marginInner = margin + thickness; // margin to the
outer
            cv::Rect PhiInner2(marginInner, marginInner,
Phi.size().width-2*marginInner, Phi.size().height-2*marginInner); //
choose inner ROI
            Phi(PhiInner2).convertTo(Phi(PhiInner2), -1, 1.0, 1); //
add 1, inner as +1
        }

        void ImgInfo(){  // cout the information
            ImgType();
            channels();
            cout << GetImgMat().size() <<  endl;
        }

        void ImgType(){
            cout << "type: "<< Img.type() << endl;
        }

        void channels(){
            cout << "channels:  " << Img.channels() << endl;;
        }

        void ExtractChannel(){ // get 1 channel
            std::vector<cv::Mat> channels;  // vector of each channel
            cv::split(Img, channels);
            cv::Mat oneChannel = channels[0];
            oneChannel.convertTo(oneChannel, CV_32F); // convert to 32-
bit floating point
            Img = oneChannel;
            cv::imshow("OneChannel",Img);
            cv::waitKey(0);
            cv::imwrite("onechannel.png", Img,
{cv::IMWRITE_PNG_COMPRESSION, 0});
        }

        void setEpsilon(float x=0.001){
            epsilon = x;
        }
```

// updataC1()

```
        float updataC1(){ // return the change
            float numerator=0;
            float denominator=0;

            for (int i = 0; i < Img.rows; i++) {
                for (int j = 0; j < Img.cols; j++) {
                    // get local point, use the short name instead of
at<>..
                    float phi_local = Phi.at<float>(i,j);

                    float u_local;
                    u_local = FMat_at(Img, i,j);

                    numerator += u_local*Heaviside(phi_local);
                    denominator += Heaviside(phi_local);
                }
            }
            float delta = abs(C1-numerator/denominator) ;
            C1 = numerator/denominator ;
            return delta;
        }
```

// updataC2()

```
    float updataC2(){ // return change
        float numerator=0;
        float denominator=0;

        for (int i = 0; i < Img.rows; i++) {
            for (int j = 0; j < Img.cols; j++) {
                // get local point, use the short name instead of
at<>..
                float phi_local = Phi.at<float>(i,j);
                // float u_local = Img.at<uchar>(i,j);  // uchar
for CV_8U
                float u_local = FMat_at(Img, i,j);

                numerator += u_local*(1-Heaviside(phi_local));
                denominator += (1-Heaviside(phi_local));
            }
        }
        float delta = abs(C2-numerator/denominator) ;
        C2 = numerator/denominator ;
        return delta;
    }
```

// Heaviside()

```
    float Heaviside(float x){ // approximately distinguish: x>=0
get 1 and x<0 get 0
        return ( 1+(2/PI_F)*(atan(x/epsilon)) )/2.0;
    }

    void resetPhi(){ // to reset the phi into -1 or 1 ??
        for (int i = 0; i < Img.rows; i++) {
            for (int j = 0; j < Img.cols; j++) {
                float phi_local = Phi.at<float>(i,j);
                Phi.at<float>(i,j) =
(2/PI_F)*(atan(phi_local/epsilon))  ; // into -1 or 1
            }
        }
    }
```

```
        float Dif_H(float x){ // no use?
            return (1/PI_F)*( epsilon/( x*x+ epsilon*epsilon) ) ;
        }
```

// NormalizedGradient()

```
        void NormalizedGradient(Mat Phi, Mat& Gx, Mat& Gy){ // Gx, Gy
to store gradient in x/y order
            // original Phi, only +1, -1

            // Calculate the gradient in the x and y directions
            cv::Sobel(Phi, Gx, CV_32F, 1, 0); //X order
            cv::Sobel(Phi, Gy, CV_32F, 0, 1); // Y order

            // Access the gradient values
            for (int i = 0; i < Img.rows; i++) {
                for (int j = 0; j < Img.cols; j++) {
                    float gx = Gx.at<float>(i, j);
                    float gy = Gy.at<float>(i, j);
                    float len = sqrt(gx*gx + gy*gy);  // len of the
gradient
                    if(len < 1e-6){ // gx=gy=0
                        // len = 1 ; // for 0 in denominator! the gx,gy
get 0 naturally
                        Gx.at<float>(i, j) = 0;
                        Gy.at<float>(i, j) = 0; // set 0
                    }else{
                        Gx.at<float>(i, j) = gx/len;
                        Gy.at<float>(i, j) = gy/len; // normalized
                    }
                }
            }
        }
```

// divNormalizedPhi()

```
        void divNormalizedPhi(Mat& div, Mat NGx, Mat NGy){ //
normalized Gx,Gy
            // get div
            div = cv::Mat::zeros(Img.size(),CV_32F); // initialize! //
32F for float TYPE
```

```
            cv::Mat gradientX, gradientY; // gx, gy in Mat
            cv::Sobel(NGx, gradientX, CV_32F, 1, 0); //X order for
Phi_x
            cv::Sobel(NGy, gradientY, CV_32F, 0, 1); // Y order for
Phi_y

            // Access the gradient values
            for (int i = 0; i < Img.rows; i++) {
                for (int j = 0; j < Img.cols; j++) {
                    float gx = gradientX.at<float>(i, j);
                    float gy = gradientY.at<float>(i, j);
                    div.at<float>(i,j) = gx+gy;  // get div
                }
            }
        }

        void setDeltT(int digit_T=2){ // set delta t for gradient
decline
            digitT = digit_T;
            delta_t = pow(0.1, digitT);
        }
```

// updatePhi()

```
        float updatePhi(){
            Mat NGX, NGY;
            NormalizedGradient(Phi, NGX, NGY);

            Mat div;
            divNormalizedPhi(div, NGX, NGY);  // get div

            float delta=0;

            for (int i = 0; i < Img.rows; i++) {
                for (int j = 0; j < Img.cols; j++) {
                    // get local point, use the short name instead of
at<>..
                    float phi_local = Phi.at<float>(i,j);
                    float div_local =  div.at<float>(i,j);  // div
local!
                    float u_local = FMat_at(Img, i,j);
                    // updat Phi
                    float delta_local
=  Dif_H(phi_local)*( Nu*div_local - Lambda1*pow(u_local-C1 ,2) +
Lambda2*pow(u_local-C2, 2) );
                    Phi.at<float>(i,j) += delta_t* delta_local;

                    delta += delta_local;
                }
            }
            return delta;
        }

        void setStop(float stopc1=1e-6, float stopc2=1e-6, float
stopPhi=1e-6){
            stopC1Delta = stopc1;
            stopC2Delta = stopc2;
            stopPhiDelta = stopPhi;
        }

        bool stop(float deltaC1, float deltaC2, float deltaPhi){
            if( deltaC1 < stopC1Delta && deltaC2 < stopC2Delta &&
deltaPhi < stopPhiDelta ){ // may more strict?
                return true;
            }else{
                return false;
            }
        }
```

// fit()

```
        void fit( int max_iter = 100 ){

            bool finishFlag = false;

            Max_Iter = max_iter;

            int maxtimes;
            for(int i=0; i<Max_Iter; i++){
                ProgressShow(Max_Iter, i);

                float deltaC1 = updataC1();
                float deltaC2 = updataC2();
                float deltaPhi = updatePhi();
                // ban stoppage  // it seems that, each time just small
changes, so stop condition is not suitable!
                // if(stop( deltaC1, deltaC2, deltaPhi) && false) {
                //     finishFlag = true;
                //     maxtimes = i+1;
                //     cout << endl;
                //     cout << "finish at index "<< i+1 << endl;
                //     break;
                // }
            }

            LastPhiAdjust();

            fitInfoReport(finishFlag, maxtimes);
        }
```

// LastPhiAdjust()

```
        void LastPhiAdjust(){  // make Phi easier to show the edge,
change from 32F into 8U?
            // edge between negative and positive
            // directly, negative(outside) into 0, positive(inside)
into 255
            for(int i=0; i<Phi.rows; i++){
                for(int j=0; j<Phi.cols; j++){
                    float phi_local = Phi.at<float>(i,j);
                    if(phi_local >=0){
                        Phi.at<float>(i,j) = 255; // white
                    }else{
                        Phi.at<float>(i,j) = 0;  // black
                    }
                }
            }
            Phi.convertTo(Phi, CV_8U); // no value change, just change
type
        }

        void fitInfoReport(bool finishFlag, int maxtimes){
            // CV reports
            cout << endl;
            cout<< "Img size: "<< Img.size() << endl;
            cout << "Phi original margin: " << Margin << endl;
            cout << "Delta_t: " << delta_t << endl;
            // cout << cv::format("stop condition: stopC1Delta: %f,
stopC2Delta: %f, stopPhiDelta: %f ", stopC1Delta, stopC2Delta,
stopPhiDelta)  << endl;
            // cout << "Max_Iteration: " << Max_Iter << endl;
            cout << "C1: " << C1 << " C2: " << C2 << endl;

            if(finishFlag){
                cout << "finish up to stop condition, at times: "<<
maxtimes << endl;
            }else{
                cout << "finish up to max_iter: " << Max_Iter << endl;
            }
        }
```

// LastPhiOutput()

```
        void LastPhiOutput(string pathbase){ // save Phi in the
designated folder
            // save Phi
            cv::imwrite( pathbase + "/" +
cv::format("Phi_Mar%d_C1_%d_C2_%d_Iter%d_deltT_1e-%d.png",
Margin ,static_cast<int>(C1), static_cast<int>(C2),Max_Iter, digitT ),
Phi);
        }
        Mat getPhi(){
            return Phi;
        }

        void showPhi(string windowName = "Phi"){  // can set the
windowname
            cv::namedWindow(windowName, cv::WINDOW_NORMAL); // then
drag for changing size
            imshow(windowName, Phi);
            cv::waitKey(0);
        }

        void showImg(string windowName = "Img"){
            cv::namedWindow(windowName, cv::WINDOW_NORMAL);
            imshow(windowName, Img);
            cv::waitKey(0);
        }
        Mat GetImgMat(){
            return Img;
        }

        void scaleSize(float scale){ // should operate before Phi
operation
            int h = Img.cols;
            int w = Img.rows;
            // resize(Img, Img, Size(int(w*scale),
int(h*scale)),0,0,INTER_LINEAR);
            resize(Img, Img, cv::Size(Img.size().width*scale,
Img.size().height*scale) ,0,0,INTER_LINEAR);
            // cv::namedWindow("Scale", cv::WINDOW_NORMAL);
            cv::imshow("Scale", Img);
            cv::imwrite("mikasa1_onechannell_resize4.png", Img);
            cv::waitKey(0);
        }

};
```

testMyCVTotal()

```
void testMyCVTotal(){ // total test, different size of mikasa picture,
automatically store Phi results in the corresponding folder
    // in ./mikasaTest or ./mikasa :  4 pic: resize1,2,3,4 respectively:
396*534, 118*160; 59*80; 29*40
    // how to set marginList?
    // for max_iter: 10000, delta_t: 0.1 is suitable, 0.01 is too small!

    fs::path path1 = "./mikasaTest"; // test // ./ can represent the
current path!

    // iteration traverse
    fs::directory_iterator list(path1); // it works!!
    for(auto& it:list){  // resize 1,2,3,4
        fs::path file = it.path().filename();
        fs::path filenameNoExtension = file.stem();
        // cout<< filename << endl;

        string newFolderStr = "./PhiRes_" +
filenameNoExtension.string(); // folderName
        fs::path newFolder(newFolderStr);
        if( ! fs::exists( newFolder ) ){
            if( fs::create_directory(newFolder) ){
            cout << "create: "<< newFolder << endl;
            }
        }

        // operate the file. use new folder as output folder
        cv::Mat img = cv::imread(file.string(), cv::IMREAD_UNCHANGED);

        vector<int>DeltaTdigit({1,2});

        // different MarList for different size?
        vector<int>MarList;
        vector<int>MarList1({1,10,30,50,100});
        vector<int>MarList2( {1,10,20,30,40});
        vector<int>MarList3({1,5,10,20});
        vector<int>MarList4({1,5,10});

        string filenameNoExtensionStr = filenameNoExtension.string();
        int resizeNum
=  static_cast<int>(filenameNoExtensionStr.at(filenameNoExtensionStr.si
ze()-1)) - '0';
        switch (resizeNum){
            case 1:
                MarList = MarList1;
                break;
            case 2:
                MarList = MarList2;
                break;
            case 3:
                MarList = MarList3;
                break;
            case 4:
                MarList = MarList4;
                break;
            default:
                break;

        }

        for(auto digit : DeltaTdigit){  // iterator ! auto!
            for(auto mar : MarList){
                myCV cv = myCV(img, mar); // margin of Phi edge to
outer:5, thickness of Phi edge is 1  // a margin bigger than 1 is
better for cv! suitable margin is important
```

```
                cv.ImgInfo();
                cv.setDeltT(digit);  // 0.1 is enough for 10000 iter, 
0.01 too small
                cv.fit(1e4); // max_iter
                cv.LastPhiOutput(newFolderStr);
            }
        }

    }

}
```

Main()

```
int main(){
    cout << "I am a fool but I can make it" << endl;

    testMyCVTotal();

    cout << "YES!" << endl;

    return 0;
}
```

实验二 RSF