

MATLAB 连连看大作业

实验报告

班级：无 56

姓名：暴志鹏

学号：2015010675

一、制作自己的连连看

1.1 运行 linkgame

本题目没有实验代码。

在 Matlab 中设置正确的工作路径，在命令行输入 linkgame 即可运行连连看。若要运行外挂，需要在打开 Matlab 时给予管理员权限，且允许按键精灵管理员权限运行。图为在 mac os 下运行连连看截图。



1.2 改写 detect.m

代码见 codes/linkgame/detect.m。

对于本题，我采取的方法是分类讨论，我认为连连看的消去情况可以分成三种情况：直线相连、有一个转折和有两个转折。对此，我在 detect.m 下内置了三个子函数 detect1,

detect2, detect3 来分别处理。

Detect1 想法较为简单，实现略为繁琐。

```
function b1 = detect1(mtx, x1, y1, x2, y2)
b1 = 0;
if (x1==x2)
    if(y1>y2)
        tmp = y2;
        y2 = y1;
        y1 = tmp;
    end
    d = mtx(x1,y1+1:y2-1);
    if (d*d'==0)
        b1 = 1;
    end
else
    if(y1==y2)
        if(x1>x2)
            tmp = x2;
            x2 = x1;
            x1 = tmp;
        end
        d = mtx(x1+1:x2-1,y1);
        if(d'*d==0)
            b1 = 1;
        end
    end
end
end
end
```

detect2 的思路我认为只需要以两个坐标点为对角线构造一个矩阵，借助矩阵加以判断即可。用 matlab 内置的矩阵操作实现很容易。

```
function b2 = detect2(mtx, x1, y1, x2, y2)
b2 = 0;
imp1 = mtx(x2,y1);
imp2 = mtx(x1,y2);
```

```

if (imp1^2*imp2^2==0)
    if(x1>x2)
        tmpx = x1;
        x1 = x2;
        x2 = tmpx;
        tmpy = y1;
        y1 = y2;
        y2 = tmpy;
    end
    mi = min(y1,y2);
    ma = max(y1,y2);
    l1 = mtx(x1+1:x2-1,y1);
    l2 = mtx(x1+1:x2-1,y2);
    r1 = mtx(x1,mi+1:ma-1);
    r2 = mtx(x2,mi+1:ma-1);
    if(l1'*l1+r2*r2'+mtx(x2,y1)^2==0 |
l2'*l2+r1*r1'+mtx(x1,y2)^2==0)
        b2 = 1;
    end
end
end

```

detect3 是建立在前两者的基础上，对一个元素的横向纵向区域点进行依次判断，看其是否和两个给定点分别满足直线可消和一个折点可消。

```

function b3 = detect3(mtx, x1, y1, x2, y2)
[m, n] = size(mtx);
row = zeros(m,1);
line = zeros(1,n+2);
mtx = [row,mtx,row];
mtx = [line;mtx;line];
x1 = x1+1;
y1 = y1+1;
x2 = x2+1;
y2 = y2+1;
b3 = 0;
for count = 1:m+2
    if(count == x1 | count == x2)
        continue
    end

```

```

    if(mtx(count,y1)==0 && detect2(mtx,count,y1,x2,y2)
&& detect1(mtx,count,y1,x1,y1))
        b3 = 1;
        break;
    end
end
if(b3==0)
    for count = 1:n+2
        if(count == y1 | count ==y2)
            continue;
        end
        if(mtx(x1,count)==0 &&
detect2(mtx,x1,count,x2,y2)&&detect1(mtx,x1,count,x1,y1))
            b3 = 1;
            break;
        end
    end
end
end

```

在主函数中，只需要依次调用三个子函数即可。

1.3 修改 omg.m

代码见 codes/linkgame/omg.m

对于本题目，我没有特地设计算法，而是用了简单的遍历查找。

从坐标最小点开始，找所有和它下表相同的点，依次判断是否可消，

若全部不可消，则换下一个点，知道找到可以消除的点对。

需要说明的是，对于本题目确实有改进算法，即不借助 detect.m，直接从边界点入手进行横向纵向延时进行深度优先搜索。

但是出于以下考虑，我没有使用改进算法：1)就题目的终极目的而言，omg 的时间复杂度远小于做匹配滤波的时间复杂度，不会影响外挂运行时间。2)改进算法需要考虑多种情况，容易出问题且没有与

detect 相联系。3)改进算法由于是从一点开始中心扩展，很容易造成四周小精灵的堆积从而形成死局，经室友测试，死局率较高。

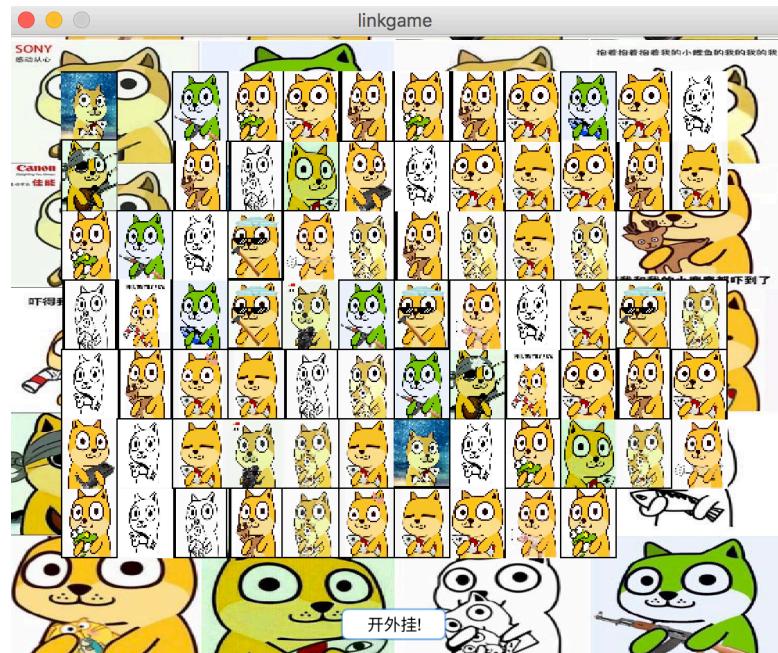
核心算法如下：

```
if(mtx(x1,y1)==0)
    continue;
end
tmp=mtx(x1,y1);
mtx(x1,y1)=0;
[line,row]=find(mtx==tmp);
mtx(x1,y1)=tmp;
for j=1:length(line)
    x2=line(j);
    y2=row(j);
    if detect(mtx,x1,y1,x2,y2)
        mtx(x1,y1)=0;
        mtx(x2,y2)=0;
        steps=[steps,x1,y1,x2,y2];
        break;
    end
end
```

1.4 自由发挥环节

代码见 codes /linkgame/bg.mat pics.mat

研究老师给的文件发现背景和小精灵图片是存储在 pics.mat 和 bg.mat 中，修改这两个 mat 文件，即可改变图像界面。我修改它们为下图，难度超高。（高能预警_(:з)∠）_）：

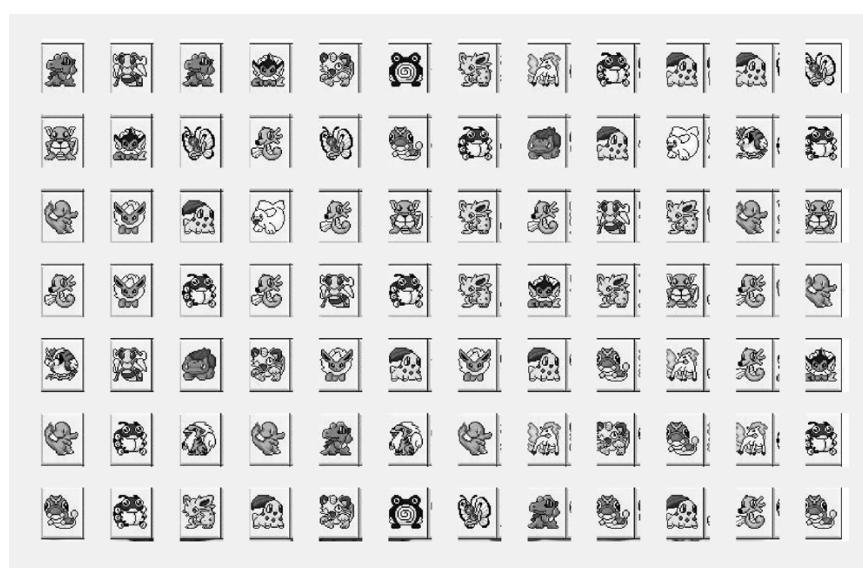


二、 攻克别人的连连看

2.1 对 graygroundtruth 进行分割。

代码见 codes/process/q2_1.m

本题目初始运行结果如下图 1，改进运行结果如下图 2。本题目实际上用多种方法都可以解出结果，最终代码选择了与后面题目一致的算法。具体优化过程见 2.2.



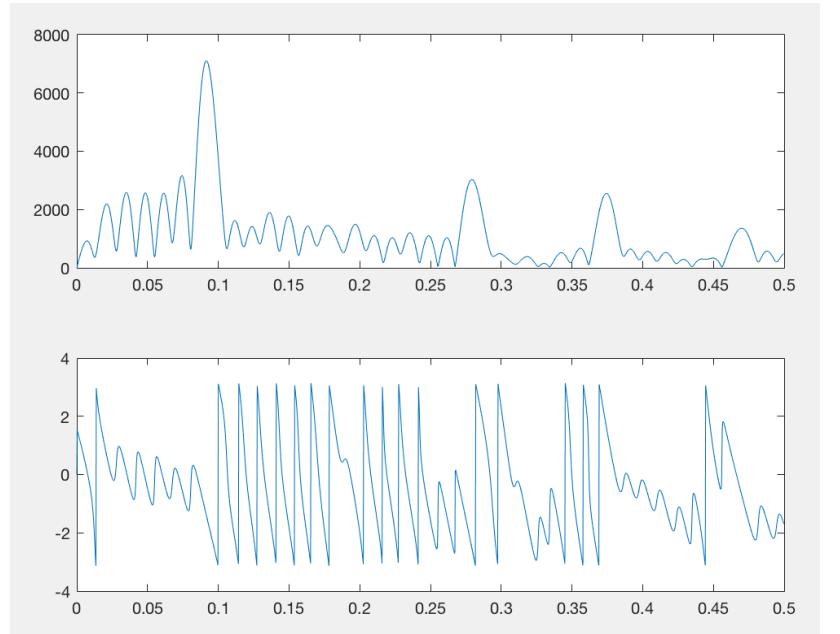


2.2 对 graycapture 进行分割

代码见 codes/process/q2_2.m gaotong.m q2_2.mat 。

对于本题目的算法，我进行了三次改进。起初对于灰度截图我是采用了选择灰度最小值点的周期性分界的。但是这显然不适合拍照图片。

分割结果完全是错误的。之后我尝试采用老师的方法，用幅度谱来解决这个问题。



分割结果如下：

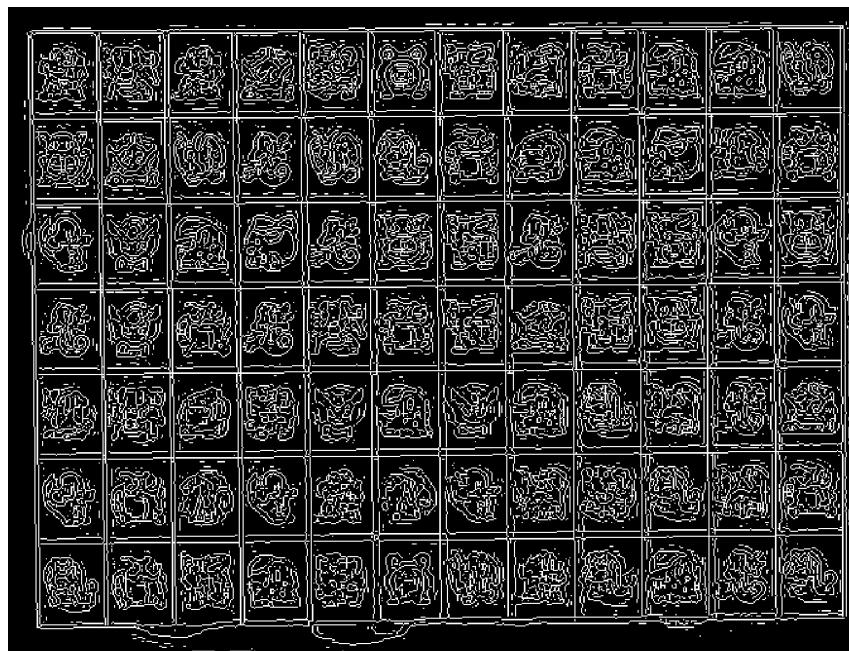


可以发现存在两个比较大的问题，其一是幅度谱基频不一定总是最大且并非所有高倍频峰值都很显著。用类似之前音乐合成的方法确实可以解决此问题，但是准确的仍然不够高。此外，对于照片照歪的情况并不能解决的很好。

对于以上两个问题我分别做了解决。

对于第一个问题，我最终决定不采用傅里叶变换法，而是沿用之前的方法，不过需要对图片进行预处理。首先我想到的是图片边缘提取。

用 Matlab 内置的 edge 函数可以很好的完成此操作。效果较好的参数调整结果如下图：

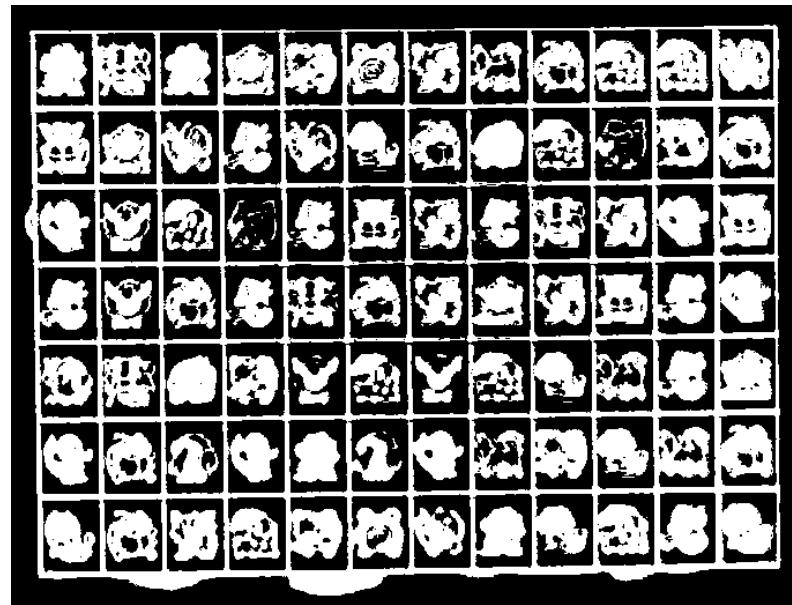


此图确实可以较好的分割，但后期处理相关时效果很差。故而弃用这个方法。

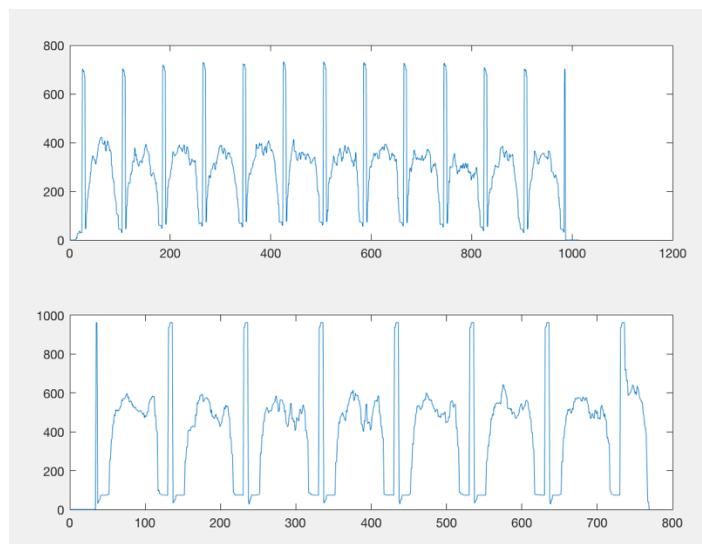
但是写报告的时候忽然意识到后期的相关处理上也做了优化，对相关处理和分割是用了不同的图片处理方式，而边缘提取在图片分割的效果实际上比之后改进的二值化方法好很多，其原因在于边缘提取后小精灵仅仅是轮廓为白色，对于横列均值峰值的影响会小很多。但写报告时时间已经所剩无几，大改程序已经来不及了。

最终程序我采用的是二值化的方法来处理问题。用 Matlab 内置的 im2bw 即可实现，而且适当调节参数可以同时起到去噪的作用。

图片预处理结果：



行列均值分布：



之后要解决的问题就是如何取到完整的小精灵，在二值化之后这个问题得到很好的解决。我们只需要对原有的图片做稍微的扩大保证白边框可以完全进来，然后根据白边灰度最大这一条件锁定边框即可。最终分割结果如下：



小精灵得到了完美的分割。

本题用到的关键代码：

二值化及参数选择：

```
data = double(~im2bw(data,0.85));
```

选取图片宽度和初始位置：

```
[m,n] = size(data);
[~,p_l] = findpeaks(-line,'MinPeakDistance',n/20);
[~,p_c] = findpeaks(-column,'MinPeakDistance',m/11);
ls = p_l(1);
cs = p_c(1);
dl = diff(p_l);
dc = diff(p_c);
dl = dl(find(abs(dl-mean(dl))<8));
dc = dc(find(abs(dc-mean(dc))<8));
wl = mean(dl);
wc = mean(dc);
ln = length(p_l)-1;
cn = length(p_c)-1;
```

图片去边缘：

```
s1 = 10;
```

```

s2 = 25;
[ll1,ll2] = size(data);
g_pic = cell(1,cn*ln);
for i = 1:cn
    for j =1:ln
        tag = (i-1)*12+j;
        subplot(7,12,tag);

        p1 = round(max(1,cs+(i-1)*wc+1-s1));
        p2 = round(min(ll1,cs+i*wc+s1));
        p3 = round(max(1,ls+(j-1)*wl+1-s1));
        p4 = round(min(ll2,ls+j*wl+s1));
        image = data(p1:p2,p3:p4);

        ave1 = sum(image,2);
        ave2 = sum(image,1);
        l1 = length(ave1);
        l2 = length(ave2);

        d1 = ave1(1:s2);
        d2 = ave1(l1-s2+1:l1);
        d3 = ave2(1:s2);
        d4 = ave2(l2-s2+1:l2);

        pos1 = find(d1 == max(d1));
        pos1 = pos1(1);
        pos2 = find(d2 == max(d2));
        pos2 = pos2(1);
        pos3 = find(d3 == max(d3));
        pos3 = pos3(1);
        pos4 = find(d4 == max(d4));
        pos4 = pos4(1);
        imshow(image);
        images = origin_data(p1+pos1+5:p2+pos2-s2-
5,p3+pos3+5:p4+pos4-s2-5);

```

讨论：对于 capture 图像，相当于引入了噪声，最直观的影响就是单纯地从像素域直接分会很难。对于这种图片的处理思路应该是进行图像处理，思路大体有两种，一种是进入变换域用傅里叶分析；另一种是先对原图进行去噪，变换为较干净的图片再处理。在实验过程中，我两种方法都做了尝试，最终还是选择了自己比较擅

长的图片去噪来完成本题。

2.3&2.4

代码见 codes/linkgame/q2_3.m gaotong.m q2_4.m。

3 小题和 4 小题本质上是一个整体，脱离 4 小题单独讨论 3 小题没有说明意义，故而一起书写报告。

此部分主要分为三个部分：高通滤波与处理、匹配滤波和图片显示。

a. 高通滤波

此部分实际上在前一问代码中已经顺势处理好。

此处的技术细节主要集中在两个方面，其一是对滤波对象的选择，其二是滤波后图像的处理。

对于高通滤波对象的选择，可以是对原图片整体做滤波，也可以是对分割之后的小图片做滤波，简单分析我们就可以得到对整体做滤波效果会更好。从我检索到的高通滤波原理和代码中，我发现高通滤波和像素点到图片中心点的距离有关，相对来说总距离范围越大，能够做的操作也越精确。

对于图像的后期处理上我还是出了不少问题，起初的时候是直接对未除直流分量的结果做后续处理，得到的结果并不理想。下图 1 为人工标定后得到的最大不相似结果。而依据这个结果设置大体阈值，找阈值下的最大不相似结果，大概有 30% 的噪声点（即实际上是分类正确的结果混入）。



分析原因后我发现应该是高通之后 jpg 图像全部为正值，所以相关的结果永远非负，客观上会造成区分度下降。于是我对高通之后对结果进行了去直流分量，但效果仍然不佳，久久不能得到原因。之后查看. mat 文件的数据时发现问题在于我之前把高通之后的图片存储为一个小图，在下一问再读取出来，但是写成图片后 Matlab 会自动将负数全部转化为 0。于是我做了一部分改进，有存储图片转化为存储 mat 文件以避免失真。

需要说明的是，高通滤波的函数非原创，参考了百度文库的资料。地址在 readme.txt 中呈现。

b. 匹配滤波

用 Matlab 内置的 xcorr2 函数可以实现匹配滤波，代码如下：

```
for i = 1:s
    im1 = double(g_pic{i});
    x11 = max(max(xcorr2(im1)));
    for j = i+1:s
        im2 = double(g_pic{j});
        x22 = max(max(xcorr2(im2)));
        x12 = max(max(xcorr2(im1,im2)));
```

```

    rate = x12/sqrt(x11*x22);
    pair(i,j) = rate;
end

```

c. 图片显示

我在 2.2 问时直接把分割好的图片存储在了 capture 目录下。
直接按照编号读取显示即可。

代码如下：

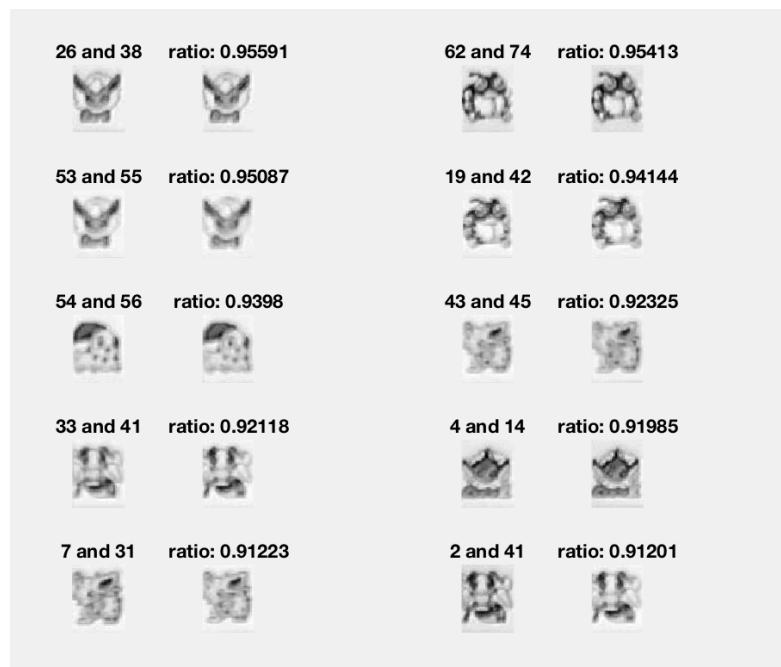
```

subplot(5,6,3*i-2);
mm = max(max(tmp));
[m,n] = find(tmp == mm);
im1 = imread([addr,num2str(m),'.jpg']);
im2 = imread([addr,num2str(n),'.jpg']);
imshow(im1);
title([num2str(m), ' and ', num2str(n)]);
subplot(5,6,3*i-1);
imshow(im2);
title(['ratio: ', num2str(mm)]);
tmp(m,n) = 0;

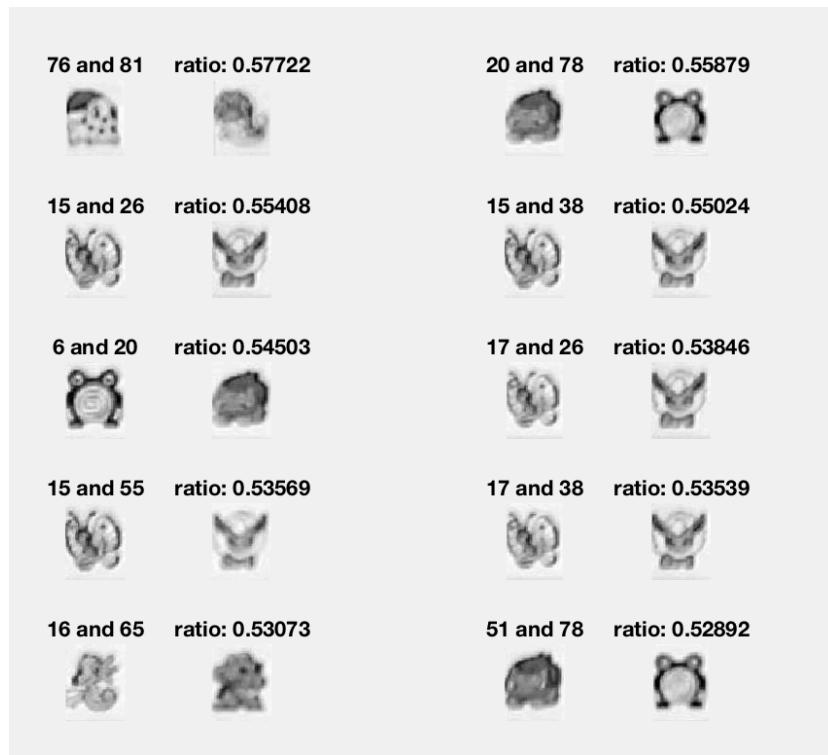
```

最终的试验结果如下：

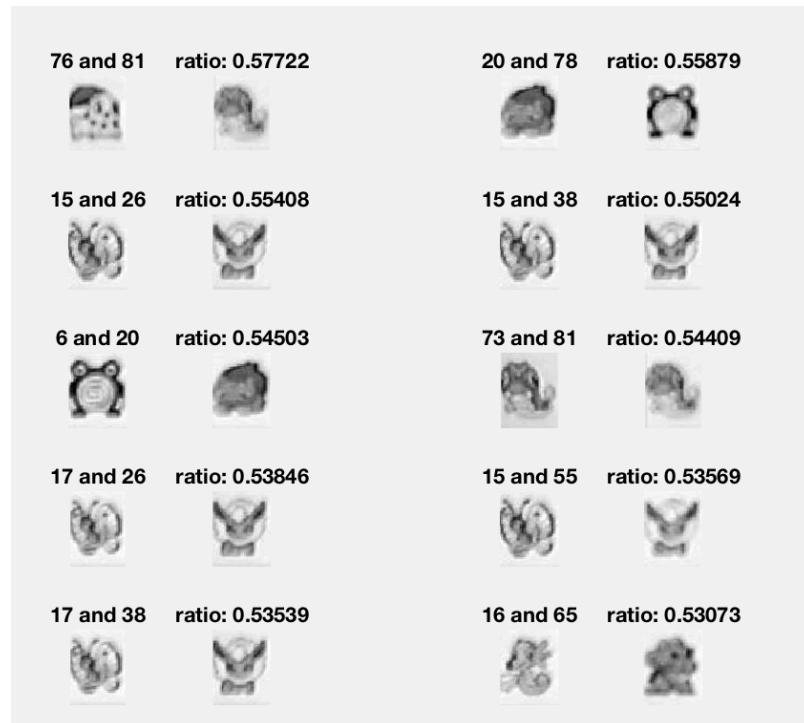
2.3 结果：



2.4 人工标定后的结果:



2.4 阈值标定后的结果:



这里需要说明的是，阈值是对相关性排序后，选择差值最大处作为分界点，分界点之上认为是正确分类，分界点之下认为是错误分类。效果很好，只有一个毛毛虫是分错的。

对于这个结果，与我的主管感受一致，错误分类最大相似度都发生在很像的几个小精灵上，而两个毛毛虫被归入“不同类”从图片上可以看到原因应该是在于光线原因以及后一只毛毛虫被拍糊了，影响了判断。

2.5 建立索引矩阵

代码见 codes/process/q2_5.m classify.m。

参考之前的几问结果，我们要做的就是借助相关性矩阵来完成索引矩阵。

制约本题的两个关键是相关矩阵的准确性和算法的有效性。

基于前几问的优化，我已经尽最大可能保证了相关性矩阵的精确度，此处主要来讨论算法的有效性。我起初采用的算法是类似 k-NN 的一种算法，即是按照小精灵次序进行排序，初始化所有小精灵索引为 0，对于任意小精灵，考虑它之前标号的小精灵，选择其中最相似的一个，如果它们的相似度高于某阈值，则归为一类，否则归为新类。

使用上述算法，我通过不断调整阈值参数，分类后与标准结果进行比对，最好时候能达到 82/84 的正确率。之后无论如何优化都无法进一步全部分对。

于是我决定推翻重写算法。用了一种新的算法：

基于前几问得到的结果，我可以认为 2.4 问所找到的“阈值”之上的小精灵完全是一样的配对。那么，视这些对为有效配对。

对于所有配对进行遍历：

- a. 配对的两个小精灵均为归类，则将它们归为新类。
- b. 配对的两个小精灵有一个归类完成，把另一个划入该类。
- c. 配对的两个小精灵不是同类，则把这两类所有小精灵置为一类。
- d. 配对的两个小精灵已经是同类， continue。

运行这个算法后，准确度有了质的飞跃，直接全部分对。源代码如下(classify.m)：

```
function class = classify(pair,edge)
[m,n] = size(pair);
class = zeros(1,m);
count = 1;
number = 0;
for i=1:m
    for j=1:n
        if pair(i,j)>edge
            number = number +1;
            if(class(i)==class(j))
                if(class(i)>0)
                    continue
                else
                    class(i) = count;
                    class(j) = count;
                    count = count+1;
                end
            else
                if(class(i)*class(j)>0)
                    if(class(i)<class(j))
                        ppos = find(class == class(j));
                        class(ppos) = class(i);
                    end
                end
            end
        end
    end
end
```

```

        else
            ppos = find(class == class(i));
            class(ppos) = class(j);
        end
    else
        if(class(i)>0)
            class(j) = class(i);
        else
            class(i) = class(j);
        end
    end
end
end

```

经过本题我意识到，如果想建立一个全部正确的索引表，两方面事情都不可或缺，一是要有准确的相关矩阵，另一方面是要有强有力的算法。这也是对于任何编程方面的事情所不可或缺的点。

2.6 模拟连连看

源代码见 codes/process/q2_6.m

把得到的索引阵交给 omg，得到 steps 信息，再图片中输出根据步数把图片块涂黑即可模拟出来。

源代码如下：

```

close all;
clear all;
clc;

load q2_5.mat;

name = 'graycapture.jpg';
image_ori = imread(name);

```

```

image_data = image_ori;

start_r = cs;
start_l = ls;
width_r = wc;
width_l = wl;

origin = res_mtx;

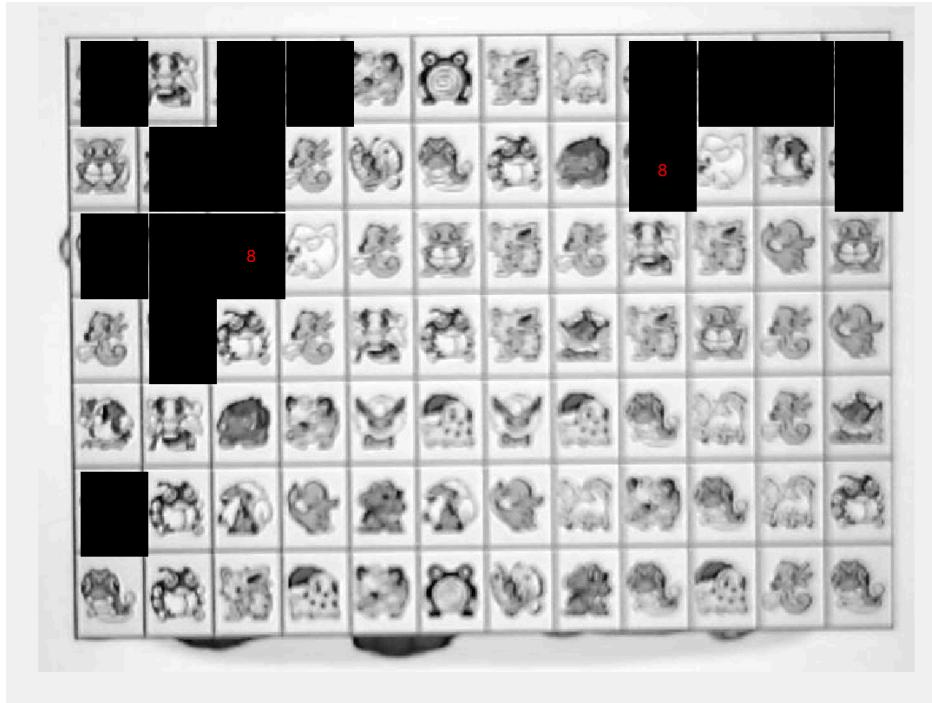
steps = omg(origin);

figure;
imshow(image_data);

number = steps(1);
steps = steps(2:length(steps));
for i=1:number
    pause;
    x1 = steps(1);
    y1 = steps(2);
    x2 = steps(3);
    y2 = steps(4);
    steps = steps(5:length(steps));
    image_data(start_r+(x1-
1)*width_r+1:start_r+x1*width_r,start_l+(y1-
1)*width_l+1:start_l+y1*width_l)=0;
    image_data(start_r+(x2-
1)*width_r+1:start_r+x2*width_r,start_l+(y2-
1)*width_l+1:start_l+y2*width_l)=0;
    imshow(image_data);
    text(start_l+(y1-0.5)*width_l,start_r+(x1-
0.5)*width_r,num2str(i),'horiz','center','color','re
d');
    text(start_l+(y2-0.5)*width_l,start_r+(x2-
0.5)*width_r,num2str(i),'horiz','center','color','re
d');
end
pause(0.3);
imshow(image_data);

```

模拟情景如下：



此程序还有一些小不足，可以发现这个格子并不是完全与图片各自对齐。利用之前的修正，完全可以基本上与图片格子平齐，但是这样的结果必然是周围和格子相连处很难看，放弃。

还有一种模拟方式我认为应该也是可行且比较巧妙的：事先先人工处理，找清楚索引值和 `pics.mat` 标记之间的一一对应关系，然后将拍照的图片转化成 `pics.mat` 中的图片，输出到屏幕上形成一个真的连连看。但是这种操作一方面需要人力成本，另一方面不具有普适性，即如果我们换一副图就需要用新的标记。若想解彻底决此问题，应该还需要再做匹配滤波，我个人认为这样做的意义不大。

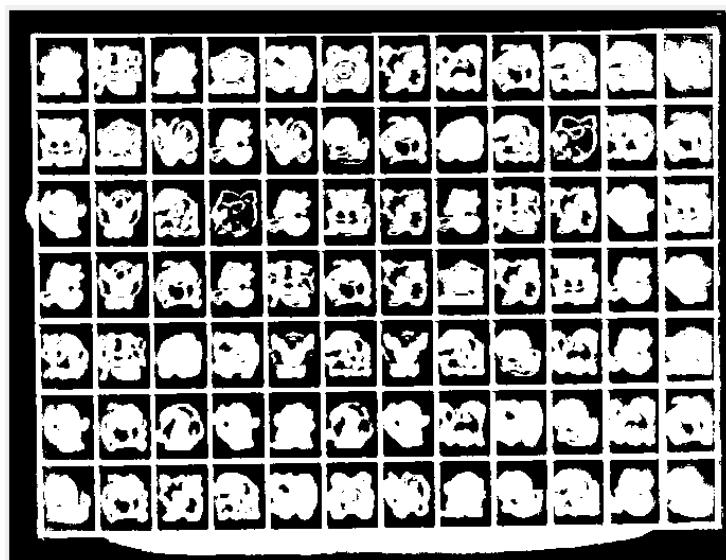
2.7 对彩色图片的处理

这部分的实验总原理上与之前的实验无异。大部分代码是可以复用的，较大的区别是在于 `rgb` 的三维图像处理，下面实验报告中代码相似部分就略去，主要侧重特殊处理的部分。

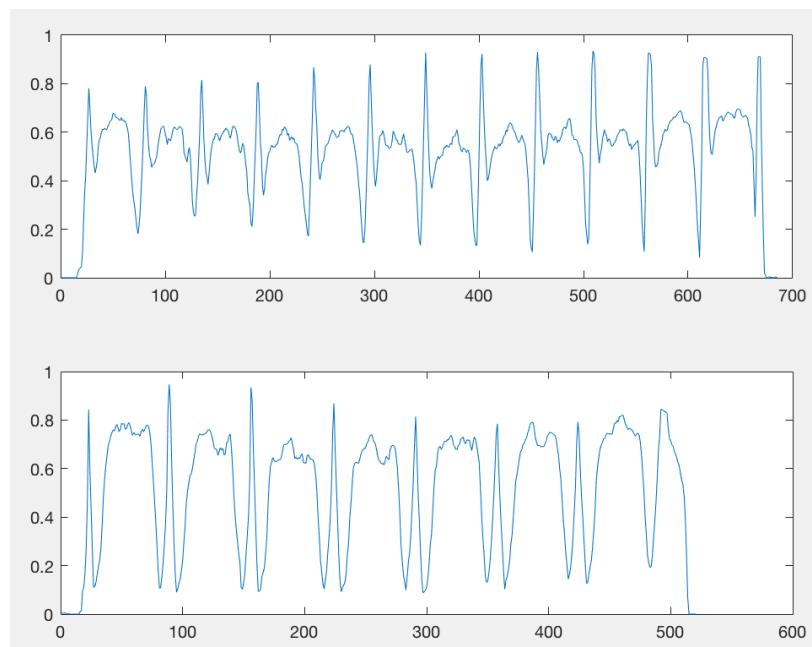
a. 图片分割

代码见 codes/process/q2_7_2.m

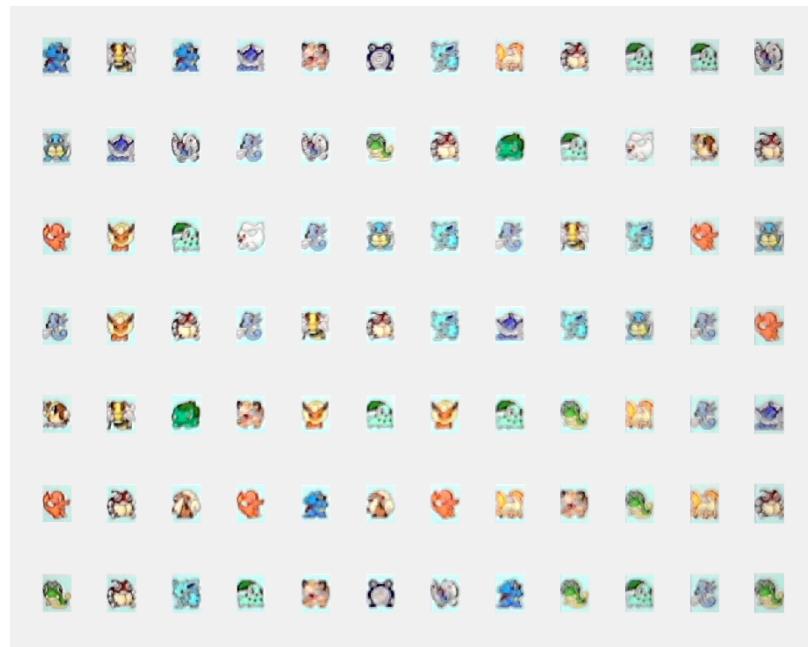
这里的主要操作时对于 rgb 图像，我们转化为二值图的时候采取了对三个维度分别取二值然后再取平均值的算法。这样可以有效避免某一色道的误差对于整体图片准确度的影响。用这种操作可以帮助优化二值化操作。



二值化结果，有效减弱了边缘误差。



均值幅度谱，峰值显著。



最终结果也显示分割较好。

特殊处理代码如下：

```
image_name = 'colorcapture.jpg';
data = imread(image_name);
data_r = data(:,:,1);
data_g = data(:,:,2);
data_b = data(:,:,3);
bw_rate = 0.85;
bwr = double(~im2bw(data_r,bw_rate));
bwg = double(~im2bw(data_g,bw_rate));
bwb = double(~im2bw(data_b,bw_rate));

data_bw = bwr+bwg+bwb;
data_bw = data_bw>1;

for j =1:3
    data_gt(:,:,j) = gaotong(data(:,:,j),30);
    data_gt(:,:,j) = data_gt(:,:,1)-
mean(mean(data_gt(:,:,j)));
end
```

b. 筛选图像相似度

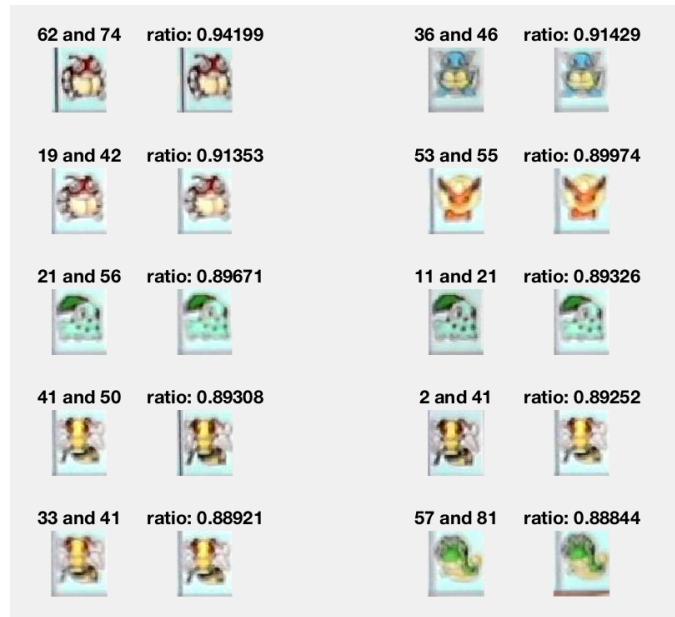
代码见 codes/process/q2_7_3.m q2_7_4.m。

依据上一问的经验，我觉得可以对高通后的图像三个通道分别做出相关矩阵，取平均值作为最终的相关矩阵。

这种思想并不只是单纯的平均主义，而是因为这个连连看图片有明显颜色趋向性，可能在某一个色道上一些图像难以区分，但是另外两个色道必然是可以区分的。而且倘若两个图片分类相同，那么它们三个色道必然相关度都很高，平均值也会很高，而不同类即使一个色道相关很高，也会被其他色道拉下来。

其实也可以选择相关度的最小值，但这种操作容易把一些受到高光影响或者模糊的同种图片分类错误。斟酌后还是选择用平均值这种简单却实用的方法。

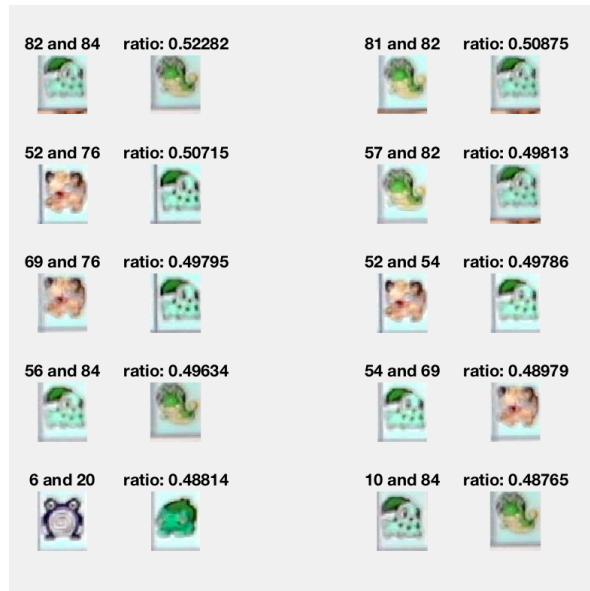
前十个分类正确的：



前十个分类错误的（人工标记）：



前十个分类错误的（阈值判断）：



可见，此时阈值判断的结果和人工标定的结果完全一致。说明经过不同色道的过滤，误判的概率下降。显著提高了有效性。

此外，对于前十个相似图片，它们的共同点大部分颜色都比较相近，对彩色图像处理时，颜色相近相关度会提高。这一方面帮助我们更好的判断，另一方面也需要有更准确的分割保证可以提取到完整的图像否则很容易误判。

c. 索引矩阵建立

代码见 codes/process/q2_7_5.m。

此部分代码与之前一摸一样。运行结果如下：

```
Accuracy Rate: 84/84  
x >>
```

d. 模拟彩色连连看

代码见 codes/process/q2_7_6.m。

这里的唯一不同就是在显示消去时，全部变黑有点难看，故而选择了改变透明度的方法来表示消去。

运行示例：



2.8 ai 外挂的实现

代码见 codes/process/ai.m user_camera.m。

录制的视频见 image/2_8.mp4。

首先来分析题目要求，user_camera 是提供图像源接口，输出屏幕截屏。Ai.m 每次接受一个截图然后返回一个 step 直到所有操作完成。

其中 ai.m 是重点所在，我认为 ai.m 主要可以有两种方法实现，其一是对每次拍到的图像做一个处理，比较现在的矩阵和初始矩阵，判断是否执行顺利并选择下一步。另一种方式是仅仅接受第一张图片，然后生成所有步骤。第一种方法对图像处理要求低但是较为繁琐；第二种方式较为简单但是对图像处理要求高。基于之前的实验结果，我选择第二种方式。

基本的实现就是综合前几题的代码。这里仅仅需要提到的就是几个技术细节：

- a. 设计全局变量 stepss 表示步数序列；number 表示总步数；count 表示当前步数。

```
persistent stepss;
persistent number;
persistent count;
if isempty(stepss)
...
end
```

- b. 对结果矩阵的进一步处理。由于定义方式不一样，经过测试后发现需要对矩阵做转置之后再做一次上下翻转才可以和题目的矩阵吻合。

```

res_mtx = classify(pair,tell);
res_mtx = reshape(res_mtx,ln,cn);
res_mtx = res_mtx';

stepss = omg(flipud(res_mtx));
number = stepss(1);
stepss = stepss(2:length(stepss));
count = 0;

```

c. 输出 step。经过测试发现，不仅仅是矩阵的关系，step 的横纵坐标也是反的，需要手动变换位置。且当步骤完成之后需要 step 所有值变成-1。

```

if(count < number)
begin = 4*count+1;
step = stepss(begin:begin+3);
step(1:2) = [step(2),step(1)];
step(3:4) = [step(4),step(3)];
count = count+1;
else
step = -1;
end

```

至此，ai.m 的所有操作完成。具体见文件 ai.m。

对于 user_camera，根据提示配置所需要的软件发现可能是 mac 双系统的原因总是报错，始终无法作出合适的接口。于是我不得不妥协，采用预先拍好的一张照片作为输入，user_camera 只做读取这张图并返回给 ai 即可。

```

function realcapture = user_camera()
name = 'picture1.jpg';
realcapture = imread(name);
end

```

调整好这些文件之后还遇到了一个问题，当我点击 auto 时鼠标点

击到正确的位置但是却无法正常消除。好像原因在于我不小心把 win 下的一个组件搞坏了。为此，我又写了一个 q2_8.m 来模拟了一遍消除过程，确定整个程序是正确的只是点击时无法正常消除。2.8.mp4 记录了这个过程。

但是需要改进的是，实际上对于图片的运算过程耗时很长，大概实际上达到了 40 秒左右，远远无法达到外挂的要求。分析原因我认为最耗时的部分在计算相关度上，查看我得到的数据，手机传过去的每一张图片都是 1280*960 像素的，大约是原来图片的 4 倍，相关矩阵的复杂度立方上涨，实际上如果采用摄像头图片质量下降后会快很多。

此外，如果要做一个真的外挂，必然是针对特定连连看的。此时，只需要将连连看的所有图片存储起来，对于拍的照片，分割后与存储的数据做判断分类即可，按照我的算法可以由立方复杂度下降到平方复杂度，如果设计好的算法有可能达到线性复杂度，有效减少时间消耗。

三、 思考和体会

本次大作业难度相比前一个有了显著的上升，对于信号与系统的知识比之前涉及的更多更广了。此外，这是一个真正的工程问题，可以以此为基础来设计一个真的连连看外挂。

通过这次大作业，我熟悉了对图像的处理技术，也通过写 Matlab 的脚本和函数进一步熟悉了 Matlab 的操作环境，及其独特的矩阵用法。有一些遗憾的应该是由于信号与系统本身掌握程度不

够理想，导致我对傅立叶变换的运用不够得心应手，对于图像的分割等处理上，我还是选择了我自己比较喜欢的去噪上，而且对整个大作业的处理也有很多是在算法层面完成的，感觉自己还是需要在更多的地方去用 Matlab 做有关信号处理方面的工作来进一步熟悉 Matlab。

还有一点的遗憾就是时间太过紧张导致一些题目没有达到自己的理想效果，且没有真正能够抽出时间写一个可用的连连看外挂。

但是，我觉得通过本次大作业我还是学到了很多东西的，体会到了信号与系统真的很有趣，用两个大作业分别熟悉了对音频、图像信号的处理，为之后进一步深入学习打下了基础。

感谢谷老师、助教的辛勤付出！