MATLAB图像处理——PROJECT1项目报告

一. 基本信息

1. 项目名称：手写汉字识别项目

2. 开发人员：3180105504 徐震

3. 开发时间：2019.07.07->2019.07.12

4. 项目报告生成时间：2019.07.07->2019.07.12

5. 开发平台：MATLAB R2018b

6. 适用平台：MATLAB R2018b Windows 10 1903测试可用

7. 依赖：MATLAB标准库及其图像处理组件

8. 字体：楷体

9. 摘要：

本项目通过**图像分块（包括扇形分块和矩形分块以及直方图）**的方法对手写汉字图像进行特征提取和相似度计算。相似度采用**开发者调整的巴氏距离**计算，并对不同分块方式得到的相似度进行**整合**。

根据相似度确定被测图片属于哪个汉字，输出与该图片最相似的十张图片。

手写汉字识别是一个历史久远的课题。与其相似的问题有如下几个：图片相似度的识别，手写数字（字母）的识别，指纹特征信息的提取以及识别，基于神经网络的图片内容识别（包括上面的手写数字（字母），以及本项目涉及的手写汉字）。其中：

- 图片相似度的识别涉及到的内容相对较容易实现，包括直方图的距离计算，图片分块后的特征矩阵距离计算，以及特征点的提取和匹配等。由于汉字繁多，对汉字进行完全抽象的特征提取在短期内并不现实，本项目的本质其实是图片相似度的对比。只不过由于汉字的特殊性，在辨识方法上有不少与一般图片不同的地方。
- 手写数字或字母也是较为总体的项目。数字只有10个，英文字母也只有26个，很容易根据他们的特征建立相对简单的分析模型，以此为起点进行神经网络和深度学习的研究门槛也相对较低。并且一些手写数字和字母的识别项目中也有分块特征提取的概念，给项目的实现带来了启发。
- 指纹信息特征的提取中，图片相似度的重要性并不很高，重点变成了特征的抓取与对比。这与本项目中的汉字识别有紧密的联系，指纹有分叉和端点，笔画同样有分叉和端点。但由于开发者能力有限，以及不同汉字间极大的差异性（指纹的粗细，间距较为固定，特征点提取相对容易，而中华文化博大精深，字体，书法家繁多，特征点提取需要考虑的因素稍多，尤其是草书会更难实现），还未做相关的特征提取。

二. 项目文档

1. 文件内容介绍

用户函数文件：

- oneImage.m
- categorize.m
- gimmeWhere.m

内部调用函数文件：

- preprocess.m
- rectFeature.m
- circFeature.m
- histFeature.m
- compare.m

- whichIsIt.m

项目报告文件：

- report1.md
- report1.pdf

单图片测试文件：

- 00.gif
- 01.gif
- 10.gif
- 11.gif

数据集以及测试文件夹（若要添加其他测试数据集，文件结构应为：.\backup\数据集\数据集\万\kai\始平公造像_01.gif等，父目录的父目录应该是这张图片的汉字）：

- backup（完整的楷书数据集文件，删除了万文件夹中错误的丈字）
- train（用于测试的数据集文件，是从完整数据集删除四分之一图片得来的，删除脚本如下（递归））

```
function train(directory)

    if ~isa(directory, 'char')
        directory = char(directory);
    end

    fileList = dir(directory);
    fileName = [];

    for i = 3:size(fileList, 1)

        if fileList(i).isdir
            train([directory, '\\', fileList(i).name]);

        else
            fileName = [fileName, string(fileList(i).name)];
        end

    end

    if size(fileName, 2) / 2 == 0
        return;
    end

    for i = 1:round(size(fileName, 2) / 4)
        delete(['.\\', directory, '\\', char(fileName(i))]);

    end

end
```

- test（用于测试的测试图片，保留了上述删除的图片，删除上述train中的图片，由此两个数据集中没有重复图片，删除脚本如下（递归））

```
function test(directory)
```

```
    if ~isa(directory, 'char')
        directory = char(directory);
    end

    fileList = dir(directory);
    fileName = [];

    for i = 3:size(fileList, 1)

        if fileList(i).isdir
            test([directory, '\\', fileList(i).name]);

        else
            fileName = [fileName, string(fileList(i).name)];
        end

    end

    if size(fileName, 2) / 2 == 0
        return;
    end

    for i = round(size(fileName, 2) / 4) + 1:size(fileName, 2)
        delete(['.\\', directory, '\\', char(fileName(i))]);

    end

end
```

- 下面提到的：**开发者的数据集、我的数据集等，指的都是用train做数据库，test做测试集得到的结果，在实际使用过程中，请调用`gimmeWhere('backup')`读取全部已有图片作为识别的数据库，理论上训练集（数据库）的图片越丰富识别的准确度相对越高。**
- 当有更丰富的数据集加入，而测试人员想直接分割数据集为数据库和测试集时，可以将数据集复制成两份，分别调用上述两个脚本。

数据文件：

- where.mat（默认储存的是调用`gimmeWhere('backup')`后的图片特征）
- pol.mat（存储的是256*256的极坐标与直角坐标对应矩阵（程序运行中并不必要））
- num.mat（用于存储每一张图片的查准率（如果按照应该查到的数目来计算就是查全率））

2. `oneImage()`函数（相似书法字检索）

- 本函数接收工作目录下的某张图片的文件名（包含拓展名）。请直接输入文件名例如`00.gif`，不要输入`.\00.gif`。
- 本函数没有返回值，在工作目录下创建以文件名称（无拓展名）为名的目录，将该图片复制进该目录，并从数据库（`backup`）中复制相似度前十的图片到该目录下，数据库中的图片名称以相似度开头。
- 若要更改数据库文件夹，请将数据集文件夹放到工作目录下，并调用一遍`gimmeWhere()`函数对数据集中图片特征提取。（默认图片的父目录的父目录是图片的类别名，例如`.\backup\数据集\数据集\万\kai\始平公造像_01.gif`中，“万”会被程序识别为图片的类别名，该注意事项对`oneImage()`函数没有影响，但由于`categorize()`函数和`oneImage()`函数共用数据集，会对`categorize()`函数有影响）

- 示例：

```
oneImage('00.gif');
oneImage('01.gif');
oneImage('10.gif');
oneImage('11.gif');
% 分别返回：
% 万万万万万无万无王节
% 万万万万万节节无凤万
% 春春春节无论春春者学
% 春春春春春春春春春春


%该示例会在工作目录下建立四个文件夹，分别放入与处理图片相似度最高的十张数据集图片，
并以六位小数标注图片的相似度数值。
```

3. `categorize()`函数（书法字体识别）

- 本函数接收工作目录下的某个文件夹。请直接输入文件夹名，例如`test`，不要输入`.\test`。
- 本函数返回对该文件夹及其子目录下所有图片的识别准确度，请保证图片文件父目录的父目录是图片的类别名，如`.\backup\数据集\数据集\万\kai\始平公造像_01.gif`中，“万”会被程序识别为图片的类别名作为是否识别正确的判断依据。
- 本函数会在屏幕上输出进行识别的图片总数和识别正确的图片总数以及准确率。
- 并且会将识别出的图片复制到工作目录下的`categorized`文件夹下，按照识别到的图片内容分别放入子目录下，例如，若对于`始平公造像_01.gif`的识别结果是：万，则该张图片会被放到`.\categorized\万\`目录下。
- 值得注意的是，若以现有数据集的体量（30个不同的汉字），随机投放的正确率应为1/30，即3%左右。
- 示例1：

```
gimmeWhere('backup');%在我的机器上大约需要15秒
% no output

categorize('backup');%在我的机器上大约需要19.5秒
% output:
%
% 357 image(s) processed.
% 357 image(s) correctly recognized.
% Accuracy: 100.0000%
%
%
%该示例会输出百分百的准确度，原因在于backup中的图片程序都已经见过，一定可以准确定位
是哪个字，并没有实际意义，但可演示categorize()函数的使用方法。

    % categorize运行结束后调用本句load num.mat;sum(num)/length(num)可以获
得查全率或查准率（进行了下两行的操作后就是查全率，不进行得到的就是查准率）
    % 若想通过num文件中的内容获取查全率，请注释掉categorize下面第58行代码，并将
其上面的那一小块的注释去掉。然后在whichIsIt中将maxnum改成一个稍大的数，例如20
    % 原理就是通过被去掉的那部分代码获得数据集中汉字的数量，暂时记为n，然后统计查到
的汉字数量，再除以n
```

- 示例2：

```
gimmeWhere('train');%在我的机器上大约需要11秒
% no output

categorize('test');%在我的机器上大约需要4.5秒
% output:
%
% 93 image(s) processed.
% 83 image(s) correctly recognized.
% Accuracy: 89.2473%

% 删除一张明显不是万字的丈字之后，准确率达到了90.2174%
% 再次调教后准确率达到了91.3043%

    % categorize运行结束后调用本句load num.mat;sum(num)/length(num)可以获得查全率或查准率（进行了下两行的操作后就是查全率，不进行得到的就是查准率）
    % 若想通过num文件中的内容获取查全率，请注释掉categorize下面第58行代码，并将其上面的那一小块的注释去掉。然后在whichIsIt中将maxnum改成一个稍大的数，例如20
    % 原理就是通过被去掉的那部分代码获得数据集中汉字的数量，暂时记为n，然后统计查到的汉字数量，再除以n
```

三. 实现过程析构及实验结果分析

1. 预处理

预处理设计的内容实现在函数`preprocess()`里。该函数也可在命令行中直接调用，对需要提取特征的图片进行预处理。输入应为矩阵（真彩色或灰度甚至二值图像都可，直接输入序列图像的序列矩阵部分也可行，程序会采用分析整合的方式得到图片的正常二值化结果）。

图片的预处理分为二值化，缩放图像，重心查找与平移图像三个部分。

对于汉字图像相似度的识别，二值化是很重要的一步。因为手写汉字的特征信息，不存在于颜色明暗饱和，不存在于灰度的高低差别，一张绿色图片和一张红色图片，从相似度上讲，或许根本没有可比性，但他们表示的却可能的确是同一个汉字，一张灰度上下限在0-50（最亮的点灰度值为50，最暗的点灰度值为0）的图像和一张灰度范围200-255的图片，可能根本无法做出任何相似度的比较，一张几乎都是黑色的，一张几乎都是白色的，可能也根本无相似度可言。但或许他们表示的的确就是同一个汉字。手写汉字的所有特征，仅仅需要逻辑值01就可以表达的很清楚，用逻辑值进行运算在一定程度上还可以大大减小运行时间，何乐而不为？

缩放图像操作的实质，是为后续的图片处理提供方便和提高速度，减小不必要的冗余计算。

试想，一张图片，只有左上角有一个汉字，而另一张图片只有右下角有汉字，两张图片相似吗，并不。但他们表示的是同一个汉字吗？还真的有可能。在本程序拥有的数据集中，通过重心识别和中心化的操作，可以让汉字识别准确度从79%上升为89%。

1.1 二值化

采用嵌套函数的方式，外函数对输入类型进行判断（3-D矩阵（真彩色）、2-D矩阵（灰度）、2-D矩阵（无大于一的值）（二值））。内函数为MATLAB内置`imbinarize()`函数，对于真彩色矩阵会先调用`rgb2gray`再对灰度图像进行二值化，以提高二值化的效果。

1.2 缩放

直接调用MATLAB内置的`imresize()`函数，当然若尺寸已经满足条件，则不进行缩放。本项目中采取256*256的处理大小。

- 注意此处若图片放大过多，则会增加无意义的运算量，大大降低程序效率，而缩小过多则会损失太多精度，导致提取的特征辨识度太低。256是项目编写者测试后认为较合理的一个大小。
- 其次，256也可方便后续的矩形拆分特征提取（作为8的倍数，可保证图片矩阵的下标都是整数，一定程度上提高运算速度，至于为何是8的倍数，将在矩形拆分特征处详细叙述）。
- 若将来要对图片进行Haar小波编码处理，256是个很方便的数值（是2的幂次）。
- **注意个别图片进行`imbinarize`之后会得到与其他图片相反的前景背景色分布，这样会导致后续的重心查找以及特征提取部分失效。**

1.3 重心查找

遍历二值图像矩阵，将黑色部分的位置值与(128, 128)的相对值求和并平均，得到重心的位置。

- 虽然(128, 128)并非图片的真正重心，但这部分细节引起的偏差几乎可以忽略不记。
- 实际操作时，并不遍历整个矩阵，而是每4个像素进行一次统计，以在尽量保证精度的前提下提高运算速度。
- 由于计算过程中需要某一特定点的坐标，故该循环难以矩阵化，所以上面一条并不会降低算法的复杂度，仅仅会降低运算的系数，但由于图像大小是固定的，所以带来的效果也不容忽视。
- 同时，由于需要特定点的横纵坐标分开读取，`find()`函数在该情况下无法使用（find函数只能返回按列排序的位置信息）。
- **补充：认真思考后发现`find()`函数可以使用。通过对find函数返回的信息进行取模和求余的运算，可以得到像素的坐标信息。经计时，这一条改变对处理时间的影响不大（总耗时稍有增加），但会提高重心查找精度（每个像素都会进行统计）。**

  重心查找函数替换前后分别为：

```
function [rcm, ccm] = cm(img)
        %         rcm = 0;
        %         ccm = 0;
        %
        %         for i = 1:4:r
        %
        %             for j = 1:4:c
        %
        %                 if ~img(i, j)
        %                     rcm = rcm + i;
        %                     ccm = ccm + j;
        %                     cnt = cnt + 1;
        %                 end
        %
        %             end
        %
        %         end
        %
        %         rcm = round(rcm / cnt);
        %         ccm = round(ccm / cnt);
```

```
        a = find(img == 0);
        rarr = mod(a, wanted(1));% wanted = [256,256] defined in the
        carr = ceil(a/wanted(2));% wanted = [256,256] outer function.
        rcm = round(sum(sum(rarr)) / size(rarr, 1));
        ccm = round(sum(sum(carr)) / size(carr, 1));
    end
```

但在本测试集下，准确度并没有明显的提高，而性能却稍有差池（零点几秒的下降），于是在程序代码中还是使用了原来的方法。

- 关于在MATLAB中使用`find()`函数代替循环运算，但经过计时测试和研究，在图片较小时实际效率与手动遍历矩阵几乎没有差异。但图片较大时会产生明显差异。测试脚本如下：

```
img = imread('rice.png');
[r, c] = size(img);
tic;
total = 0;

for i = 1:4:r

    for j = 1:4:c

        if img(i, j) == 127
            total = total + 1;
        end

    end

end

toc;

tic;
total = size(find(img == 127), 1)/16;
toc;
% 此时的时间消耗差不多


img = imread('concordaerial.png');
[r, c] = size(img);
tic;
total = 0;

for i = 1:4:r

    for j = 1:4:c

        if img(i, j) == 127
            total = total + 1;
        end

    end

end

toc;
```

```
tic;
total = size(find(img == 127), 1)/16;
toc;
% 此时的时间消耗有极大差异
% 两种算法的复杂度不同
% 但在像素数量不多的时候却不一定哪一种方式速度最快
```

1.4 图片平移

首先，用`zeros()`函数并对矩阵加一生成空白画布。根据重心位置和图片几何中心位置的差异的符号，分为四种情况，分别进行矩阵值的整体替换。

- 在这里，矩阵值的整体替换过程也可以用循环实现。但实际上将循环矩阵化之后运行效率会提高很多。
- 对边界的处理稍显棘手，应注意调试防止下标越界（不同于C语言或C++，MATLAB对下标越界有报错，给调试带来了一定方便）。

2. 特征提取

一张BW图会有什么特征呢？边缘、形状、骨架、区域、直线、曲线、笔画……等等等等，数不胜数。本项目中仅仅实现对矩形划分、扇形划分下的特征特征分析，并且分析在手写汉字识别中，直方图的作用。

本质上，矩形特征或者扇形特征是对汉字笔画特征的一种抽象，矩形特征可以看作图像在直角坐标系下的缩放变换，小到一个像素（其实将一张BW图片缩小到一个像素得到的就是其直方图），大到每一个像素的原始信息都要进行比对，都是对图像的某一部分信息做平均的过程（缩成一个像素是对整张图做平均，根本不进行缩放是对某个像素自身进行平均）。扇形特征可以看作在极坐标下对图片做缩放（对某一区块的信息做平均）的变换。而汉字与汉字最明显不同的地方在于他们的笔画。一个像素显然不能包含一个256*256大小汉字的一个笔画信息，而整张256*256图片却包含了太多的笔画信息。开发者的理论分析和实验测试都表明，当一个划分能够表达出某个汉字的某一块笔画信息时，识别的效果最优（在直角坐标系中，我的数据集得到的划分是8*8，在极坐标系中是4*20）。显然，划分为1*1的直方图并不会反应太多的汉字信息，甚至还会导致程序误判，得到比随机投放还低的认字率。

在这里，值得注意的一点是，对于开发者的数据集，直角坐标下的划分得到的最高识别准确率是79%，而极坐标可以达到89%，究其原因是汉字中的笔画在极坐标的划分下可以得到更好的表达（扇形特征对不同汉字的辨识度更高）（例如撇、捺、点、钩等笔画在不同汉字里差异更大，更容易得到辨识、但横、竖这种在汉字里位置比较统一的笔画辨识度相对较低。），并且，以汉字重心为重心的极坐标划分（扇形划分），更符合汉字的造字特征，例如“来”字，明显就有中心辐射的特点。这也是为什么在整合两种方法得到的相似度时，开发者采用了矩形：扇形=1：3的权重。

2.1 矩形拆分特征（直角坐标）

具体实现在`rectFeature()`函数中。

函数读入一张BW图片矩阵，将图片分割成8*8的64个区域，分别计算区域中的白色点（也就是值为一的点）的数量，用两个嵌套的sum函数替代循环遍历像素以提高运行效率。对于是否进行归一化处理，实际上对最终判断结果没有任何影响，所以为了省去不必要的计算，在实际运行的程序中去除归一化的部分。经过人工训练调试，发现8*8的分块方式得到的结果最为准确。

2.2 扇形拆分特征（极坐标）

具体实现在`circFeature()`函数中。

由于实现极坐标需要进行坐标变换，这里有两种思路。

1. 首先构建x、y坐标矩阵（位置上的值对于该位置的x、y坐标），通过矩阵坐标变换运算转化为极坐标的坐标矩阵，然后利用`find()`函数得到极坐标矩阵的值在事先定义的r, theta分块中的位置，统计每个分块中白色像素的个数，特征矩阵。对于是否进行归一化处理，实际上对最终判断结果没有任何影响，所以为了省去不必要的计算，在实际运行的程序中去除归一化的部分。
2. 对BW图片矩阵的每个像素进行遍历，若为白色，则对该像素的坐标做极坐标变换，判断其所属的r, theta扇形分块，该分块的特征值加一。

- 经过人工训练调试，发现对r进行4分块，对角度进行20分块时准确度最高。

2.3 直方图

具体实现在`histFeature()`中，

直接统计BW图像中的黑白像素个数即可。

- 实际使用过程中，直方图对手写汉字的识别几乎没有任何作用，最终获得的准确度甚至低于随机分类的准确度。
- 于是在实际计算中，准确率最高情况是直方图特征相似度的系数为零的时候，不如直接不进行直方图的计算以节省运算时间。

3. 数据库变量构建

具体实现于`gimmeWhere()`函数中。之所以起如此奇怪的函数名，是因为该函数会储存一个名为where的结构体变量，开发的早期，where是用于储存每张图片的位置信息的一个结构体，后来由于结构体具有极大的拓展性，发展成数据集图片特征信息的储存变量。

- 每次调用`gimmeWhere()`都会在当前目录下存储一个名为`"where.mat"`的文件，将数据集中图片的位置，类别（属于哪个汉字），矩形特征向量，扇形特征向量存储到一起。
- 调用`gimmeWhere()`函数时，请确保仅仅输入文件夹名，例如`train`，不要输入`.\train`。

值得注意的是，后续进行相似度判断时，不用重新对图片进行预处理和特征提取（每次调用`gimmeWhere()`都需要大量的时间，普遍以秒为单位，在我的机器上提取357张图片的特征耗时为15秒左右），若每张图片判断时都需要对全部数据集文件进行处理，假设判断357张图片，则仅仅是提取特征值的数据就需要一个半小时，而不进行处理直接调用where结构体变量总的判断只需要不到二十秒，调用变量的时间（对应上一种方法中的提取特征值）几乎可以忽略。

3.1 遍历（文件夹操作、递归）

数据集图片分布在不同的文件夹下，且其父文件夹的父文件夹名代表了该图片所属的汉字，经过试验发现通过递归进行文件夹处理较为方便，自动化程度高。

1. 使用MATLAB内置`dir()`函数返回每个文件夹下的文件信息（返回一个结构，包括文件名，文件位置，是否为文件夹，大小，时间等），每次递归时传入当前进入的文件夹信息，两者结合起来就得到了当前目录的所有信息。
2. 首先调整循环中的初始化值来忽略`dir()`函数得到的`'.'`、`'..'`两个目录，然后对返回的结构体按数字顺序进行检测，若为文件夹，则对该文件夹进行递归，回到1.步骤，若不是文件夹，则将文件名记录到一个数组中。
3. 对文件名数组进行遍历，处理每一张图片，将其信息储存到where结构变量中（图片的位置，类别（属于哪个汉字），矩形特征向量，扇形特征向量）。

- 由于MATLAB中嵌套函数的特性：变量可以共用，where 和 index可以视作`gimmeWhere()`函数下的全局变量，由所有的递归函数公用。

- 在对路径的处理中用到了几个小技巧：
  - 由于可通过字符数组外嵌套中括号的方式直接构建字符数组，程序会将用户输入的其他类型变量转换为char。

```
if ~isa(directory, 'char')
    directory = char(directory);
end
```

  - 在程序内部，目录分隔符使用两个反斜杠"\\"，于是定位汉字时需要一定的查找：

```
tempdir = fliplr(directory);% 将目录内容反转以进行倒序查找
n2 = find(tempdir == '\', 4);
n3 = n2(3);% 找到第三个反斜杠
n2 = n2(2);% 找到第二个反斜杠
character = tempdir(n2 + 1:n3 - 1);% 提取出的character就是图片文件父目录的父目录
character = fliplr(character);% 在本数据集下该语句并不必要
```

3.2 信息整合与储存

- 每次调用`gimmewhere()`都会在当前目录下存储一个名为`"where.mat"`的文件，将数据集中图片的位置，类别（属于哪个汉字），矩形特征向量，扇形特征向量存储到一起。

4. 单张图片特征比较

即为`oneImage()`函数的实现。

实际上，`oneImage()`函数是`whichIsIt()`函数的一个封装。真正处理一张图片的函数是`whichIsIt()`，只不过通过`oneImage()`调用时会开启其复制相似度前十文件的功能，而后面将要提到的`categorize()`函数在调用`whichIsIt()`时不会开启该功能。

这样设计的目的是为了简化对单张图片处理函数的调用，不需要用户手动开启复制功能，只需要在传入的参数中讲明要处理的图片名称即可。

上一节提到的特征提取会为每一张图片提取出特征向量，如何计算这些向量之间的“**距离**”，也是影响认字准确度的一大原因。

4.1 向量相似度的比较方法

具体实现于`compare()`函数中。

- 对于两种不同（如果包括直方图，是三种）的特征向量，用相同的模式进行相似度计算。
- 若直接采用直方图计算中的巴士距离，无论是使用哪种特征向量，最终准确度都较低，在50%上下，经检查发现该计算方法没有考虑到汉字笔画的粗细，例如，对于同一块矩形区域，如果某个字的笔画极粗，得到的特征向量里这一区域数值会很大，对于同一区域中不是同一汉字的一个粗笔画，也会得到很大的数值，两者计算巴氏距离会对相似度带来很大的贡献。反观若两个相同汉字的笔画都很细，他们的巴氏距离反而不一定很高。于是开发者在原有基础上将巴氏距离更改为：

$$\rho(p,q)=\sum_{i=1}^{n}\frac{\sqrt{p_i\cdot q_i}}{\sum_i^n p_i+\sum_i^n q_i}$$

（原公式为：）

$$\rho(p,q)=\sum_{i=1}^{n}\sqrt{p_i\cdot q_i}$$

准确率有了极大提高，达到90%（89.2473%）左右。

4.2 总体相似度的获得（三种特征相似度的整合）

- 正如前面第三节所叙述的，对于手写汉字识别系统，直方图的作用几乎为零，在考虑直方图的相似度之后，识别率反而有所降低，并且只考虑直方图获得的识别率比完全随机的识别率还低，所以程序最终摒弃了直方图的特征相似度。
- 经过多次实际检测，发现对于楷书的汉字识别，矩形特征的检测效果比扇形信号的检测效果稍差。
- 最终经过人工训练调试，发现当矩形特征检测和扇形特征检测的贡献比例约为1：3时，在开发人员的数据集（也就是现在存储在当前目录下的train 和 test）中检测效果最好。
- 由于`compare()`函数的设置，单独特征的相似度范围都在0.0-0.5之间，于是只要保证不同特征相似度整合后的系数和为2即可。即为：设矩形相似度为rectSimilarity，扇形相似度为circSimilarity：

$$similarity=a*rectSimilarity+b*circSimilarity$$
$$a+b\equiv 1$$

在本数据集和测试集下，$a=0.5$，$b=1.5$效果较好。能够达到最高准确度（89.2473%）。

4.3 更高效的前k个最大值提取方法（O(n)）

1. 要取出三百五十多张图片中相似度最高的十张图片，最简单的思路或许是将图片按照相似度进行排序（调用sort等方法）然后选取出相似度前十的图片，这种方法的时间复杂度为$O(n\log n)$。
2. 更为高效的方法是通过调用max函数等方法取得最大值。取得之后将最大值存下，然后重复进行操作一共10次，时间复杂度为$O(n)$，但若考虑系数，消耗的时间为10n（遍历矩阵的时间看作1）。
3. 另一种方式（本程序采取的方式（是开发者原创的，因此没有经过严格的数学计算或其他更深层的优化）），只对矩阵遍历一遍，但遍历过程中会将矩阵数值与当前前十大的数值进行比较，从第十个开始，若当前前十中某一个比矩阵元素数值大（例如第五个（第六个比矩阵元素数值小）），则将该矩阵元素数值记录为第六个。平均对每个像素需要额外进行5次比较（估算），若遍历矩阵的时间也可以看作1，则其消耗的时间应为5n。实现部分粘贴如下：（对于下标越界问题需要仔细调试）

```
for i = 1:size({where.rect}, 2)
    if i>10
        if i==11
            [~,maxa] = sort([similarity(1:10)]);%maxa为记录最大值位置信息
的数组。
            maxa = maxa(end:-1:1);
        end
        for j = 10:-1:1
            if similarity(j)>similarity(i)
                if j<10
                    maxa(j+2:10) = maxa(j+1:9);
                    maxa(j+1) = i;
                    j=0;
                end
                break;
            end
```

```
            end
            if j==1
                j=0;
                maxa(j+2:10) = maxa(j+1:9);
                maxa(j+1) = i;
            end
        end
    end
```

4.4 复制以及重命名图片（文件操作）

- 在MATLAB中，复制文件有一定的限制：目标文件夹必须存在，不存在无法自动新建，反而会报错说明没有写入权限。所以在复制之前都会调用`mkdir()`命令来创建目标文件夹，但要注意处理大批量数据时不要重复创建文件夹，否则MATLAB会给出警告，仅仅是在屏幕上打印警告就会严重降低MATLAB的运行速度。
- 在提取文件夹名的过程中，最初采取的方式是直接利用下标编号精确定位，这种方式固然简单易写，可一旦输入文件的名称过长就无从下手。后来考虑过不少MATLAB的内置函数，例如`fileparts()`等，但由于输入的仅仅是文件名不包括其他路径，`fileparts()`无法返回有效信息。最终采取的方式是对目录分隔符（或者后缀名分割符）进行查找，动态定位要提取的信息的下标。

```
name = img(1:find(img=='.',1)-1);% img = '00.gif'
% 得到name = '00'
```

5. 多张图片特征比较

具体实现在`categorize()`函数中。`categorize()`的主函数部分也仅仅为一个封装。

利用嵌套函数封装的作用在与，可以在不建立全局变量的情况下让递归函数调用一个在递归过程中不改变的变量，在上面的`gimmeWhere()`函数中也用到了相应的概念。当然，这种效果也可以通过在递归函数之间传递参数实现。为了逻辑清晰易懂，开发者在具体实现上选用了前者。

5.1 遍历（文件夹操作、递归）

遍历过程类似于`gimmeWhere()`函数所做的事情。打开两个函数的.m文件将会很容易发现，二者利用递归遍历文件夹的过程是完全一样的。（这种形式的递归遍历操作开发者也用在了肖老师要的批处理文件中。）

相较于利用`uigerdir()`等函数打开UI界面，开发者认为这种方式更为自动化更为灵活，省去了不必要的操作。

5.2 总体类别的获得（十个最相似图片信息整合）

正如在`oneImage()`函数中提到的，`whichIsIt()`函数会返回与输入图片相似度最高的10张图片对应的汉字（这个数字10是一个灵活的参数，可以很容易的改变为其他数字，例如5，或者50（只需要到`whichIsIt()`中更改第一个参数`maxnum`的值，`categorize()`函数会自动适应）），于是`categorize()`函数会对这十张图片的信息进行整合：

1. 首先遍历十张图片，提取其类别名称，并建立类别名与可能性的对应关系（具体形式为创建一个`cate`结构变量：category的缩写，字段`word`用于储存汉字名，字段`count`用于储存可能性）。

2. `whichIsIt()`函数返回的10个汉字是按照图片相似度排序的，自然，排名越靠前的字可能性越高
    - 一开始，开发者采用了10-index的统计方式（index是字的排名），效果并不令人满意，识别准确度甚至比不上直接选取可能性最大的图片对应的汉字。
    - 再后来采用的是1/index^2的统计方式，识别准确度较为满意。

四. 展望

1. 仍需完善的地方

1. 在`preprocess`过程中，受今日课程（laplacian边缘检测）启发，考虑到可以在处理BW图像时仅仅保留边缘，在手写汉字检测中，理论上这样做会提高汉字的识别准确度，因为这样几乎摒弃了笔画大小对汉字识别带来的影响，但不一定会提高图像的识别准确度，即最终结果里，相似度最高的可能的确是同一个字，但两者的图片在直观感受上或许会相差不少（例如两个字形状类似，但笔画粗细差异较大）。
    - **更新：在2019.07.10课后，采用了canny边缘检测的方法对preprocess之后的图片进行边缘化处理，但最终准确度却和原来相差不大（88%，89%），而边缘检测可能会带来一定的性能影响，所以在本项目中暂时将实现边缘检测功能的代码注释掉。**
2. 在`preprocess`过程中，开发者认识到，所给出的数据集图片或需要进行查找图片中汉字不一定在正中间（这一问题被`preprocess`中的重心查找部分解决），其汉字大小也不一定相同，但由于默认数据集样本中，汉字大小基本都在正常范围内，开发者就没有在与处理中增加识别汉字`bounding box`的部分，随着后期课程的进行，我认识到使用MATLAB内置的函数（例如`bwboundaries`或者`regionprops`）也可以进行汉字的`bounding box`的识别，这是开发过程中瑕疵。
3. 前期查阅文献过程中，了解过特征点提取的方法（对指纹二值图像的检测）：利用分叉点和端点的特征以及指纹的走向确定这些特征点。如果在现有的三种特征提取方式上增加该方式，或许也会带来像1.中提到的类似的影响。（需要注意的是，如果仅仅是对特征点进行**统计**，或许对图像相似度的贡献不大，但若能通过更复杂的方式对特征点进行**匹配**，图像相似度的识别能力应该也会有不小的提高，具体会在项目展望中叙述）
4. 在`whichIsIt()`函数的返回值传递过程中，会丢失图像的相似度具体信息，仅仅保留相似度的排序，如果考虑保留，或许能在上述：**总体类别的获得（十个最相似图片信息整合）**中得到更好的结果。
5. 由于开发者学识有限，精力有限，且对MATLAB编程平台仅仅是初步了解，没有为项目做UI，而是仅仅提供了命令行程序。开发者认为，作为一个完全开源的项目，以命令行程序的方式呈现反而更方便理解项目真正在做什么。不过如今的世道……颜值的确至关重要。
6. 经过一遍`preflight-review`之后，发现有个别的图片预处理之后会得到与其他图片完全相反的前景与背景色分布（其他图片：背景：白色，汉字：黑色，个别图片：背景：黑色，汉字：白色）。后来在预处理函数`preprocess`中增加了背景检查功能。有两种思路：
    1. 提取图片中某个一定是背景的点，将其颜色作为背景色。
    2. 计算黑白点的个数，数目多的作为背景色。

    在已有数据集中，两种方法都可行（第一种方法性能影响小，项目中采用第一种方法），但当输入的情况未知时，两种方法都有瑕疵。
    1. 若我们认为一定是背景的点恰巧是一个噪声点，被赋予了与其他值不同的颜色，则方法1失效。
    2. 若汉字笔画极粗，在画面中很大，则很有可能出现前景色像素数高于背景色的情况，则方法2失效。

2. 展望

本项目是手写汉字识别项目，虽然根据汉字的特点做了一定的调整，但本质上仍然是图像相似度识别（如果数据集中的图片不是汉字而是其他图片，本项目也能检测其相似度）。由于开发者学识有限，精力有限，且对MATLAB编程平台仅仅是初步了解，对更为针对化的手写汉字识别方式涉猎过少，在项目中没有涉及，包括但不限于：

- 特征点匹配（类似于SIFT算法及其变种）
- 针对汉字笔画特征的（弹性）区块划分（包括弹性扇形区块和矩形区块的划分）
- 对图片进行一定的特征提取，但摒弃特征相似度比较的概念，通过机器学习的方式，直接做特征->汉字的对应
- 通过机器学习的方式摒弃特征提取及特征相似度比较的概念，直接做图像->汉字的对应

这些形式的内容，或许是未来该项目的发展方向。

[参考文献]

钟国华, 金连文. 手写体汉字扇形弹性网格特征提取的新方法[J]. 计算机工程, 2002, 28(11):61-62.

宁博. 手写体汉字识别实验平台及笔划网格特征提取方法的研究[D]. 河北工业大学.

俞凯, 吴江琴, 庄越挺. 基于骨架相似性的书法字检索[J]. 计算机辅助设计与图形学学报, 2009, 21(6):746-751.

李晨丹, 徐进. 指纹图像预处理和特征提取算法的Matlab实现[J]. 计算机工程与科学, 2009, 31(7):61-64.

刘思慧, 江维. 基于MATLAB手写体数字识别程序设计[J]. 电子世界, 2019, 561(03):141-142.

Lowe D G . Object Recognition from Local Scale-Invariant Features[C]// iccv. IEEE Computer Society, 1999.

Lowe D G . Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2):91-110.

毛群, 王少飞. 基于Matlab的神经网络数字识别系统实现[J]. 中国西部科技, 2010, 09(19):22-24.

郭晶莹, 吴晴, 商庆瑞. 基于Matlab实现的指纹图像细节特征提取[J]. 计算机仿真, 2007, 24(1):182-185.

胡海威, 公绪成, 孙立民. 基于二值图像的指纹特征点快速提取算法[J]. 广西师范大学学报：自然科学版, 2009(3):162-165.