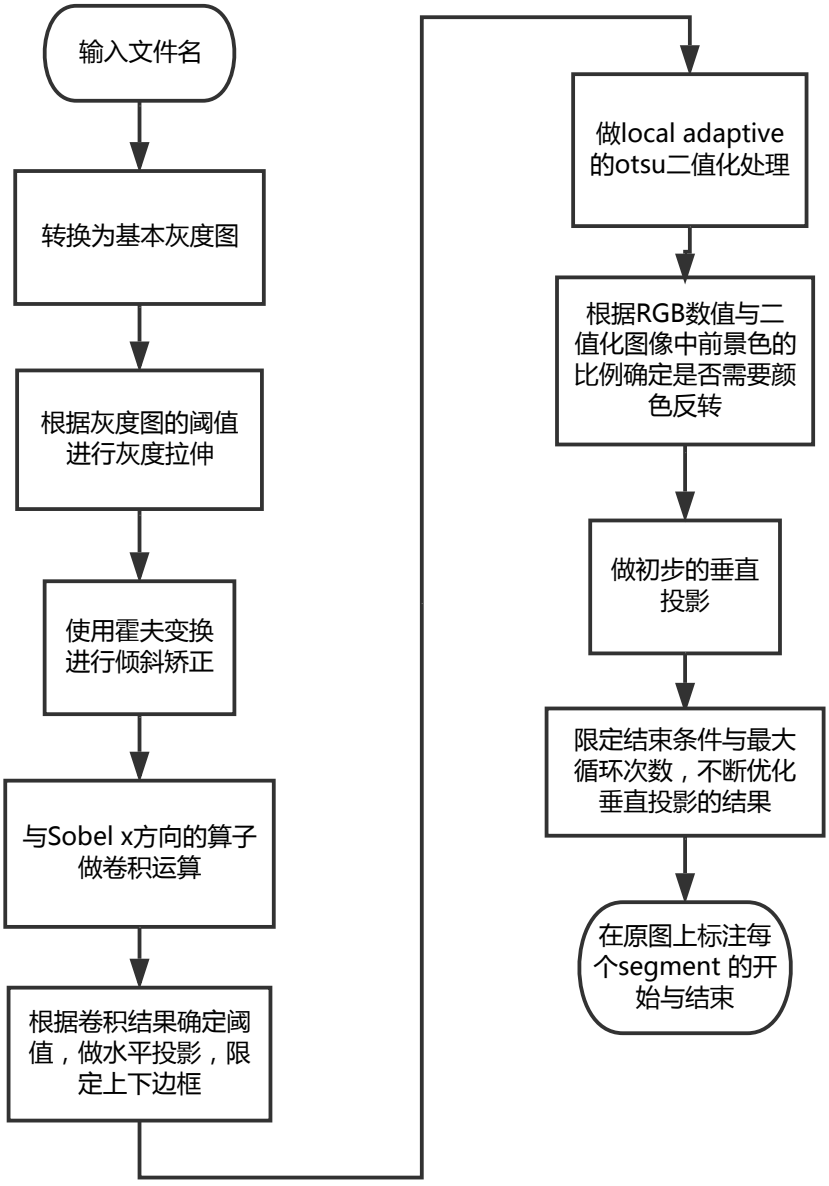
基于垂直投影的车牌分割
——适用于分辨率较差的车牌图像

顾秀烨　　3130101959

一、总体方法

在尝试了很多方法以及这些方法的多种组合后，最后我的车牌分割流程如下图所示。



二、各个步骤的说明

1. 灰度图的处理

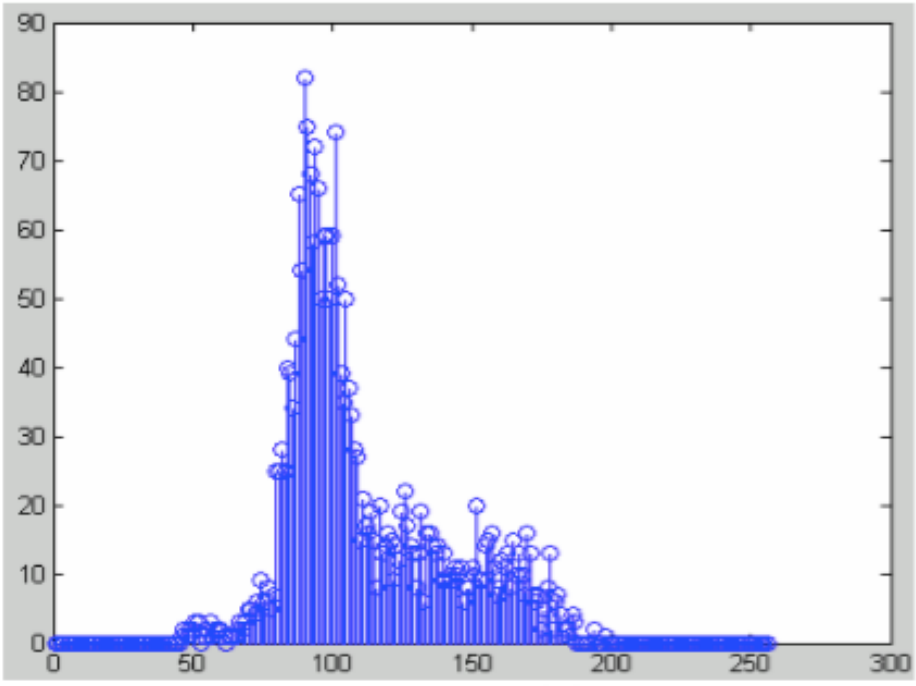
1) Transform to basic gray:

RGB -> YUV

gray value = Y in YUV color space.

2) Contrast-stretching transformation [1]

获得灰度密度图：



The gray density level graph.

The threshold of the density of gray level in this algorithm is the average of the density, which can be calculated by:

threshold = (the area of the whole graph) / 256

在确定了 threshold 以后，在密度大于 threshold 的灰度值中寻找最小和最大的灰度。使用线性灰度拉伸，将大于阈值的灰度最小值拉伸为 0，将大于阈值的灰度最大值拉伸为 255，以增强对比度，减弱不同光照对图像的影响，使得二值化时准确率更高。

2. 轻微的倾斜矫正

2.1 倾斜预处理

1) 平均模糊处理

有助于去除噪点，提高边缘提取的正确率

2) Canny 边缘检测

这一步为下一步霍夫变换做准备，根据霍夫变换的原理——使用点来投票决定直线的参数，而我需要提取的是边缘上的点。

2.2 霍夫变换，检测倾斜角度

1) 选取原图的上边框部分与下边框部分。
我的设想是通过边框的水平边界线或字符的上下切线来获取倾斜角度，所以只需要对这两部分做霍夫变换即可。否则中间区域的字符边界上的点会对直线的提取造成较大干扰。不过这也限制了这里的霍夫变换能够矫正的幅度。对于大幅倾斜的图片并不能很好的矫正。

事实上，我在中间结果中标注了用于计算倾斜角度的直线，发现由于边框的不完整，或者没有边框，会导致有些线条的选取有偏差。

另一方面，我也认为倾斜校正应该主要是 plate detection 的工作，因为在当时还有完整的边框，霍夫变换能够更好地检测到边框线条。所以，出于对倾斜车牌的处理效果的不满意，我尝试了一下去矫正在另一个文件夹（bad_phone_reg_origin）里的车牌，使用了颜色定位法，对该文件夹里的车牌进行了定位、提取、倾斜矫正、仿射变换（矫正另一种倾斜）。

2) 对上下边框部分分别使用霍夫变换函数找出边框水平边界或车牌文字的上下切线
其中涉及一些细节：
首先，设定一个较大的 threshold（Accumulator threshold parameter），逐步减小直到在所框定的区域中找到两条直线，或者 threshold 达到最小值——认为那一部分没有边框。

同时，通过设定 minLineLength 与 maxLineGap 来限定想要获得的直线的特征。
对获得的所有直线，计算直线的角度，只有角度在一定范围内的角度才会被考虑用于倾斜矫正。

对于所有有效角度，求平均值。

2.3 倾斜校正

1) 对部分车牌进行倾斜矫正

对于上一步获得的倾斜角度平均值，若该角度大于最小需要矫正的角度，则对该车牌进行旋转。

在进行旋转时，考虑到数据集中的车牌图像很模糊，霍夫变换检测到的倾斜直线可能有一定偏差，所以需要对倾斜角度做一定的调整。由于后期的分割算法鲁棒性较好，可以允许一定范围的倾斜。

同时，我也注意到，有很多车牌，尽管有比较倾斜的边框边界，但是车牌字符比较正。

所以在做倾斜校正时，减小了检测到的倾斜角度平均值——

abs(rotate_angle) = abs(detected_angle) * rotate_modify_ratio – rotate_modify

总的来说就是针对不同的倾斜程度，都在一定程度上减小了倾斜角。以免检测倾斜方向失误与矫枉过正对分割结果的影响。

3. 去除上下边框

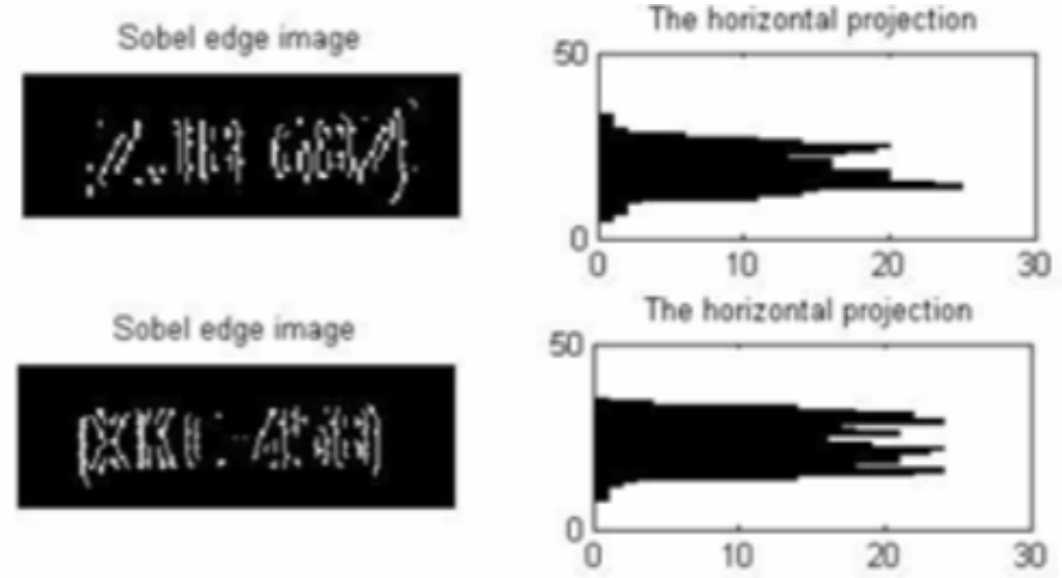
最后进行分割的方式是垂直投影，通过设定阈值进行分割，再优化分割结果。所以一些倾斜的、比较粗的边框对于我所采用的垂直投影法很不利。

在我搜索到的文献中，我找到了一种去除上下边框的方法，对其进行了一定的修改，并实现了它。

3.1 对灰度图做水平方向的 x sobel 算子

这样可以提取出 x 方向的边界，由于车牌字符的边界很多，明显多于边框与噪点的边界线，所以可以对提取出的水平边界做水平投影，区分出字符与边框。

3.2 对检测到的边界做水平投影



即记录下每一行上白色像素点的个数。

在理想情况下，字符区域的水平投影明显大于边界区域。所以通过设定一定的阈值，选择所有行的水平投影数大于该阈值的区域，即为字符区域。在没有倾斜的情况下，该阈值可以设为 0.

但在实际操作中，还是需要限定该区域的范围（由于给定的图片都是 80 个像素的高度，所以我把字符区域的范围限制在了 [15, 65] 的范围及更宽），否则很容易出错。比如说，有些车牌的字符很窄（例如 1，7）而边框又很粗而且噪点多。

4. 二值化

经过测试，我选择了 Local Adaptive 的 Otsu 法。

基于 PPT 中提出的处理光照影响的要求，我认为受光照影响的主要步骤是二值化。

所以我选择的方案是局部自适应的二值化方法（local adaptive），我观察到，在传统的全局 Otsu 方法中，对于一些高光区域，光照区域会被二值化为白色的，给垂直分割带来难度。

在 local adaptive 的处理上，我尝试了[3]中讨论的使用 Integral Images 来实现的 Sauvola 二值化算法，基本公式为

$$o(x,y) = \begin{cases} 0 & \text{if } g(x,y) \leq t(x,y) \\ 255 & \text{otherwise} \end{cases} \tag{1}$$

In Sauvola's binarization method, the threshold $t(x,y)$ is computed using the mean $m(x,y)$ and standard deviation $s(x,y)$ of the pixel intensities in a $w \times w$ window centered around the pixel (x,y):

$$t(x,y) = m(x,y)\left[1 + k\left(\frac{s(x,y)}{R} - 1\right)\right] \tag{2}$$

where R is the maximum value of the standard deviation ($R = 128$ for a greyscale document), and k is a parameter which takes positive values in the range $[0.2, 0.5]$. The local mean $m(x,y)$ and standard deviation

虽然使用了图像积分的方法，使得运算效率有了很大提升，但是可能因为 window width 等参数的设定不够好，或者算法本身不适用于低分辨率的图像，结果并不好，一些非字符区域会被二值化为白色，总体效果劣于全局 Otsu。

最后，我采用的是 local adaptive 的 Otsu 方法。

基本思路为设定一个小窗（窗宽，窗高），从左到右，从上到下滑动该小窗，每一次滑动需要与上一次的小窗有一定的覆盖，每一次计算阈值，作为小窗中元素的阈值，两个小窗之间的 overlap 区域的阈值是通过直接覆盖的方式给定的。

效果是对于高光区域有一定优化，但是 local adaptive 的效果还依赖于窗宽、窗高、图像本身的状况，所以不能完全依赖二值化消除光照的影响。

对于光照带来的不能被 local adaptive 的二值化算法消除的影响，我选择依赖垂直投影优化算法的鲁棒性。

5. 颜色判断与反转

中国的车牌种类繁多，而且拍摄车牌时受光线的影响，对车牌颜色的判断带来了很多困难。我尝试了多种方法，最终选择的方案是结合 RGB 信息与二值化后的前景色与背景色像素的比例来进行判断。

首先，在将彩色图片转换为灰度图片时，记录下所有像素 R、G、B 三种颜色值的和（sumR, sumG, sumB）.

当满足 sumG + sumB > 2.5 * sumR || sumB > sumR * 1.18 条件时，认为是蓝色车牌（有些蓝色车牌偏绿）。

如若不然，当 sumR + sumG > 2.5 * sumB 时，认为是黄色车牌。

如果上述两个条件判断都不满足，那么计算二值化结果中的白色像素点所占的比例，当 whiteRatio > 0.53 时，认为车牌背景色被判别为了白色，需要反转车牌的颜色。

当 whiteRatio < 0.47 时，认为不需要反转车牌的颜色。

当 whiteRatio 介于两个比例值之间时，由于情况特殊，需要特判。

我采用 opencv 弹出图片，并询问是否需要逆转颜色，以进行特判。（在做 hough 变换时，我学习了 opencv 的相关内容，发现它可以在程序中显示图片，对提高特判的效率很有帮助。）

在上述流程后，在 987 张图片中，只有个位数的车牌（亮度过大的蓝色车牌，黑底白字而光照偏黄的车牌）判断失误。我选择了不再加强约束，而容忍了这些判断失误，因为特判过多会给自动化系统带来一定效率上的影响，另一方面这些失误的案例可以通过学习机器学习的相关知识，在未来或许可以解决。

我也曾采用转换为 HSV 色彩空间，对于 S、V 值在一定范围内的像素，通过它们的 H 值所在的范围，采用颜色模板的方式来进行色彩判断。（参考了 EasyPR 的颜色判断方式，EasyPR 是我做完了分割，做车牌定位时发现的一款开源车牌识别系统）。但是效果比我自己朴素的 RGB 判断差很多。

另一种尝试是图像主题色彩提取算法，我观察了一下网上别人使用中位切分法（Median cut）、八叉树算法（Octree）、聚类（K-means）算法的结果，决定采用 Median cut color quantization 算法进行尝试，使用 python 进行了测试，发现该算法会提取出环境中具有一定面积的干扰色，另一方面，或许还是因为我无法很好地对提取到的颜色进行判断，所以 color quantization 算法在判断车牌颜色上的表现并不出众。

6. 初步垂直投影[4]

首先，对颜色反转过后的二值化图像，计算每一个横坐标上，所有白色像素点的个数。

在主程序中给定一个初始阈值，选取垂直投影值均大于该阈值的区域作为一个分割片段：

记录下每个片段的起始 x 坐标与结束 x 坐标，返回。

8. 垂直投影分割结果的优化

在上一步中，由于每个车牌的情况都不同，会有一些因为车牌边框未清理干净而造成的粘连片段，也会有一些断裂的片段（比如汉字“川”，即使在最理想的情况下也会被分割开）。

我的算法是：
1）去除左右的边框，当最左边（最右边）的片段的横坐标落在一定区域内，并且该片段的宽度小于一定阈值时，认为该片段是边框，移除。

考虑到一些车牌的断裂的汉字（“川”字，最左边的一撇与边框从图像的角度上几乎没有明显区别），左边框的宽度阈值我设得很小，可能会在结果中留下一些边框，需要字符识别这一步进一步的处理。

2）去除车牌中间地域编号与车牌序号之间的一点
由于我初步选择的算法是根据获得的片段的宽度与距离的平均值来进行合并或进一步分割，车牌中间这一点如果被识别为一个字符的话会给平均值带来一定的扰动。所以我设计了去除干扰点的步骤。而结果表明干扰点特征明显，去除步骤的结果很好。

给干扰点设定横坐标的范围，对可能为干扰点的片段进行在一定纵坐标区域（排除遗留边框的干扰）内的逐列扫描，当某一列中的限定区域内的白色像素点的个数大于一定的值时，判定为非干扰点片段。同时统计扫描时进入白色像素点区域的次数，当次数大于 2 时，也判定为非干扰点片段（干扰点为一个白点，白色像素区域应该是连续的，考虑边缘可能呈锯齿形，设定为进入前景色像素区域的次数最大值为 2）。

3）扫描所有片段，对片段宽度大于初始分割宽度的片段进行分割（一分为二，下次判断从分割结果中的第一个片段开始继续判断是否需要分割）。

在分割程序中，逐步提高阈值（起始值为初步垂直投影时使用的初始阈值）直到可以将给定片段一分为二为止。

4）进入优化循环，每次循环，计算相邻分割片段之间的距离（由于数字 1 等类似字符的片段的宽度很小，所以无法使用宽度进行判断）。

遍历所有的距离值，当两个片段的距离小于给定的最小距离时，进行合并，当两个距离大于给定的最大距离、并且其中较宽的片段的宽度大于最小分裂宽度时，对该片段进行分割。每当分割发生变化导致距离发生变化时，进入循环结束判断块，若判断失败，则重新计算宽度值，进行下一轮优化。

循环结束条件为，循环次数大于给定值，或者车牌分割得到 6-8 个片段、每两个相邻片段间的距离均处于给定范围内，则可提前结束循环。6-8 个片段的选择是由于，虽然车牌字符数是不定的，但是由于是基于中国车牌（7 个字符）的，给定+/- 1 的合理偏差。

在初步尝试时，结合了几篇论文上给出的方案，我选择根据距离平均值来进行判断是否需要合并或进一步分割，但是效果并不是很好，我们都知道，平均值受极端值的影响很大，如果要将标准差等值加上去也并不是很好做。但是，当车牌的字符数完全不确定时，以平均值（与标准差）为判断依据是目前最普遍的做法，也是最不受字符个数干扰的做法。

5）如果每相邻两个片段之间的空隙允许的话，给每个片段的左右坐标均拓宽 1，（减缓类似'7','J'字符端点处垂直投影值较小带来的分割偏差）

6）将得到的所有片段的起始值与结束值在经过倾斜矫正的车牌上标注（起始值以红线标注、结束值以绿线标注）。

三、示例

以下图为例，以图示的形式给出处理过程：

原图

灰度化并进行灰度拉伸后的结果

使用 Canny 算子检测到边缘，并对上下边框区域进行霍夫变换检测到边界线

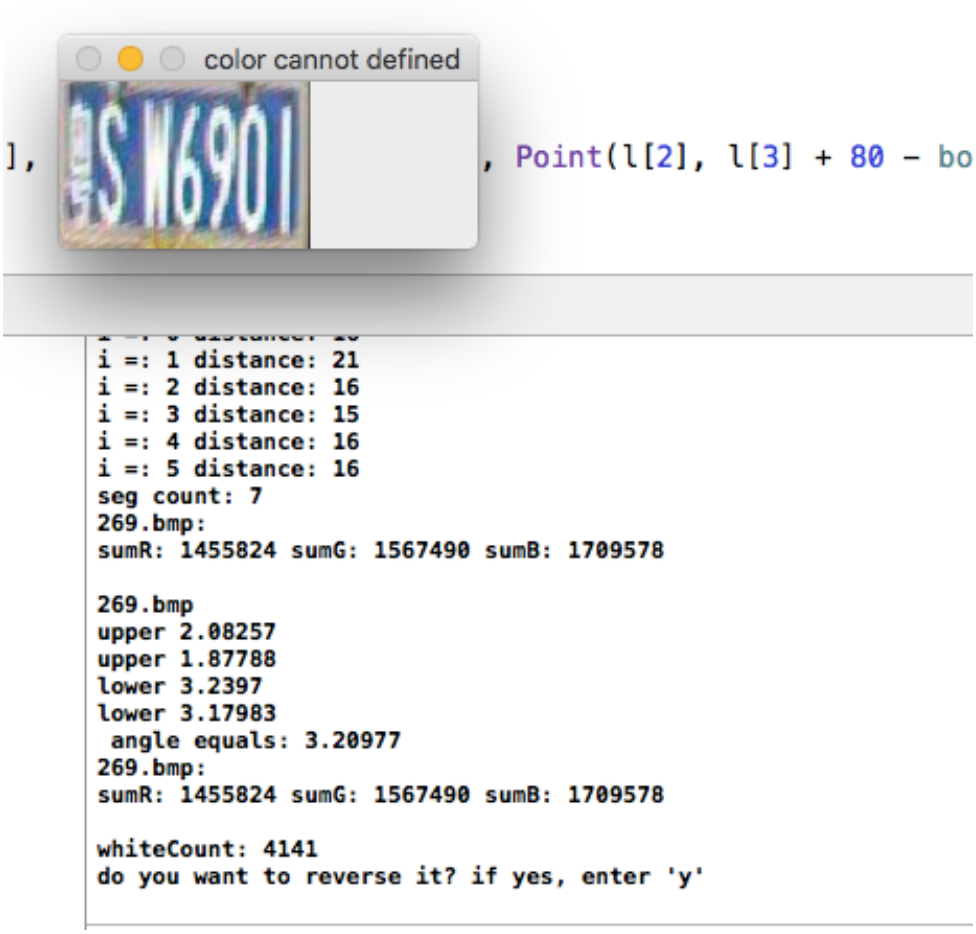
旋转后的结果

除去上下边框并进行二值化的结果，其后进行了颜色判断与反转

初始化垂直分割的结果（这里每个片段的起始坐标界线与结束坐标界线均标注为红色）

优化后的分割结果（可以看到，很好地去除了左右边框与中间的干扰点，并且分割开了每一个字符）

颜色反转特判示例：



四、最后结果

我对给定的 987 张车牌均做了处理，其中除了倾斜角度特别厉害，或者特别模糊（人眼无法辨别）以及颜色判断出现失误导致未能正确反转颜色的车牌，其余车牌的分割效果均很好。

样本的分割结果如下图所示：

所有 987 张图片的结果我也附在了附件中。

五、工程运行环境与运行方法说明

运行环境：
系统：osx 10.11 系统
IDE：xcode （系统需安装 opencv3）

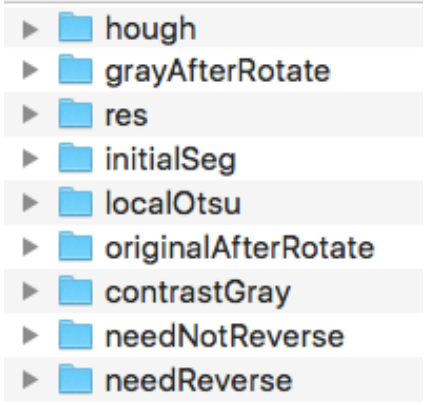
附件中的工程已经将所有的步骤都封装起来了。
但是需要首先将所有 jpg 图片转换成按照顺序命名的 bmp 图像（车牌分割的工程除了倾斜矫正部分使用了 opencv，其余部分都是使用 C++手写的）

可以使用如下 python 代码：

```
#!/usr/bin/python

from PIL import Image
import glob

count = 1

for filename in glob.glob(r'./*.jpg'):
  im = Image.open(filename)
  im.save(r'./bmp/'+str(count)+'.bmp','BMP')
```

```
  count = count + 1

for filename in glob.glob(r'./*.png'):
  im = Image.open(filename)
  im.save(r'./bmp/'+str(count)+'.bmp', 'BMP')
  count = count + 1
```

如果要运行，需要修改 main 函数中的 seg.openPath 为 987 张 bmp 图片所在的绝对路径，修改 seg.globalPath 为结果的保存路径，在保存路径中，按照现在的设定，需要建有以下文件夹（最终结果保存在 res 中）：



部分参考文献：

[1] A Novel Approach for License Plate Character Segmentation. Feng Yang, Zheng Ma, Mei Xie.
[2] Segmentation of Characters on Car License Plates. Xiangjian He, Lihong Zheng, Qiang Wu, Wenjing Jia, Bijan Samali and Marimuthu Palaniswami.
[3] Efficient Implementation of Local Adaptive Thresholding Techniques Using Integral Images. Faisal Shafait, Daniel Keysers, Thomas M. Breuel.
[4] 基于垂直投影的车牌字符分割方法. 冉令峰.
[5] EasyPR. https://github.com/liuruoze/EasyPR
[6] Color Quantization. http://www.leptonica.com/color-quantization.html
[7] A New Algorithm for Character Segmentation of License Plate. Yungang Zhang, Changshui Zhang.
[8] 基于 Hough 变换的车牌倾斜检测算法. 包明, 路小波.
[9] Hough transformation. http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm

Table of Contents