基于迭代（自动阈值）算法的医学图像增强方法

1 算法原理介绍

1.1 基于灰度阈值的图像分割原理

采用灰度阈值对图像进行分割是图像分割的基本方法之一。通过设定灰度阈值，把图像的象素点按灰度等级进行分类，把图像分割成若干子图像。在实际应用中，最常用的一种分割方法是将图像分割成高灰度区和低灰度区两部分，组成一幅二值图像。该分割方法按式 1 对图像进行操作。

$$g(x,y)=\begin{cases}255\,, f(x,y)\geq T\\ 0\,, f(x,y)<T\end{cases} \qquad (1)$$

其中，T 为预先设定的灰度阈值，f 为输入图像，g 为输出图像。基于灰度阈值对图像进行分割的基本条件是式(1)中阈值 T 的选择。虽然采用人工方法往往可以获得较理想的阈值，但是在很多情况下，需要计算机自动完成阈值的选择，这样就要求有合适的算法对图像的灰度直方图进行分析，选择合适的阈值。在实际中常采用迭代算法或 Ostu 法对阈值进行自动计算[1]，本文仅介绍迭代算法。迭代算法的基本思想是：首先设定一个阈值的估计值；采用一定的算法反复对该估计值进行修正，保证每次修正后的结果都优于前一次；当进行一定次数的修正之后，结果趋于收敛，即相邻两次的结果的差异较小，当该差异小到可接受范围时，表明一个理想的阈值已经求出。最后利用该阈值按式(1)对图像进行操作，即完成了图像的自动阈值分割。

但是，一些图片由于照度不均、阴影、对比度差异等，使得如果采用同一阈值对整幅图片进行处理（即全局阈值）时会出现不兼容图像各处的情况，使得分割效果变差。这时，可以考虑将图片分割成若干子图片，将每个子图片按自动阈值算法进行处理，然后再将各个子图片的处理结果合并成整体结果输出，该方法称局部阈值或动态阈值法。

1.2 基于图像自动阈值分割的边缘检测

采用全局阈值或局部阈值获得的二值图像可以方便地应用在图像的边缘检测当中。由于图像已经转换为二值图像，所以对图像中边缘信息的提取较为方便，仅需判断某像素是区域内部点还是边缘点即可。实际操作中，对二值图像中黑色（或白色）点的四邻域进行判断，若该点的四邻域均为黑色（或白色），即可判断该点为区域内部点而不是边缘点。依次对图像中每个象素的四邻域进行判断，即可得到图像的边缘信息。

2 程序设计

本次实验中，程序采用 Matlab 语言编写。采用基于迭代的自动阈值算法分别实现了对图像的全局阈值分割、局部阈值分割，并利用全局阈值分割的结果，实现图片的边缘检测，增强了图片中的边缘信息。

2.1 迭代算法的实现[2]

根据 1.1 节中介绍的迭代算法思想，采用如下语句实现利用迭代算法找出用于图像分割的阈值。

```
t=mean(gray_image(:));                          %设置估计值
is_done=false;                                  %迭代完成标志位
count=0;                                        %迭代计数归零
while ~is_done                                  %迭代循环
    r1=find(gray_image<=t);                     %按前次结果 t 对图像二分
    r2=find(gray_image>t);
    temp1=mean(block(r1));                      %求 r1 区域均值
    if isnan(temp1);                            %保证结果非 NaN
        temp1=0;
    end
    temp2=mean(block(r2));                      %求 r2 区域均值
    if isnan(temp2)                             %保证结果非 NaN
        temp2=0;
    end
    t_new=(temp1+temp2)/2;
    is_done=abs(t_new-t)<1;                     %检测两次迭代计算结果的差值
    t=t_new;                                    %更新结果
    count=count+1;                              %迭代计数增 1
    if count>=1000                              %判断是否迭代次数过多
        Error='Error:Cannot find the ideal threshold.'      %出错提示
        Return                                  %退出程序
    end
end
```

该程序中，首先将估计值设为灰度图像的灰度均值。该估计值对最终处理结果不产生影响，但该值越接近最终结果，则迭代计算的次数越少，增加程序效率。程序中采用 is_done 标志来决定是否继续进行迭代计算，is_done 标志由判断两次迭代结果的差值决定，本程序中设定若两次迭代结果相差小于 1 则视为迭代结束。在某些极端情况下，该算法无法找出理想阈值，迭代计算将陷入死循环，所以在程序的迭代循环中，除了进行迭代结果的计算外，还增加了迭代次数的检测，当迭代运算进行了 1000 次仍没有找出理想阈值时，视为迭代失败，退出程序。同时，还对区域求均值得操作结果作了检验，如果结果为 NaN，就将结果更正为 0，这样增强了整个程序的健壮性，防止程序陷入死循环。

2.2 全局阈值图像分割的实现

对图像进行全局阈值分割按照图 1 的流程进行。

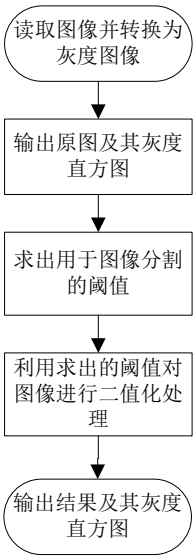

图 1 全局阈值图像分割程序流程

其中，自动求阈值采用 2.1 节中的程序实现。对图像进行二值化处理采用如下程序进行。

```
[m,n]=size(gray_image);                    %获得原图尺寸
result=zeros(m,n);                         %申请内存空间用来存放结果
result(r1)=255;                            %将图像二值化处理
resule(r2)=0;
```

经过上述程序，得到的二值化图像就存放在了数组 result 中。全局阈值图像分割程序的整体代码见附录 I。

2.3 局部阈值图像分割的实现

局部阈值图像分割的原理与全局阈值分割相似，只是在使用 2.1 节的程序前，将图像分割成若干子图像进行处理，在处理后，在将各自的结果拼接起来。该部分程序流程图见图 2。

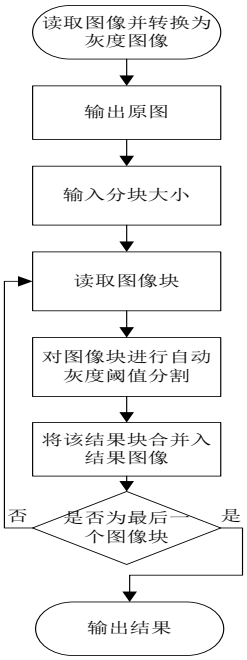

图 2 局部阈值图像分割程序流程

该程序中，对每个图像块进行自动阈值分割的程序与 2.1 节中的程序相同，不再赘述。采如下程序对图像进行分块与合并结果。

```
block_size=input('Block size=');                    %输入分块大小
for i=1:block_size:m                                %依次对图像分块并处理
    for j=1:block_size:n
        if ((i+block_size)>m)&&((j+block_size)>n)
            block=gray_image(i:end,j:end);          %右下角区块
        elseif ((i+block_size)>m)&&((j+block_size)<=n)
            block=gray_image(i:end,j:j+block_size-1);  %最右列区块
        elseif ((i+block_size)<=m)&&((j+block_size)>n)
            block=gray_image(i:i+block_size-1,j:end);  %最下行区块
        else
            block=gray_image(i:i+block_size-1,j:j+block_size-1); %普通区块
        end
        … … … …  %该部分为基于迭代的灰度阈值分割程序，同 2.1 节，略
        if ((i+block_size)>m)&&((j+block_size)>n)
            result(i:end,j:end)=block;              %右下角区块
        elseif ((i+block_size)>m)&&((j+block_size)<=n)
            result(i:end,j:j+block_size-1)=block;   %最右列区块
        elseif ((i+block_size)<=m)&&((j+block_size)>n)
            result(i:i+block_size-1,j:end)=block;   %最下行区块
        else
            result(i:i+block_size-1,j:j+block_size-1)=block; %普通区块
        end
    end
end
```

在程序中，由于无法保证图片的尺寸能被输入的区块大小整除，即最右列和最下行区块可能为非完整区块，所以在分块时要对这些区块特殊处理。程序中采用 if 语句来判定区块类型，并选择合适的操作。该部分完整代码见附录 II。

2.4 利用二值图像进行图像边缘的检测

利用迭代算法得到了图像的二值图像后，可利用该二值图像进行图像边缘的检测。从二值图像中提取图像的边缘较为方便，既对图像中黑色或白色区域中每一个像素的四邻域进行判断即可判断该像素是区域内点还是边缘点。用于边缘检测的代码如下：

```
edge=zeros(m,n);
for k=2:1:m-1
    for j=2:1:n-1
        if result(k,j)==255                         %白色区域
            if ((result(k,j)==255)&&(result(k+1,j)==255)&&
              (result(k-1,j)==255)&&(result(k,j+1)==255)&&
              (result(k,j-1)==255))                 %判断四邻域
                edge(k,j)=255;                      %区域内点
            else
```

```
                edge(k,j)=0;                    %边缘点
            end
        else
            edge(k,j)=255;                      %其他区域
        end
    end
end
```

经过上述代码，将在 edge 数组中得到图像的边缘。将得到边缘合并到原图，即可完成图像的边缘增强。

采用如下代码完成图像边缘与原图像的合并。

```
mix=gray_image;
mix=uint8(mix);
for k=1:1:m
    for j=1:1:n
        if edge(k,j)==0
            mix(k,j)=0;
        end
    end
end
```

经上述程序，最终的结果存放在 mix 数组中。该部分完整程序见附录 III。

3 结果分析

本实验中，选用头部 MRI 照片（图 3）用于评估上述程序的性能。

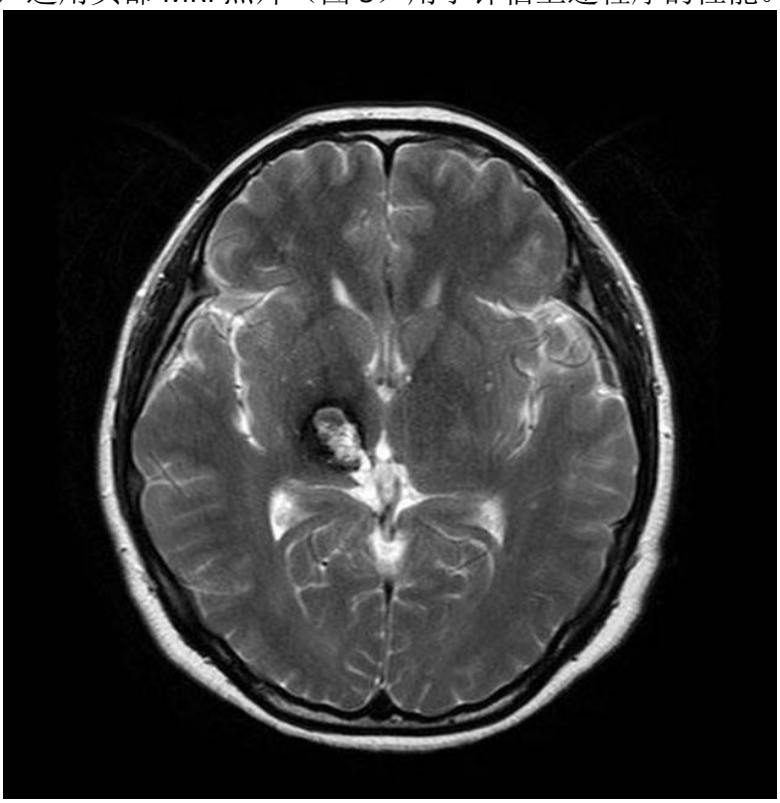

图 3 脑瘤患者头部 MRI 照片

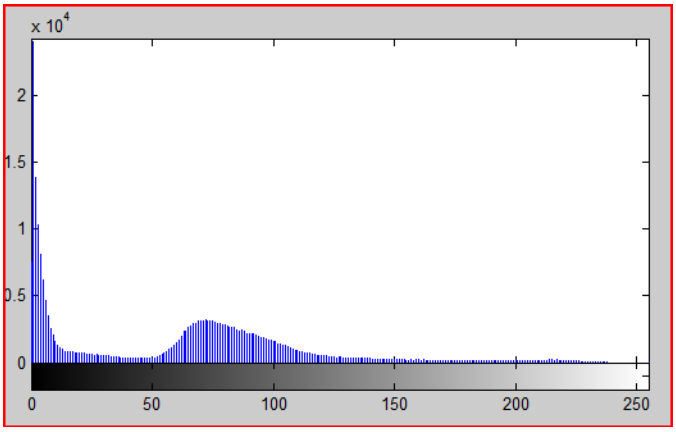

图 4 图 3 的灰度直方图

3.1 全局阈值分割结果

采用基于迭代算法的全局阈值分割对图 3 进行操作，经 2 次迭代运算，得到结果图 5，自动求得的分割阈值为 53，即图 4 中的谷点。

图 5 全局阈值分割得到的二值图像

3.2 局部阈值分割结果

采用局部阈值对图像进行分割，分割的结果与选择的子图像区块大小有直接关系。分别以 10×10、50×50 和 100×100 为区块大小进行处理，结果分别见图 6（a）、6（b）和 6（c）。

图 6 局部阈值分割结果

从图 6 的结果中可以看出，区块的大小对局部阈值分割的结果有很大的影响。当选择区块较小时，由于很多区块仅处于背景区域中，所以将背景中的噪声被放大了。

3.3 边缘检测及增强的结果

对利用结果图 5，进行边缘检测，得到图像的边缘见图 7。

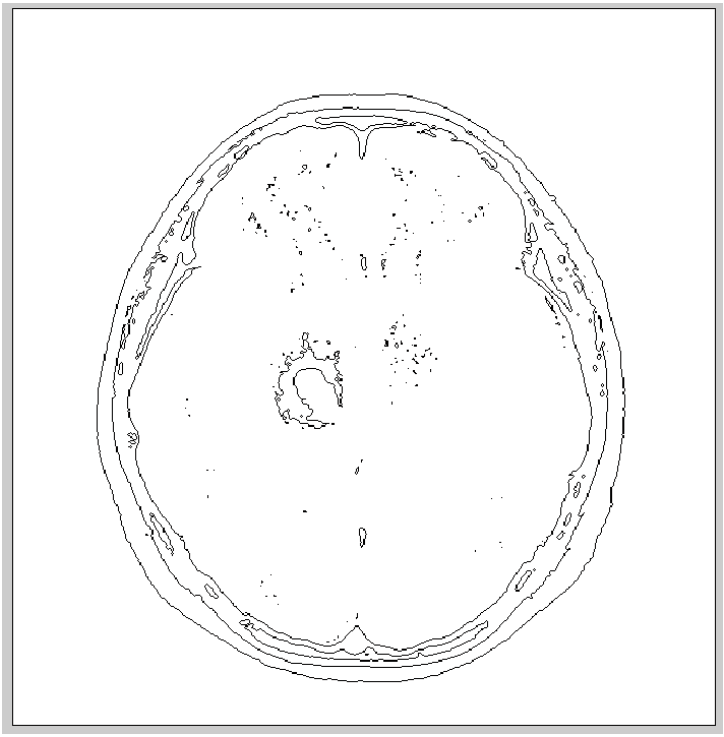

图 7 边缘检测的结果

将图 7 与原图合并，得到最终结果图 8。

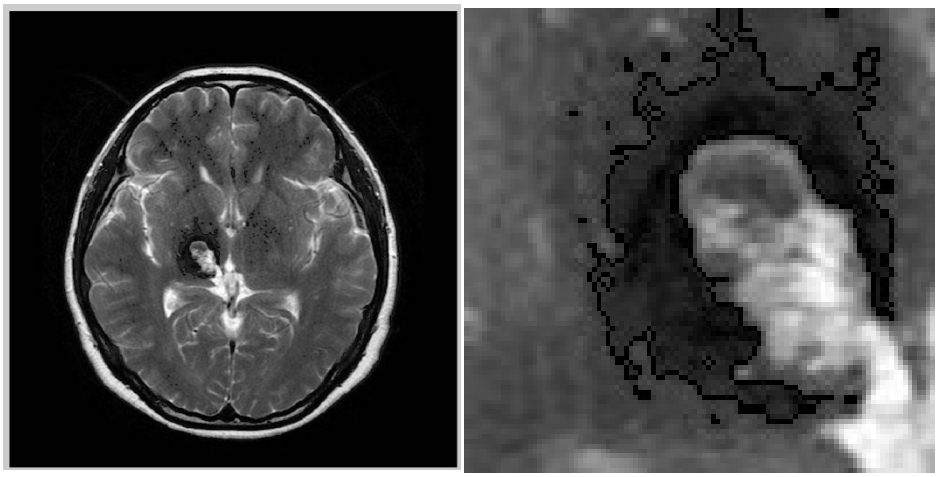

图 8 边缘增强结果及该图局部放大

4 结论

基于迭代算法能够叫高效地自动求出灰度阈值，对于大多数图像，迭代算法的结果都是收敛的，即能在有限迭代次数内求理想阈值。该迭代算法可以用于全局阈值分割和局部阈值分割处理中。对于一般的图像，采用全局阈值分割能获得较好的分割效果，并且利用分割后得到的二值图像可以方便地提取出图像的边缘信息，用于图像边缘提取与增强，得到较清晰的图像轮廓。但是，对于由于曝光不均等原因导致的不同位置灰度差异过大的图片，采用全局阈值分割得到的结果往往不理想。对于这样的图片，可以采用局部阈值分割进行处理。采用局部阈值分割时，分割的结果受对图片进行分块的大小的影响。分块时，图片的分块不宜过小，因为如果分块过小，每个区块内部的像素过少，统计规律不明显，分割效果不好，且会对背景中的噪声敏感。

参考文献

[1] 陈冬岚,刘京南,余玲玲.几种图像分割阈值选取方法的比较与研究[J].电气技术与自动化,2003,(1):77-80.

[2] 姚敏.数字图像处理(M).北京,机械工业出版社,2006.

附录 I：全局阈值分割程序

```
original_image=imread('w:\head.jpg');
gray_image=rgb2gray(original_image);
subplot(1,2,1);
imshow(original_image);
subplot(1,2,2);
imhist(gray_image);
gray_image=double(gray_image);
t=mean(gray_image(:));
is_done=false;
count=0;
while ~is_done
    r1=find(gray_image<=t);
    r2=find(gray_image>t);
    temp1=mean(block(r1));                    %求 r1 区域均值
    if isnan(temp1);                          %保证结果非 NaN
        temp1=0;
    end
    temp2=mean(block(r2));                    %求 r2 区域均值
    if isnan(temp2)                           %保证结果非 NaN
        temp2=0;
    end
    t_new=(temp1+temp2)/2;
    is_done=abs(t_new-t)<1;
    t=t_new;
    count=count+1;
    if count>=1000
        Error='Error:Cannot find the ideal threshold.'
        return
    end
end
count                                         %输出迭代次数
t                                             %输出阈值计算结果
[m,n]=size(gray_image);
result=zeros(m,n);
result(r1)=255;                               %图像二值化
resule(r2)=0;
result=uint8(result);
figure
imshow(result);
```

附录 II：局部阈值分割程序

```
original_image=imread('w:\leg.jpg');
gray_image=rgb2gray(original_image);
subplot(1,2,1);
imshow(original_image);
subplot(1,2,2);
gray_image=double(gray_image);
[m,n]=size(gray_image);
result=zeros(m,n);
block_size=input('Block size=');                    %输入分块大小
for i=1:block_size:m
    for j=1:block_size:n
        if ((i+block_size)>m)&&((j+block_size)>n)      %分块
            block=gray_image(i:end,j:end);
        elseif ((i+block_size)>m)&&((j+block_size)<=n)
            block=gray_image(i:end,j:j+block_size-1);
        elseif ((i+block_size)<=m)&&((j+block_size)>n)
            block=gray_image(i:i+block_size-1,j:end);
        else
            block=gray_image(i:i+block_size-1,j:j+block_size-1);
        end
        t=mean(block(:));
        t_org=t;
        is_done=false;
        count=0;
        while ~is_done                                %迭代算阈值
            r1=find(block<=t);
            r2=find(block>t);
            temp1=mean(block(r1));
            if isnan(temp1);
                temp1=0;
            end
            temp2=mean(block(r2));
            if isnan(temp2)
                temp2=0;
            end
            t_new=(temp1+temp2)/2;
            is_done=abs(t_new-t)<1;
            t=t_new;
            count=count+1;
            if count>=1000
                Error='Error:Cannot find the ideal threshold.'
                return
            end
        end
        block(r1)=255;
```

```
            block(r2)=0;
            if ((i+block_size)>m)&&((j+block_size)>n)                    %合并结果
                  result(i:end,j:end)=block;
            elseif ((i+block_size)>m)&&((j+block_size)<=n)
                 result(i:end,j:j+block_size-1)=block;
            elseif ((i+block_size)<=m)&&((j+block_size)>n)
                  result(i:i+block_size-1,j:end)=block;
            else
                  result(i:i+block_size-1,j:j+block_size-1)=block;
            end
      end
end
resule=uint8(result);
figure
imshow(result);
```

附录 III：基于全局阈值分割的边缘增强程序

```
original_image=imread('w:\head.jpg');
gray_image=rgb2gray(original_image);
subplot(2,2,1)
imshow(original_image);
subplot(2,2,2);
imhist(gray_image);
gray_image=double(gray_image);
t=mean(gray_image(:));
is_done=false;
count=0;
while ~is_done                                  %迭代算阈值
      r1=find(gray_image<=t);
      r2=find(gray_image>t);
      temp1=mean(block(r1));                    %求 r1 区域均值
      if isnan(temp1);                          %保证结果非 NaN
            temp1=0;
      end
      temp2=mean(block(r2));                    %求 r2 区域均值
      if isnan(temp2)                           %保证结果非 NaN
            temp2=0;
      end
      t_new=(temp1+temp2)/2;
      is_done=abs(t_new-t)<1;
      t=t_new;
      count=count+1;
      if count>=1000
            Error='Error:Cannot find the ideal threshold.'
            return
      end
end
  [m,n]=size(gray_image);
result=zeros(m,n);
result(r1)=255;
resule(r2)=0;
result=uint8(result);
subplot(2,2,3);
imshow(result);                                 %输出阈值分割结果
edge=zeros(m,n);                                %提取边缘
for k=2:1:m-1
      for j=2:1:n-1
            if result(k,j)==255
                  if ((result(k,j)==255)&&(result(k+1,j)==255)
                    &&(result(k-1,j)==255)&&(result(k,j+1)==255)
                    &&(result(k,j-1)==255))         %判断四邻域
                        edge(k,j)=255;
```

```
                else
                    edge(k,j)=0;
                end
            else
                edge(k,j)=255;
            end
        end
end
subplot(2,2,4);
imshow(edge);                                   %输出边缘
mix=gray_image;
mix=uint8(mix);                                 %合并边缘与原图
for k=1:1:m
    for j=1:1:n
        if edge(k,j)==0
            mix(k,j)=0;
        end
    end
end
figure
imshow(mix);                                    %输出最终结果
```