# 数字图象处理

# 小作业五

2019010485　自 91　刘祖炎*

2021 年 11 月 21 日

## 1　任务一：脊线分割

### *1.1* 算法原理及代码分析

根据题目提供的 Gabor 滤波增强后的指纹图片，仿照综合作业一的方法进行脊线分割。首先，利用高斯滤波器进行平滑滤波，使得后续脊线更加平滑，此后利用 *imbinarize* 函数进行图像二值化处理。程序代码如下：

```
img = gaussFilter(img);
img = imbinarize(img, 'adaptive', 'ForegroundPolarity',
    'dark', 'Sensitivity', binarizeParam);

function Period = gaussFilter(Period)
    filter = fspecial('gaussian', [3, 3], 1);
    Period = imfilter(Period, filter, 'replicate', 'same');
end
```

其中，*binarizeParam* 为根据图片预先设定的参数。

完成图像二值化后，需要利用 *bwareaopen* 函数进行形态学运算，去除二值图中的缺陷，包括脊线上的空洞、脊线轮廓上的不规则凹凸、白背景上的孤岛等。具体去除方法为，设定小面积需要去除的阈值，并两次应用 *bwareaopen* 函数分别在白色背景、黑色背景的图片上，以分别去除不同类型的缺陷。程序代码如下：

```
img = bwareaopen(img, openParam, 8);
img = ~img;
img = bwareaopen(img, openParam, 8);
```

其中，*openParam* 为根据图片预先设定的参数。

*liuzuyan19@mails.tsinghua.edu.cn

### 1.2 运行结果

以第一张指纹图片为例，程序运行结果如下所示。其中，图1为原图，2为经过高斯滤波后的灰度图，3为运用 *imbinarize* 处理后的二值化图，4为进行形态学运算后的二值化图。明显可以看到，高斯滤波可以使图像更平滑，经过形态学处理后，大量缺陷部分得到修复。

图 1: 原图

图 2: 高斯滤波结果

图 3: 二值化结果

图 4: 形态学处理结果

## 2 任务二：脊线细化

### 2.1 算法原理及代码分析

对得到的二值化图，首先需要利用 *bwmorph* 函数进行脊线细化。细化后，利用 *bwareaopen* 函数去除过短的线段。程序代码如下：

```
1 img = bwmorph(img, 'thin', inf);
2 img = bwareaopen(img, 5, 8);
```

其中，*'thin'* 表示细化，*inf* 表示细化至单像素宽。

完成细化后，需要对所得图片进行后处理，去除短线、毛刺、桥接等缺陷。去除后，利用 *bwareaopen* 函数去除过短的线段。程序代码如下：

```
img = Cutting(img, iterParam1, iterParam2);
img = bwareaopen(img, 5, 8);

function result = Cutting(img, iterParam1, iterParam2)
    endpoint = zeros(3, 3, 8);
    endpoint(:, :, 1) = reshape([0,1,0,-1,1,-1,-1,-1,-1], [3,3]);
    endpoint(:, :, 2) = reshape([-1,-1,0,-1,1,1,-1,-1,0], [3,3]);
    endpoint(:, :, 3) = reshape([-1,-1,-1,-1,1,-1,0,1,0], [3,3]);
    endpoint(:, :, 4) = reshape([0,-1,-1,1,1,-1,0,-1,-1], [3,3]);
    endpoint(:, :, 5) = reshape([1,-1,-1,-1,1,-1,-1,-1,-1], [3,3]);
    endpoint(:, :, 6) = reshape([-1,-1,1,-1,1,-1,-1,-1,-1], [3,3]);
    endpoint(:, :, 7) = reshape([-1,-1,-1,-1,1,-1,-1,-1,1], [3,3]);
    endpoint(:, :, 8) = reshape([-1,-1,-1,-1,1,-1,1,-1,-1], [3,3]);
    iter_time = iterParam1;
    [height, width] = size(img);
    img_init = img;
    for i = 1:iter_time
        hit = zeros(height, width, 8);
        for j = 1:8
            hit(:, :, j) = bwhitmiss(img, endpoint(:, :, j));
        end
        for j = 1:8
            img = img - hit(:, :, j);
        end
    end
    img_rec = bwmorph(img, 'endpoints', inf);
    se = strel('disk', iterParam2);
    img_rec = imdilate(img_rec, se) & (img_init);

    result = img | img_rec;
    imshow(result);
end
```

其中，*Cutting* 函数为后处理函数。基本原理与 PPT 中介绍的修剪算法相同。我们假设毛刺的长度不超过 *iterParam*1 像素，通过迭代删除端点并重复上述操作 *iterParam*1 次，即可将毛刺部分突出的端点进行修剪。

定义如图5所示的八种端点形态，在代码中定义为 *endpoint* 数组。利用 *bwhitmiss* 函数在原图中寻找满足条件的端点位置，找到后将对应位置的像素值置零，即完成了一次端点的修剪。迭代上述操作 *iterParam*1 次，即可

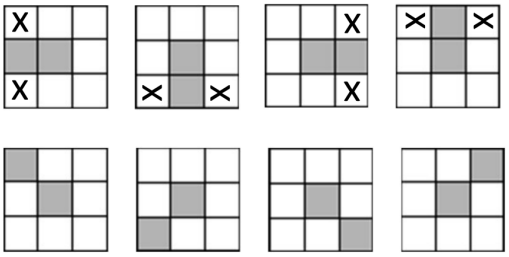

图 5: 端点形态

其中，寻找端点的公式如下所示。该公式利用 *bwhitmiss* 函数实现。

$$X_2 = \bigcup_{k=1}^{8} (X_1 \otimes B^k) \tag{1}$$

由于正常的端点也在上述操作中被缩短，故需要对其进行恢复。恢复算法的公式如下所示。该公式利用 *bwmorph* 函数先找到端点后，利用 *imdilate* 函数对端点进行扩张实现。

$$X_3 = (X_2 \oplus H) \cap A \tag{2}$$

最后，将延长后的端点与原图进行并运算，公式如下所示。

$$X_4 = X_1 \cup X_3 \tag{3}$$

## *2.2* 运行结果

以第一张指纹图片为例，运行结果如下所示。其中，图6为直接进行细化后的图片，7为对端点进行修剪后的图片，8为对修剪完成后的图片进行恢复的图片。可以看到，每一步均实现了预期效果，后处理完成的图片基本不存在毛刺部分。

图 6: 细化结果

图 7: 修剪结果

图 8: 恢复结果

# 3 任务三：细节点检测

## 3.1 算法原理及代码分析

细节点检测的方式为计算每个像素 $p$ 的交叉数 $cn(p)$，计算公式为：

$$cn(p) = \frac{1}{2}\sum_{i=0}^{7}|f(p_{(i+1)mod\ 8}) - f(p_i)| \tag{4}$$

其中，$f(p_i)$ 表示像素 $p_i$ 处的图像值，交叉数为 1 表示端点，交叉数为 3 表示分叉点，其他交叉数表示非细节点。参照上述公式，编写细节点检测代码如下：

```
dir = zeros(9,2);
dir(:, 1) = [-1,-1,-1,0,1,1,1,0,-1];
dir(:, 2) = [-1,0,1,1,1,0,-1,-1,-1];
[height, width] = size(img);
cn = zeros(height, width);
for i = 2:height - 1
    for j = 2:width - 1
        if (img(i, j) == 0)
            continue
        end
        for x = 1:8
            cn(i,j) = cn(i,j)+ abs(img(i+dir(x,1),j+dir(x,2))
                    - img(i+dir(x+1,1),j+dir(x+1,2)));
        end
    end
end
[endPoint_x, endPoint_y] = find(cn == 2);  %端点
[branPoint_x, branPoint_y] = find(cn == 6);  %分支点
```

上述代码中，$endPoint\_x, endPoint\_y$ 为端点坐标，$branPoint\_x, branPoint\_y$ 为分支点坐标。

完成细节点检测后，需要将上述坐标点绘制在细化后的指纹图像上。利用 *rectangle* 函数在画布上进行绘制，代码如下。

```
img_plot = ~img;
figure(1);
imshow(img_plot);
hold on;
for idx = 1:length(endPoint_x)
    rectangle('Position', [endPoint_y(idx), endPoint_x(idx), 3, 3],
     'LineWidth', 1, 'EdgeColor', 'r');
end
for idx = 1:length(branPoint_x)
    rectangle('Position', [branPoint_y(idx), branPoint_x(idx), 3, 3],
     'LineWidth', 1, 'EdgeColor', 'b');
```

```
12 end
```

### 3.2 运行结果

以第一张指纹图片为例，程序运行结果如图9所示。可以看到，程序准确识别出了端点与分支点 (红色表示端点，蓝色表示分支点)。

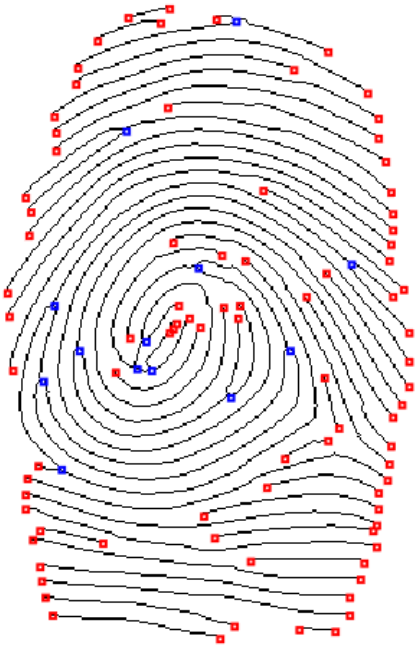

图 9: 细节点

## 4 任务四：细节点验证

### 4.1 算法原理及代码分析

任务三所得的细节点包含了指纹外轮廓的伪细节点。需要进行进一步验证以排除干扰。验证原理为，根据图片背景色获取前景分割矩阵，并利用形态学膨胀将前景分割矩阵向内膨胀 $maskParam$ 像素。对每一个细节点，若处于前景分割矩阵外部，则将其舍弃。代码如下。

```
1 if length(greyParam) > 1
2     ROI_mask = (img_init == greyParam(1) / 255);
3     ROI_mask = ROI_mask | (img_init == greyParam(2) / 255);
4 else
5     ROI_mask = (img_init == greyParam / 255);
6 end
7 ROI_mask = bwareaopen(ROI_mask, 32, 8);
```

```
se = strel('disk', maskParam);
ROI_mask = imdilate(ROI_mask, se);

for i = length(endPoint_x):-1:1
    if (ROI_mask(endPoint_x(i), endPoint_y(i)) == 1)
        endPoint_x(i) = [];
        endPoint_y(i) = [];
    end
end
```

需要注意的是，由于第二张指纹图片的背景色并非同一种颜色，故此处需要对两种灰度背景进行处理。对获取到的前景分割矩阵，也需要通过 *bwareaopen* 函数去除小面积部分的干扰。

### *4.2* 运行结果

以第一张指纹图片为例，程序运行结果如下所示。可以看到，程序成功去除了外侧的干扰点。

图 10: 前景分割结果

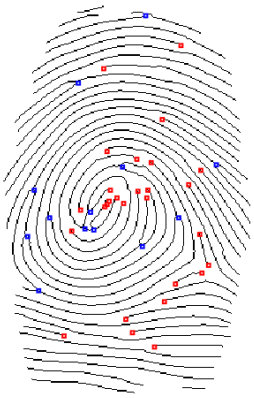

图 11: 细节点验证结果

## 5 参数选择与程序结果

运行程序时，修改 *code.m* 中第一行文件名为对应的图片名即可。程序将输出脊线分割、脊线细化、细化后处理、细节点标记四张图片，分别命名为 *step*1.*png*、*step*2.*png*、*step*3.*png*、*step*4.*png*。

### *5.1* 指纹图一

通过参数调节，如下设置参数：

表 1: 参数设置

| 参数内容 | 参数值 |
|---|---|
| 二值化阈值 | 0.56 |
| 形态学阈值 | 64 |
| 背景灰度 | 126 |
| 向内膨胀值 | 20 |
| 修剪长度 | 12 |
| 恢复长度 | 8 |

每一步的图片输出结果分别如图12、13、14、15、16所示。

图 12: 二值化结果

图 13: 细化结果

图 14: 细化后处理结果

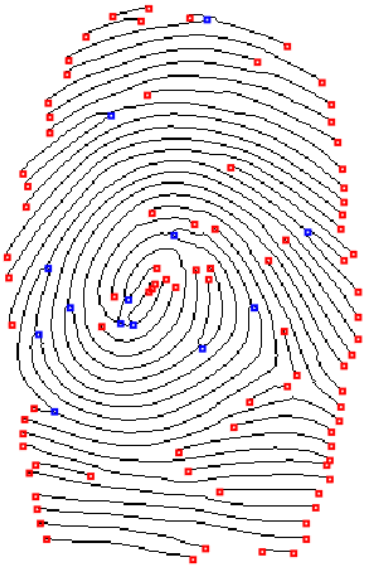

图 15: 细节点检测结果

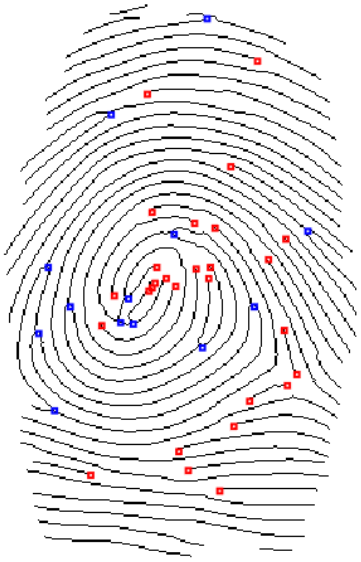

图 16: 细节点处理结果

## *5.2* 指纹图二

通过参数调节，如下设置参数：

表 2: 参数设置

| 参数内容 | 参数值 |
|---|---|
| 二值化阈值 | 0.58 |
| 形态学阈值 | 64 |
| 背景灰度 | 124, 125 |
| 向内膨胀值 | 30 |
| 修剪长度 | 12 |
| 恢复长度 | 8 |

每一步的图片输出结果分别如图17、18、19、20、21所示。

图 17: 二值化结果

图 18: 细化结果

图 19: 细化后处理结果

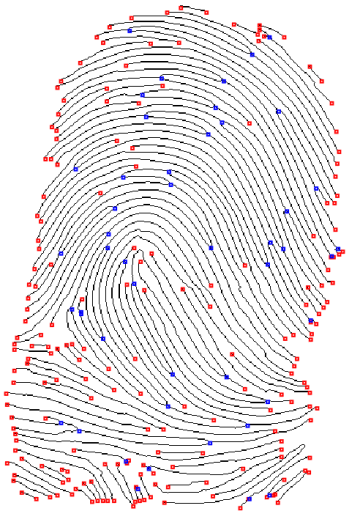

图 20: 细节点检测结果

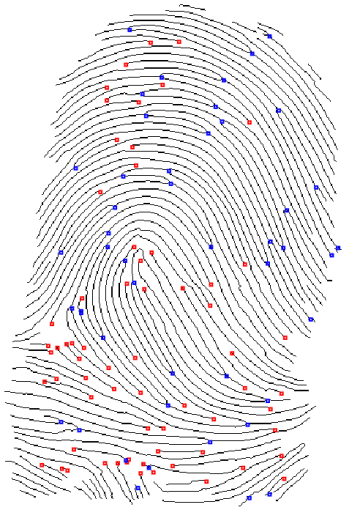

图 21: 细节点处理结果

### *5.3* 指纹图三

通过参数调节，如下设置参数：

表 3: 参数设置

| 参数内容 | 参数值 |
|---|---|
| 二值化阈值 | 0.58 |
| 形态学阈值 | 64 |
| 背景灰度 | 126 |
| 向内膨胀值 | 10 |
| 修剪长度 | 10 |
| 恢复长度 | 5 |

每一步的图片输出结果分别如图22、23、24、25、26所示。

图 22: 二值化结果

图 23: 细化结果

图 24: 细化后处理结果

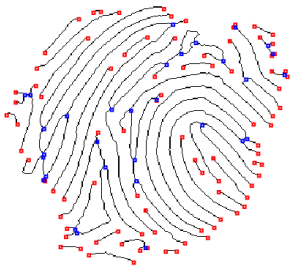

图 25: 细节点检测结果

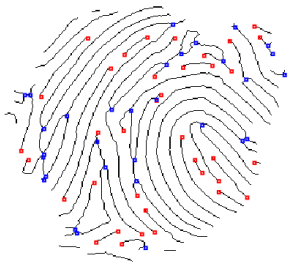

图 26: 细节点处理结果

# 6 遇到的困难与解决方案

由于此次作业指导较为详细，步骤较为明确，故没有遇到太多困难，大部分时间花费在参数选取与最终结果调试上。

- 修剪后端点恢复问题

  由于修剪后端点会以相等的距离被缩短，故需要设计合适的算法对缩短后的端点进行恢复。笔者此处参考了 PPT 中的思路，对完成修剪后的端点进行一定程度的延长处理。此外，笔者也尝试了一些其他的算法，包括在修剪前对端点进行寻迹，计算端点与最近的分叉点之间的距离，只修剪长度短于阈值的端点等方式。最终，笔者选择了 PPT 中提供的算法，并实现了较好的效果。

- 二值化与细化步骤

  笔者发现，二值化、形态学处理与细化步骤对最终修剪与细节点验证的结果影响较大。因此，笔者尝试通过一些方式在二值化与细化过程中即提高图像的质量。利用的方法主要包括：高斯滤波、$imbinarize$ 函数中的 $'adaptive'$ 选项、合适的形态学处理参数与方式等。细化的后处理步骤只能在很小的范围内对图像进行优化，事实上，通过得到质量较高的初始图能够很大程度提高结果的美观程度。

# 7 收获与心得体会

通过此次作业，笔者对指纹的了解有了进一步的加深。此前虽然已经通过数次作业进行了多种不同方式的指纹处理，但仍然没有尝试将指纹与实际应用联系起来。通过对指纹细节点进行识别，笔者通过几次作业的循序渐进，实现了指纹从获取到最终进行识别的完整算法。

完成本次作业后，笔者充分理解了上课所讲的形态学处理的各种知识，进一步充实了自己对图像处理的了解。