# 程序设计思维 大作业报告

1700012803 徐思睿

# 问题介绍

在图像中提取出文字一直是我非常感兴趣的问题，OCR（光学字符识别）是一个非常重要的应用工具，我们日常生活中经常需要扫描文件，有些时候就需要将纸质文件输入到电脑上，如果能直接通过扫描就能让文字尽数上传到电脑，那就会非常的方便。

这次大作业我设计了一个完整的pipeline，实现了一个从图像中检测出字母，识别出字母，并且交互可视化的全部过程。（仅用英文，中文字符过于复杂）

之前只会熟练地使用了matlab，趁着这个机会，学习了opencv、numpy、sklearn等相关工具，可以说是受益匪浅。作为单人完成的大作业，可以说是费了好大的劲才完成这么多的代码，也是对自己的一个锻炼。

我认为这次任务的难点在于：

1. 涉及到数据处理、图像处理、模式识别、可视化多个方面，需要学习大量的知识。
2. 任务可以分成字母检测、字母识别两个部分，其实每一个部分都非常难，都可以作为一个大作业去完成。现在需要一个完整的pipeline去整合两个过程。

我认为这次大作业我的主要贡献点在于：

1. 实现了在图像中检测、识别英文字母，并且实现可视化的完整过程。
2. 在图像分割检测阶段尝试了opencv多种算法，包括霍夫变换、最大稳定极值区域、连通域算法，并且自己整合设计了一套算法，加入了非极大抑制过程，实现了精准、干净的分割。
3. 在文字识别阶段，没有像网上或者说很多传统的做法下载一个数据集训练，而是自己生成了一个字母的数据集，没有利用到任何的外界资源。
4. 实现可自己生成带文字的图片，同样也在网上截了带文字的图进行测试。

具体各个文件的操作说明请务必查看文件夹内的README文件

超过95%的代码为自己实现！

# Pipeline介绍

## 文字检测部分

文字检测属于目标检测问题的一个重要部分，但却与其它目标检测问题大相径庭。文字图像的纹理通常长宽比较大，文字本身大小很小，且文字的常以序列排列好出现。

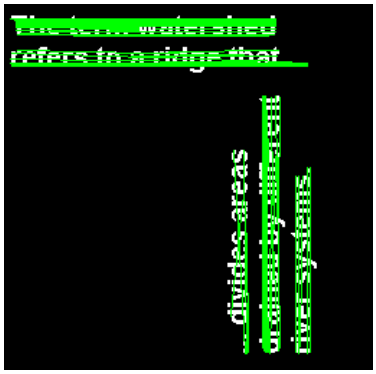
我在给定的数据上分别实现了三种比较常见的文字检测（实际上在这种情况下更像单纯的纹理检测），并且基于其中的两种方法做了更进一步的改进。

- **霍夫变换**

用来检测出可以用公式描述的形状，比如直线，圆，椭圆等等。其原理简而言之是通过将图像空间中的一点转化为参数空间中的直线集（由于不是重点，略过）

在opencv中，有自带的函数cv2.HoughLinesP或者cv2.HoughLines可以直接实现相应的功能。

检测结果：



可以看到仅仅利用几何约束就能将文字部分检测出，我们只需提取出这一部分的图片就可以进行下一步操作。

代码在./lib/houghTransform.py

核心代码如下，需要将图像进行边缘检测后送入函数中，边缘检测这里使用了canny算子。

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gaus = cv2.GaussianBlur(gray,(5,5),0)
edges = cv2.Canny(gaus, 50, 200)
#hough transform
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 50, minLineLength=60,
maxLineGap=10)
```

- **最大稳定极值区域（MSER-Maximally Stable Extremal Regions）**

  MSER的基本原理是对一幅灰度图像（灰度值为0～255）取阈值进行二值化处理，阈值从0到255依次递增。阈值的递增类似于分水岭算法中的水面的上升，随着水面的上升，有一些较矮的丘陵会被淹没，如果从天空往下看，则大地分为陆地和水域两个部分，这类似于二值图像。在得到的所有二值图像中，图像中的某些连通区域变化很小，甚至没有变化，则该区域就被称为最大稳定极值区域。这类似于当水面持续上升的时候，有些被水淹没的地方的面积没有变化。（参考https://blog.csdn.net/zhaocj/java/article/details/40742191）

  1.对于图像灰度的仿射变化具有不变性

  2.稳定性，区域的支持集相对灰度变化稳定

  3.可以检测不同精细程度的区域

  MSER提取过程

  1.使用一系列灰度阈值对图像进行二值化处理

  2.对于每个阈值得到的二值图像，得到相应的黑色区域与白色区域

  3.在比较宽的灰度阈值范围内保持形状稳定的区域就是MSERs

  4.评判标准： dA/dt

  A: 二值图像区域面积，t: 灰度阈值

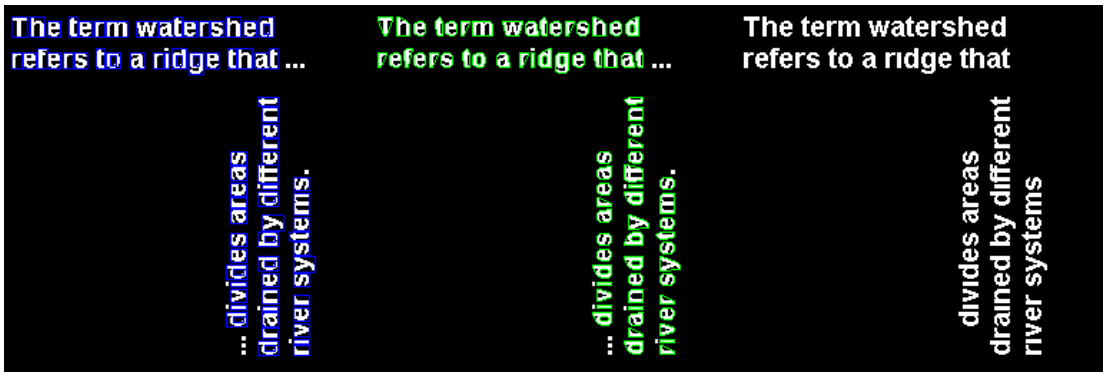
（参考百度百科MSER）

实际上算法借鉴了求取连通域这一大类问题的思想。

在opencv中，有自带的函数cv2.MSER_create().detectRegions()可以直接实现相应的功能。

检测结果：

左图是bbox框图，中间用凸包围住每个字母，右图是利用mask来去除图像中与文字无关区域后的图。



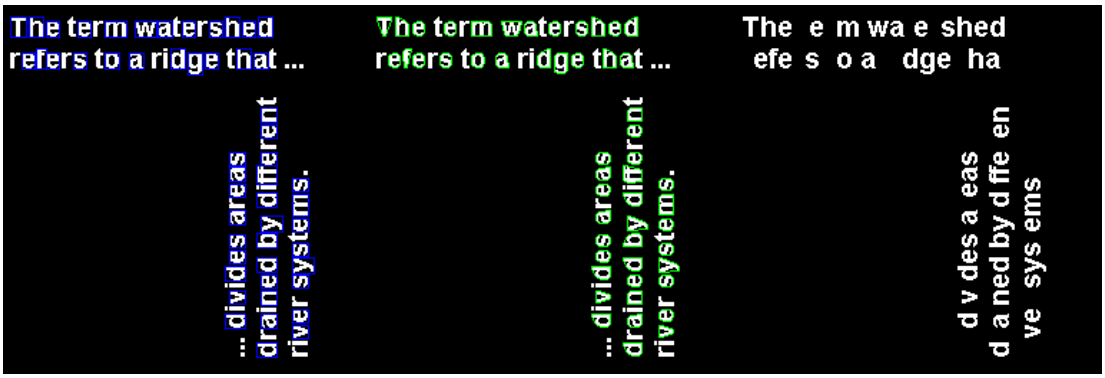

从检测结果来看基本上所有文字都可以被框出，但是在一些情况下**出现了较多重合的框以及一些大框**。

代码在./lib/mser.py，凸包的实现参考了网上代码。

核心代码：

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
mser = cv2.MSER_create(*args, **kwargs)
regions, boxes = mser.detectRegions(gray)
```

值得一提的是cv2.MSER_create()中有很多的参数，其中min_area是检测出的连通域的最小面积，如果面积小于给的值，则该区域被删除。如果设置的过小则会有很多噪点（不能光靠滤波解决的问题），如果设置的过大，则一些小的文字不能被检测出来，如下图所示：

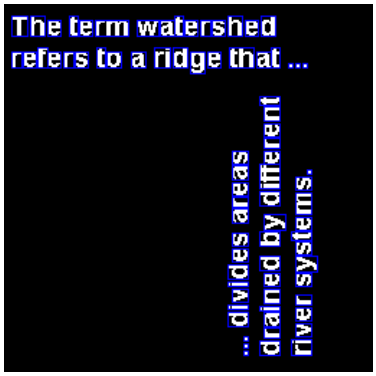



- **连通域算法**

  通过直接求取图像中的所有4-连通域来求得所有字母所在的位置。

  利用了skimage中的measure.label来获得连通域的若干信息。

  检测结果：




  可以看到基本上所有的字母都被检测出。但是由于有些字母之间隔的比较近就被认为是同一连通域，则出现了更大的框。这是由于在连通域算法之前做了一次滤波的原因。

  代码在./lib/connect.py，核心代码如下：

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray,(3,3),0)
    # 二值化
    if np.mean(gray) < 128:
        ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU +
cv2.THRESH_BINARY)
    else:
        ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU +
cv2.THRESH_BINARY_INV)
    labeled_img, num = label(binary, connectivity=1, background=0,
return_num=True, *args, **kwargs)
    return regionprops(labeled_img)
```

  其中比较重要的点是要对灰度图像先做一次滤波（cv2.GaussianBlur），否则会有很多的小噪点被认为是连通区域而被检测出，其次是需要考虑连通域中不能被认为是背景，因此需要在做二值化操作时对于例如白底黑字的图片做一些处理（cv2.threshold、大津算法）。这里我采用的是通过比较图像灰度值的均值来判断背景是黑底还是白底，从而进行不同的处理。

- **我的改进算法-结合mser和直接连通域的方法**

  代码在./pipeline.py中的boost函数。测试结果如下：

mser方法的问题在于不能自适应判断文字的大小，且会有很多重复的框；直接采用连通域算法由于高斯滤波，容易得到一些融合的框，因此我考虑融合两种算法，首先先用连通域算法求出一些候选区域，然后利用这些候选区域的属性来指导mser的检测。具体如下：

  - **利用连通域中得到的最小最大面积作为参数送入mser函数；**
- **在mser生成的矩形框中利用非极大抑制non_max_suppression来删除一些不必要的候选框，解决mser中出现的问题**
- 非极大抑制，顾名思义就是抑制不是极大值的元素，可以理解为局部最大搜索。本次实现了非极大抑制算法的简易版本，效果良好，
  - **利用mser中region的面积与所有连通域中最接近的面积的差值作为非极大抑制中的概率值，指导模型保留那些与连通域大小类似的区域。**

  我也尝试了如果图片中出现了不同大小的字体，采用kmeans和silhouette_score去指导不同字体在非极大抑制过程中概率值的生成，但目前效果不是很好（该部分代码被注释掉）

## 字母识别部分

采用了一些简单的模式识别手段。

- **数据生成**

  做识别的任务，数据集必不可少，我首先生成了一些字母图片的数据，手动挑选了一些字体来生成（mac系统）几百张图片。

  代码在./create_data.py，利用了 PIL 库中的 Image, ImageDraw, ImageFont, ImageFilter

- **数据分类**

  我采用了两种方法，第一种方法利用检测到字母和生成数据的SIFT特征（cv2.xfeatures2d.SIFT_create()）进行brute force匹配，以匹配数作为评价指标，判断检测到的字母属于哪一类。但这种方法效果不佳，原因可能是不同图片SIFT descriptor本身数量就不同，单纯比较数量会让descriptor数量多的更容易胜出。代码在./pipeline.py的pipeline函数

  **第二种方法**先将生成数据通过相同的检测算法二值化，并抠出图像中只有字母的部分（与我们检测到的东西保持一致），然后将所有图片resize成20*15的大小，直接将整个图片作为特征。训练SVM分类器并测试之前检测到的字母。（利用了sklearn）

## 可视化交互

利用opencv实现了鼠标触发的可视化交互，当鼠标移动到图片上某个位置时，自动标出检测出的字母的bounding box，以及识别出来的字母。

代码在./demo.py，录制了一个视频在./demo.mp4。