EasyPR 开发详解（1）

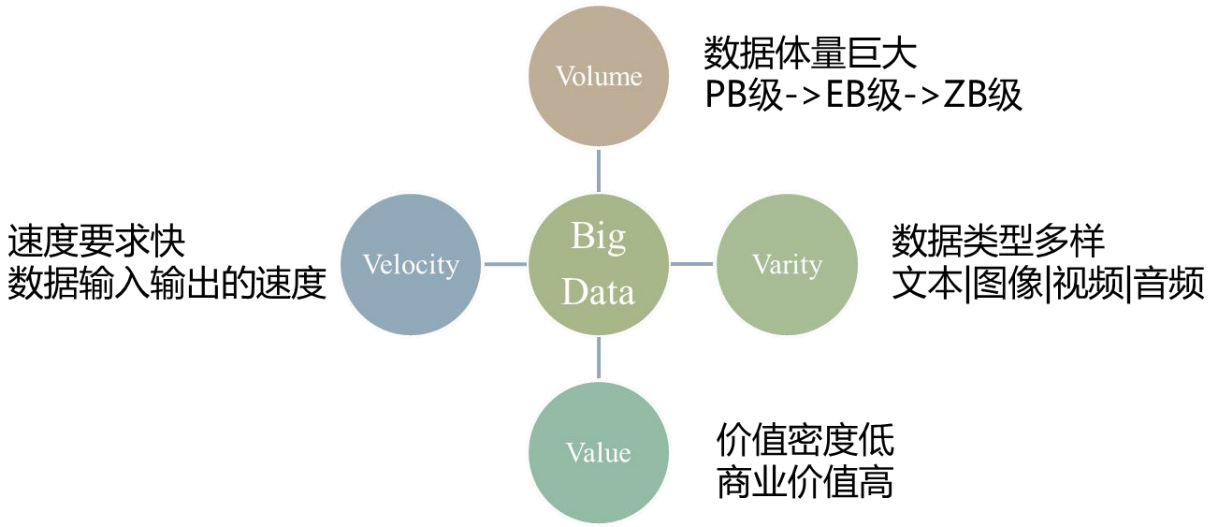
在上篇文档中已经简单的介绍了 EasyPR，现在在本文档中详细的介绍 EasyRP 的开发过程。

正如淘宝诞生于一个购买来的 LAMP 系统，EasyPR 也有它诞生的原型，起源于 CSDN 的一个博客，作者以读书笔记的形式记述了通过阅读“”这本书完成的一个车牌系统的雏形。

这个系统有几个特点：1.将车牌系统划分为了两个过程，即车牌检测和字符识别。2.整个系统是针对西班牙的车牌开发的，与中文车牌不同。3.系统的训练模型来自于原书。作者基于这个系统，诞生了开发一个适用于中文的，且适合与协作开发的开源车牌系统，也就是 EasyPR。

当然了，现在车牌系统满大街都是，随便上下百度首页都是大量的广告，一些甚至宣称自己实现了 99%的识别率。那么，作者为什么还要开发这个系统呢？这主要是基于时势与机遇的原因。

众所皆知，现在是大数据的时代。那么，什么是大数据？可能有些人认为这个只是一个概念或着炒作。但是大数据确是实实在在有着基础理论与科学研究背景的一门技术，其中包含着分布式计算、内存计算、机器学习、计算机视觉、语音识别、自然语言处理等众多计算机界崭新的技术，而且是这些技术综合的产物。事实上，大数据的“大”包含着 4 个特征，即 4V 理念，包括 Volume（体量）、Varity（多样性）、Velocity（速度）、Value（价值）。



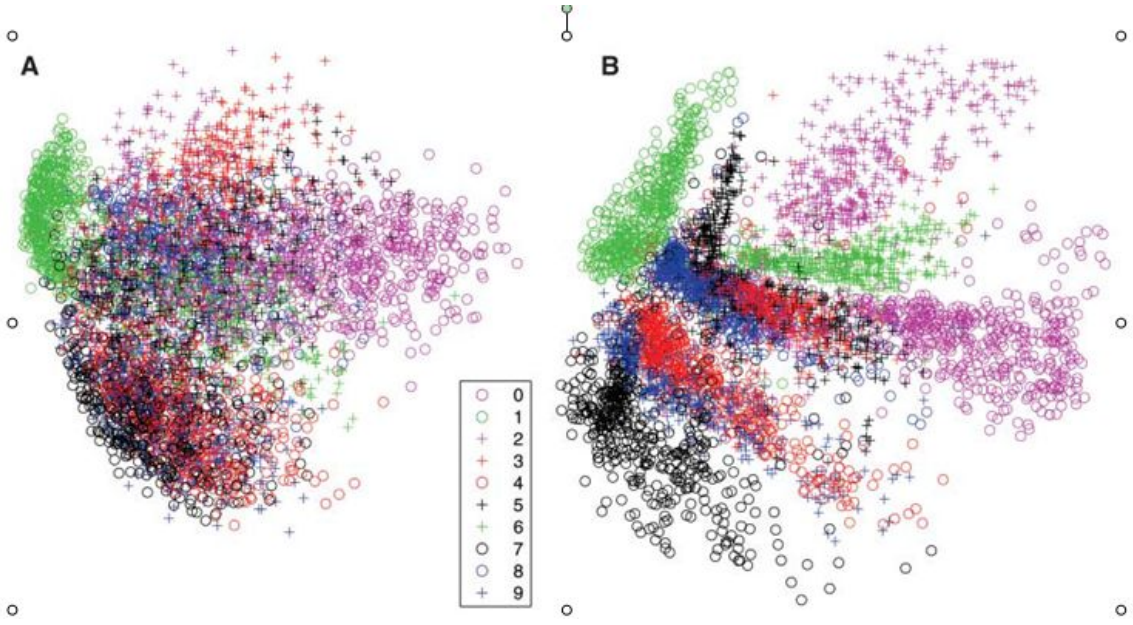
也就是说大数据技术不仅包含数据量的大，也包含处理数据的复杂，和处理数据的速度，以及数据中蕴含的价值。而车牌识别这个系统，虽然传统，古老，确是包含了所有这四个特侦的一个大数据技术的缩影。在车牌识别中，你需要处理的数据是图像中海量的像素单元，你处理的数据不再是传统的结构化数据，而是图像这种非结构化数据，如果不能在很短的时间内识别出车牌，系统对海量数据的处理就会被拖慢，虽然一副抓拍图像中有很多的信息，但可能仅仅只有那一小块的信息（车牌）以及车身的颜色是你关心，而且这些信息都蕴含着价值。

也就是说，车牌识别系统事实上就是现在火热的大数据技术在某个领域的一个聚焦，通过了解车牌识别系统，可以很好的帮助你理解大数据技术的内涵，也能清楚的认识到大数据的价值。很神奇吧，也许你觉得车牌识别系统很低端，这不是随便大街上都有的么，而你又认为大数据技术很高端，似乎高大上的感觉。其实两者本质上是一样的。另外对于觉得大数据技术是虚幻的炒作念头的同学，你们也可以了解一下车牌识别系统，就能知道大数据落在实地，事实上已经不知不觉进入我们的生活很长时间了，像一些其他的如抢票系统，语音助手等，都是大数据技术的真真切切的体现。所谓再虚幻的概念落到实处，就成了下里巴人，应该就是这个意思。所以对于炒概念要有所警觉，但是不能因此排除一切，要了解具体的技术内涵，才能更好的利用技术为我们服务。

除了帮忙我们更好的理解大数据技术，使我们跟的上时代，开发一个车牌系统还有其他原因。那就是、现在的车牌系统，仍然还有许多待解决的挑战。这个可能很多同学有疑问，你别骗我，百度上我随便一搜都是 99%，只要多少多少元，就可以 99%。但是事实上，车牌识别系统业界一直都没有一个成熟的百分百适用的方案。一些 90%以上的车牌识别系统都是跟高清摄像机做了集成，由摄像头传入的高分辨率图片进入识别系统，可以达到较高的识别率。但是如果图像分辨率一旦下来，或者图里的车牌脏了的话，那么很遗憾，识别率远远不如我们的肉眼。也就是说，距离真正的智能的车牌识别系统，目前已有的系统还有许多挑战。什么时候能够达到人眼的精度以及识别速率，估计那时候才算是完整成熟的。

那么，有同学问，就没有办法进一步优化了么。答案是有的，这个就需要谈到目前火热的深度学习与计算机视觉技术，使用多隐层的深度神经网络也许能够解决这个问题。但是目前 EasyPR 并没有采用这种技术，或许以后会采用。但是这个方向是有的。也就是说，通过研究车牌识别系统，也许会让你一领略当今人工智能与计算机视觉技术最尖端的研究方向，即深度学习技术。怎么样，听了是不是很心动？最后扯一下，前端时间非常火热 Google 大脑技术和百度深度学习研究院，都是跟深度学习相关的。

下图是一个深度学习（右）与传统技术（左）的对比，可以看出深度学习对于数据的分类能力的优势。



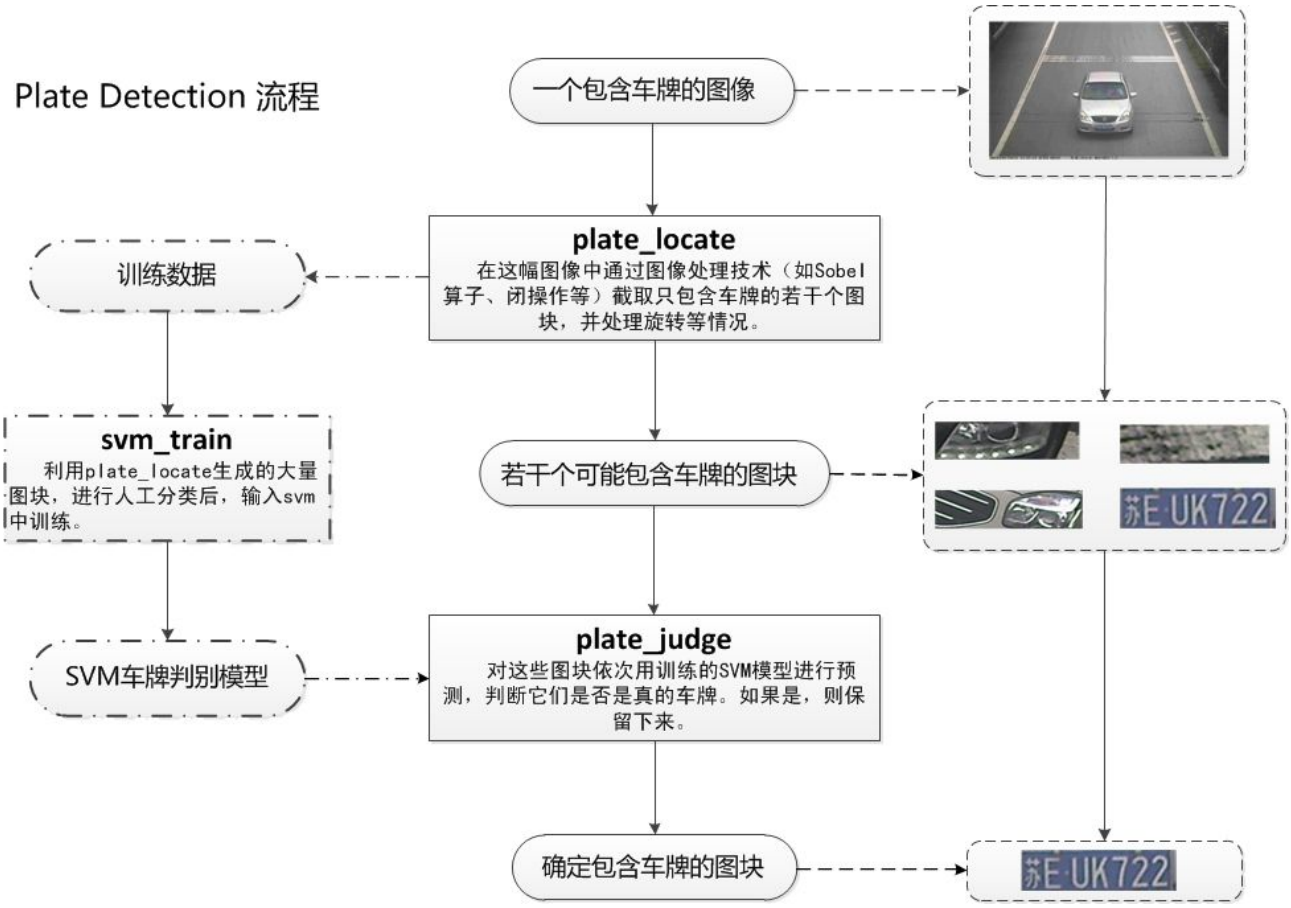
总结一下：开发一个车牌识别系统可以让你了解最新的时势---大数据的内涵，同时，也有机遇让你了解最新的人工智能技术---深度学习。因此，不要轻易的小看这门技术中蕴含的价值。

好，谈价值就说这么多。现在，我简单的介绍一下 EasyPR 的具体过程。

在上一篇文档中，我们了解到 EasyPR 包括两个部分，但实际上为了更好进行模块化开发，EasyPR 被划分成了六个模块，其中每个模块的准确率与速度都影响着整个系统。

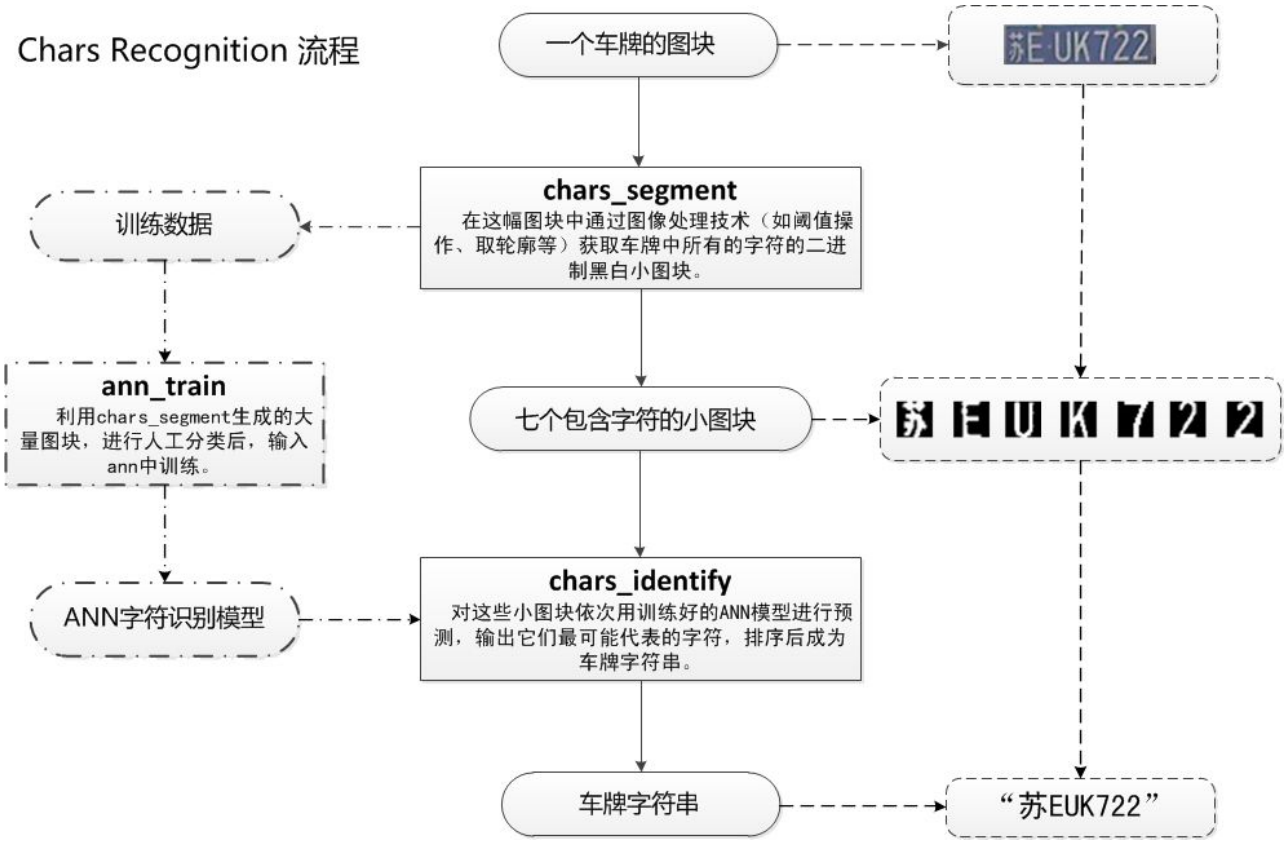
具体说来，EasyPR 中 PlateDetect 与 CharsRecognize 各包括三个模块。

PlateDetect 包括的是车牌定位，SVM 训练，车牌判断三个过程，见下图。



通过 PlateDetect 过程我们获得了许多可能是车牌的图块，将这些图块进行手工分类，聚集一定数量后，放入 SVM 模型中训练，得到 SVM 的一个判断模型，在实际的车牌过程中，我们再把所有可能是车牌的图块输入 SVM 判断模型，通过 SVM 模型自动的选择出实际上真正是车牌的图块。

PlateDetect 过程结束后，我们获得一个图片中我们真正关心的部分--车牌。那么下一步该如何处理呢。下一步就是根据这个车牌图片，生成一个车牌号字符串的过程，也就是 CharsRecognisze 的过程，具体见下图。



在 CharsRecognise 过程中，一副车牌图块首先会进行灰度化，二值化，然后使用一系列算法获取到车牌的每个字符的分割图块。获得海量的这些字符图块后，进行手工分类（这个步骤非常耗时间，后面会介绍如何加速这个处理的方法），然后喂入神经网络的 MLP 模型中，进行训练。在实际的车牌识别过程中，将得到 7 个字符图块放入训练好的神经网络模型，通过模型来预测每个图块所表示的具体字符，例如图片中就输出了“苏 EUK722”，（这个车牌只是示例，切勿以为这个车牌有什么特定选取目标。车主既不是作者，也不是什么深仇大恨，仅仅为学术说明选择而已）。

至此一个完整的车牌识别过程就结束了，但是在每一步的处理过程中，有许多的优化方法和处理策略。尤其是车牌定位和字符分割这两块，非常重要，它们不仅生成实际数据，还生成训练数据，因此会直接影响到模型的准确性，以及模型判断的最终结果。这两部分会是作者重点介绍的模块，至于 SVM 模型与 ANN 模型，由于使用的是 OpenCV 提供的类，因此可以直接看 openCV 的源码或者机器学习介绍的书，来了解训练与判断过程。

EasyPR 开发详解（2）车牌定位

这篇文章是一个系列中的第三篇。前两篇的地址贴下：介绍、详解 1。我撰写这系列文章的目的是：1、普及车牌识别中相关的技术与知识点；2、帮助开发者了解 EasyPR 的实现细节；3、增进沟通。

EasyPR 的项目地址在这：GitHub。要想运行 EasyPR 的程序，首先必须配置好 openCV，具体可以参照这篇文章。

在前两篇文章中，我们已经初步了解了 EasyPR 的大概内容，在本篇内容中我们开始深入 EasyRP 的程序细节。了解 EasyPR 是如何一步一步实现一个车牌的识别过程的。根据 EasyPR 的结构，我们把它分为六个部分，前三个部分统称为“Plate Detect”过程。主要目的是在一副图片中发现仅包含车牌的图块，以此提高整体识别的准确率与速度。这个过程非常重要，如果这步失败了，后面的字符识别过程就别想了。而“Plate Detect”过程中的三个部分又分别称之为“Plate Locate”，“SVM train”,“Plate judge”，其中最重要的部分是第一步“Plate Locate”过程。本篇文章中就是主要介绍“Plate Locate”过程，并且回答以下三个问题：

1.此过程的作用是什么，为什么重要？

2.此过程是如何实现车牌定位这个功能的？

3.此过程中的细节是什么，如何进行调优？

1. “Plate Locate”的作用与重要性

在说明“Plate Locate”的作用与重要性之前，请看下面这两幅图片。

图 1 两幅包含车牌的不同形式图片

左边的图片是作者训练的图片（作者大部分的训练与测试都是基于此类交通抓拍图片），右边的图片则是在百度图片中“车牌”获得（这个图片也可以称之为生活照片）。右边图片的问题是一个网友评论时问的。他说 EasyPR 在处理百度图片时的识别率不高。确实如此，由于工业与生活应用目的不同，拍摄的车牌的大小，角度，色泽，清晰度不一样。而对图像处理技术而言，一些算法对于图像的形式以及结构都有一定的要求或者假设。因此在一个场景下适应的算法并不适用其他场景。目前 EasyPR 所有的功能都是基于交通抓拍场景的图片制作的，因此也就导致了其无法处理生活场景中这些车牌照片。

那么是否可以用一致的“Plate Locate”过程中去处理它？答案是也许可以，但是很难，而且最后即便处理成功，效率也许也不尽如人意。我的推荐是：对于不同的场景要做不同的适配。尽管“Plate Locate”过程无法处理生活照片的定位，但是在后面的字符识别过程中两者是通用的。可以对 EasyPR 的“Plate Locate”做改造，同时仍然使用整体架构，这样或许可以处理。

有一点事实值得了解到是，在生产环境中，你所面对的图片形式是固定的，例如左边的图片。你可以根据特定的图片形式来调优你的车牌程序，使你的程序对这类图片足够健壮，效率也够高。在上线以后，也有很好的效果。但当图片形式调整时，就必须要调整你的算法了。在“Plate Locate”过程中，有一些参数可以调整。如果通过调整这些参数就可以使程序良好工作，那最好不过。当这些参数也不能够满足需求时，就需要完全修改 EasyPR 的实现代码，因此需要开发者了解 EasyPR 是如何实现 plateLocate 这一过程的。

在 EasyPR 中，“Plate Locate”过程被封装成了一个“CPlateLocate”类，通过“plate_locate.h”声明，在“plate_locate.cpp”中实现。

CPlateLocate 包含三个方法以及数个变量。方法提供了车牌定位的主要功能，变量则提供了可定制的参数，有些参数对于车牌定位的效果有非常明显的影响，例如高斯模糊半径、Sobel 算子的水平与垂直方向权值、闭操作的矩形宽度。CPlateLocate 类的声明如下：

```
class CPlateLocate
{public:
    CPlateLocate();

    //! 车牌定位
    int plateLocate(Mat, vector<Mat>& );

    //! 车牌的尺寸验证
    bool verifySizes(RotatedRect mr);

    //! 结果车牌显示    Mat showResultMat(Mat src, Size rect_size, Point2f center);

    //! 设置与读取变量
    //...
protected:
    //! 高斯模糊所用变量
    int m_GaussianBlurSize;

    //! 连接操作所用变量
    int m_MorphSizeWidth;
    int m_MorphSizeHeight;

    //! verifySize 所用变量
    float m_error;
    float m_aspect;
    int m_verifyMin;
    int m_verifyMax;

    //! 角度判断所用变量
    int m_angle;

    //! 是否开启调试模式，0 关闭，非 0 开启
    int m_debug;
};
```

注意，所有 EasyPR 中的类都声明在命名空间 easypr 内，这里没有列出。CPlateLocate 中最核心的方法是 plateLocate 方法。它的声明如下：

```
//! 车牌定位
int plateLocate(Mat, vector<Mat>& );
```

方法有两个参数，第一个参数代表输入的源图像，第二个参数是输出数组，代表所有检索到的车牌图块。返回值为 int 型，0 代表成功，其他代表失败。plateLocate 内部是如何实现的，让我们再深入下看看。

2. “Plate Locate”的实现过程

plateLocate 过程基本参考了 taotao1233 的博客的处理流程，但略有不同。

plateLocate 的总体识别思路是：如果我们的车牌没有大的旋转或变形，那么其中必然包括很多垂直边缘（这些垂直边缘往往缘由车牌中的字符），如果能够找到一个包含很多垂直边缘的矩形块，那么有很大的可能性它就是车牌。

依照这个思路我们可以设计一个车牌定位的流程。设计好后，再根据实际效果进行调优。下面的流程是经过多次调整与尝试后得出的，包含了数月来作者针对测试图片集的一个最佳过程（这个流程并不一定适用所有情况）。plateLocate 的实现代码在这里不贴了，Git 上有所有源码。plateLocate 主要处理流程图如下：

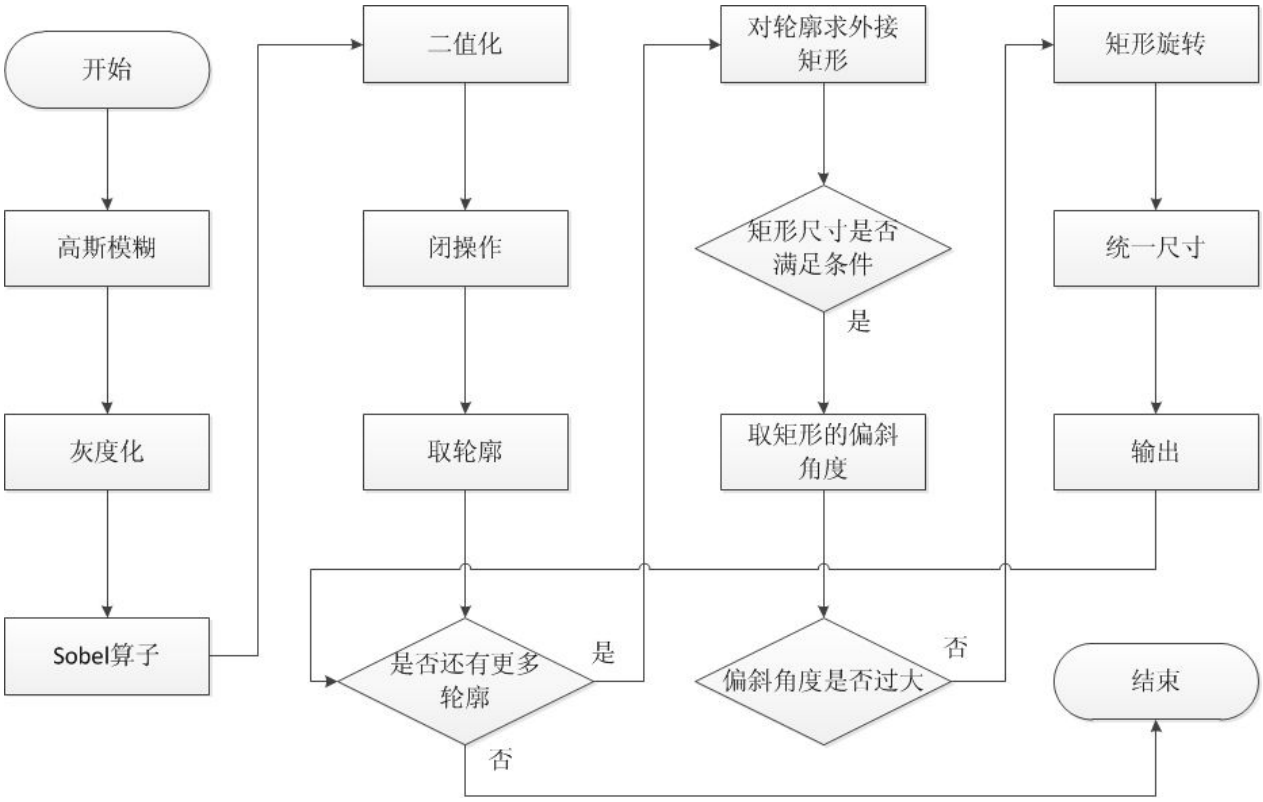


图 2 plateLocate 流程图

下面会一步一步参照上面的流程图，给出每个步骤的中间临时图片。这些图片可以在 1.01 版的 CPlateLocate 中设置如下代码开启调试模式。

```
CPlateLocate plate;
plate.setDebug(1);
```

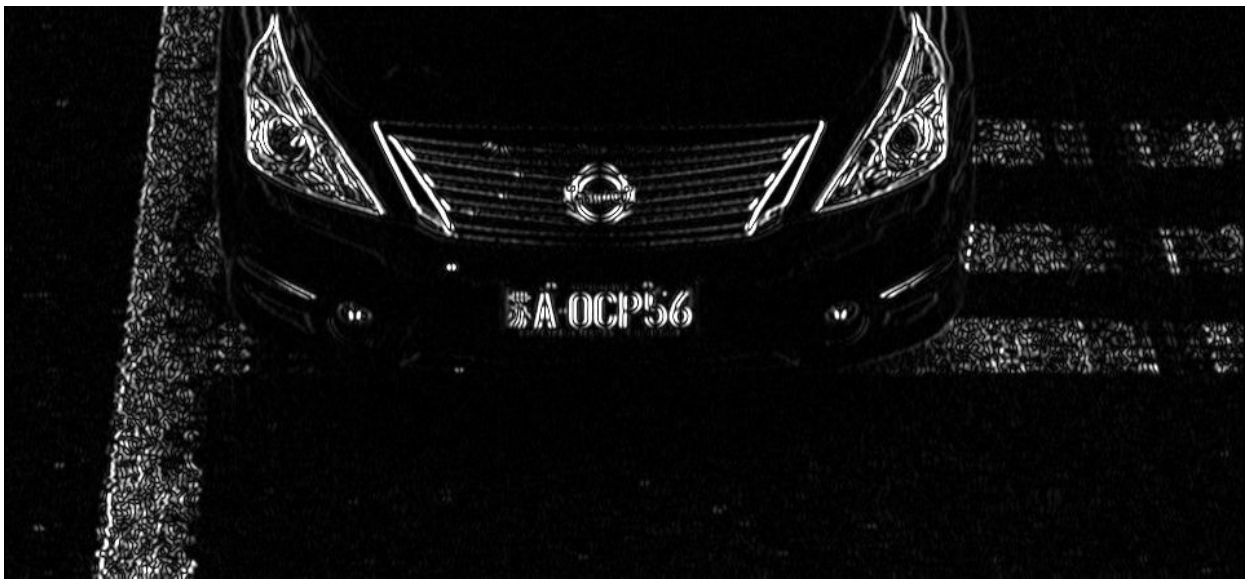
临时图片会生成在 tmp 文件夹下。对多个车牌图片处理的结果仅会保留最后一个车牌图片的临时图片。

1、原始图片。

2、经过高斯模糊后的图片。经过这步处理，可以看出图像变的模糊了。这步的作用是为接下来的 Sobel 算子去除干扰的噪声。

3、将图像进行灰度化。这个步骤是一个分水岭，意味着后面的所有操作都不能基于色彩信息了。此步骤是利是弊，后面再做分析。

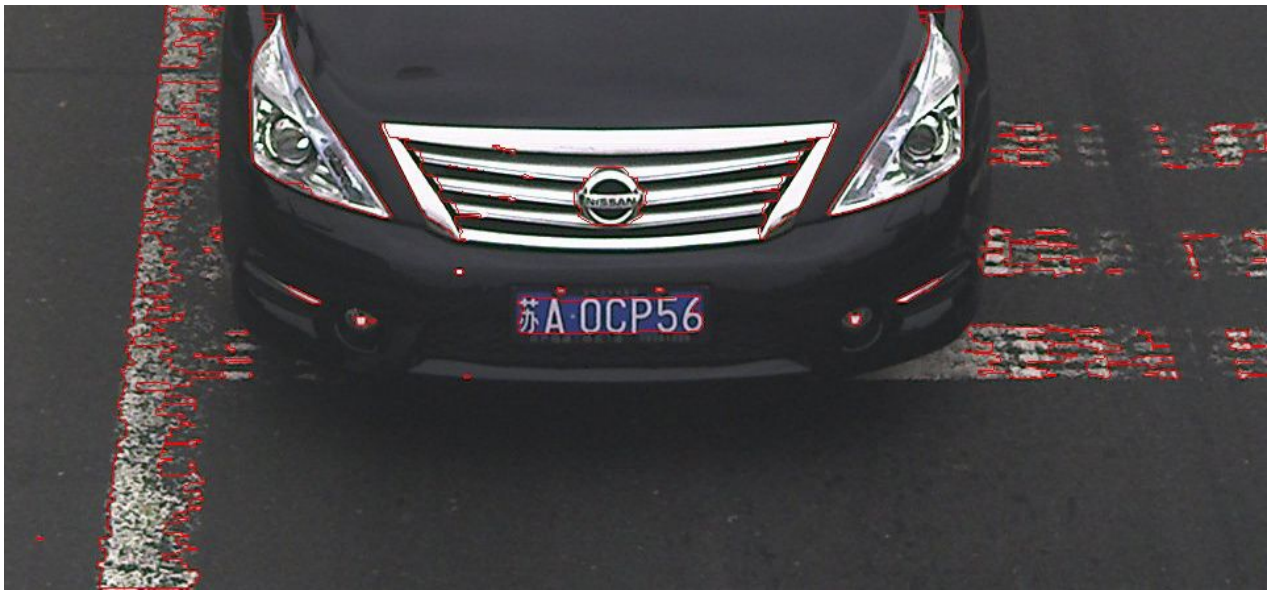
4、对图像进行 Sobel 运算，得到的是图像的一阶水平方向导数。这步过后，车牌被明显的区分出来。



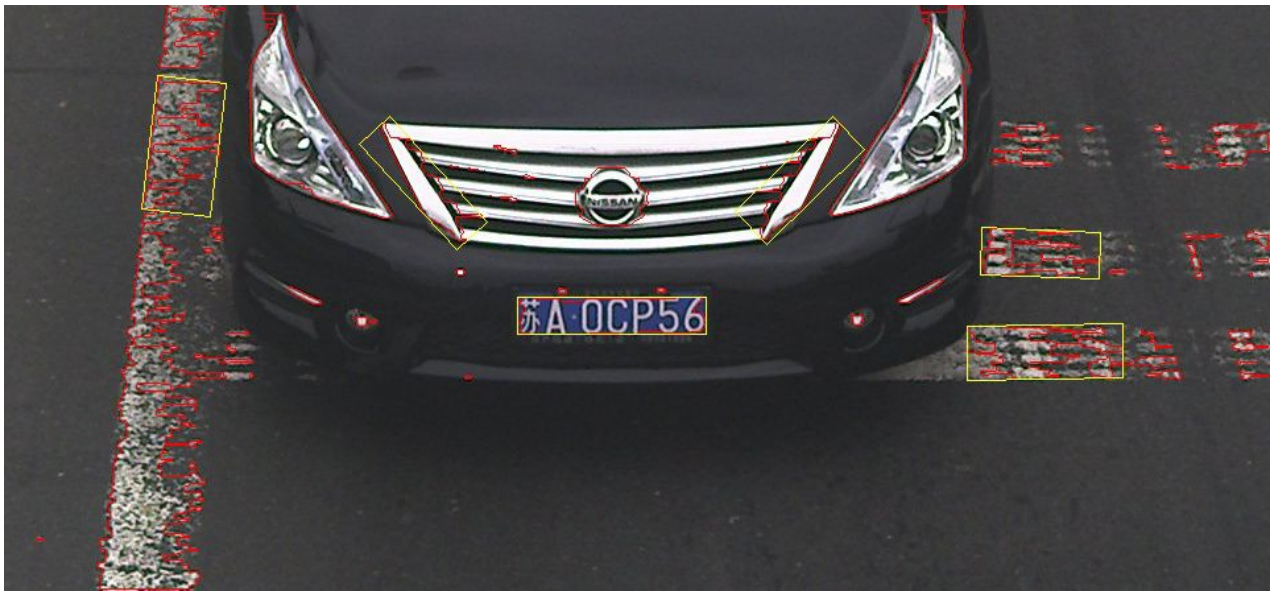
5、对图像进行二值化。将灰度图像（每个像素点有 256 个取值可能）转化为二值图像（每个像素点仅有 1 和 0 两个取值可能）。

6、使用闭操作。对图像进行闭操作以后，可以看到车牌区域被连接成一个矩形装的区域。

7、求轮廓。求出图中所有的轮廓。这个算法会把全图的轮廓都计算出来，因此要进行筛选。



8、筛选。对轮廓求最小外接矩形，然后验证，不满足条件的淘汰。经过这步，仅仅只有六个黄色边框的矩形通过了筛选。



8、角度判断与旋转。把倾斜角度大于阈值（如正负 30 度）的矩形舍弃。左边第一、二、四个矩形被舍弃了。余下的矩形进行微小的旋转，使其水平。

10、统一尺寸。上步得到的图块尺寸是不一样的。为了进入机器学习模型，需要统一尺寸。统一尺寸的标准宽度是 136，长度是 36。这个标准是对千个测试车牌平均后得出的通用值。下图为最终的三个候选”车牌“图块。

这些"车牌"有两个作用：一、积累下来作为支持向量机（SVM）模型的训练集，以此训练出一个车牌判断模型；二、在实际的车牌检测过程中，将这些候选"车牌"交由训练好的车牌判断模型进行判断。如果车牌判断模型认为这是车牌的话就进入下一步即字符识别过程，如果不是，则舍弃。

3. "Plate Locate" 的深入讨论与调优策略

好了，说了这么多，读者想必对整个"Plate Locate"过程已经有了一个完整的认识。那么让我们一步步审核一下处理流程中的每一个步骤。回答下面三个问题:这个步骤的作用是什么？省略这步或者替换这步可不可以？这个步骤中是否有参数可以调优的？通过这几个问题可以帮助我们更好的理解车牌定位功能，并且便于自己做修改、定制。

EasyPR 开发详解（3）车牌定位深度分析（一）

在上篇文章中我们了解了 PlateLocate 的过程中的所有步骤。在本篇文章中我们对前 3 个步骤，分别是高斯模糊、灰度化和 Sobel 算子进行分析。

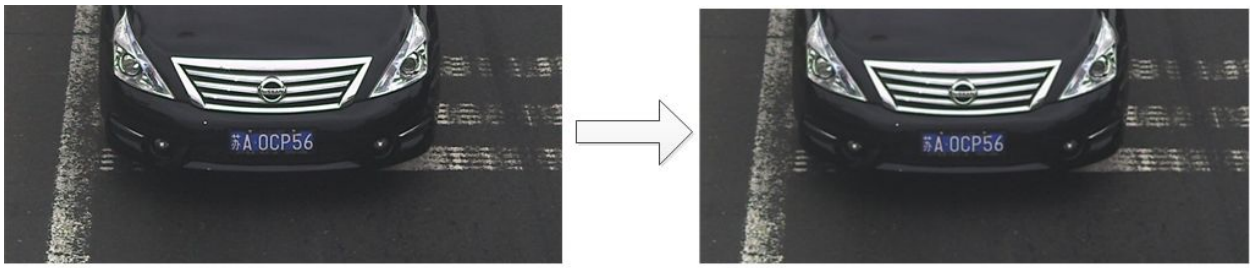
一、高斯模糊

1.目标

对图像去噪，为边缘检测算法做准备。

2. 效果

在我们的车牌定位中的第一步就是高斯模糊处理。



原图片　　高斯模糊后效果

图 1 高斯模糊效果

3. 理论

详细说明可以看这篇:阮一峰讲高斯模糊。

高斯模糊是非常有名的一种图像处理技术。顾名思义，其一般应用是将图像变得模糊，但同时高斯模糊也应用在图像的预处理阶段。理解高斯模糊前，先看一下平均模糊算法。平均模糊的算法非常简单。见下图，每一个像素的值都取周围所有像素（共 8 个）的平均值。

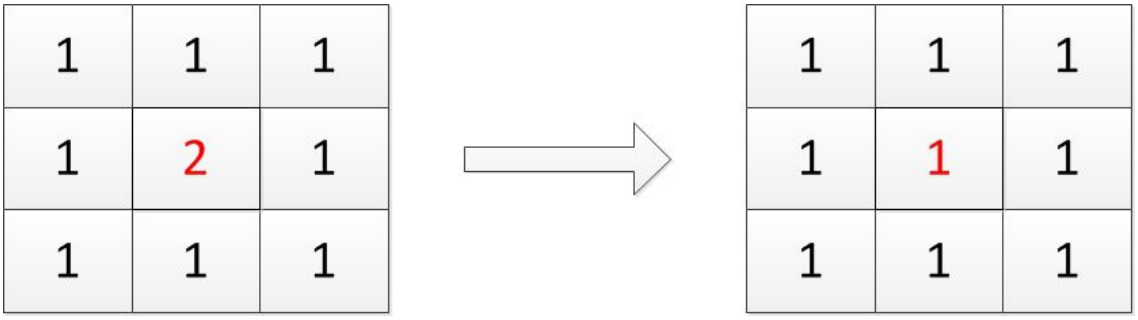


图 2 平均模糊示意图

在上图中，左边红色点的像素值本来是 2，经过模糊后，就成了 1（取周围所有像素的均值）。在平均模糊中，周围像素的权值都是一样的，都是 1。如果周围像素的权值不一样，并且与二维的高斯分布的值一样，那么就叫做高斯模糊。

在上面的模糊过程中，每个像素取的是周围一圈的平均值，也称为模糊半径为 1。如果取周围三圈，则称之为半径为 3。半径增大的话，会更加深模糊的效果。

4. 实践

在 PlateLocate 中是这样调用高斯模糊的。

```
    //高斯模糊。Size 中的数字影响车牌定位的效果。     GaussianBlur( src, src_blur, Size(m_Gaussia
nBlurSize, m_GaussianBlurSize),
        0, 0, BORDER_DEFAULT );
```

其中 Size 字段的参数指定了高斯模糊的半径。值是 CPlateLocate 类的 m_GaussianBlurSize 变量。由于 opencv 的高斯模糊仅接收奇数的半径，因此变量为偶数值会抛出异常。

这里给出了 opencv 的高斯模糊的 API(英文，2.48 以上版本）。

高斯模糊这个过程一定是必要的么。笔者的回答是必要的，倘若我们将这句代码注释并稍作修改，重新运行一下。你会发现 plateLocate 过程在闭操作时就和原来发生了变化。最后结果如下。

图 3 不采用高斯模糊后的结果

可以看出，车牌所在的矩形产生了偏斜。最后得到的候选"车牌"图块如下：

图 4 不采用高斯模糊后的“车牌”图块

如果不使用高斯模糊而直接用边缘检测算法，我们得到的候选"车牌"达到了 8 个！这样不仅会增加车牌判断的处理时间，还增加了判断出错的概率。由于得到的车牌图块中车牌是斜着的，如果我们的字符识别算法需要一个水平的车牌图块，那么几乎肯定我们会无法得到正确的字符识别效果。

高斯模糊中的半径也会给结果带来明显的变化。有的图片，高斯模糊半径过高了，车牌就定位不出来。有的图片，高斯模糊半径偏低了，车牌也定位不出来。因 此、高斯模糊的半径既不宜过高，也不能过低。CPlateLocate 类中的值为 5 的静态常量 DEFAULT_GAUSSIANBLUR_SIZE，标示 着推荐的高斯模糊的半径。这个值是对于近千张图片经过测试后得出的综合定位率最高的一个值。在 CPlateLocate 类的构造函数 中，m_GaussianBlurSize 被赋予了 DEFAULT_GAUSSIANBLUR_SIZE 的值，因此，默认的高斯模糊的半径就是 5。如果不 是特殊情况，不需要修改它。

在数次的实验以后，必须承认，保留高斯模糊过程与半径值为 5 是最佳的实践。为应对特殊需求，在 CPlateLocate 类中也应该提供了方法修改高斯半径的值，调用代码（假设需要一个为 3 的高斯模糊半径）如下：

```
CPlateLocate plate;
plate.setGaussianBlurSize(3);
```

目前 EasyPR 的处理步骤是先进行高斯模糊，再进行灰度化。从目前的实验结果来看，基于色彩的高斯模糊过程比灰度后的高斯模糊过程更容易检测到边缘点。

二、灰度化处理

1.目标

为边缘检测算法准备灰度化环境。

2.效果

灰度化的效果如下。

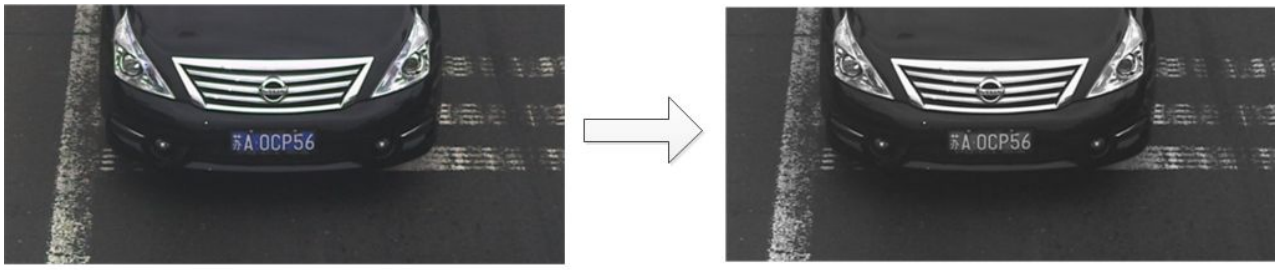


图 5 灰度化效果

3.理论

在灰度化处理步骤中，争议最大的就是信息的损失。无疑的，原先 plateLocate 过程面对的图片是彩色图片，而从这一步以后，就会面对的是灰度图片。在前面，已经说过这步骤是利是弊是需要讨论的。

无疑，对于计算机而言，色彩图像相对于灰度图像难处理多了，很多图像处理算法仅仅只适用于灰度图像，例如后面提到的 Sobel 算子。在这种情况下，你除 了把图片转成灰度图像再进行处理别无它法，除非重新设计算法。但另一方面，转化成灰度图像后恰恰失去了最丰富的细节。要知道，真实世界是彩色的，人类对于 事物的辨别是基于彩色的框架。甚至可以这样说，因为我们的肉眼能够区别彩色，所以我们对于事物的区分，辨别，记忆的能力就非常的强。

车牌定位环节中去掉彩色的利弊也是同理。转换成灰度图像虽然利于使用各种专用的算法，但失去了真实世界中辨别的最重要工具---色彩的区分。举个简单的例 子，人怎么在一张图片中找到车牌？非常简单，一眼望去，一个合适大小的矩形，蓝色的、或者黄色的、或者其他颜色的在另一个黑色，或者白色的大的跟车形类似 的矩形中。这个过程非常直观，明显，而且可以排除模糊，色泽，不清楚等很多影响。如果使用灰度图像，就必须借助水平，垂直求导等方法。

未来如果 PlateLocate 过程可以使用颜色来判断，可能会比现在的定位更清楚、准确。但这需要研究与实验过程，在 EasyPR 的未来版本中可能会 实现。但无疑，使用色彩判断是一种趋势，因为它不仅符合人眼识别的规律，更趋近于人工智能的本质，而且它更准确，速度更快。

4.实践

在 PlateLocate 过程中是这样调用灰度化的。

```
cvtColor( src_blur, src_gray, CV_RGB2GRAY );
```

这里给出了 opencv 的灰度化的 API(英文，2.48 以上版本）。

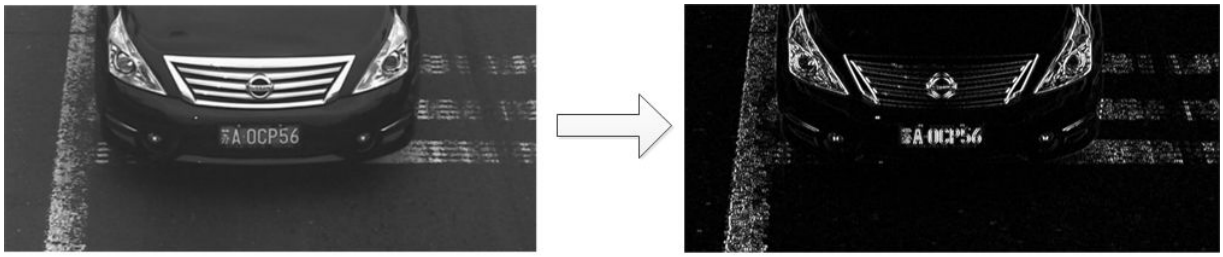
三.Sobel 算子

1.目标

检测图像中的垂直边缘，便于区分车牌。

2.效果

下图是 Sobel 算子的效果。



原图片　　Sobel算子效果

图 6 Sobel 效果

3.理论

如果要说哪个步骤是 plateLocate 中的核心与灵魂，毫无疑问是 Sobel 算子。没有 Sobel 算子，也就没有垂直边缘的检测，也就无法得到车牌 的可能位置，也就没有后面的一系列的车牌判断、字符识别过程。通过 Sobel 算子，可以很方便的得到车牌的一个相对准确的位置，为我们的后续处理打好坚实 的基础。在上面的 plateLocate 的执行过程中可以看到，正是通过 Sobel 算子，将车牌中的字符与车的背景明显区分开来，为后面的二值化与闭操作 打下了基础。那么 Sobel 算子是如何运作的呢？

Soble 算子原理是对图像求一阶的水平与垂直方向导数，根据导数值的大小来判断是否是边缘。请详见 CSDN 小魏的博客（小心她博客里把 Gx 和 Gy 弄反了）。

为了计算方便，Soble 算子并没有真正去求导，而是使用了周边值的加权和的方法，学术上称作“卷积”。权值称为“卷积模板”。例如下图左边就是 Sobel 的 Gx 卷积模板（计算垂直边缘），中间是原图像，右边是经过卷积模板后的新图像。

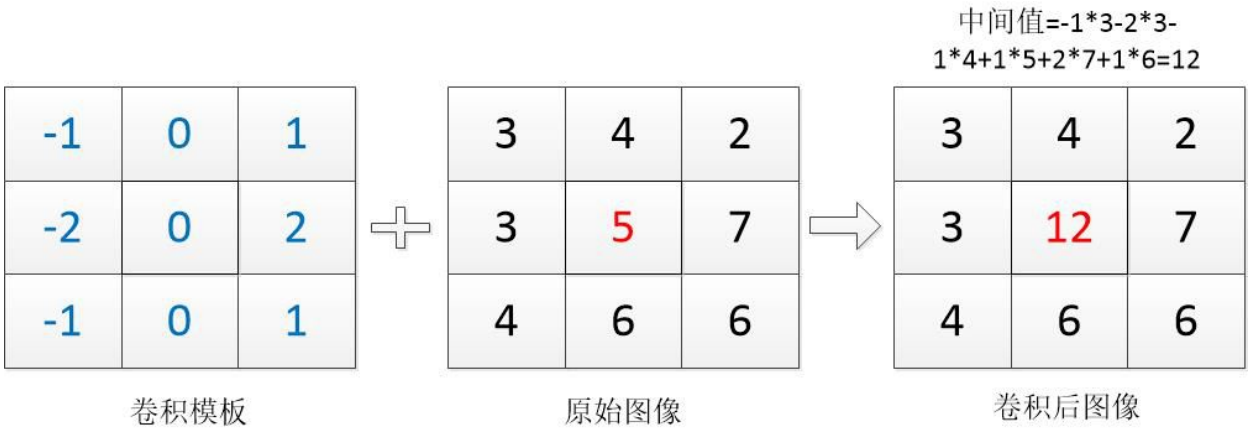


图 7　Sobel 算子 Gx 示意图

在这里演示了通过卷积模板，原始图像红色的像素点原本是 5 的值，经过卷积计算（- 1 * 3 - 2 * 3 - 1 * 4 + 1 * 5 + 2 * 7 + 1 * 6 = 12）后红色像素的值变成了 12。

4.实践

在代码中调用 Soble 算子需要较多的步骤。

```
/// Generate grad_x and grad_y    Mat grad_x, grad_y;
Mat abs_grad_x, abs_grad_y;

/// Gradient X
//Scharr( src_gray, grad_x, ddepth, 1, 0, scale, delta, BORDER_DEFAULT );
Sobel( src_gray, grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT );
convertScaleAbs( grad_x, abs_grad_x );

/// Gradient Y
//Scharr( src_gray, grad_y, ddepth, 0, 1, scale, delta, BORDER_DEFAULT );
Sobel( src_gray, grad_y, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT );
convertScaleAbs( grad_y, abs_grad_y );

/// Total Gradient (approximate)
addWeighted( abs_grad_x, SOBEL_X_WEIGHT, abs_grad_y, SOBEL_Y_WEIGHT, 0, grad );
```

这里给出了 opencv 的 Sobel 的 API(英文，2.48 以上版本）

在调用参数中有两个常量 SOBEL_X_WEIGHT 与 SOBEL_Y_WEIGHT 代表水平方向和垂直方向的权值，默认前者是 1，后者是 0，代表仅仅做水平方向求导，而不做垂直方向求导。这样做的意义是，如果我们做了垂直方向求导，会检测出很多水平边缘。水平边缘多也许有利于生成更精确的轮廓，但是由于有些车子前端太多的水平边缘了，例如车头排气孔，标志等等，很多的水平边缘会误导我们的连接结果，导致我们得不到一个恰好的车牌位置。例如，我们对于测试的图做如下实验，将 SOBEL_X_WEIGHT 与 SOBEL_Y_WEIGHT 都设置为 0.5（代表两者的权值相等），那么最后得到的闭操作后的结果图为

由于 Sobel 算子如此重要，可以将车牌与其他区域明显区分出来，那么问题就来了，有没有与 Sobel 功能类似的算子可以达到一致的效果，或者有没有比 Sobel 效果更好的算子？

Sobel 算子求图像的一阶导数，Laplace 算子则是求图像的二阶导数，在通常情况下，也能检测出边缘，不过 Laplace 算子的检测不分水平和垂直。下图是 Laplace 算子与 Sobel 算子的一个对比。

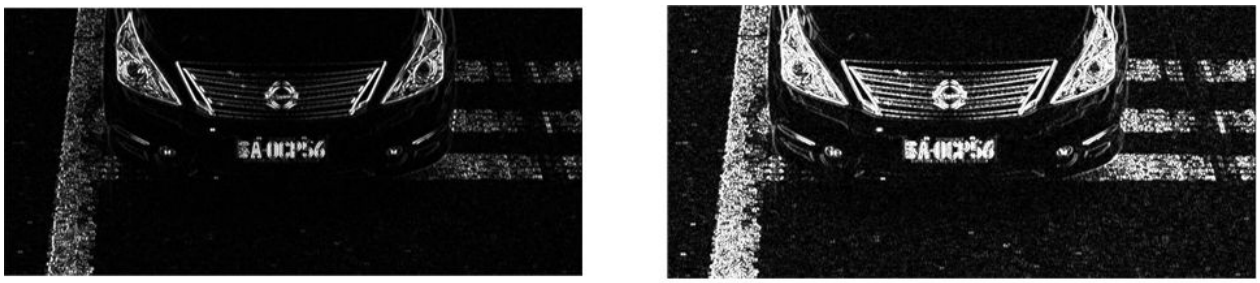
Sobel算子效果

Laplace算子效果

图 8　Sobel 与 Laplace 示意图

可以看出，通过 Laplace 算子的图像包含了水平边缘和垂直边缘，根据我们刚才的描述。水平边缘对于车牌的检测一般无利反而有害。经过对近百幅图像的测试，Sobel 算子的效果优于 Laplace 算子，因此不适宜采用 Laplace 算子替代 Sobel 算子。

除了 Sobel 算子，还有一个算子，Shcarr 算子。但这个算子其实只是 Sobel 算子的一个变种，由于 Sobel 算子在 3*3 的卷积模板上计算往往不太精确，因此有一个特殊的 Sobel 算子，其权值按照下图来表达，称之为 Scharr 算子。下图是 Sobel 算子与 Scharr 算子的一个对比。



Sobel算子效果　　Scharr算子效果

图 9　Sobel 与 Scharr 示意图

一般来说，Scharr 算子能够比 Sobel 算子检测边缘的效果更好，从上图也可以看出。但是，这个"更好"是一把双刃剑。我们的目的并不是画出图像的边缘，而是确定车牌的一个区域，越精细的边缘越会干扰后面的闭运算。因此，针对大量的图片的测试，Sobel 算子一般都优于 Scharr 算子。

关于 Sobel 算子更详细的解释和 Scharr 算子与 Sobel 算子的同异，可以参看官网的介绍：Sobel 与 Scharr。

综上所述，在求图像边缘的过程中，Sobel 算子是一个最佳的契合车牌定位需求的算子，Laplace 算子与 Scharr 算子的效果都不如它。

有一点要说明的：Sobel 算子仅能对灰度图像有效果，不能将色彩图像作为输入。因此在进行 Soble 算子前必须进行前面的灰度化工作。

EasyPR 开发详解（4）车牌定位深度分析（二）

在上一篇深度分析与调优讨论中，我们介绍了高斯模糊，灰度化和 Sobel 算子。在本文中，会分析剩余的定位步骤。

根据前文的内容，车牌定位的功能还剩下如下的步骤，见下图中未涂灰的部分。

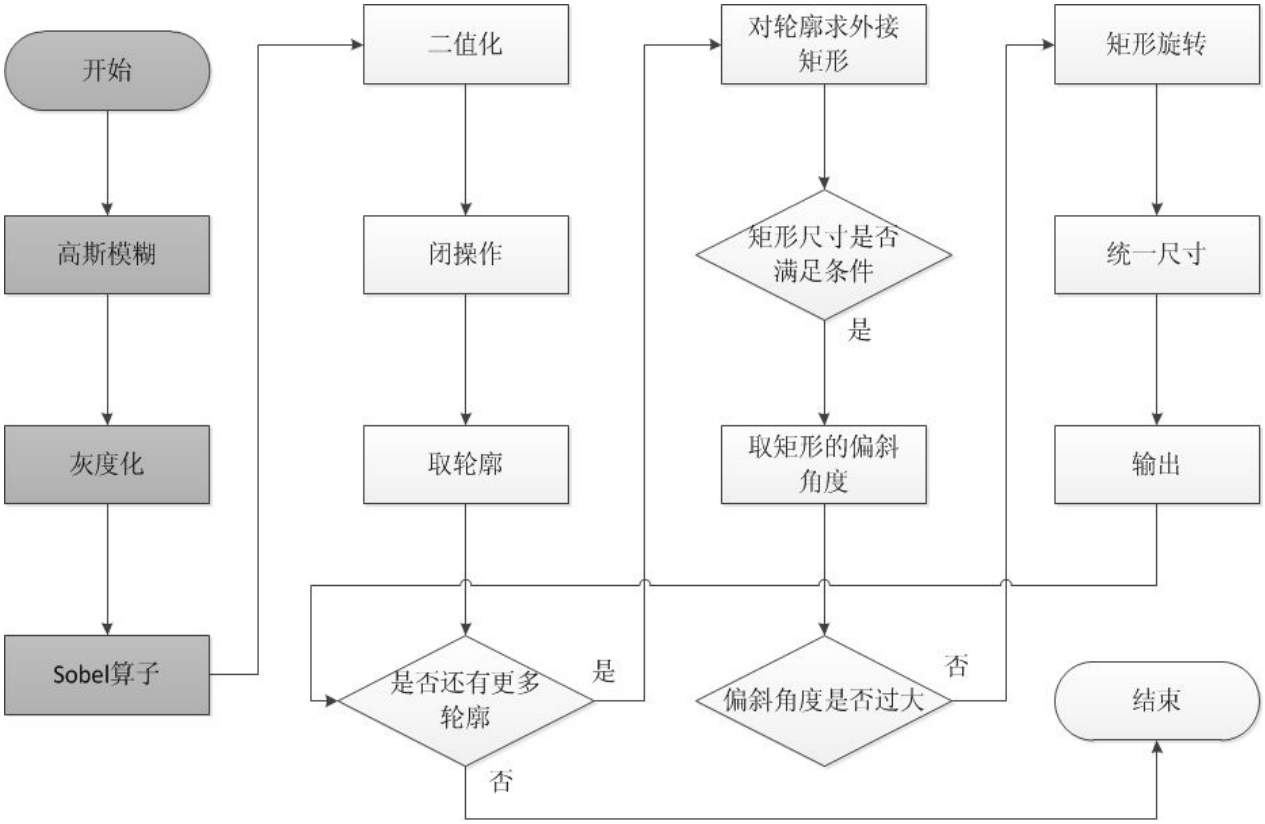


图1 车牌定位步骤

我们首先从 Soble 算子分析出来的边缘来看。通过下图可见，Sobel 算子有很强的区分性，车牌中的字符被清晰的描绘出来，那么如何根据这些信息定位出车牌的位置呢？

图2 Sobel 后效果

我们的车牌定位功能做了个假设，即车牌是包含字符图块的一个最小的外接矩形。在大部分车牌处理中，这个假设都能工作的很好。我们来看下这个假设是如何工作的。首先，我们通过二值化处理将 Sobel 生成的灰度图像转变为二值图像。

四.二值化

二值化算法非常简单，就是对图像的每个像素做一个阈值处理。

1.目标

为后续的形态学算子 Morph 等准备二值化的图像。

2.效果

经过二值化处理后的图像效果为下图，与灰度图像仔细区分下，二值化图像中的白色是没有颜色强与暗的区别的。

图3 二值化后效果

3.理论

在灰度图像中，每个像素的值是0-255之间的数字，代表灰暗的程度。如果设定一个阈值 T，规定像素的值 x 满足如下条件时则：

if x < t then x = 0; if x >= t then x = 1。

如此一来，每个像素的值仅有{0,1}两种取值，0代表黑、1代表白，图像就被转换成了二值化的图像。在上面的公式中，阈值 T 应该取多少？由于不同图像 的光造程度不同，导致作为二值化区分的阈值 T 也不一样。因此一个简单的做法是直接使用 opencv 的二值化函数时加上自适应阈值参数。如下：

```
threshold(src, dest, 0, 255, CV_THRESH_OTSU+CV_THRESH_BINARY);
```

通过这种方法，我们不需要计算阈值的取值，直接使用即可。

threshold 函数是二值化函数，参数 src 代表源图像，dest 代表目标图像，两者的类型都是 cv::Mat 型，最后的参数代表二值化时的选项，
CV_THRESH_OTSU 代表自适应阈值，CV_THRESH_BINARY 代表正二值化。正二值化意味着像素的值越接近0，越可能被赋值为0，反之则为1。而另外一种二值化方法表示反二值化，其含义是像素的值越接近0，越可能被赋值1,，计算公式如下：

if x < t then x = 1; if x >= t then x = 0,

如果想使用反二值化，可以使用参数 CV_THRESH_BINARY_INV 代替 CV_THRESH_BINARY 即可。在后面的字符识别中我们会同时 使用到正二值化与反二值化两种例子。因为中国的车牌有很多类型，最常见的是蓝牌和黄牌。其中蓝牌字符浅，背景深，黄牌则是字符深，背景浅，因此需要正二值 化方法与反二值化两种方法来处理，其中正二值化处理蓝牌，反二值化处理黄牌。

五.闭操作

闭操作是个非常重要的操作，我会花很多的字数与图片介绍它。

1. 目标

将车牌字母连接成为一个连通域，便于取轮廓。

2. 效果

我们这里看下经过闭操作后图像连接的效果。

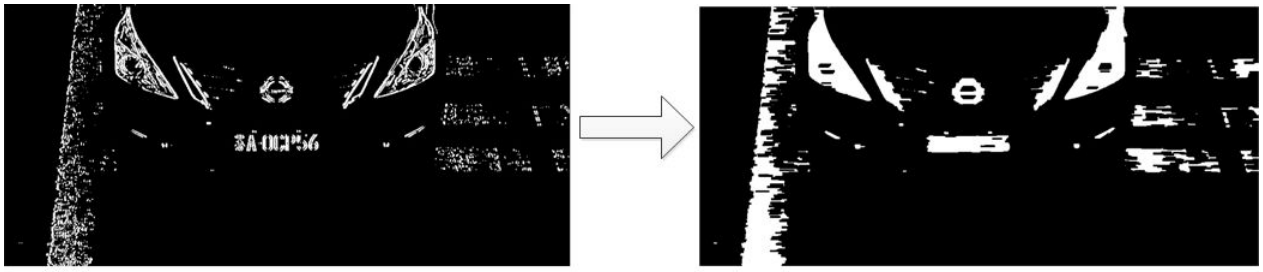


图4 闭操作后效果

3. 理论

在做闭操作的说明前，必须简单介绍一下腐蚀和膨胀两个操作。

在图像处理技术中，有一些的操作会对图像的形态发生改变，这些操作一般称之为形态学操作。形态学操作的对象是二值化图像。
有名的形态学操作中包括腐蚀，膨胀，开操作，闭操作等。其中腐蚀，膨胀是许多形态学操作的基础。

腐蚀操作：
顾名思义，是将物体的边缘加以腐蚀。具体的操作方法是拿一个宽 m,高 n 的矩形作为模板，对图像中的每一个像素 x 做如下处理：像素 x 至于模板的中心，根据 模版的大小，遍历所有被模板覆盖的其他像素，修改像素 x 的值为所有像素中最小的值。这样操作的结果是会将图像外围的突出点加以腐蚀。如下图的操作过程：

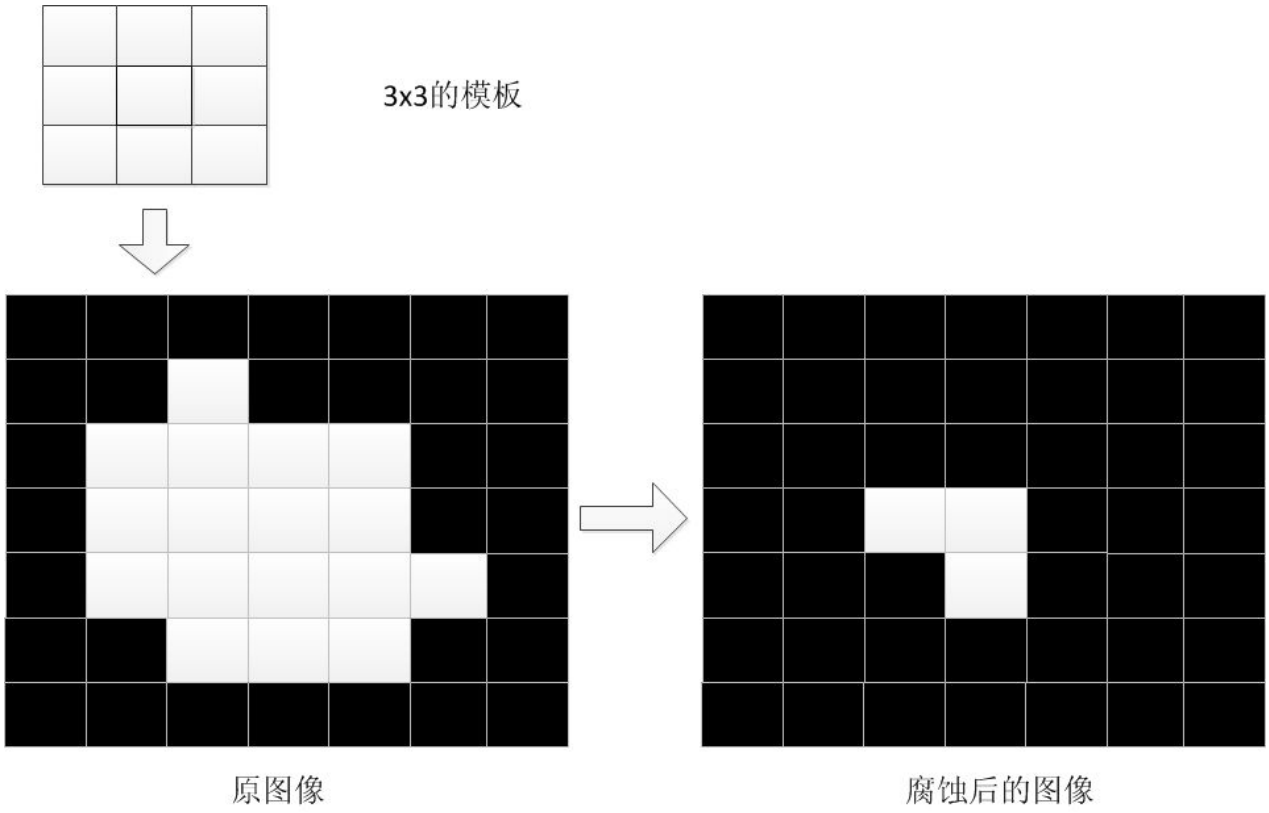
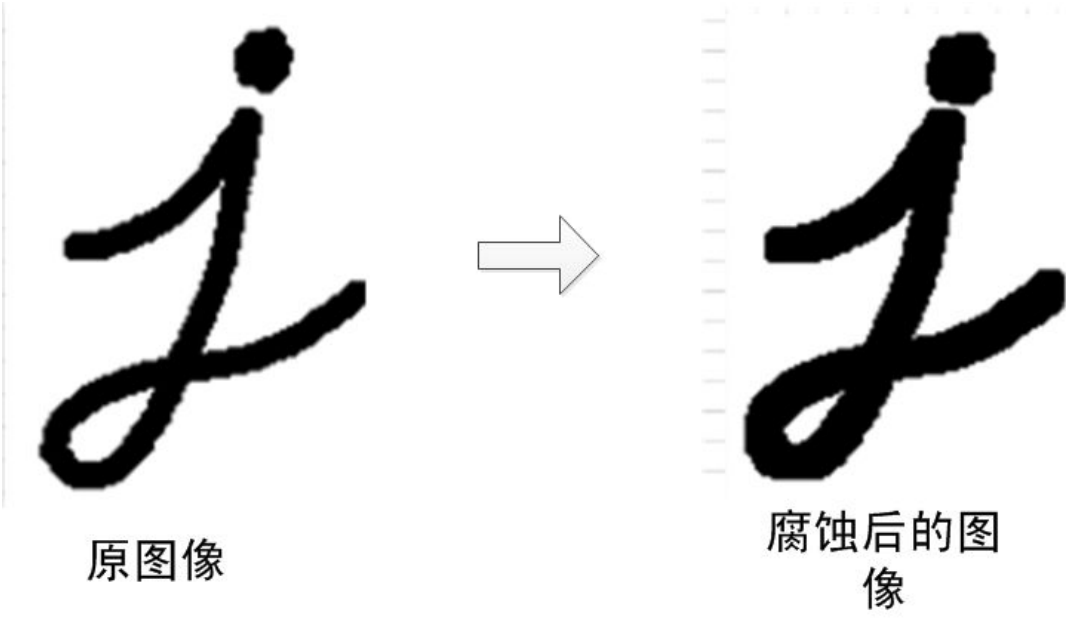


图5 腐蚀操作原理

上图演示的过程是背景为黑色，物体为白色的情况。腐蚀将白色物体的表面加以“腐蚀”。在 opencv 的官方教程中，是以如下的图示说明腐蚀过程的，与我 上面图的区别在于：背景是白色，而物体为黑色（这个不太符合一般的情况，所以我没有拿这张图作为通用的例子）。读者只需要了解背景为不同颜色时腐蚀也是不 同的效果就可以了。



腐蚀的结果是：亮区(背景)变细，而黑色区域(字母)则变大了。

注意这张图像中的字母为黑色，背景为白色，而不是一般意义的背景为黑色，前景为白色。

图6 腐蚀操作原理2

膨胀操作：

膨胀操作与腐蚀操作相反，是将图像的轮廓加以膨胀。操作方法与腐蚀操作类似，也是拿一个矩形模板，对图像的每个像素做遍历处理。不同之处在于修改像素的值不是所有像素中最小的值，而是最大的值。这样操作的结果会将图像外围的突出点连接并向外延伸。如下图的操作过程：

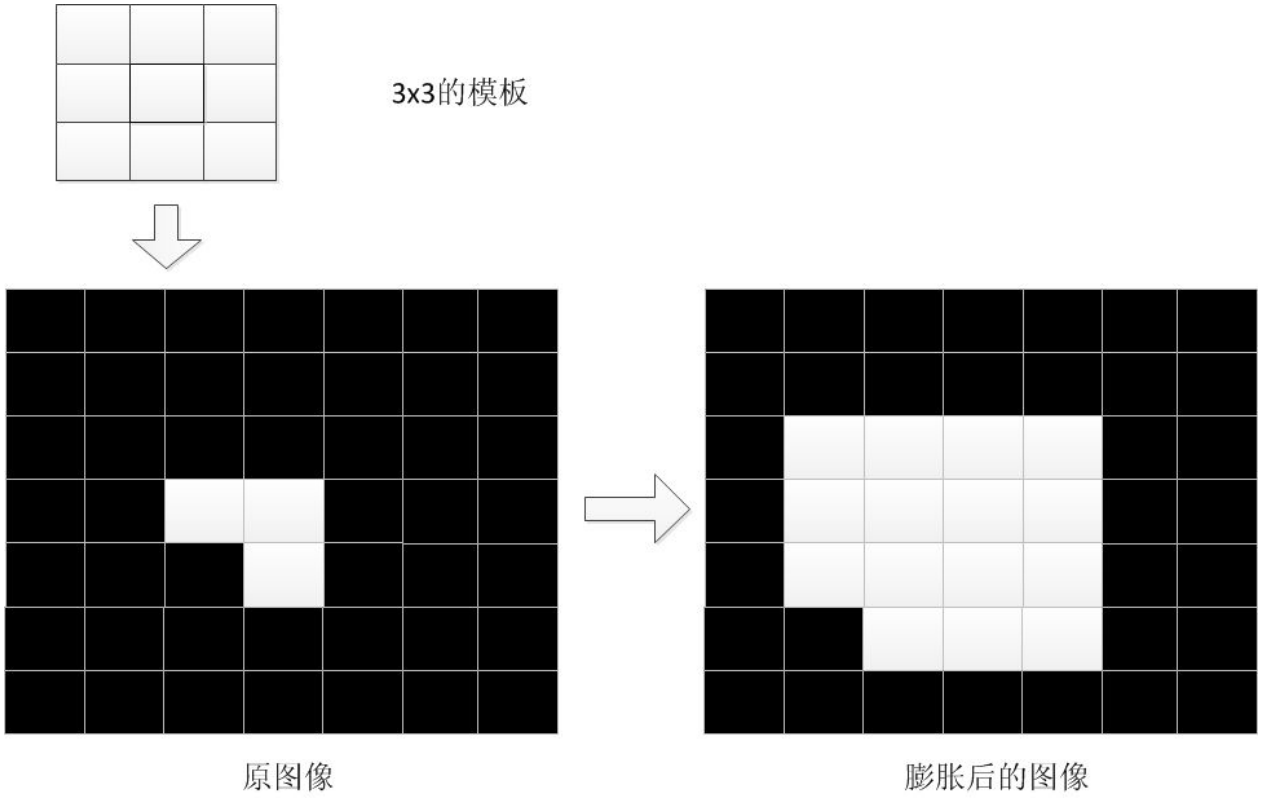
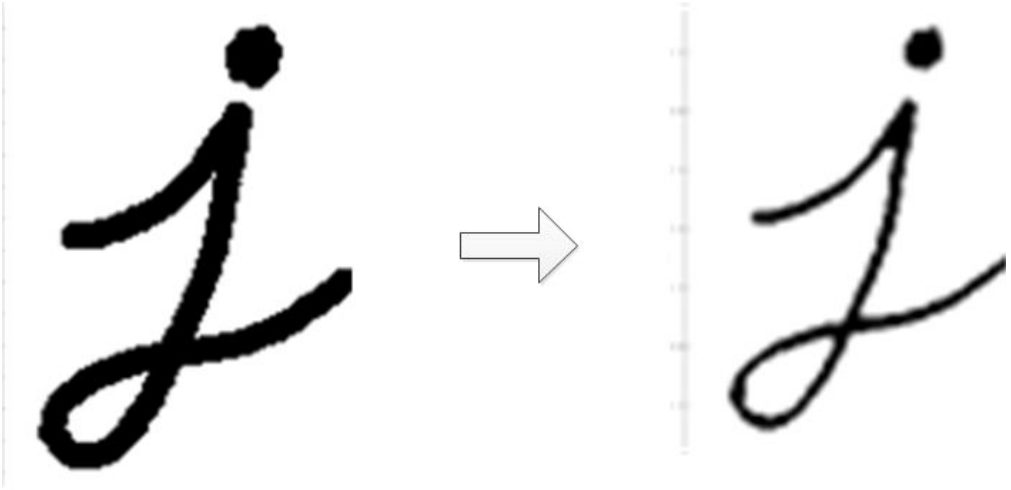


图7 膨胀操作原理

下面是在 opencv 的官方教程中，膨胀过程的图示：



膨胀的结果是：背景(白色)膨胀，而黑色字母缩小了。

注意这张图像中的字母为黑色，背景为白色，而不是一般意义的背景为黑色，前景为白色。

图8 膨胀操作原理2

开操作：

开操作就是对图像先腐蚀，再膨胀。其中腐蚀与膨胀使用的模板是一样大小的。为了说明开操作的效果，请看下图的操作过程：

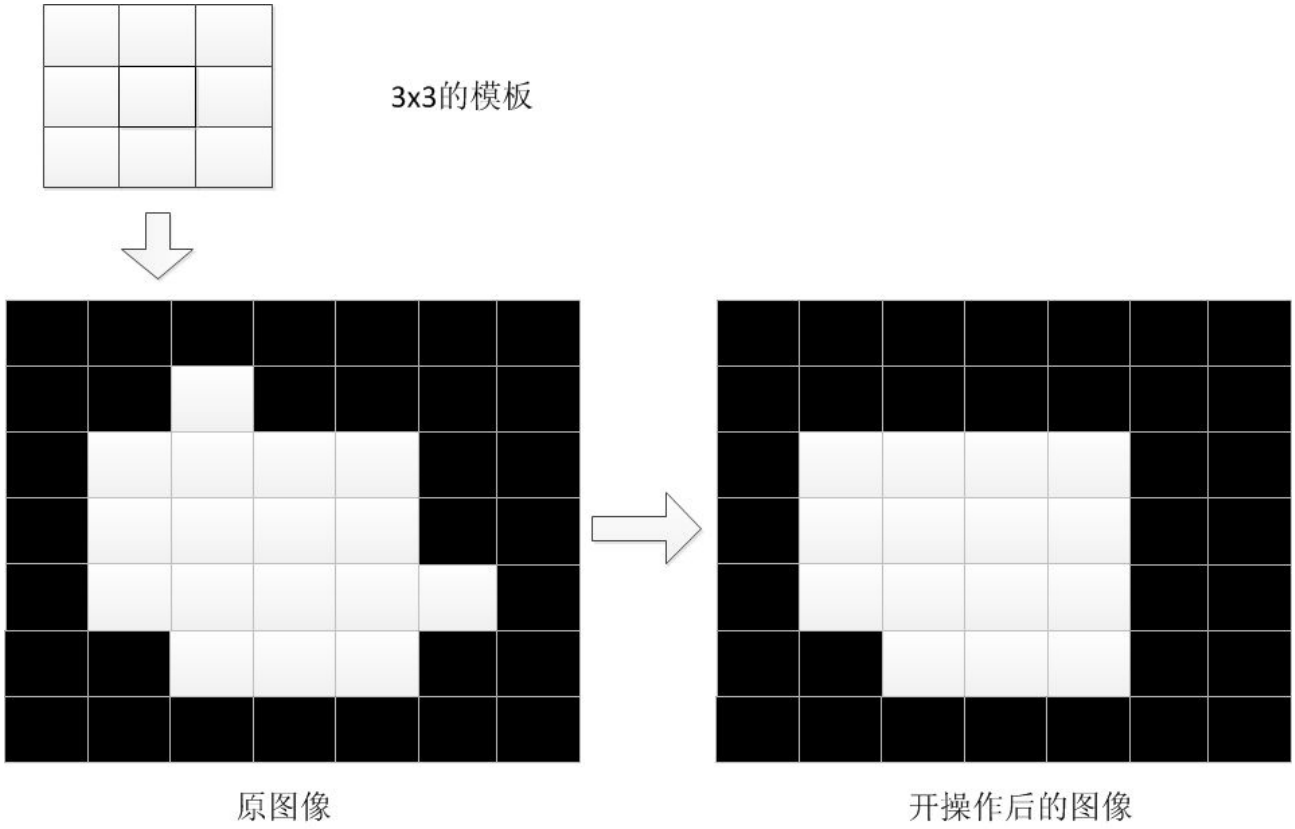


图9 开操作原理

由于开操作是先腐蚀，再膨胀。因此可以结合图5和图7得出图9，其中图5的输出是图7的输入，所以开操作的结果也就是图7的结果。

闭操作：

闭操作就是对图像先膨胀，再腐蚀。闭操作的结果一般是可以将许多靠近的图块相连称为一个无突起的连通域。在我们的图像定位中，使用了闭操作去连接所有的 字符小图块，然后形成一个车牌的大致轮廓。闭操作的过程我会讲的细致一点。

为了说明字符图块连接的过程。在这里选取的原图跟上面三个操作的原图不大一样， 是一个由两个分开的图块组成的图。原图首先经过膨胀操作，将两个分开的图块结合起来（注意我用偏白的灰色图块表示由于膨胀操作而产生的新的白色）。接着通 过腐蚀操作，将连通域的边缘和突起进行削平（注意我用偏黑的灰色图块表示由于腐蚀被侵蚀成黑色图块）。最后得到的是一个无突起的连通域（纯白的部分）。

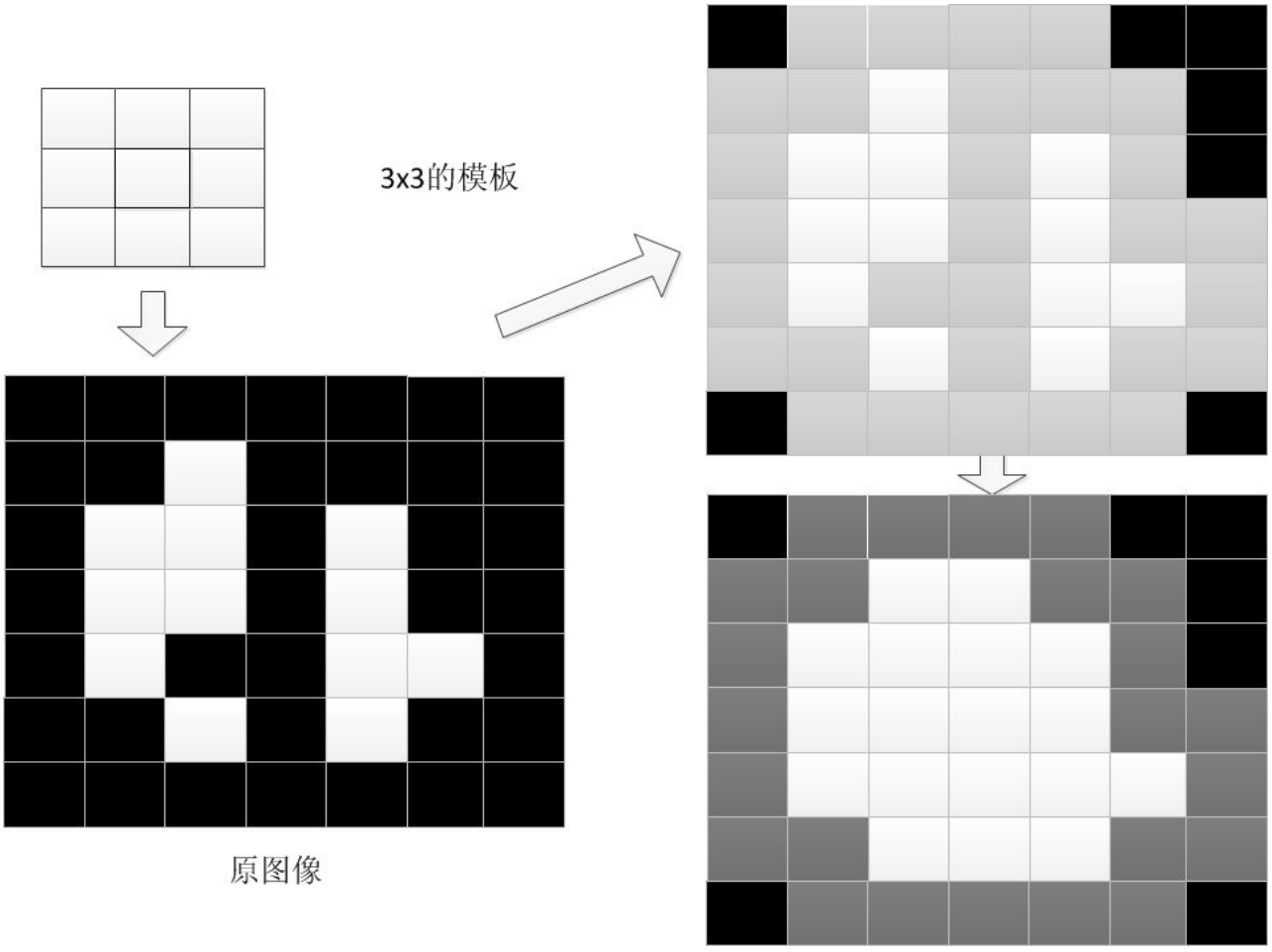


图10 闭操作原理

4. 代码

在 opencv 中，调用闭操作的方法是首先建立矩形模板，矩形的大小是可以设置的，由于矩形是用来覆盖以中心像素的所有其他像素，因此矩形的宽和高最好是奇数。

通过以下代码设置矩形的宽和高。

```
Mat element = getStructuringElement(MORPH_RECT, Size(m_MorphSizeWidth,
m_MorphSizeHeight) );
```

在这里，我们使用了类成员变量，这两个类成员变量在构造函数中被赋予了初始值。宽是17，高是3.

设置完矩形的宽和高以后，就可以调用形态学操作了。opencv 中所有形态学操作有一个统一的函数，通过参数来区分不同的具体操作。例如 MOP_CLOSE 代表闭操作，MOP_OPEN 代表开操作。

```
morphologyEx(img_threshold, img_threshold, MORPH_CLOSE, element);
```

如果我对二值化的图像进行开操作，结果会是什么样的？下图是图像使用闭操作与开操作处理后的一个区别：

闭操作后效果

开操作后效果

图11 开与闭的对比

晕，怎么开操作后图像没了？原因是：开操作第一步腐蚀的效果太强，直接导致接下来的膨胀操作几乎没有效果，所以图像就变几乎没了。

可以看出，使用闭操作以后，车牌字符的图块被连接成了一个较为规则的矩形，通过闭操作，将车牌中的字符连成了一个图块，同时将突出的部分进行裁剪，图块 成为了一个类似于矩形的不规则图块。我们知道，车牌应该是一个规则的矩形，因此获取规则矩形的办法就是先取轮廓，再接着求最小外接矩形。

这里需要注意的是，矩形模板的宽度，17是个推荐值，低于17都不推荐。

为什么这么说，因为有一个”断节“的问题。中国车牌有一个特点，就是表示城市的字母与右边相邻的字符距离远大于其他相邻字符之间的距离。如果你设置的不够大，结果导致左边的字符与右边的字符中间断开了，如下图：

图12 “断节”效果

这种情况我称之为“断节”如果你不想字符从中间被分成"苏 A"和"7EUK22"的话，那么就必须把它设置大点。

另外还有一种讨厌的情况，就是右边的字符第一个为1的情况，例如苏 B13GH7。在这种情况下，由于1的字符的形态原因，导致跟左边的 B 的字符的距离更 远，在这种情况下，低于17都有很大的可能性会断节。下图说明了矩形模板宽度过小时（例如设置为7）面对不同车牌情况下的效果。其中第二个例子选取了苏 E 开头的车牌，由于 E 在 Sobel 算子运算过后仅存有左边的竖杠，因此也会导致跟右边的字符相距过远的情况！

图13 “断节”发生示意

宽度过大也是不好的，因为它会导致闭操作连接不该连接的部分，例如下图的情况。

图14 矩形模板宽度过大

这种情况下，你取轮廓获得矩形肯定会大于你设置的校验规则，即便通过校验了，由于图块中有不少不是车牌的部分，会给字符识别带来麻烦。

因此，矩形的宽度是一个需要非常细心权衡的值，过大过小都不好，取决于你的环境。至于矩形的高度，3是一个较好的值，一般来说都能工作的很好，不需要改变。

记得我在前一篇文章中提到，工业用图片与生活场景下图片的区别么。笔者做了一个实验，下载了30多张左右的百度车牌图片。用 plateLocate 过程去识别他们。如果按照下面的方式设置参数，可以保证90%以上的定位成功率。

```
CPlateLocate plate;

plate.setDebug(1);

plate.setGaussianBlurSize(5);

plate.setMorphSizeWidth(7);

plate.setMorphSizeHeight(3);

plate.setVerifyError(0.9);

plate.setVerifyAspect(4);

plate.setVerifyMin(1);
```

```
plate.setVerifyMax(30);
```

在 EasyPR 的下一个版本中，会增加对于生活场景下图片的一个模式。只要选择这个模式，就适用于百度图片这种日常生活抓拍图片的效果。但是，仍然有一些图片是 EasyPR 不好处理的。或者可以说，按照目前的边缘检测算法，难以处理的。

请看下面一张图片：

图15 难以权衡的一张图片

这张图片最麻烦的地方在于车牌左右两侧凹下去的边侧，这个边缘在 Sobel 算子中非常明显，如果矩形模板过长，很容易跟它们连接起来。更麻烦的是这个车 牌属于上面说的“断节”很容易发生的类型，因为车牌右侧字符的第一个字母是“1”，这个导致如果矩形模板过短，则很容易车牌断成两截。结果最后导致了如下 的情况。

如果我设置矩形模板宽度为12，则会发生下面的情况：

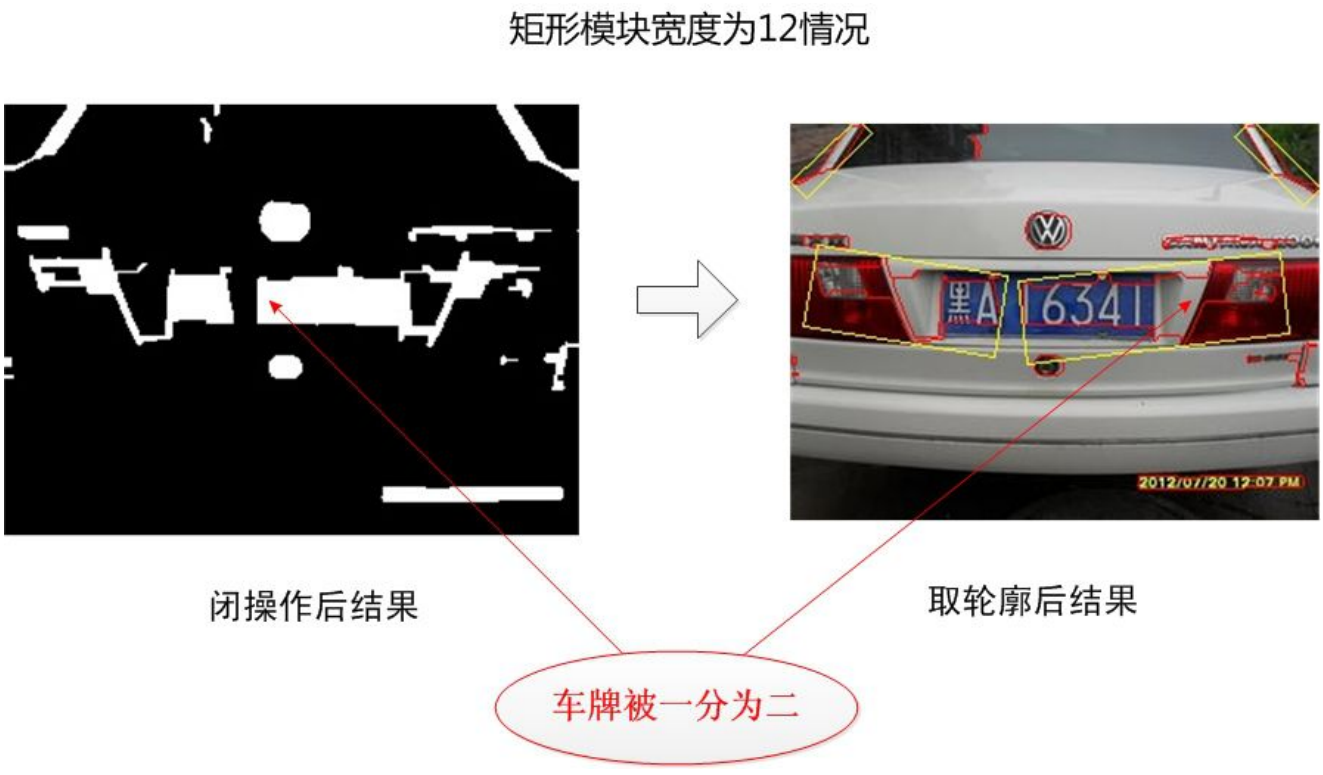


图16 车牌被一分为二

如果我增加矩形模板宽度到13，则又会发生下面的情况。

图17 车牌区域被不不正确的放大

因此矩形模板的宽度是个整数值，在12和13中间没有中间值。这个导致几乎没有办法处理这幅车牌图像。

上面的情况属于车尾车牌的一种没办法解决的情况。下面所说的情况属于车头的情况，相比前者，错误检测的几率高的多！为什么，因为是一类型车牌无法处理。要问我这家车是哪家，我只能说：碰到开奥迪 Q5及其系列的，早点嫁了吧。伤不起。

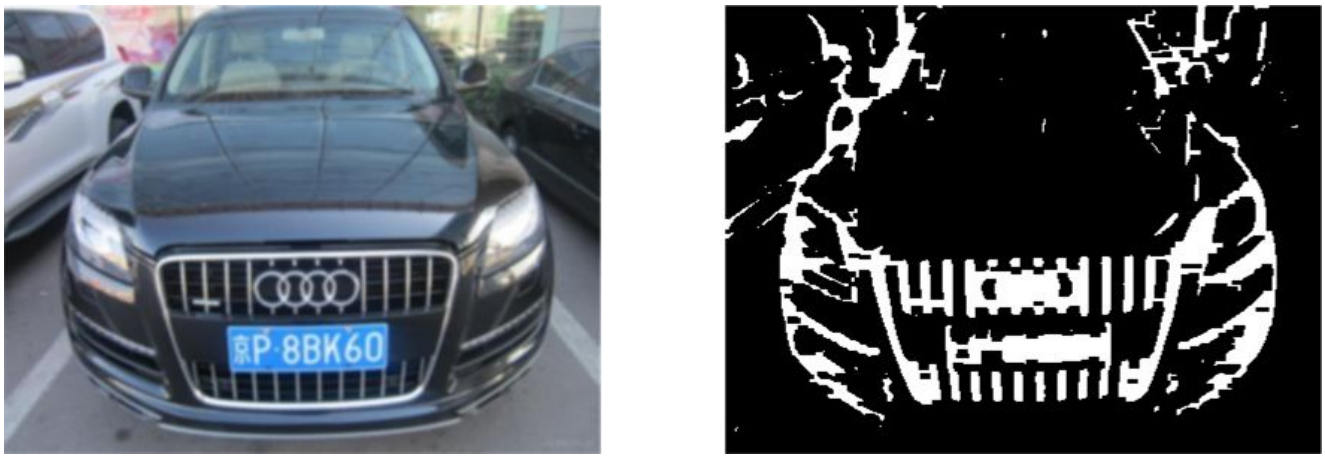


图18 奥迪 Q5前部垂直边缘太多

这么多的垂直边缘，极为容易检错。已经试过了，几乎没有办法处理这种车牌。只能替换边缘检测这种思路，采用颜色区分等方法。奥体 Q 系列前脸太多垂直边缘了，给跪。

六. 取轮廓

取轮廓操作是个相对简单的操作，因此只做简短的介绍。

1. 目标

将连通域的外围勾画出来，便于形成外接矩形。

2. 效果

我们这里看下经过取轮廓操作的效果。

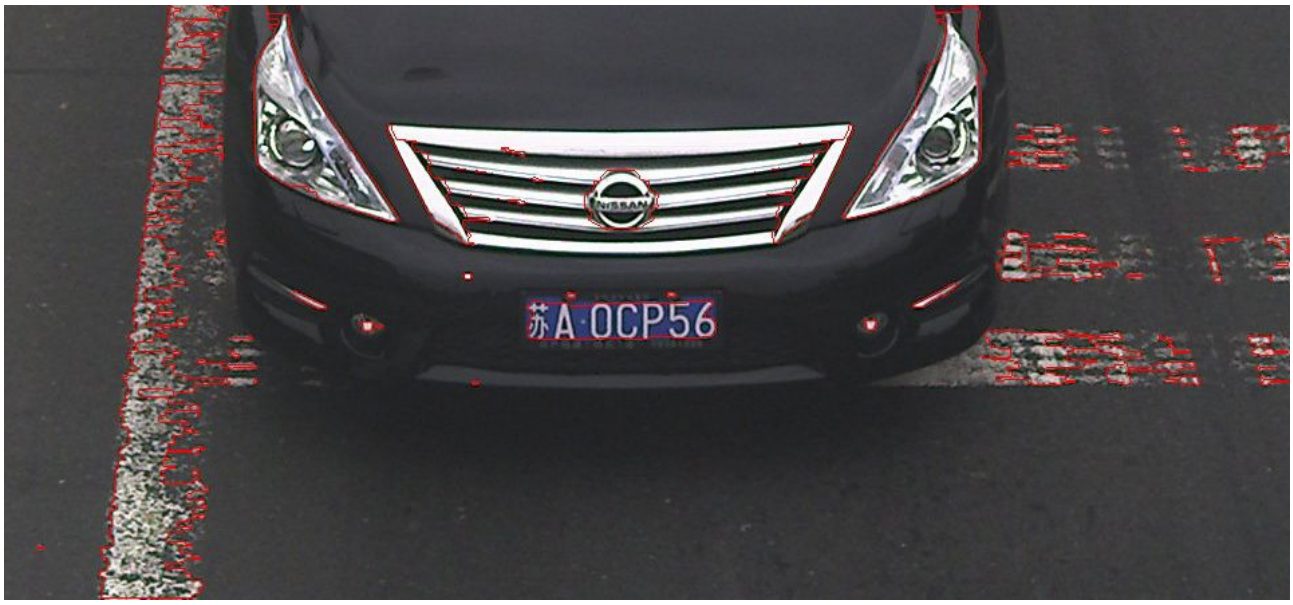


图19 取轮廓操作

在图中，红色的线条就是轮廓，可以看到，有非常多的轮廓。取轮廓操作就是将图像中的所有独立的不与外界有交接的图块取出来。然后根据这些轮廓，求这些轮 廓的最小外接矩形。这里面需要注意的是这里用的矩形是 RotatedRect，意思是可旋转的。因此我们得到的矩形不是水平的，这样就为处理倾斜的车牌打 下了基础。

取轮廓操作的代码如下：

```
1     vector< vector< Point> > contours;
2    findContours(img_threshold,
3          contours, // a vector of contours
4          CV_RETR_EXTERNAL, // 提取外部轮廓
5          CV_CHAIN_APPROX_NONE); // all pixels of each contours
```

七. 尺寸判断

尺寸判断操作是对外接矩形进行判断，以判断它们是否是可能的候选车牌的操作。

1. 目标

排除不可能是车牌的矩形。

2. 效果

经过尺寸判断，会排除大量由轮廓生成的不合适尺寸的最小外接矩形。效果如下图：

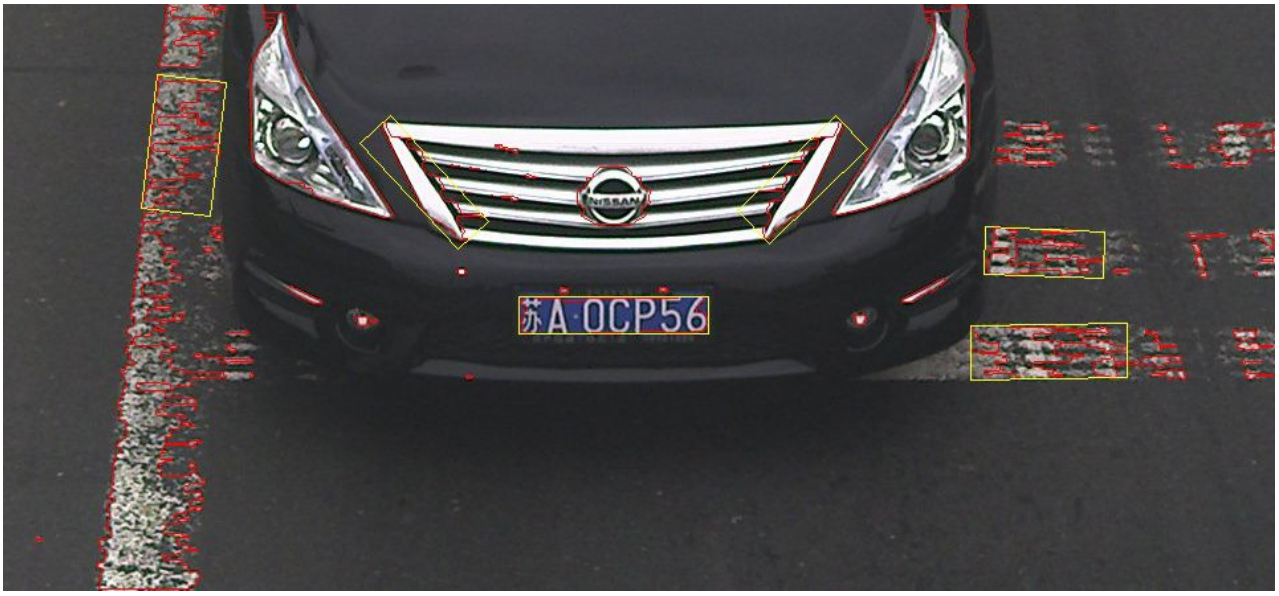


图20 尺寸判断操作

通过对图像中所有的轮廓的外接矩形进行遍历，我们调用 CplateLocate 的另一个成员方法 verifySizes，代码如下：

⊞View Code

在原先的 verifySizes 方法中，使用的是针对西班牙车牌的检测。而我们的系统需要检测的是中国的车牌。因此需要对中国的车牌大小有一个认识。

中国车牌的一般大小是440mm*140mm，面积为440*140，宽高比为3.14。verifySizes 使用如下方法判断矩形是否是车牌：

1. 设立一个偏差率 error，根据这个偏差率计算最大和最小的宽高比 rmax、rmin。判断矩形的 r 是否满足在 rmax、rmin 之间。
2. 设定一个面积最大值 max 与面积最小值 min。判断矩形的面积 area 是否满足在 max 与 min 之间。

以上两个条件必须同时满足，任何一个不满足都代表这不是车牌。

偏差率和面积最大值、最小值都可以通过参数设置进行修改，且他们都有一个默认值。如果发现 verifySizes 方法无法发现你图中的车牌，试着修改这些参数。

另外，verifySizes 方法是可选的。你也可以不进行 verifySizes 直接处理，但是这会大大加重后面的车牌判断的压力。一般来说，合理的 verifySizes 能够去除90%不合适的矩形。

八. 角度判断

角度判断操作通过角度进一步排除一部分车牌。

1. 目标

排除不可能是车牌的矩形。

通过 verifySizes 的矩形，还必须进行一个筛选，即角度判断。一般来说，在一副图片中，车牌不太会有非常大的倾斜，我们做如下规定：如果一个矩形的偏斜角度大于某个角度（例如30度），则认为不是车牌并舍弃。

对上面的尺寸判断结果的六个黄色矩形应用角度判断后结果如下图：

图21 角度判断后的候选车牌

可以看出，原先的6个候选矩形只剩3个。车牌两侧的车灯的矩形被成功筛选出来。角度判断会去除 verifySizes 筛选余下的7%矩形，使得最终进入车牌判断环节的矩形只有原先的全部矩形的3%。

角度判断以及接下来的旋转操作的代码如下：

⊞View Code

九. 旋转

旋转操作是为后面的车牌判断与字符识别提高成功率的关键环节。

1. 目标

旋转操作将偏斜的车牌调整为水平。

2. 效果

假设待处理的图片如下图：

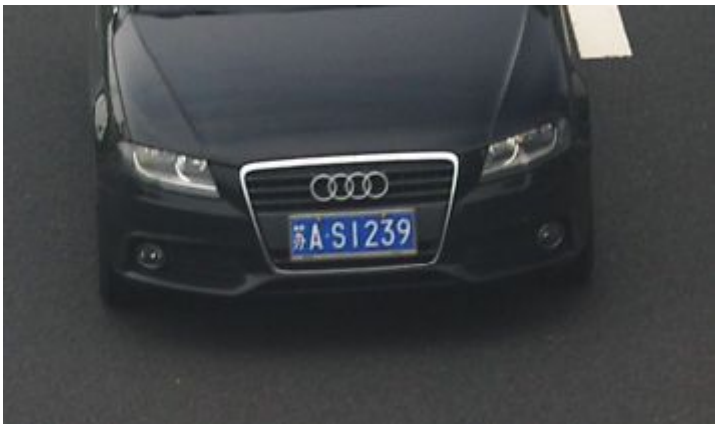


图22 倾斜的车牌

使用旋转与不适用旋转的效果区别如下图：

图23 旋转的效果

可以看出，没有旋转操作的车牌是倾斜，加大了后续车牌判断与字符识别的难度。因此最好需要对车牌进行旋转。

在角度判定阈值内的车牌矩形，我们会根据它偏转的角度进行一个旋转，保证最后得到的矩形是水平的。调用的 opencv 函数如下：

```
1                Mat rotmat = getRotationMatrix2D(minRect.center, angle, 1);

2               Mat img_rotated;

3                warpAffine(src, img_rotated, rotmat, src.size(), CV_INTER_CUBIC);
```

这个调用使用了一个旋转矩阵，属于几何代数内容，在这里不做详细解释。

十.大小调整

结束了么？不，还没有，至少在我们把这些候选车牌导入机器学习模型之前，需要确保他们的尺寸一致。

机器学习模型在预测的时候，是通过模型输入的特征来判断的。我们的车牌判断模型的特征是所有的像素的值组成的矩阵。因此，如果候选车牌的尺寸不一致，就无法被机器学习模型处理。因此需要用 resize

方法进行调整。

我们将车牌 resize 为宽度136，高度36的矩形。为什么用这个值？这个值一开始也不是确定的，我试过许多值。最后我将近千张候选车牌做了一个统计，取它们的平均宽度与高度，因此就有了136和36这个值。所以，这个是一个统计值，平均来说，这个值的效果最好。

大小调整调用了 CplateLocate 的最后一个成员方法 showResultMat，代码很简单，贴下，不做细讲了。

View Code

十一.总结

通过接近10多个步骤的处理，我们才有了最终的候选车牌。这些过程是一环套一环的，前步骤的输出是后步骤的输入，而且顺序也是有规则的。目前针对我的测 试图片来说，它们工作的很好，但不一定适用于你的情况。车牌定位以及图像处理算法的一个大的问题就是他的弱鲁棒性，换一个场景可能就得换一套工作方式。因 此结合你的使用场景来做调整吧，这是我为什么要在这里费这么多字数详细说明的原因。如果你不了解细节，你就不可能进行修改，也就无法使它适合你的工作需 求。

讨论：

车牌定位全部步骤了解后，我们来讨论下。这个过程是否是一个最优的解？

毫无疑问，一个算法的好坏 除了取决于它的设计思路，还取决于它是否充分利用了已知的信息。如果一个算法没有充分利用提供的信息，那么它就有进一步优化的空间。EasyPR 的 plateLocate 过程就是如此，在实施过程中它相继抛弃掉了色彩信息，没有利用纹理信息，因此车牌定位的过程应该还有优化的空间。如果 plateLocate 过程无法良好的解决你的定位问题，那么尝试下能够利用其他信息的方法，也许你会大幅度提高你的定位成功率。

车牌定位讲完后，下面就是机器学习的过程。不同于前者，我不会重点说明其中的细节，而是会概括性的说明每个步骤的用途以及训练的最佳实践。在下一个章节中，我会首先介绍下什么是机器学习，为什么它如今这么火热，机器学习和大数据的关系，欢迎继续阅读。

EasyPR 开发详解（5）颜色定位与偏斜扭正

本篇文章介绍 EasyPR 里新的定位功能：颜色定位与偏斜扭正。希望这篇文档可以帮助开发者与使用者更好的理解 EasyPR 的设计思想。

让我们先看一下示例图片，这幅图片中的车牌通过颜色的定位法进行定位并从偏斜的视角中扭正为正视角（请看右图的左上角）。

图 1 新版本的定位效果

下面内容会对这两个特性的实现过程展开具体的介绍。首先介绍颜色定位的原理，然后是偏斜扭正的实现细节。

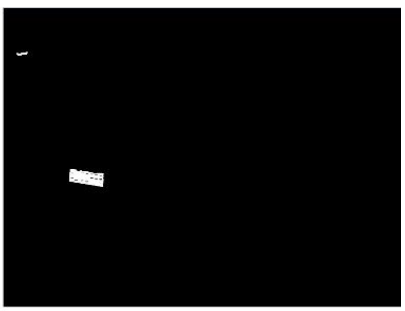
一. 颜色定位

1. 起源

在前面的介绍里，我们使用了 Sobel 查找垂直边缘的方法，成功定位了许多车牌。但是，Sobel 法最大的问题就在于面对垂直边缘交错的情况下，无法准确地定位车牌。例如下图。为了解决这个问题，可以考虑使用颜色信息进行定位。

原始图像

Sobel定位效果
定位失败



颜色定位效果
非常精准

图 2 颜色定位与 Sobel 定位的比较

如果将颜色定位与 Sobel 定位加以结合的话，可以使车牌的定位准确率从 75%上升到 94%。

2. 方法

关于颜色定位首先我们想到的解决方案就是：利用 RGB 值来判断。

这个想法听起来很自然：如果我们想找出一幅图像中的蓝色部分，那么我们只需要检查 RGB 分量（RGB 分量由 Red 分量--红色，Green 分量--绿色，Blue 分量--蓝色共同组成）中的 Blue 分量就可以了。一般来说，Blue 分量是个 0 到 255 的值。如果我们设定一个阈值，并且检查每个像素的 Blue 分量是否大于它，那我们不就可以得知这些像素是不是蓝色的了么？这个想法虽然很好，不过存在一个问题，我们该怎么来选择这个阈值？这是第一个问题。

即便我们用一些方法决定了阈值以后，那么下面的一个问题就会让人抓狂，颜色是组合的，即便蓝色属性在 255（这样已经很'蓝'了吧），只要另外两个分量配合（例如都为 255），你最后得到的不是蓝色，而是黑色。

这还只是区分蓝色的问题，黄色更麻烦，它是由红色和绿色组合而成的，这意味着你需要考虑两个变量的配比问题。这些问题让选择 RGB 颜色作为判断的难度大到难以接受的地步。因此必须另想办法。

为了解决各种颜色相关的问题，人们发明了各种颜色模型。其中有一个模型，非常适合解决颜色判断的问题。这个模型就是 HSV 模型。

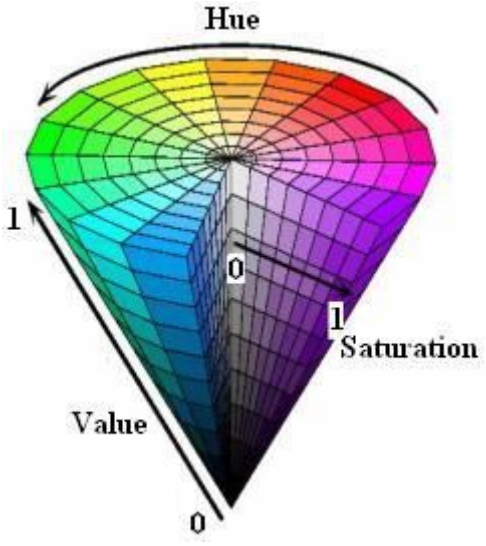


图 3 HSV 颜色模型

HSV 模型是根据颜色的直观特性创建的一种圆锥模型。与 RGB 颜色模型中的每个分量都代表一种颜色不同的是，HSV 模型中每个分量并不代表一种颜色，而分别是：色调（H），饱和度（S），亮度（V）。

H 分量是代表颜色特性的分量，用角度度量，取值范围为 0～360，从红色开始按逆时针方向计算，红色为 0，绿色为 120，蓝色为 240。S 分量代表颜色的饱和信息，取值范围为 0.0～1.0，值越大，颜色越饱和。V 分量代表明暗信息，取值范围为 0.0～1.0，值越大，色彩越明亮。

H 分量是 HSV 模型中唯一跟颜色本质相关的分量。只要固定了 H 的值，并且保持 S 和 V 分量不太小，那么表现的颜色就会基本固定。为了判断蓝色车牌颜色的范围，可以固定了 S 和 V 两个值为 1 以后，调整 H 的值，然后看颜色的变化范围。通过一段摸索，可以发现当 H 的取值范围在 200 到 280 时，这些颜色都可以被认为是蓝色车牌的颜色范畴。于是我们可以用 H 分量是否在 200 与 280 之间来决定某个像素

是否属于蓝色车牌。黄色车牌也是一样的道理，通过观察，可以发现当 H 值在 30 到 80 时，颜色的值可以作为黄色车牌的颜色。

这里的颜色表来自于这个网站。

下图显示了蓝色的 H 分量变化范围。

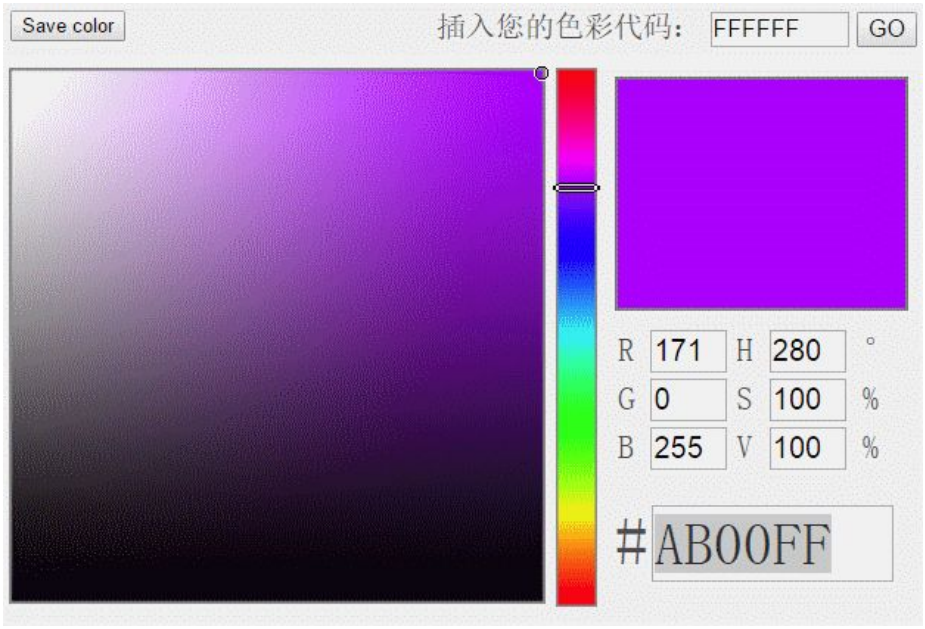


图 4 蓝色的 H 分量区间

下图显示了黄色的 H 分量变化范围。

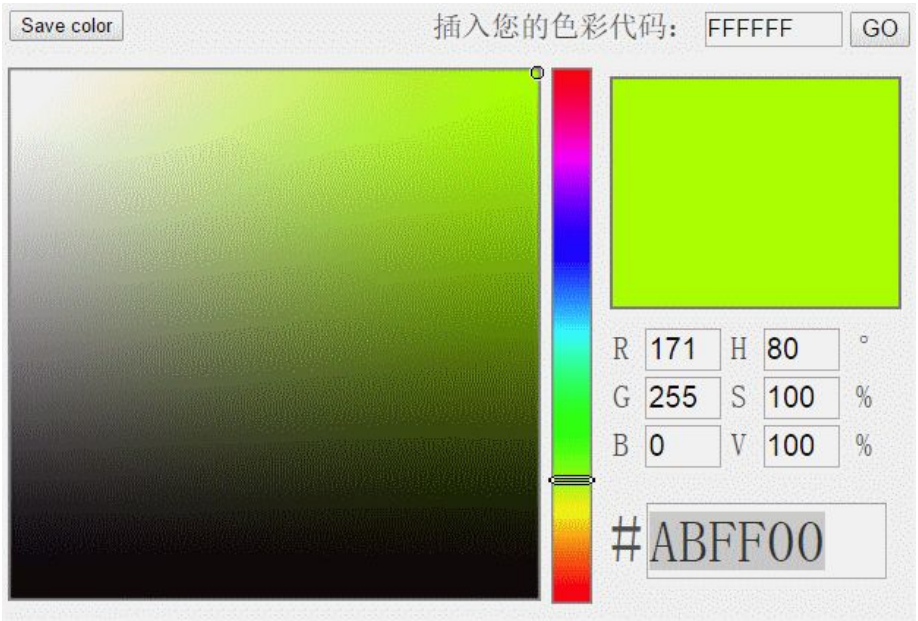


图 5 黄色的 H 分量区间

光判断 H 分量的值是否就足够了？

事实上是不足的。固定了 H 的值以后，如果移动 V 和 S 会带来颜色的饱和度和亮度的变化。当 V 和 S 都达到最高值，也就是 1 时，颜色是最纯正的。降低 S，颜色越发趋向于变白。降低 V，颜色趋向于变黑，当 V 为 0 时，颜色变为黑色。因此，S 和 V 的值也会影响最终颜色的效果。

我们可以设置一个阈值，假设 S 和 V 都大于阈值时，颜色才属于 H 所表达的颜色。

在 EasyPR 里，这个值是 0.35，也就是 V 属于 0.35 到 1 且 S 属于 0.35 到 1 的一个范围，类似于一个矩形。对 V 和 S 的阈值判断是有必要的，因为很多车牌周身的车身，都是 H 分量属于 200-280，而 V 分量或者 S 分量小于 0.35 的。通过 S 和 V 的判断可以排除车牌周围车身的干扰。

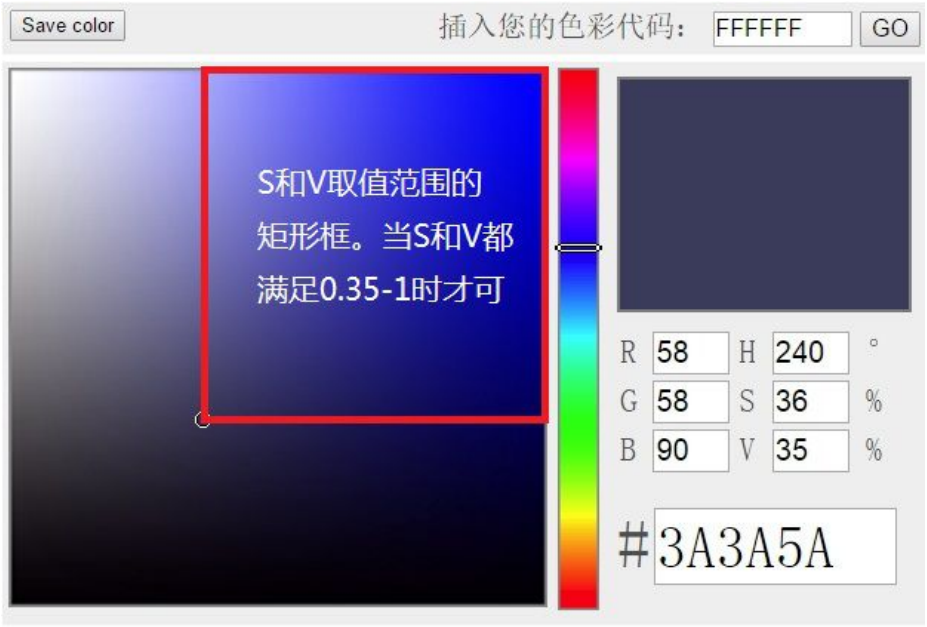


图 6 V 和 S 的区间

明确了使用 HSV 模型以及用阈值进行判断以后，下面就是一个颜色定位的完整过程。

第一步，将图像的颜色空间从 RGB 转为 HSV，在这里由于光照的影响，对于图像使用直方图均衡进行预处理；

第二步，依次遍历图像的所有像素，当 H 值落在 200-280 之间并且 S 值与 V 值也落在 0.35-1.0 之间，标记为白色像素，否则为黑色像素；

第三步，对仅有白黑两个颜色的二值图参照原先车牌定位中的方法，使用闭操作，取轮廓等方法将车牌的外接矩形截取出来做进一步的处理。

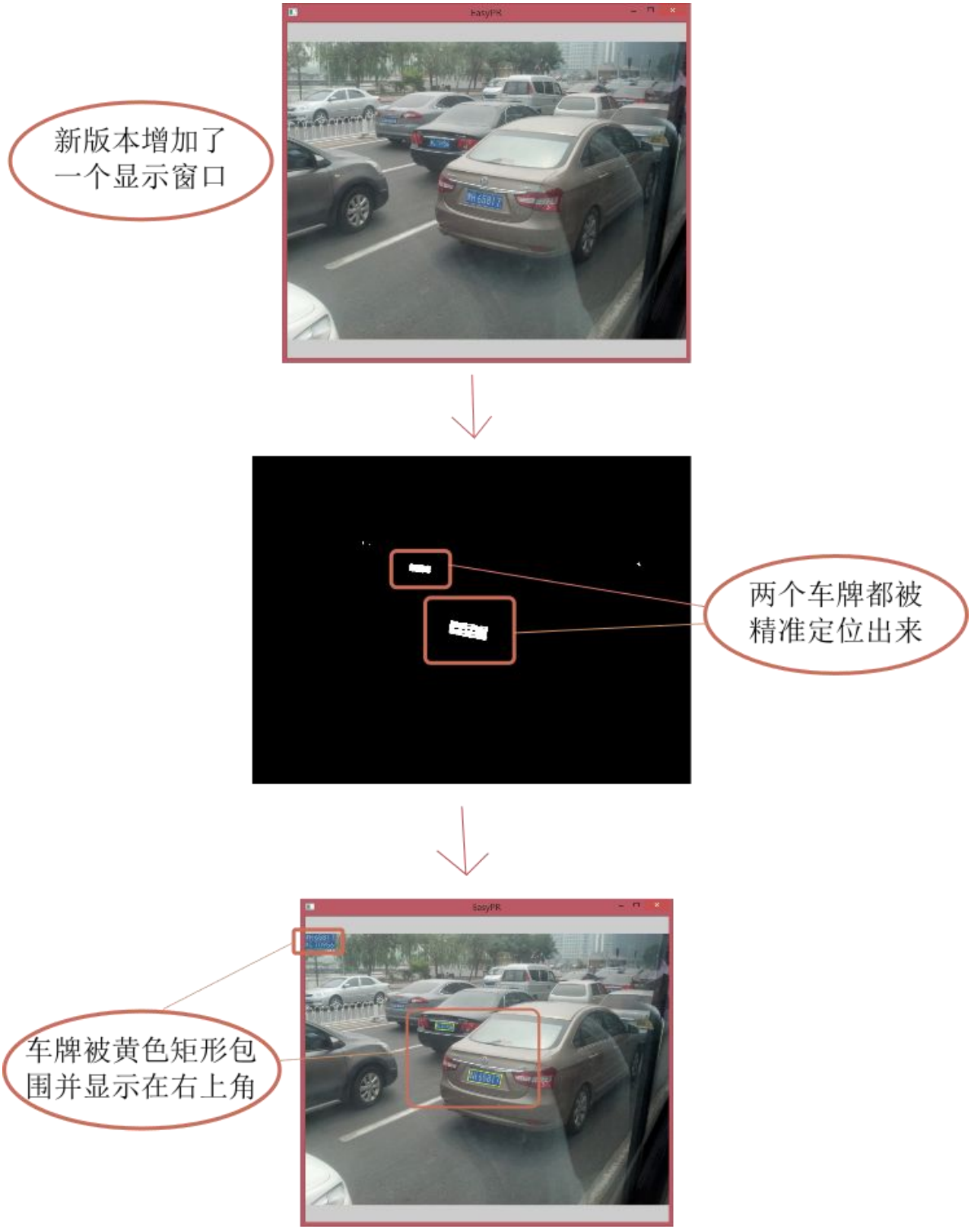


图 7 蓝色定位效果

以上就完成了一个蓝色车牌的定位过程。我们把对图像中蓝色车牌的寻找过程称为一次与蓝色模板的匹配过程。代码中的函数称之为 colorMatch。一般说来，一幅图像需要进行一次蓝色模板的匹配，还要进行一次黄色模板的匹配，以此确保蓝色和黄色的车牌都被定位出来。

黄色车牌的定位方法与其类似，仅仅只是 H 阈值范围的不同。事实上，黄色定位的效果一般好的出奇，可以在非常复杂的环境下将车牌极为准确的定位出来，这可能源于现实世界中黄色非常醒目的原因。

图 8 黄色定位效果

从实际效果来看，颜色定位的效果是很好的。在通用数据测试集里，大约 70%的车牌都可以被定位出来（一些颜色定位不了的，我们可以用 Sobel 定位处理）。

在代码中有些细节需要注意：

一. opencv 为了保证 HSV 三个分量都落在 0-255 之间（确保一个 char 能装的下），对 H 分量除以了 2，也就是 0-180 的范围，S 和 V 分量乘以了 255，将 0-1 的范围扩展到 0-255。我们在设置阈值的时候需要参照 opencv 的标准，因此对参数要进行一个转换。

二. 是 v 和 s 取值的问题。对于暗的图来说，取值过大容易漏，而对于亮的图，取值过小则容易跟车身混淆。因此可以考虑最适应的改变阈值。

三. 是模板问题。目前的做法是针对蓝色和黄色的匹配使用了两个模板，而不是统一的模板。统一模板的问题在于担心蓝色和黄色的干扰问题，例如黄色的车与蓝色的牌的干扰，或者蓝色的车和黄色牌的干扰，这里面最典型的例子就是一个带有蓝色车牌的黄色出租车，在很多城市里这已经是“标准配置”。因此需要将蓝色和黄色的匹配分别用不同的模板处理。

了解完这三个细节以后，下面就是代码部分。

```
View Code
```

3. 不足

以上说明了颜色定位的设计思想与细节。那么颜色定位是不是就是万能的？答案是否定的。在色彩充足，光照足够的情况下，颜色定位的效果很好，但是在面对光线不足的情况，或者蓝色车身的情况时，颜色定位的效果很糟糕。下图是一辆蓝色车辆，可以看出，车牌与车身内容完全重叠，无法分割。

图 9 失效的颜色定位

碰到失效的颜色定位情况时需要使用原先的 Sobel 定位法。

目前的新版本使用了颜色定位与 Sobel 定位结合的方式。首先进行颜色定位，然后根据条件使用 Sobel 进行再次定位，增加整个系统的适应能力。

为了加强鲁棒性，Sobel 定位法可以用两阶段的查找。也就是在已经被 Sobel 定位的图块中，再进行一次 Sobel 定位。这样可以增加准确率，但会降低了速度。一个折衷的方案是让用户决定一个参数

m_maxPlates 的值，这个值决定了你在一幅图里最多定位多少车牌。系统首先用颜色定位出候选车牌，然后通过 SVM 模型来判断是否是车牌，最后统计数量。如果这个数量大于你设定的参数，则认为车牌已经定位足够了，不需要后一步处理，也就不会进行两阶段的 Sobel 查找。相反，如果这个数量不足，则继续进行 Sobel 定位。

综合定位的代码位于 CPlateDectec 中的的成员函数 plateDetectDeep 中，以下是 plateDetectDeep 的整体流程。

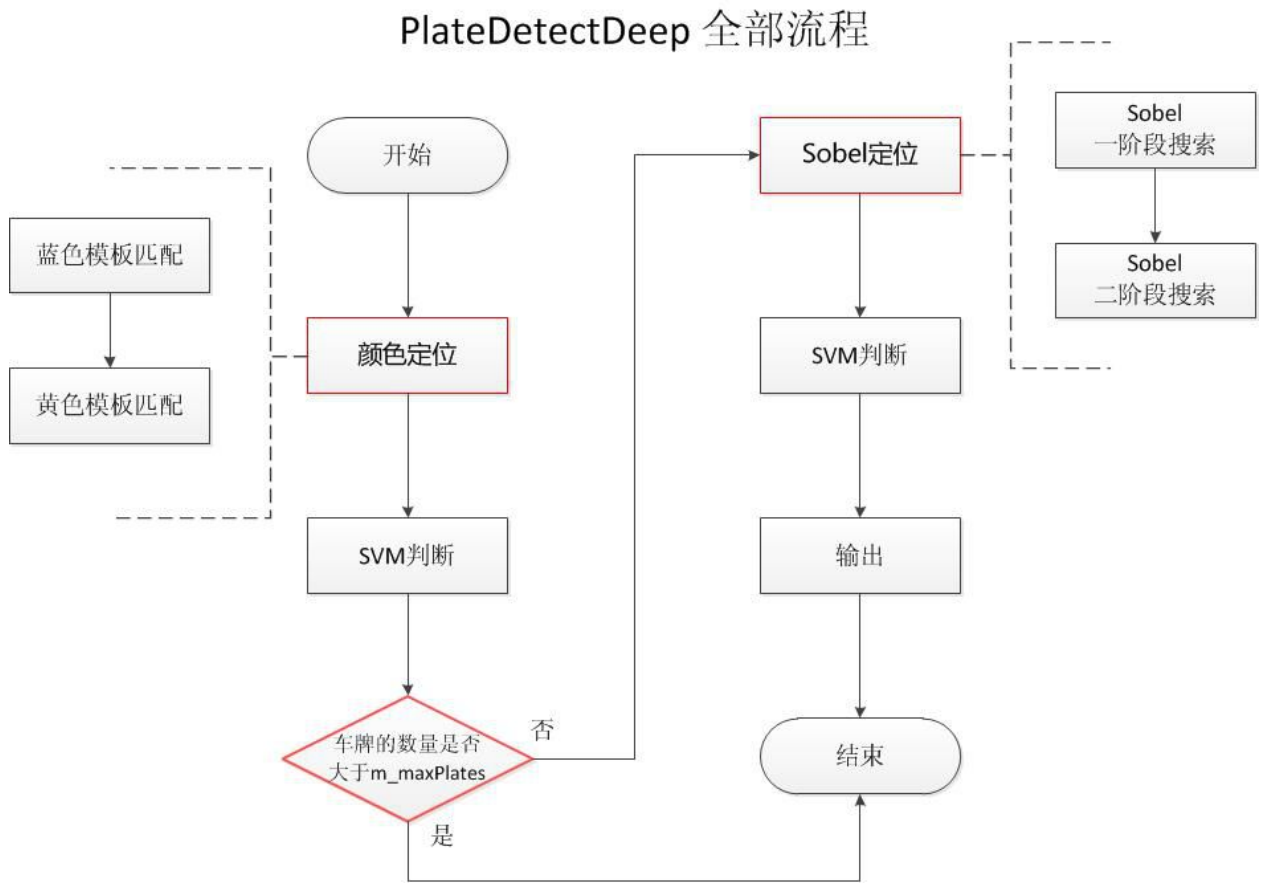


图 10 综合定位全部流程

有没有颜色定位与 Sobel 定位都失效的情况？有的。这种情况下可能需要使用第三类定位技术--字符定位技术。这是 EasyPR 发展的一个方向，这里不展开讨论。

二. 偏斜扭转

解决了颜色的定位问题以后，下面的问题是：在定位以后，我们如何把偏斜过来的车牌扭正呢？

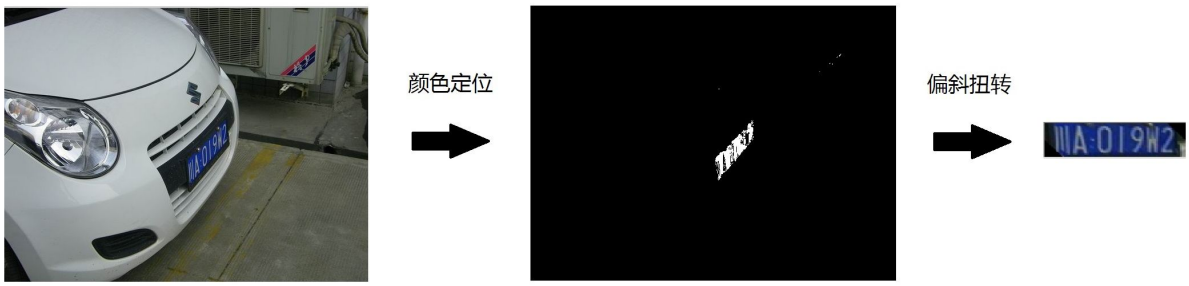


图 11 偏斜扭转效果

这个过程叫做偏斜扭转过程。其中一个关键函数就是 opencv 的仿射变换函数。但在具体实施时，有很多需要解决的问题。

1. 分析

在任何新的功能开发之前，技术预研都是第一步。

在这篇文档介绍了 opencv 的仿射变换功能。效果见下图。

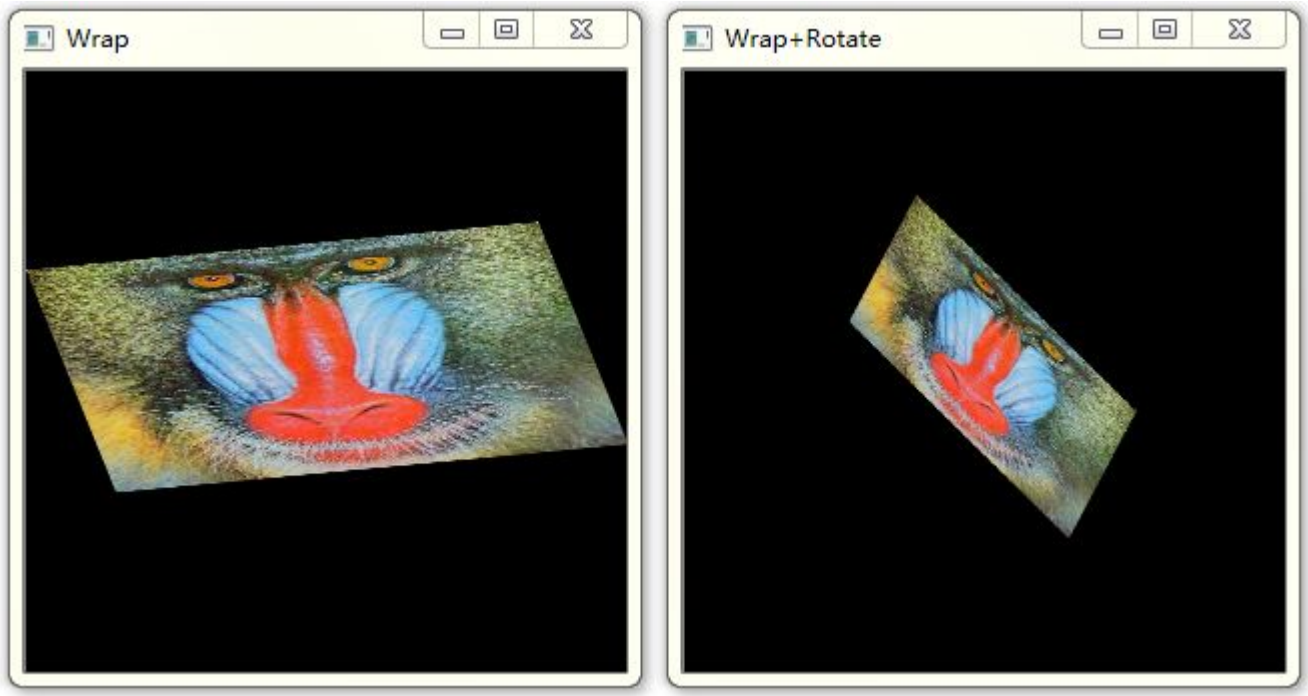


图 12 仿射变换效果

仔细看下，貌似这个功能跟我们的需求很相似。我们的偏斜扭转功能，说白了，就是把对图像的观察视角进行了一个转换。

不过这篇文章里的代码基本来自于另一篇官方文档。官方文档里还有一个例子，可以矩形扭转成平行四边形。而我们的需求正是将平行四边形的车牌扭正成矩形。这么说来，只要使用例子中对应的反函数，应该就可以实现我们的需求。从这个角度来看，偏斜扭转功可以实现。确定了可行性以后，下一步就是思考如何实现。

在原先的版本中，我们对定位出来的区域会进行一次角度判断，当角度小于某个阈值（默认 30 度）时就会进行全图旋转。

这种方式有两个问题：

一是我们的策略是对整幅图像旋转。对于 opencv 来说，每次旋转操作都是一个矩形的乘法过程，对于非常大的图像，这个过程是非常消耗计算资源的；

二是 30 度的阈值无法处理示例图片。事实上，示例图片的定位区域的角度是-50 度左右，已经大于我们的阈值了。为了处理这样的图片，我们需要把我们的阈值增大，例如增加到 60 度，那么这样的结果是带来候选区域的增多。

两个因素结合，会大幅度增加处理时间。为了不让处理速度下降，必须想办法规避这些影响。

一个方法是不再使用全图旋转，而是区域旋转。其实我们在获取定位区域后，我们并不需要定位区域以外的图像。

倘若我们能划出一块小的区域包围定位区域，然后我们仅对定位区域进行旋转，那么计算量就会大幅度降低。而这点，在 opencv 里是可以实现的，我们对定位区域 RotatedRect 用 boundingRect()方法获取外接矩形，再使用 Mat(Rect ...)方法截取这个区域图块，从而生成一个小的区域图像。于是下面的所有旋转等操作都可以基于这个区域图像进行。

在这些设计决定以后，下面就来思考整个功能的架构。

我们要解决的问题包括三类，第一类是正的车牌，第二类是倾斜的车牌，第三类是偏斜的车牌。前两类是前面说过的，第三类是本次新增的功能需求。第二类倾斜车牌与第三类车牌的区别见下图。

图 13 两类不同的旋转

通过上图可以看出，正视角的旋转图片的观察角度仍然是正方向的，只是由于路的不平或者摄像机的倾斜等原因，导致矩形有一定倾斜。这类图块的特点就是在 RotataedRect 内部，车牌部分仍然是个矩形。偏斜视角的图片的观察角度是非正方向的，是从侧面去看车牌。这类图块的特点是在 RotataedRect 内部，车牌部分不再是个矩形，而是一个平行四边形。这个特性决定了我们需要区别的对待这两类图片。

一个初步的处理思路就是下图。

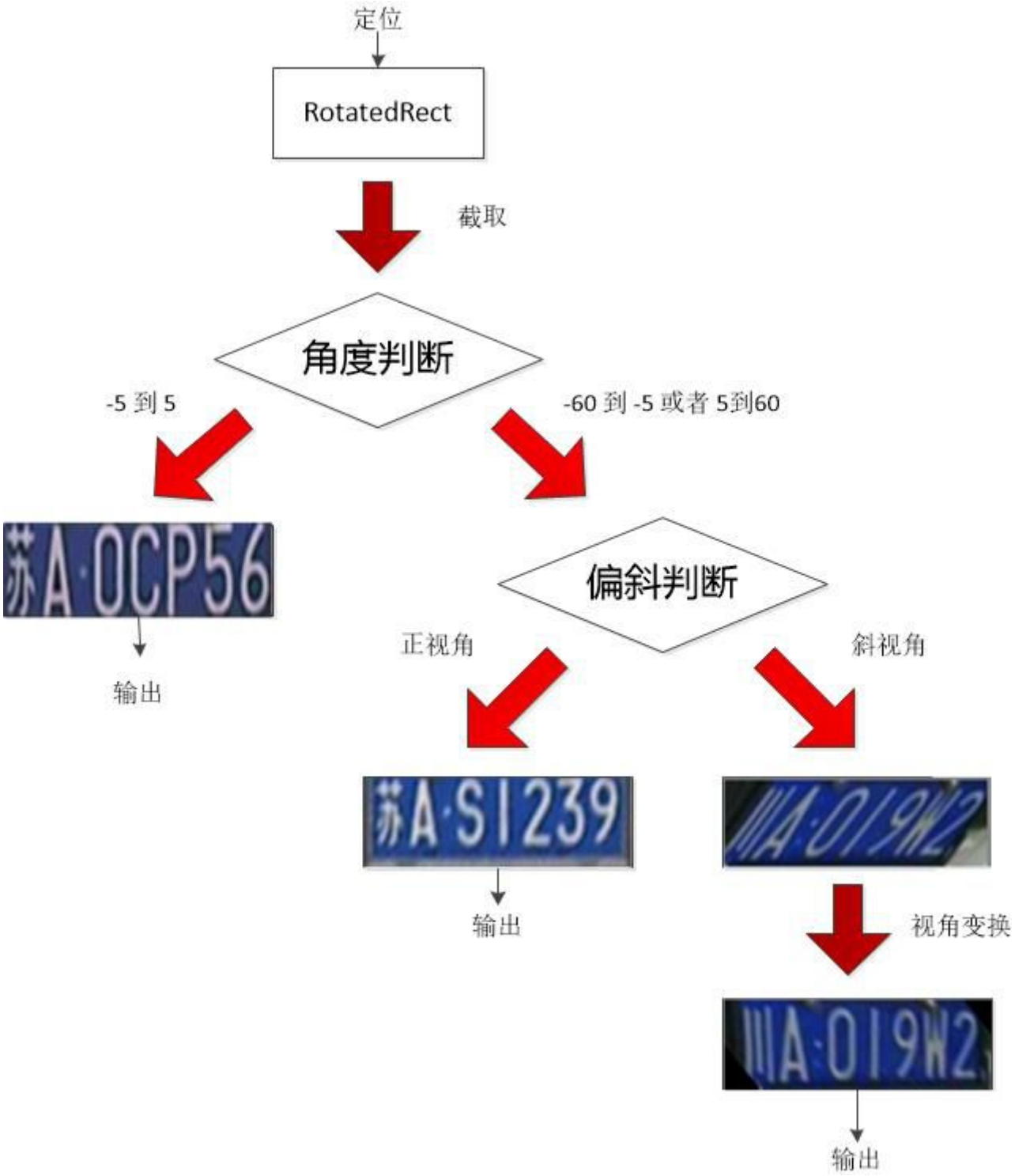


图 14 分析实现流程

简单来说，整个处理流程包括下面四步：

1.感兴趣区域的截取
2.角度判断
3.偏斜判断
4.仿射变换

接下来按照这四个步骤依次介绍。

2. ROI 截取

如果要使用区域旋转，首先我们必须从原图中截取出一个包含定位区域的图块。

opencv 提供了一个从图像中截取感兴趣区域 ROI 的方法，也就是 Mat(Rect ...)。这个方法会在 Rect 所在的位置，截取原图中一个图块，然后将其赋值到一个新的 Mat 图像里。遗憾的是这个方法不支持 RotataedRect，同时 Rect 与 RotataedRect 也没有继承关系。因此布不能直接调用这个方法。

我们可以使用 RotataedRect 的 boudingRect()方法。这个方法会返回一个 RotataedRect 的最小外接矩形，而且这个矩形是一个 Rect。因此将这个 Rect 传递给 Mat(Rect...)方法就可以截取出原图的 ROI 图块，并获得对应的 ROI 图像。

需要注意的是，ROI 图块和 ROI 图像的区别，当我们给定原图以及一个 Rect 时，原图中被 Rect 包围的区域称为 ROI 图块，此时图块里的坐标仍然是原图的坐标。当这个图块里的内容被拷贝到一个新的 Mat 里时，我们称这个新 Mat 为 ROI 图像。ROI 图像里仅仅只包含原来图块里的内容，跟原图没有任何关系。所以图块和图像虽然显示的内容一样，但坐标系已经发生了改变。在从 ROI 图块到 ROI 图像以后，点的坐标要计算一个偏移量。

下一步的工作中可以仅对这个 ROI 图像进行处理，包括对其旋转或者变换等操作。

示例图片中的截取出来的 ROI 图像如下图：

图 15 截取后的 ROI 图像

在截取中可能会发生一个问题。如果直接使用 boundingRect()函数的话，在运行过程中会经常发生这样的异常。OpenCV Error: Assertion failed (0 <= roi.x && 0 <= roi.width && roi.x + roi.width <= m.cols && 0 <= roi.y && 0 <= roi.height && roi.y + roi.height <= m.rows) incv::Mat::Mat，如下图。

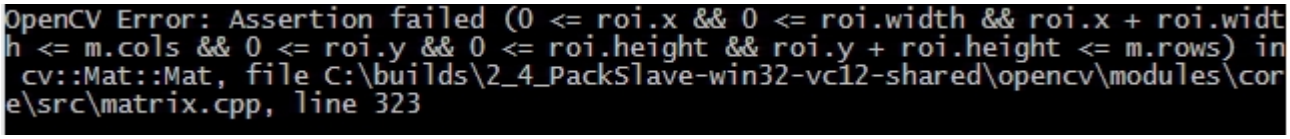


图 16 不安全的外接矩形函数会抛出异常

这个异常产生的原因在于，在 opencv2.4.8 中（不清楚 opencv 其他版本是否没有这个问题），boundingRect()函数计算出的 Rect 的四个点的坐标没有做验证。这意味着你计算一个 RotataedRect

的最小外接矩形 Rect 时，它可能会给你一个负坐标，或者是一个超过原图片外界的坐标。于是当你把 Rect 作为参数传递给 Mat(Rect ...)的话，它会提示你所要截取的 Rect 中的坐标越界了！

解决方案是实现一个安全的计算最小外接矩形 Rect 的函数，在 boundingRect()结果之上，对角点坐标进行一次判断，如果值为负数，就置为 0，如果值超过了原始 Mat 的 rows 或 cols，就置为原始 Mat 的这些 rows 或 cols。

这个安全函数名为 calcSafeRect(...)，下面是这个函数的代码。

View Code

3. 扩大化旋转

好，当我通过 calcSafeRect(...)获取了一个安全的 Rect，然后通过 Mat(Rect ...)函数截取了这个感兴趣图像 ROI 以后。下面的工作就是对这个新的 ROI 图像进行操作。

首先是判断这个 ROI 图像是否要旋转。为了降低工作量，我们不对角度在-5 度到 5 度区间的 ROI 进行旋转（注意这里讲的角度针对的生成 ROI 的 RotataedRect，ROI 本身是水平的）。因为这么小的角度对于 SVM 判断以及字符识别来说，都是没有影响的。

对其他的角度我们需要对 ROI 进行旋转。当我们对 ROI 进行旋转以后，接着把转正后的 RotataedRect 部分从 ROI 中截取出来。

但很快我们就会碰到一个新问题。让我们看一下下图，为什么我们截取出来的车牌区域最左边的“川”字和右边的“2”字发生了形变？为了搞清这个原因，作者仔细地研究了旋转与截取函数，但很快发现了形变的根源在于旋转后的 ROI 图像。

仔细看一下旋转后的 ROI 图像，是否左右两侧不再完整，像是被截去了一部分？

图 17 旋转后图像被截断

要想理解这个问题，需要理解 opencv 的旋转变换函数的特性。作为旋转变换的核心函数，affinTransform 会要求你输出一个旋转矩阵给它。这很简单，因为我们只需要给它一个旋转中心点以及角度，它就能计算出我们想要的旋转矩阵。旋转矩阵的获得是通过如下的函数得到的：

```
Mat rot_mat = getRotationMatrix2D(new_center, angle, 1);
```

在获取了旋转矩阵 rot_mat，那么接下来就需要调用函数 warpAffine 来开始旋转操作。这个函数的参数包括一个目标图像、以及目标图像的 Size。目标图像容易理解，大部分 opencv 的函数都会需要这个参数。我们只要新建一个 Mat 即可。那么目标图像的 Size 是什么？在一般的观点中，假设我们需要旋转一个图像，我们给 opencv 一个原始图像，以及我需要在某个旋转点对它旋转一个角度的需求，那么

opencv 返回一个图像给我即可，这个图像的 Size 或者说大小应该是 opencv 返回给我的，为什么要我来告诉它呢？

你可以试着对一个正方形进行旋转，仔细看看，这个正方形的外接矩形的大小会如何变化？当旋转角度还小时，一切都还好，当角度变大时，明显我们看到的外接矩形的大小也在扩增。在这里，外接矩形被称为视框，也就是我需要旋转的正方形所需要的最小区域。随着旋转角度的变大，视框明显增大。

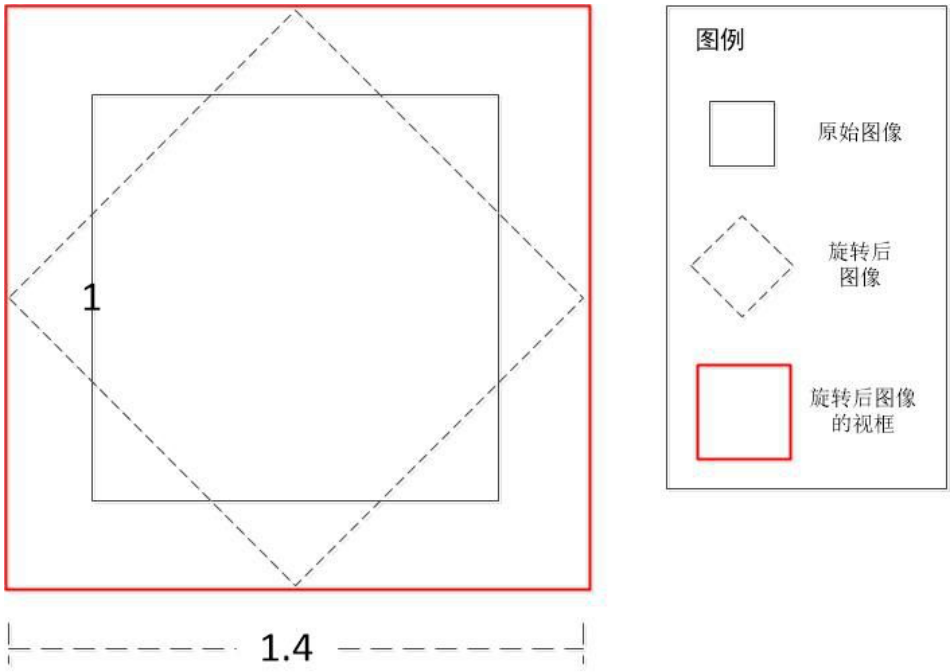


图 18 矩形旋转后所需视框增大

在图像旋转完以后，有三类点会获得不同的处理，一种是有原图像对应点且在视框内的，这些点被正常显示；一类是在视框内但找不到原图像与之对应的点，这些点被置 0 值（显示为黑色）；最后一类是有原图像与之对应的点，但不在视框内的，这些点被悲惨的抛弃。

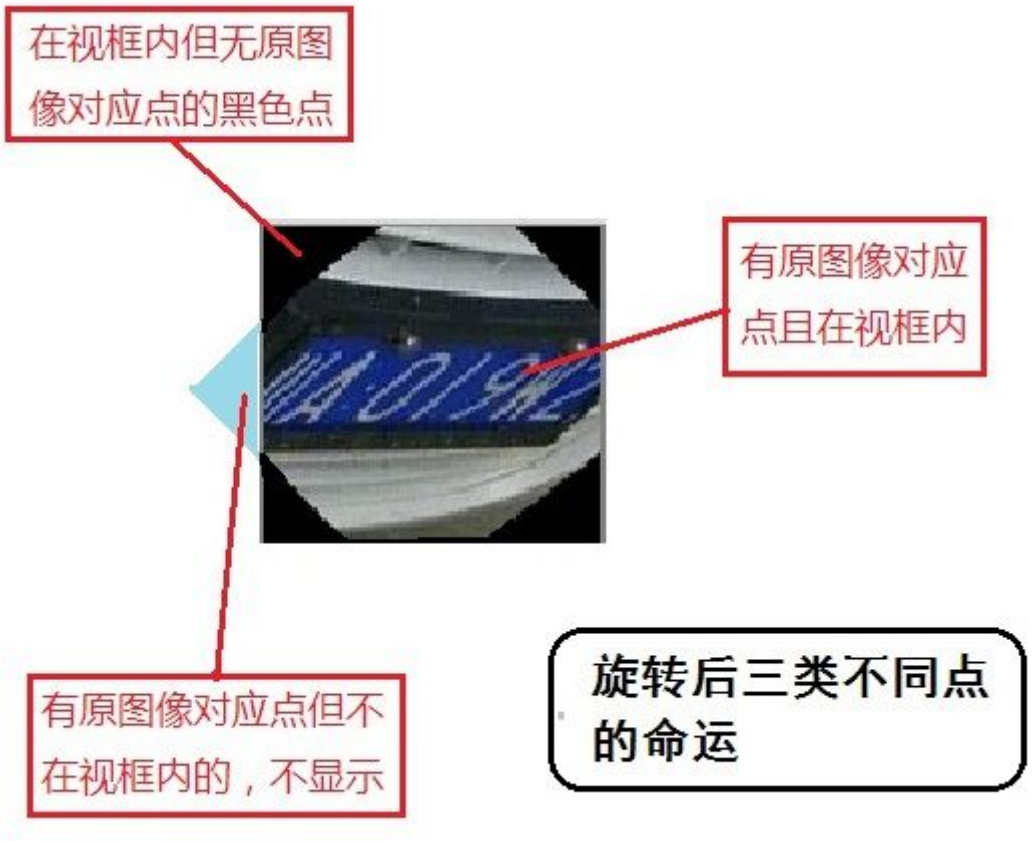


图 19 旋转后三类不同点的命运

这就是旋转后不同三类点的命运，也就是新生成的图像中一些点呈现黑色（被置 0），一些点被截断（被抛弃）的原因。如果把视框调整大点的话，就可以大幅度减少被截断点的数量。所以，为了保证旋转后的图像不被截断，因此我们需要计算一个合理的目标图像的 Size，让我们的感兴趣区域得到完整的显示。

下面的代码使用了一个极为简单的策略，它将原始图像与目标图像都进行了扩大化。首先新建一个尺寸为原始图像 1.5 倍的新图像，接着把原始图像映射到新图像上，于是我们得到了一个显示区域(视框)扩大化后的原始图像。显示区域扩大以后，那些在原图像中没有值的像素被置了一个初值。

接着调用 warpAffine 函数，使用新图像的大小作为目标图像的大小。warpAffine 函数会将新图像旋转，并用目标图像尺寸的视框去显示它。于是我们得到了一个所有感兴趣区域都被完整显示的旋转后图像。

这样，我们再使用 getRectSubPix()函数就可以获得想要的车牌区域了。

图 20 扩大化旋转后图像不再被截断

以下就是旋转函数 rotation 的代码。

```
View Code
```

4. 偏斜判断

当我们对 ROI 进行旋转以后，下面一步工作就是把 RotataedRect 部分从 ROI 中截取出来，这里可以使用 getRectSubPix 方法，这个函数可以在被旋转后的图像中截取一个正的矩形图块出来，并赋值到一个新的 Mat 中，称为车牌区域。

下步工作就是分析截取后的车牌区域。车牌区域里的车牌分为正角度和偏斜角度两种。对于正的角度而言，可以看出车牌区域就是车牌，因此直接输出即可。而对于偏斜角度而言，车牌是平行四边形，与矩形的车牌区域不重合。

如何判断一个图像中的图形是否是平行四边形？

一种简单的思路就是对图像二值化，然后根据二值化图像进行判断。图像二值化的方法有很多种，假设我们这里使用一开始在车牌定位功能中使用的大津阈值二值化法的话，效果不会太好。因为大津阈值是自适应阈值，在完整的图像中二值出来的平行四边形可能在小的局部图像中就不再是。最好的办法是使用在前面定位模块生成后的原图的二值图像，我们通过同样的操作就可以在原图中截取一个跟车牌区域对应的二值化图像。

下图就是一个二值化车牌区域获得的过程。

图 21 二值化的车牌区域

接下来就是对二值化车牌区域进行处理。为了判断二值化图像中白色的部分是平行四边形。一种简单的做法就是从图像中选择一些特定的行。计算在这个行中，第一个全为 0 的串的长度。从几何意义上来看，这就是平行四边形斜边上某个点距离外接矩形的长度。

假设我们选择的这些行位于二值化图像高度的 1/4，2/4，3/4 处的话，如果是白色图形是矩形的话，这些串的大小应该是相等或者相差很小的，相反如果是平行四边形的话，那么这些串的大小应该不等，并且呈现一个递增或递减的关系。通过这种不同，我们就可以判断车牌区域里的图形，究竟是矩形还是平行四边形。

偏斜判断的另一个重要作用就是，计算平行四边形倾斜的斜率，这个斜率值用来在下面的仿射变换中发挥作用。我们使用一个简单的公式去计算这个斜率，那就是利用上面判断过程中使用的串大小，假设二值化图像高度的 1/4，2/4，3/4 处对应的串的大小分别为 len1，len2，len3，车牌区域的高度为 Height。一个计算斜率 slope 的计算公式就是：(len3-len1)/Height*2。

Slope 的直观含义见下图。

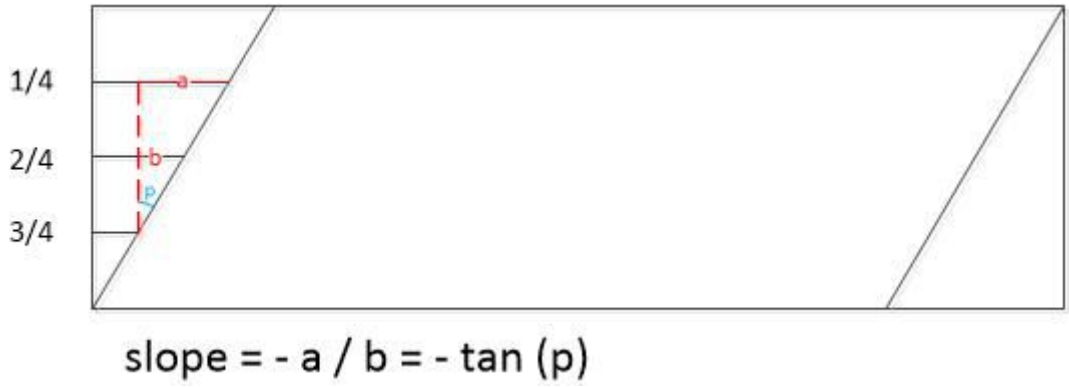


图 22 slope 的几何含义

需要说明的，这个计算结果在平行四边形是右斜时是负值，而在左斜时则是正值。于是可以根据 slope 的正负判断平行四边形是右斜或者左斜。在实践中，会发生一些公式不能应对的情况，例如像下图这种情况，斜边的部分区域发生了内凹或者外凸现象。这种现象会导致 len1,len2 或者 len3 的计算有误，因此 slope 也会不准。

图 23 内凹现象

为了实现一个鲁棒性更好的计算方法，可以用(len2-len1)/Height*4 与(len3-len1)/Height*2 两者之间更靠近 tan(angle)的值作为 solpe 的值（在这里，angle 代表的是原来 RotataedRect 的角度）。

多采取了一个 slope 备选的好处是可以避免单点的内凹或者外凸，但这仍然不是最好的解决方案。在最后的讨论中会介绍一个其他的实现思路。

完成偏斜判断与斜率计算的函数是 isdeflection，下面是它的代码。

```
View Code
```

5. 仿射变换

俗话说：行百里者半九十。前面已经做了如此多的工作，应该可以实现偏斜扭转功能了吧？但在最后的道路中，仍然有问题等着我们。

我们已经实现了旋转功能，并且在旋转后的区域中截取了车牌区域，然后判断车牌区域中的图形是一个平行四边形。下面要做的工作就是把平行四边形扭正成一个矩形。

图 24 从平行四边形车牌到矩形车牌

首先第一个问题就是解决如何从平行四边形变换成一个矩形的问题。opencv 提供了一个函数 warpAffine，就是仿射变换函数。注意，warpAffine 不仅可以让图像旋转（前面介绍过），也可以进行仿射变换，真是一个多才多艺的函数。o

通过仿射变换函数可以把任意的矩形拉伸成其他的平行四边形。opencv 的官方文档里给了一个示例，值得注意的是，这个示例演示的是把矩形变换为平行四边形，跟我们想要的恰恰相反。但没关系，我们先看一下它的使用方法。

图 25 opencv 官网上对 warpAffine 使用的示例

warpAffine 方法要求输入的参数是原始图像的左上点，右上点，左下点，以及输出图像的左上点，右上点，左下点。注意，必须保证这些点的对应顺序，否则仿射的效果跟你预想的不一样。通过这个方法介绍，我们可以大概看出，opencv 需要的是三个点对（共六个点）的坐标，然后建立一个映射关系，通过这个映射关系将原始图像的所有点映射到目标图像上。

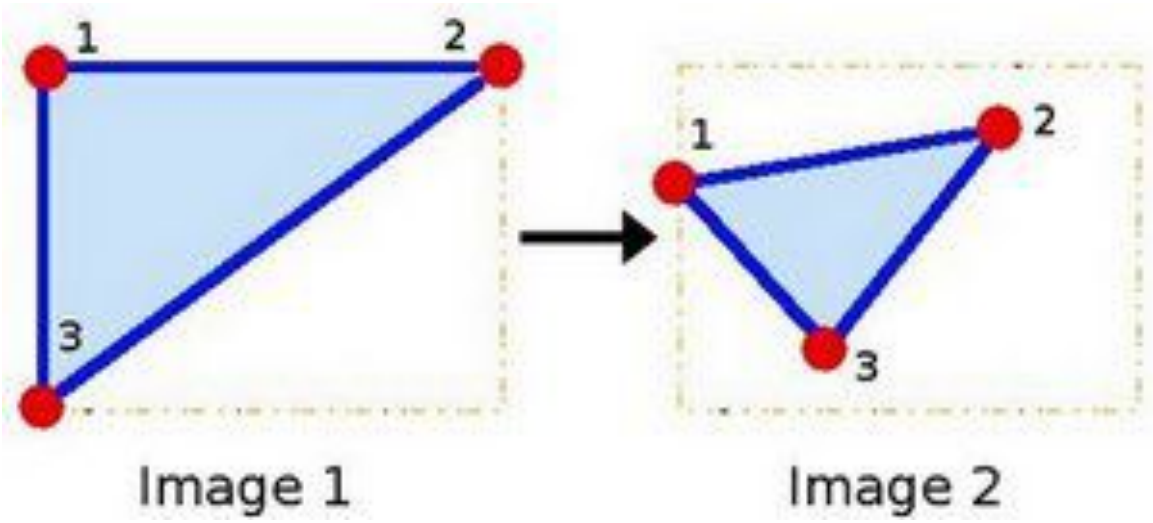


图 26 warpAffine 需要的三个对应坐标点

再回来看一下我们的需求，我们的目标是把车牌区域中的平行四边形映射为一个矩形。让我们做个假设，如果我们选取了车牌区域中的平行四边形车牌的三个关键点，然后再确定了我们希望将车牌扭正成的矩形的三个关键点的话，我们是否就可以实现从平行四边形车牌到矩形车牌的扭正？

让我们画一幅图像来看看这个变换的作用。有趣的是，把一个平行四边形变换为矩形会对包围平行四边形车牌的区域带来影响。

例如下图中，蓝色的实线代表扭转前的平行四边形车牌，虚线代表扭转后的。黑色的实线代表矩形的车牌区域，虚线代表扭转后的效果。可以看到，当蓝色车牌被扭转为矩形的同时，黑色车牌区域则被扭转为平行四边形。

注意，当车牌区域扭变为平行四边形以后，需要显示它的视框增大了。跟我们在旋转图像时碰到的情形一样。

图 27 平行四边形的扭转带来的变化

让我们先实际尝试一下仿射变换吧。

根据仿射函数的需要，我们计算平行四边形车牌的三个关键点坐标。其中左上点的值(xdiff,0)中的 xdiff 就是根据车牌区域的高度 height 与平行四边形的斜率 slope 计算得到的：

```
xidff = Height * abs(slope)
```

为了计算目标矩形的三个关键点坐标，我们首先需要把扭转后的原点坐标调整到平行四边形车牌区域左上角位置。见下图。

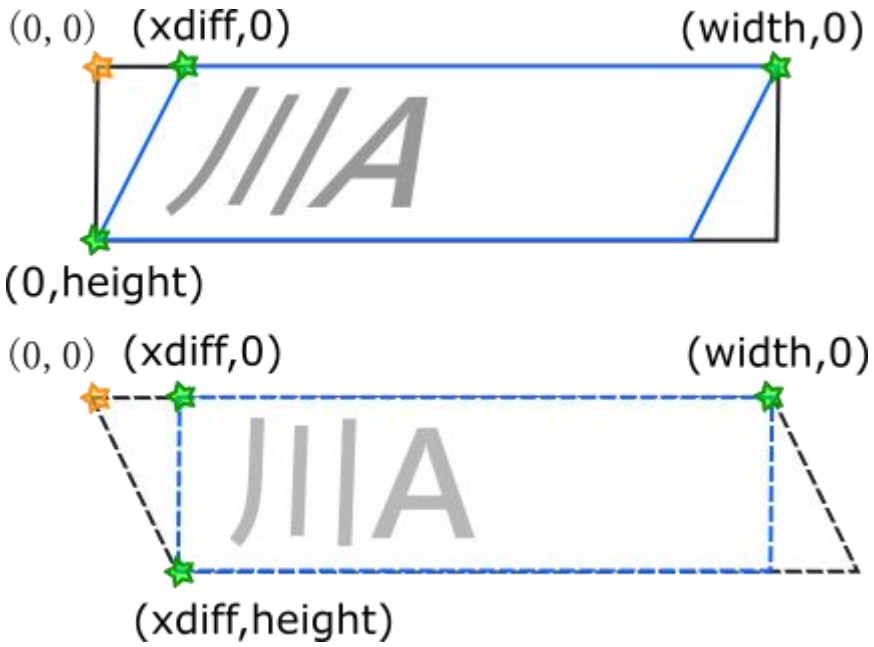


图 28 原图像的坐标计算

依次推算关键点的三个坐标。它们应该是

```
plTri[0] = Point2f(0 + xiff, 0);
plTri[1] = Point2f(width - 1, 0);
plTri[2] = Point2f(0, height - 1);

dstTri[0] = Point2f(xiff, 0);
dstTri[1] = Point2f(width - 1, 0);
dstTri[2] = Point2f(xiff, height - 1);
```

根据上图的坐标，我们开始进行一次仿射变换的尝试。

opencv 的 warpAffine 函数不会改变变换后图像的大小。而我们给它传递的目标图像的大小仅会决定视框的大小。不过这次我们不用担心视框的大小，因为根据图 27 看来，哪怕视框跟原始图像一样大，我们也足够显示扭正后的车牌。

看看仿射的效果。晕，好像效果不对，视框的大小是足够了，但是图像往右偏了一些，导致最右边的字母没有显示全。

图 29 被偏移的车牌区域

这次的问题不再是目标图像的大小问题了，而是视框的偏移问题。仔细观察一下我们的视框，倘若我们想把车牌全部显示的话，视框往右偏移一段距离，是不是就可以解决这个问题呢？为保证新的视框中心能够正好与车牌的中心重合，我们可以选择偏移 xidff/2 长度。正如下图所显示的一样。

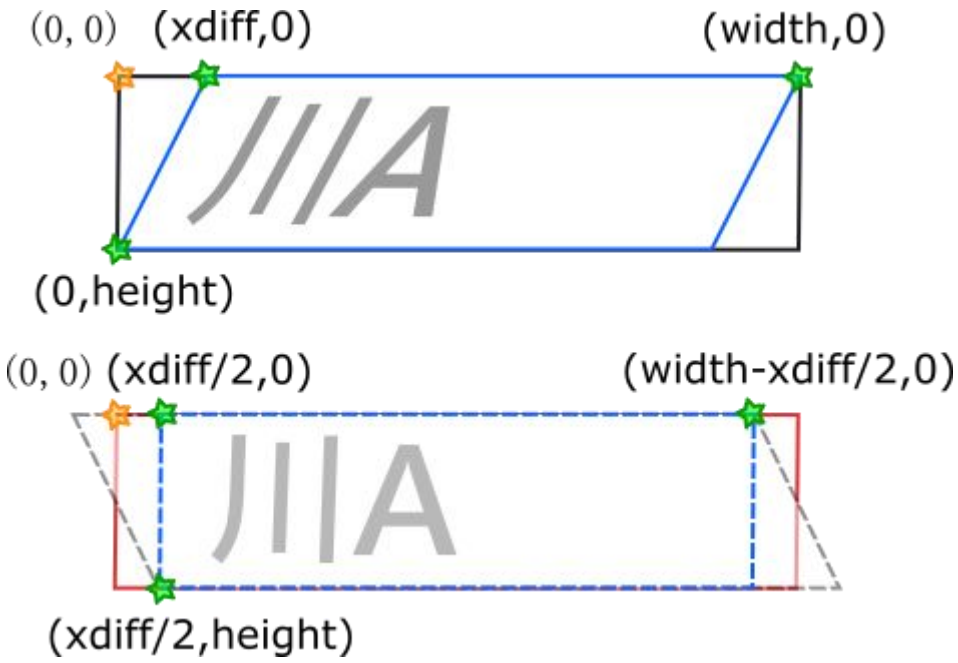


图 30 考虑偏移的坐标计算

视框往右偏移的含义就是目标图像 Mat 的原点往右偏移。如果原点偏移的话，那么仿射后图像的三个关键点的坐标要重新计算，都需要减去 xidff/2 大小。

重新计算的映射点坐标为下：

```
        plTri[0] = Point2f(0 + xiff, 0);
        plTri[1] = Point2f(width - 1, 0);
        plTri[2] = Point2f(0, height - 1);

        dstTri[0] = Point2f(xiff/2, 0);
        dstTri[1] = Point2f(width - 1 - xiff + xiff/2, 0);
        dstTri[2] = Point2f(xiff/2, height - 1);
```

再试一次。果然，视框被调整到我们希望的地方了，我们可以看到所有的车牌区域了。这次解决的是 warpAffine 函数带来的视框偏移问题。

图 31 完整的车牌区域

关于坐标调整的另一个理解就是当中心点保持不变时，平行四边形扭正为矩形时恰好是左上的点往左偏移了 xdiff/2 的距离，左下的点往右偏移了 xdiff/2 的距离，形成一种对称的平移。可以使用 ps 或者 inkspace 类似的矢量制图软件看看“斜切”的效果，

如此一来，就完成了偏斜扭正的过程。需要注意的是，向左倾斜的车牌的视框偏移方向与向右倾斜的车牌是相反的。我们可以用 slope 的正负来判断车牌是左斜还是右斜。

6. 总结

通过以上过程，我们成功的将一个偏斜的车牌经过旋转变换等方法扭正过来。

让我们回顾一下偏斜扭正过程。我们需要将一个偏斜的车牌扭正，为了达成这个目的我们首先需要对图像进行旋转。因为旋转是个计算量很大的函数，所以我们需要考虑不再用全图旋转，而是区域旋转。在旋转过程中，会发生图像截断问题，所以需要使用扩大化旋转方法。旋转以后，只有偏斜视角的车牌才需要扭正，正视角的车牌不需要，因此还需要一个偏斜判断过程。如此一来，偏斜扭正的过程需要旋转，区域截取，扩大化，偏斜判断等等过程的协助，这就是整个流程中有这么多步需要处理的原因。

下图从另一个视角回顾了偏斜扭正的过程，主要说明了偏斜扭转中的两次"截取"过程。

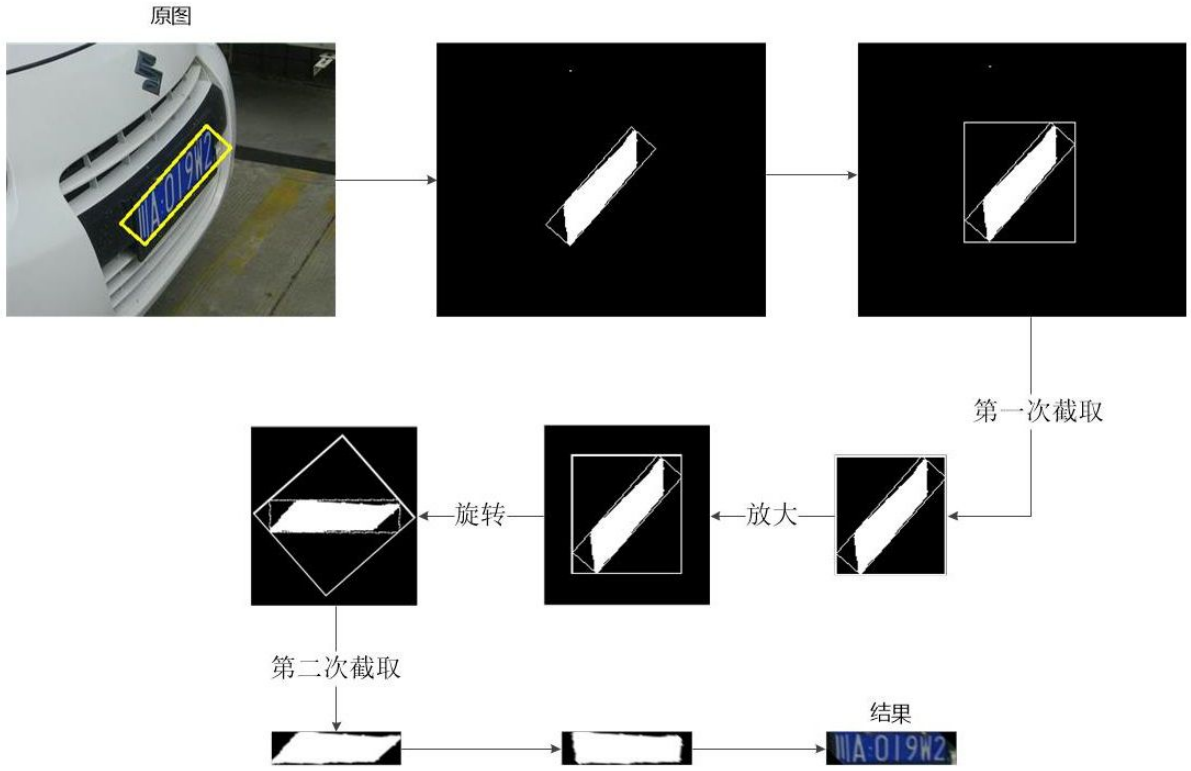


图 32 偏斜扭正全过程

1. 首先我们获取 RotatedRect，然后对每个 RotatedRect 获取外界矩形，也就是 ROI 区域。外接矩形的计算有可能获得不安全的坐标，因此需要使用安全的获取外界矩形的函数。
2. 获取安全外接矩形以后，在原图中截取这部分区域，并放置到一个新的 Mat 里，称之为 ROI 图像。这是本过程中第一次截取，使用 Mat(Rect ...)函数。
3. 接下来对 ROI 图像根据 RotatedRect 的角度展开旋转，旋转的过程中使用了放大化旋转法，以此防止车牌区域被截断。
4. 旋转完以后，我们把已经转正的 RotatedRect 部分截取出来，称之为车牌区域。这是本过程中第二次截取，与第一次不同，这次截取使用 getRectSubPix()方法。
5. 接下里使用偏斜判断函数来判断车牌区域里的车牌是否是倾斜的。

6. 如果是，则继续使用仿射变换函数 wrapAffine 来进行扭正处理，处理过程中要注意三个关键点的坐标。
7. 最后使用 resize 函数将车牌区域统一化为 EasyPR 的车牌大小。

整个过程有一个统一的函数--deskew。下面是 deskew 的代码。

```
View Code
```

最后是改善建议：

角度偏斜判断时可以用白色区域的轮廓来确定平行四边形的四个点，然后用这四个点来计算斜率。这样算出来的斜率的可能鲁棒性更好。

三. 总结

本篇文档介绍了颜色定位与偏斜扭转等功能。其中颜色定位属于作者一直想做的定位方法，而偏斜扭转则是作者以前认为不可能解决的问题。这些问题现在都基本被攻克了，并在这篇文档中阐述，希望这篇文档可以帮助到读者。

作者希望能在这片文档中不仅传递知识，也传授我在摸索过程中积累的经验。因为光知道怎么做并不能加深对车牌识别的认识，只有经历过失败，了解哪些思想尝试过，碰到了哪些问题，是如何解决的，才能帮助读者更好地认识这个系统的内涵。

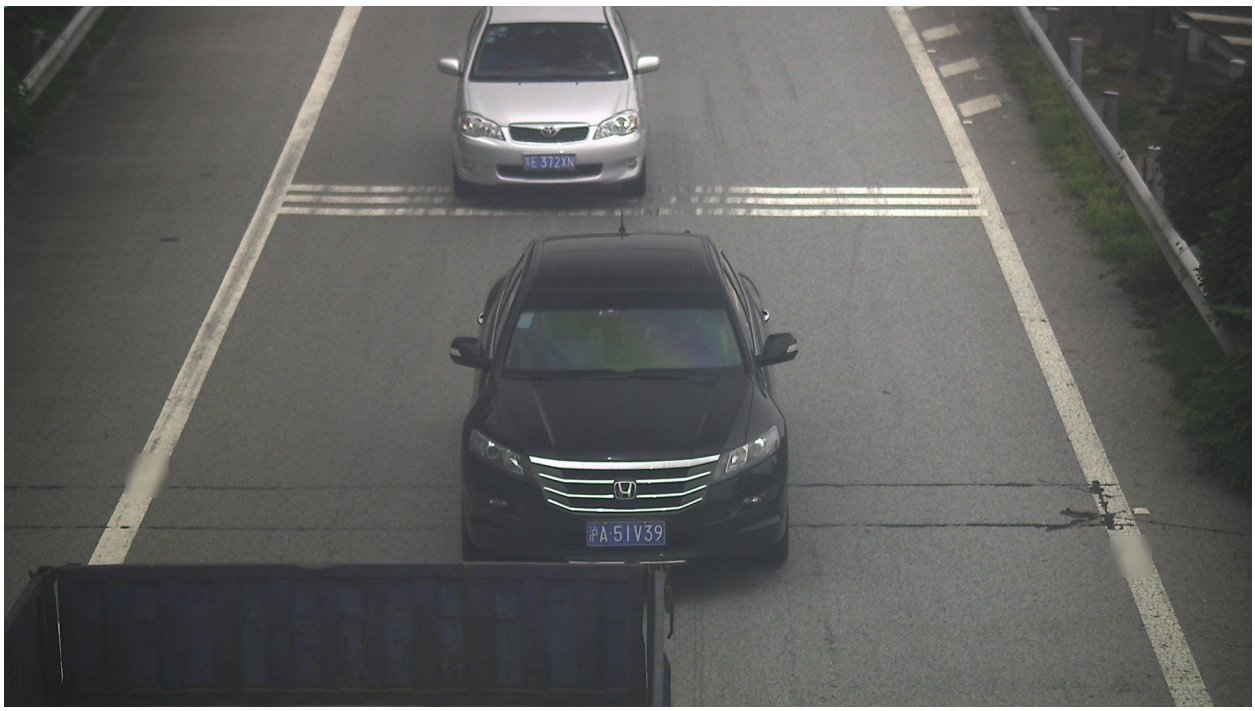
EasyPR 开发详解（6）SVM 开发详解

在前面的几篇文章中，我们介绍了 EasyPR 中车牌定位模块的相关内容。本文开始分析车牌定位模块后续步骤的车牌判断模块。车牌判断模块是 EasyPR 中的基于机器学习模型的一个模块，这个模型就是作者前文中从机器学习谈起中提到的 SVM（支持向量机）。

我们已经知道，车牌定位模块的输出是一些候选车牌的图片。但如何从这些候选车牌图片中甄选出真正的车牌，就是通过 SVM 模型判断/预测得到的。

图 1 从候选车牌中选出真正的车牌

简单来说，EasyPR 的车牌判断模块就是将候选车牌的图片一张张地输入到 SVM 模型中，然后问它，这是车牌么？如果 SVM 模型回答不是，那么就继续下一张，如果是，则把图片放到一个输出列表里。最后把列表输入到下一步处理。由于 EasyPR 使用的是列表作为输出，因此它可以输出一副图片中所有的车牌，不像一些车牌识别程序，只能输出一个车牌结果。



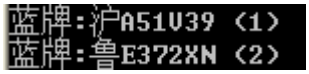


图 2 EasyPR 输出多个车牌

现在，让我们一步步地，进入这个 SVM 模型的核心看看，它是如何做到判断一副图片是车牌还是不是车牌的？本文主要分为三个大的部分：

1. SVM 应用：描述如何利用 SVM 模型进行车牌图片的判断。
2. SVM 训练：说明如何通过一系列步骤得到 SVM 模型。
3. SVM 调优：讨论如何对 SVM 模型进行优化，使其效果更加好。

一.SVM 应用

人类是如何判断一个张图片所表达的信息呢？简单来说，人类在成长过程中，大脑记忆了无数的图像，并且依次给这些图像打上了标签，例如太阳，天空，房子，车子等等。你们还记得当年上幼儿园时的那些教科书么，上面一个太阳，下面是文字。图像的组成事实上就是许多个像素，由像素组成的这些信息被输入大脑中，然后得出这个是什么东西的回答。我们在 SVM 模型中一开始输入的原始信息也是图像的所有像素，然后 SVM 模型通过对这些像素进行分析，输出这个图片是否是车牌的结论。

图 3 通过图像来学习

SVM 模型处理的是最简单的情况，它只要回答是或者不是这个"二值"问题，比从许多类中检索要简单很多。

我们可以看一下 SVM 进行判断的代码：

View Code

首先我们读取这幅图片，然后把这幅图片转为 OPENCV 需要的格式；

```
Mat p = histeq(inMat).reshape(1, 1);
p.convertTo(p, CV_32FC1);
```

接着调用 svm 的方法 predict；

```
int response = (int)svm.predict(p);
```

perdict 方法返回的值是 1 的话，就代表是车牌，否则就不是；

```
if (response == 1)
{
    resultVec.push_back(inMat);
}
```

svm 是类 CvSVM 的一个对象。这个类是 opencv 里内置的一个机器学习类。

```
CvSVM svm;
```

opencv 的 CvSVM 的实现基于 libsvm（具体信息可以看 opencv 的官方文档的介绍 ）。

libsvm 是台湾大学林智仁(Lin Chih-Jen)教授写的一个世界知名的 svm 库(可能算是目前业界使用率最高的一个库)。官方主页地址是这里。

libsvm 的实现基于 SVM 这个算法，90 年代初由 Vapnik 等人提出。国内几篇较好的解释 svm 原理的博文：cnblog 的 LeftNotEasy(解释的易懂)，pluskid 的博文(专业有配图)。

作为支持向量机的发明者，Vapnik 是一位机器学习界极为重要的大牛。最近这位大牛也加入了 Facebook。

图 4 SVM 之父 Vapnik

svm 的 perdict 方法的输入是待预测数据的特征，也称之为 features。在这里，我们输入的特征是图像全部的像素。由于 svm 要求输入的特征应该是一个向量，而 Mat 是与图像宽高对应的矩阵，因此在输入前我们需要使用 reshape(1,1)方法把矩阵拉伸成向量。除了全部像素以外，也可以有其他的特征，具体看第三部分“SVM 调优”。

predict 方法的输出是 float 型的值，我们需要把它转变为 int 型后再进行判断。如果是 1 代表就是车牌，否则不是。这个"1"的取值是由你在训练时输入的标签决定的。标签，又称之为 label，代表某个数据

的分类。如果你给 SVM 模型输入一个车牌，并告诉它，这个图片的标签是 5。那么你这边判断时所用的值就应该是 5。

以上就是 svm 模型判断的全过程。事实上，在你使用 EasyPR 的过程中，这些全部都是透明的。你不需要转变图片格式，也不需要调用 svm 模型 preditct 方法，这些全部由 EasyPR 在内部调用。

那么，我们应该做什么？这里的关键在于 CvSVM 这个类。我在前面的机器学习论文中介绍过，机器学习过程的步骤就是首先你搜集大量的数据，然后把这些数据输入模型中训练，最后再把生成的模型拿出来使用。

训练和预测两个过程是分开的。也就是说你们在使用 EasyPR 时用到的 CvSVM 类是我在先前就训练好的。我是如何把我训练好的模型交给各位使用的呢？CvSVM 类有个方法，把训练好的结果以 xml 文件的形式存储，我就是把这个 xml 文件随 EasyPR 发布，并让程序在执行前先加载好这个 xml。这个 xml 的位置就是在文件夹 Model 下面--svm.xml 文件。

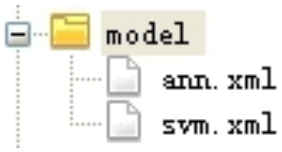


图 5 model 文件夹下的 svm.xml

如果看 CPlateJudge 的代码，在构造函数中调用了 LoadModel()这个方法。

```
CPlateJudge::CPlateJudge()
{
    //cout << "CPlateJudge" << endl;
    m_path = "model/svm.xml";
    LoadModel();
}
```

LoadModel()方法的主要任务就是装载 model 文件夹下 svm.xml 这个模型。

```
void CPlateJudge::LoadModel()
{
    svm.clear();
    svm.load(m_path.c_str(), "svm");
}
```

如果你把这个 xml 文件换成其他的，那么你就可以改变 EasyPR 车牌判断的内核，从而实现你自己的车牌判断模块。

后面的部分全部是告诉你如何有效地实现一个自己的模型（也就是 svm.xml 文件）。如果你对 EasyPR 的需求仅仅在应用层面，那么到目前的了解就足够了。如果你希望能够改善 EasyPR 的效果，定制一个自己的车牌判断模块，那么请继续往下看。

二.SVM 训练

恭喜你！从现在开始起，你将真正踏入机器学习这个神秘并且充满未知的领域。至今为止，机器学习很多方法的背后原理都非常复杂，但众多的实践都证明了其有效性。与许多其他学科不同，机器学习界更为关注的是最终方法的效果，也就是偏重以实践效果作为评判标准。因此非常适合从工程的角度入手，通过自己动手实践一个项目里来学习，然后再转入理论。这个过程已经被证明是**有效**的，本文的作者在开发 EasyPR 的时候，还没有任何机器学习的理论基础。后来的知识是将通过学习相关课程后获取的。

简而言之，SVM 训练部分的目标就是通过一批数据，然后生成一个代表我们模型的 xml 文件。

EasyPR 中所有关于训练的方法都可以在 svm_train.cpp 中找到(1.0 版位于 train/code 文件夹下，1.1 版位于 src/train 文件夹下)。

一个训练过程包含 5 个步骤，见下图：

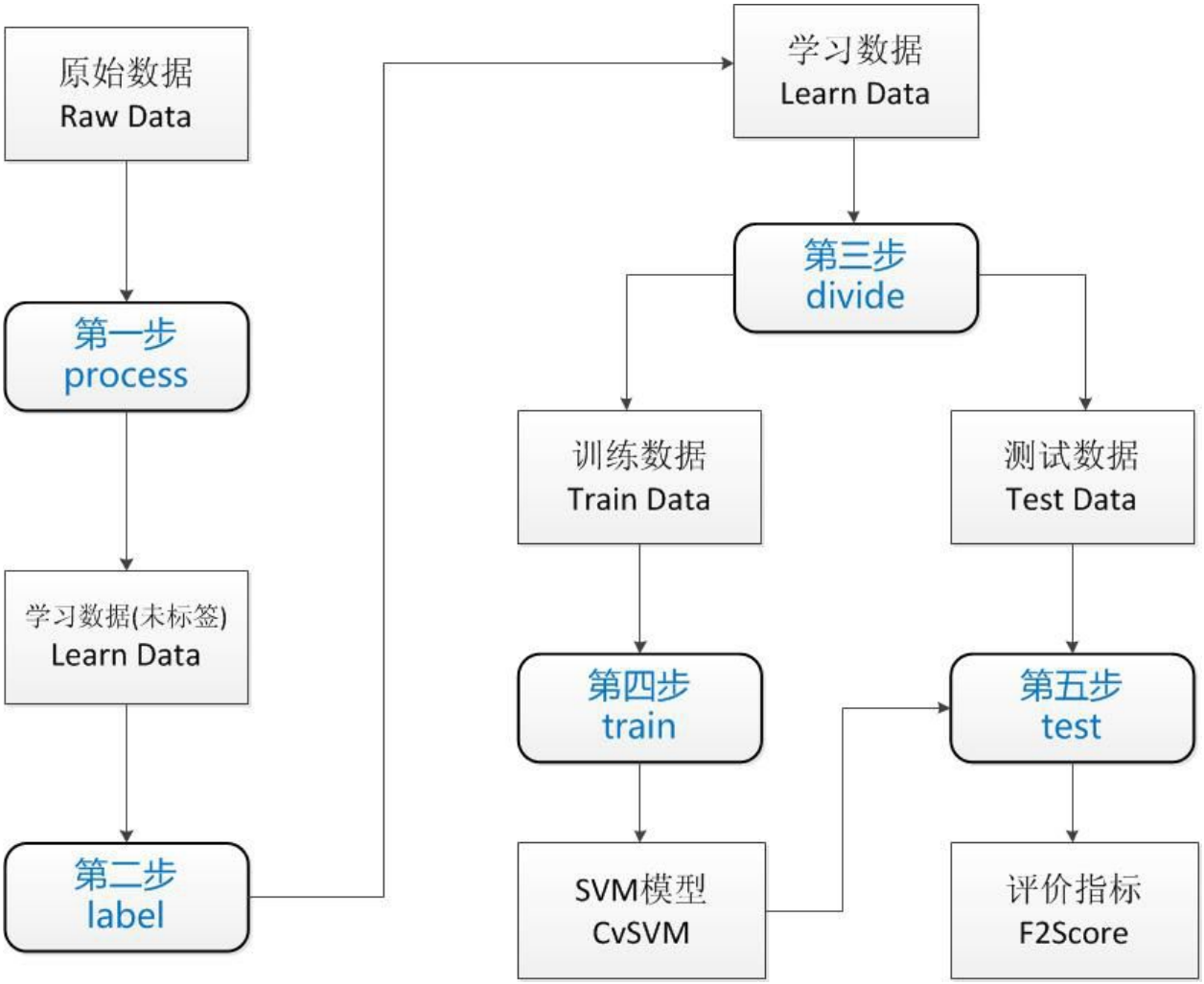


图 6 一个完整的 SVM 训练流程

下面具体讲解一下这 5 个步骤，步骤后面的括号里代表的是这个步骤主要的输入与输出。

1. preprocss（原始数据->学习数据(未标签)）

预处理步骤主要处理的是原始数据到学习数据的转换过程。原始数据（raw data），表示你一开始拿到的数据。这些数据的情况是取决你具体的环境的，可能有各种问题。学习数据（learn data），是可以被输入到模型的数据。

为了能够进入模型训练，必须将原始数据处理为学习数据，同时也可能进行了数据的筛选。比方说你有 10000 张原始图片，出于性能考虑，你只想用 1000 张图片训练，那么你的预处理过程就是将这 10000 张处理为符合训练要求的 1000 张。你生成的 1000 张图片中应该包含两类数据：真正的车牌图片和不是车牌的图片。如果你想让你的模型能够区分这两种类型。你就必须给它输入这两类的数据。

通过 EasyPR 的车牌定位模块 PlateLocate 可以生成大量的候选车牌图片，里面包括模型需要的车牌和非车牌图片。但这些候选车牌是没有经过分类的，也就是说没有标签。下步工作就是给这些数据贴上标签。

2. label （学习数据(未标签)->学习数据）

训练过程的第二步就是将未贴标签的数据转化为贴过标签的学习数据。我们所要做的工作只是将车牌图片放到一个文件夹里，非车牌图片放到另一个文件夹里。在 EasyPR 里，这两个文件夹分别叫做 HasPlate 和 NoPlate。如果你打开 train/data/plate_detect_svm 后，你就会看到这两个压缩包，解压后就是打好标签的数据(1.1 版本在同层 learn data 文件夹下面)。

如果有人问我开发一个机器学习系统最耗时的步骤是哪个，我会毫不犹豫的回答："贴标签"。诚然，各位看到的压缩包里已经有打好标签的数据了。但各位可能不知道作者花在贴这些标签上的时间。粗略估计，整个 EasyPR 开发过程中有 70%的时间都在贴标签。SVM 模型还好，只有两个类，训练数据仅有 1000 张。到了 ANN 模型那里，字符的类数有 40 多个，而且训练数据有 4000 张左右。那时候的贴标签过程，真是不堪回首的回忆，来回移动文件导致作者手经常性的非常酸。后来我一度想找个实习生帮我做这些工作。但转念一想，这些苦我都不愿承担，何苦还要那些小伙子承担呢。"己所不欲，勿施于人"。算了，既然这是机器学习者的命，那就欣然接受吧。幸好在这段磨砺的时光，我逐渐掌握了一个方法，大幅度减少了我贴标签的时间与精力。不然，我可能还未开始写这个系列的教程，就已经累吐血了。开发 EasyPR1.1 版本时，新增了一大批数据，因此又有了贴标签的过程。幸好使用这个方法，使得相关时间大幅度减少。这个方法叫做**逐次迭代自动标签法**。在后面会介绍这个方法。

贴标签后的车牌数据如下图：

图 7 在 HasPlate 文件夹下的图片

贴标签后的非车牌数据下图：

图 8 在 NoPlate 文件夹下的图片

拥有了贴好标签的数据以后，下面的步骤是分组，也称之为 divide 过程。

3. divide （学习数据->分组数据）

分组这个过程是 EasyPR1.1 版新引入的方法。

在贴完标签以后，我拥有了车牌图片和非车牌图片共几千张。在我直接训练前，不急。先拿出 30%的数据，只用剩下的 70%数据进行 SVM 模型的训练，训练好的模型再用这 30%数据进行一个效果测试。这 30%数据充当的作用就是一个评判数据测试集，称之为 test data，另 70%数据称之为 train data。于是一个完整的 learn data 被分为了 train data 和 test data。

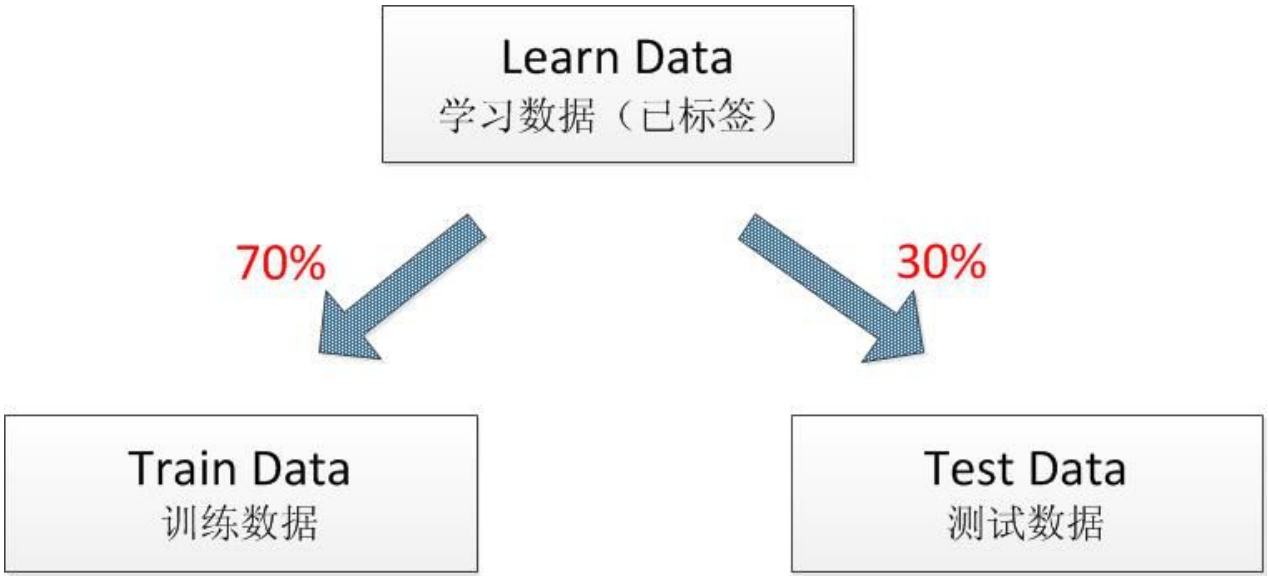


图 9 数据分组过程

在 EasyPR1.0 版是没有 test data 概念的，所有数据都输入训练，然后直接在原始的数据上进行测试。直接在原始的数据集上测试与单独划分出 30%的数据测试效果究竟有多少不同？

事实上，我们训练出模型的根本目的是为了对未知的，新的数据进行预测与判断。

当使用训练的数据进行测试时，由于模型已经考虑到了训练数据的特征，因此很难将这个测试效果推广到其他未知数据上。如果使用单独的测试集进行验证，由于测试数据集跟模型的生成没有关联，因此可以很好的反映出模型推广到其他场景下的效果。**这个过程就可以简单描述为你不可以拿你给学生的复习提纲卷去考学生，而是应该出一份考察知识点一样，但题目不一样的卷子。前者的方式无法区分出真正学会的人和死记硬背的人，而后者就能有效地反映出哪些人才是真正"学会"的。**

在 divide 的过程中，注意无论在 train data 和 test data 中都要保持数据的标签，也就是说车牌数据仍然归到 HasPlate 文件夹，非车牌数据归到 NoPlate 文件夹。于是，车牌图片 30%归到 test data 下面的 hasplate 文件夹，70%归到 train data 下面的 hasplate 文件夹，非车牌图片 30%归到 test data 下面的 noplate 文件夹，70%归到 train data 下面的 noplate 文件夹。于是在文件夹 train 和 test 下面又有两个子文件夹，他们的结构树就是下图：

图 10 分组后的文件树

divide 数据结束以后，我们就可以进入真正的机器学习过程。也就是对数据的训练过程。

4. train （训练数据->模型）

模型在代码里的代表就是 CvSVM 类。在这一步中所要做的就是加载 train data，然后用 CvSVM 类的 train 方法进行训练。这个步骤只针对的是上步中生成的总数据 70%的训练数据。

具体来说，分为以下几个子步骤：

1) 加载待训练的车牌数据。见下面这段代码。

View Code

注意看，车牌图像我存储在的是一个 vector<Mat>中，而标签数据我存储在的是一个 vector<int>中。我将 train/HasPlate 中的图像依次取出来，存入 vector<Mat>。每存入一个图像，同时也往 vector<int>中存入一个 int 值 1，也就是说图像和标签分别存在不同的 vector 对象里，但是保持一一对应的关系。

2) 加载待训练的非车牌数据，见下面这段代码中的函数。基本内容与加载车牌数据类似，不同之处在于文件夹是 train/NoPlate，并且我往 vector<int>中存入的是 int 值 0，代表无车牌。

View Code

3) 将两者合并。目前拥有了两个 vector<Mat>和两个 vector<int>。将代表车牌图片和非车牌图片数据的两个 vector<Mat>组成一个新的 Mat--trainingData，而代表车牌图片与非车牌图片标签的两个 vector<int>组成另一个 Mat--classes。接着做一些数据类型的调整，以让其符合 svm 训练函数 train 的要求。这些做完后，数据的准备工作基本结束，下面就是参数配置的工作。

View Code

4) 配置 SVM 模型的训练参数。SVM 模型的训练需要一个 CvSVMParams 的对象，这个类是 SVM 模型中训练对象的参数的组合，如何给这里的参数赋值，是很有讲究的一个工作。注意，这里是 SVM 训练的核心内容，也是最能体现一个机器学习专家和新手区别的地方。机器学习最后模型的效果差异有很大因素取决与模型训练时的参数，尤其是 SVM，有非常多的参数供你配置(见下面的代码)。参数众多是一个问题，更为显著的是，机器学习模型中参数的一点微调都可能带来最终结果的巨大差异。

```
    CvSVMParams SVM_params;
    SVM_params.svm_type = CvSVM::C_SVC;
    SVM_params.kernel_type = CvSVM::LINEAR; //CvSVM::LINEAR;
    SVM_params.degree = 0;
    SVM_params.gamma = 1;
    SVM_params.coef0 = 0;
    SVM_params.C = 1;
    SVM_params.nu = 0;
    SVM_params.p = 0;
    SVM_params.term_crit = cvTermCriteria(CV_TERMCRIT_ITER, 1000, 0.01);
```

opencv 官网文档对 CvSVMParams 类的各个参数有一个详细的解释。如果你上过 SVM 课程的理论部分，你可能对这些参数的意思能搞的明白。但在这里，我们可以不去管参数的含义，因为我们有更好的方法去解决这个问题。

- **degree** - Parameter degree of a kernel function (POLY).
- **gamma** - Parameter γ of a kernel function (POLY / RBF / SIGMOID).
- **coef0** - Parameter coef0 of a kernel function (POLY / SIGMOID).
- **Cvalue** - Parameter C of a SVM optimization problem (C_SVC / EPS_SVR / NU_SVR).
- **nu** - Parameter ν of a SVM optimization problem (NU_SVC / ONE_CLASS / NU_SVR).
- **p** - Parameter ϵ of a SVM optimization problem (EPS_SVR).

图 11 SVM 各参数的作用

这个原因在于：EasyPR1.0 使用的是 liner 核，也称之为线型核，因此 degree 和 gamma 还有 coef0 三个参数没有作用。同时，在这里 SVM 模型用作的问题是分类问题，那么 nu 和 p 两个参数也没有影响。最后唯一能影响的参数只有 Cvalue。到了 EasyPR1.1 版本以后，默认使用的是 RBF 核，因此需要调整的参数多了一个 gamma。

以上参数的选择都可以用自动训练(train_auto)的方法去解决，在下面的 SVM 调优部分会具体介绍 train_auto。

5) 开始训练。OK！数据载入完毕，参数配置结束，一切准备就绪，下面就是交给 opencv 的时间。我们只要将前面的 trainingData，classes，以及 CvSVMParams 的对象 SVM_params 交给 CvSVM 类的 train 函数就可以。另外，直接使用 CvSVM 的构造函数，也可以完成训练过程。例如下面这行代码：

```
CvSVM svm(trainingData, classes, Mat(), Mat(), SVM_params);
```

训练开始后，慢慢等一会。机器学习中数据训练的计算量往往是非常大的，即便现代计算机也要运行很长时间。具体的时间取决于你训练的数据量的大小以及模型的复杂度。在我的 2.0GHz 的机器上，训练 1000 条数据的 SVM 模型的时间大约在 1 分钟左右。

训练完成以后，我们就可以用 CvSVM 类的对象 svm 去进行预测了。如果我们仅仅需要这个模型，现在可以把它存到 xml 文件里，留待下次使用：

```
FileStorage fsTo("train/svm.xml", cv::FileStorage::WRITE);
svm.write(*fsTo, "svm");
```

5. test （测试数据->评判指标）

记得我们还有 30%的测试数据了么？现在是使用它们的时候了。将这些数据以及它们的标签加载如内存，这个过程与加载训练数据的过程是一样的。接着使用我们训练好的 SVM 模型去判断这些图片。

下面的步骤是对我们的模型做指标评判的过程。首先，测试数据是有标签的数据，这意味着我们知道每张图片是车牌还是不是车牌。另外，用新生成的 svm 模型对数据进行判断，也会生成一个标签，叫做“预测标签”。“预测标签”与“标签”一般是存在误差的，这也就是模型的误差。这种误差有两种情况：1.这副图片是真的车牌，但是 svm 模型判断它是“非车牌”；2.这幅图片不是车牌，但 svm 模型判断它是“车牌”。无疑，这两种情况都属于 svm 模型判断失误的情况。我们需要设计出来两个指标，来分别评测这两种失误情况发生的概率。这两个指标就是下面要说的“准确率”（precision）和“查全率”（recall）。

准确率是统计在我已经预测为车牌的图片中，真正车牌数据所占的比例。假设我们用 ptrue_rtrue 表示预测(p)为车牌并且实际(r)为车牌的数量，而用 ptrue_rfalse 表示实际不为车牌的数量。

准确率的计算公式是：

$$\frac{ptrue\_rtrue}{ptrue\_rtrue + ptrue\_rfalse}$$

图 12 precise 准确率

查全率是统计真正的车牌图片中，我预测为车牌的图片所占的比例。同上，我们用 ptrue_rtrue 表示预测与实际都为车牌的数量。用 pfalse_rtrue 表示实际为车牌，但我预测为非车牌的数量。

查全率的计算公式是：

$$\frac{ptrue\_rtrue}{ptrue\_rtrue + pfalse\_rtrue}$$

图 13 recall 查全率

recall 的公式与 precision 公式唯一的区别在于右下角。precision 是 ptrue_rfalse，代表预测为车牌但实际不是的数量；而 recall 是 pfalse_rtrue，代表预测是非车牌但其实是车牌的数量。

简单来说，precision 指标的期望含义就是要**"查的准"**，recall 的期望含义就是**"不要漏"**。查全率还有一个翻译叫做"召回率"。但很明显，召回这个词没有反映出查全率所体现出的不要漏的含义。

值得说明的是，precise 和 recall 这两个值自然是越高越好。但是如果一个高，一个低的话效果会如何，如何跟两个都中等的情况进行比较？为了能够数字化这种比较。机器学习界又引入了 FScore 这个数值。当 precise 和 recall 两者中任一者较高，而另一者较低是，FScore 都会较低。两者中等的情况下 Fscore 表现比一高一低要好。当两者都很高时，FScore 会很高。

FScore 的计算公式如下图：

$$Fscore = \frac{2 \times precise \times recall}{precise + recall}$$

图 14 Fscore 计算公式

模型测试以及评价指标是 EasyPR1.1 中新增的功能。在 svm_train.cpp 的最下面可以看到这三个指标的计算过程。

训练心得

通过以上 5 个步骤，我们就完成了模型的准备，训练，测试的全部过程。下面，说一说过程中的几点心得。

1. 完善 EasyPR 的 plateLocate 功能

在 1.1 版本中的 EasyPR 的车牌定位模块仍然不够完善。如果你的所有的图片符合某种通用的模式，参照前面的车牌定位的几篇教程，以及使用 EasyPR 新增的 Debug 模式，你可以将 EasyPR 的 plateLocate 模块改造为适合你的情况。于是，你就可以利用 EasyPR 为你制造大量的学习数据。通过原始数据的输入，然后通过 plateLocate 进行定位，再使用 EasyPR 已有的车牌判断模块进行图片的分类，于是你就可以得到一个基本分好类的学习数据。下面所需要做的就是人工核对，确认一下，保证每张图片的标签是正确的，然后再输入模型进行训练。

2. 使用"逐次迭代自动标签法"。

上面讨论的贴标签方法是在 EasyPR 已经提供了一个训练好的模型的情况下。如果一开始手上任何模型都没有，该怎么办？假设目前手里有成千上万个通过定位出来的各种候选车牌，手工一个个贴标签的话，岂不会让人累吐血？在前文中说过，我在一开始贴标签过程中碰到了这个问题，在不断被折磨与痛苦中，我发现了一个好方法，大幅度减轻了这整个工作的痛苦性。

当然，这个方法很简单。我如果说出来你一定也不觉得有什么奇妙的。但是如果在你准备对 1000 张图片进行手工贴标签时，相信我，使用这个方法会让你最后的时间节省一半。如果你需要雇 10 个人来贴标签的话，那么用了这个方法，可能你最后一个人都不用雇。

这个方法被我称为"逐次迭代自动标签法"。

方法核心很简单。就是假设你有 3000 张未分类的图片。你从中选出 1%，也就是 30 张出来，手工给它们每个图片进行分类工作。好的，如今你有了 30 张贴好标签的数据了，下步你把它直接输入到 SVM 模型中训练，获得了一个简单粗旷的模型。之后，你从图片集中再取出 3%的图片，也就是 90 张，**然后用刚训练好的模型对这些图片进行预测**，根据预测结果将它们自动分到 hasplate 和 noplate 文件夹下面。分完以后，你到这两个文件夹下面，看看哪些是预测错的，把 hasplate 里预测错的**移动**到 noplate 里，反之，把 noplate 里预测错的移动到 hasplate 里。

接着，你把一开始手工分类好的那 30 张图片，结合调整分类的 90 张图片，总共 120 张图片再输入 svm 模型中进行训练。于是你获得一个比最开始粗旷模型更精准点的模型。然后，你从 3000 张图片中再取出 6%的图片来，用这个模型再对它们进行预测，分类....

以上反复。你每训练出一个新模型，用它来预测后面更多的数据，然后自动分类。这样做最大的好处就是你只需要移动那些被分类错误的图片。其他的图片已经被正 确的归类了。注意，在整个过程中，你每次只需要对新拿出的数据进行人工确认，因为前面的数据已经分好类了。因此，你最好使用两个文件夹，一个是已经分好类 的数据，另一个是自动分类数据，需要手工确认的。这样两者不容易乱。

每次从未标签的原始数据库中取出的数据不要多，最好不要超过上次数据的两倍。这样可以保证你的模型的准确率稳步上升。如果想一口吃个大胖子，例如用 30 张图片训练出的模型，去预测 1000 张数据，那最后结果跟你手工分类没有任何区别了。

整个方法的原理很简单，就是**不断迭代循环细化**的思想。跟软件工程中迭代开发过程有异曲同工之妙。你只要理解了其原理，很容易就可以复用在任何其他机器学习模型的训练中，从而大幅度（或者部分）减轻机器学习过程中贴标签的巨大负担。

回到一个核心问题，对于开发者而言，什么样的方法才是自己实现一个 svm.xml 的最好方法。有以下几种选择。

1.你使用 EasyPR 提供的 svm.xml，这个方式等同于你没有训练，那么 EasyPR 识别的效率取决于你的环境与 EasyPR 的匹配度。运气好的话，这个效果也会不错。但如果你的环境下车牌跟 EasyPR 默认的不一样。那么可能就会有点问题。

2.使用 EasyPR 提供的训练数据，例如 train/data 文件下的数据，这样生成的效果等同于第一步的，不过你可以调整参数，试试看模型的表现会不会更好一点。

3.使用自己的数据进行训练。这个方法的适应性最好。首先你得准备你原始的数据，并且写一个处理方法，能够将原始数据转化为学习数据。下面你调用 EasyPR 的 PlateLocate 方法进行处理，将候选车牌图片从原图片截取出来。你可以使用逐次迭代自动标签思想，使用 EasyPR 已有的 svm 模型对这些候选图片进行预标签。然后再进行肉眼确认和手工调整，以生成标准的贴好标签的数据。后面的步骤就可以按照分组，训练，测试等过程顺次走下去。如果你使用了 EasyPR1.1 版本，后面的这几个过程已经帮你实现好代码了，你甚至可以直接在命令行选择操作。

以上就是 SVM 模型训练的部分，通过这个步骤的学习，你知道如何通过已有的数据去训练出一个自己的模型。下面的部分，是对这个训练过程的一个思考，讨论通过何种方法可以改善我最后模型的效果。

三.SVM 调优

SVM 调优部分，是通过对 SVM 的原理进行了解，并运用机器学习的一些调优策略进行优化的步骤。

在这个部分里，最好要懂一点机器学习的知识。同时，本部分也会讲的尽量通俗易懂，让人不会有理解上的负担。在 EasyPR1.0 版本中，SVM 模型的代码完全参考了 mastering opencv 书里的实现思路。从 1.1 版本开始，EasyPR 对车牌判断模块进行了优化，使得模型最后的效果有了较大的改善。

具体说来，本部分主要包括如下几个子部分:1.RBF 核；2.参数调优；3.特征提取；4.接口函数；5.自动化。

下面分别对这几个子部分展开介绍。

1.RBF 核

SVM 中最关键的技巧是核技巧。"核"其实是一个函数，通过一些转换规则把低维的数据映射为高维的数据。在机器学习里，数据跟向量是等同的意思。例如，一个 [174, 72]表示人的身高与体重的数据就是一个两维的向量。在这里，维度代表的是向量的长度。（务必要区分"维度"这个词在不同语境下的含义，有的时候我们会说向量是一维的，矩阵是二维的，这种说法针对的是数据展开的层次。机器学习里讲的维度代表的是向量的长度，与前者不同）

简单来说，低维空间到高维空间映射带来的好处就是可以利用高维空间的线型切割模拟低维空间的非线性分类效果。也就是说，SVM 模型其实只能做线型分类，但是在线型分类前，它可以通过核技巧把数据映射到高维，然后在高维空间进行线型切割。高维空间的线型切割完后在低维空间中最后看到的效果就是划出了一条复杂的分线型分类界限。**从这点来看，SVM 并没有完成真正的非线性分类，而是通过其它方式达到了类似目的，可谓"曲径通幽"。**

SVM 模型总共可以支持多少种核呢。根据官方文档，支持的核类型有以下几种：

1. liner 核，也就是无核。
2. rbf 核，使用的是高斯函数作为核函数。
3. poly 核，使用多项式函数作为核函数。
4. sigmoid 核，使用 sigmoid 函数作为核函数。

liner 核和 rbf 核是所有核中应用最广泛的。

liner 核，虽然名称带核，但它其实是无核模型，也就是没有使用核函数对数据进行转换。因此，它的分类效果仅仅比逻辑回归好一点。在 EasyPR1.0 版中，我们的 SVM 模型应用的是 liner 核。我们用的是图像的全部像素作为特征。

rbf 核，会将输入数据的特征维数进行一个维度转换，具体会转换为多少维？这个等于你输入的训练量。假设你有 500 张图片，rbf 核会把每张图片的数据转 换为 500 维的。如果你有 1000 张图片，rbf 核会把每幅图片的特征转到 1000 维。这么说来，随着你输入训练数据量的增长，数据的维数越多。更方便在高维空间下的分类效果，因此最后模型效果表现较好。

既然选择 SVM 作为模型，而且 SVM 中核心的关键技巧是核函数，那么理应使用带核的函数模型，充分利用数据高维化的好处，利用高维的线型分类带来低维空间下的非线性分类效果。但是，rbf 核的使用是需要条件的。

当你的数据量很大，但是每个数据量的维度一般时，才适合用 rbf 核。相反，当你的数据量不多，但是每个数据量的维数都很大时，适合用线型核。

在 EasyPR1.0 版中，我们用的是图像的全部像素作为特征，那么根据车牌图像的 136×36 的大小来看的话，就是 4896 维的数据，再加上我们输入的 是彩色图像，也就是说有 R，G，B 三个通道，那么数量还要乘以 3，也就是 14688 个维度。这是一个非常庞大的数据量，你可以把每幅图片的数据理解为长度 为 14688 的向量。这个时候，每个数据的维度很大，而数据的总数很少，如果用 rbf 核的话，相反效果反而不如无核。

在 EasyPR1.1 版本时，输入训练的数据有 3000 张图片，每个数据的特征改用直方统计，共有 172 个维度。这个场景下，如果用 rbf 核的话，就会将每个数据的维度转化为与数据总数一样的数量，也就是 3000 的维度，可以充分利用数据高维化后的好处。

因此可以看出，为了让 EasyPR 新版使用 rbf 核技巧，我们给训练数据做了增加，扩充了两倍的数据，同时，减小了每个数据的维度。以此满足了 rbf 核的使用条件。通过使用 rbf 核来训练，充分发挥了非线性模型分类的优势，因此带来了较好的分类效果。

但是，使用 rbf 核也有一个问题，那就是参数设置的问题。在 rbf 训练的过程中，参数的选择会显著的影响最后 rbf 核训练出模型的效果。因此必须对参数进行最优选择。

2.参数调优

传统的参数调优方法是人手完成的。机器学习工程师观察训练出的模型与参数的对应关系，不断调整，寻找最优的参数。由于机器学习工程师大部分时间在调整模型的参数，也有了**"机器学习就是调参"**这个说法。

幸好，opencv 的 svm 方法中提供了一个自动训练的方法。也就是由 opencv 帮你，不断改变参数，训练模型，测试模型，最后选择模型效果最好的那些参数。整个过程是全自动的，完全不需要你参与，你只需要输入你需要调整参数的参数类型，以及每次参数调整的步长即可。

现在有个问题，如何验证 svm 参数的效果？你可能会说，使用训练集以外的那 30%测试集啊。但事实上，机器学习模型中专门有一个数据集，是用来验证参数效果的。也就是交叉验证集(cross validation set，简称 validate data) 这个概念。

validate data 就是专门从 train data 中取出一部分数据，用这部分数据来验证参数调整的效果。比方说现在有 70%的训练数据，从中取出 20%的数据，剩下 50%数据用来训练，再用训练出来的模型在 20%数据上进行测试。这 20%的数据就叫做 validate data。真正拿来训练的数据仅仅只是 50%的数据。

正如上面把数据划分为 test data 和 train data 的理由一样。为了验证参数在新数据上的推广性，我们不能用一个训练数据集，所以我们要把训练数据集再细分为 train data 和 validate data。在 train data 上训练，然后在 validate data 上测试参数的效果。所以说，在一个更一般的机器学习场景中，机器学习工程师会把数据分为 train data，validate data，以及 test data。在 train data 上训练模型，用 validate data 测试参数，最后用 test data 测试模型和参数的整体表现。

说了这么多，那么，大家可能要问，是不是还缺少一个数据集，需要再划分出来一个 validate data 吧。但是答案是 No。opencv 的 train_auto 函数帮你完成了所有工作，你只需要告诉它，你需要划分多少个子分组，以及 validate data 所占的比例。然后 train_auto 函数会自动帮你从你输入的 train data 中划分出一部分的 validate data，然后自动测试，选择表现效果最好的参数。

感谢 train_auto 函数！既帮我们划分了参数验证的数据集，还帮我们一步步调整参数，最后选择效果最好的那个参数，可谓是节省了调优过程中 80%的工作。

train_auto 函数的调用代码如下：

```
svm.train_auto(trainingData, classes, Mat(), Mat(), SVM_params, 10,
               CvSVM::get_default_grid(CvSVM::C),
               CvSVM::get_default_grid(CvSVM::GAMMA),
               CvSVM::get_default_grid(CvSVM::P),
               CvSVM::get_default_grid(CvSVM::NU),
               CvSVM::get_default_grid(CvSVM::COEF),
               CvSVM::get_default_grid(CvSVM::DEGREE),
               true);
```

你唯一需要做的就是泡杯茶，翻翻书，然后慢慢等待这计算机帮你处理好所有事情(时间较长，因为每次调整参数又得重新训练一次)。作者最近的一次训练的耗时为 1 个半小时)。

训练完毕后，看看模型和参数在 test data 上的表现把。99%的 precise 和 98%的 recall。非常棒，比任何一次手工配的效果都好。

3.特征提取

在 rbf 核介绍时提到过，输入数据的特征的维度现在是 172，那么这个数字是如何计算出来的？现在的特征用的是直方统计函数，也就是先把图像二值化，然后统计图像中一行元素中 1 的数目，由于输入图像有 36 行，因此有 36 个值，再统计图像中每一列中 1 的数目，图像有 136 列，因此有 136 个值，两者相加正好等于 172。新的输入数据的特征提取函数就是下面的代码：

```
// ！EasyPR 的 getFeatures 回调函数// ！本函数是获取垂直和水平的直方图图值 void getHistogramFeatur
es(const Mat& image, Mat& features)
{
    Mat grayImage;
    cvtColor(image, grayImage, CV_RGB2GRAY);
    Mat img_threshold;
    threshold(grayImage, img_threshold, 0, 255, CV_THRESH_OTSU+CV_THRESH_BINARY);
    features = getTheFeatures(img_threshold);
}
```

我们输入数据的特征不再是全部的三原色的像素值了，而是抽取过的一些特征。从原始的图像到抽取后的特征的过程就被称为特征提取的过程。在 1.0 版中没有特征提取的概念，是直接把图像中全部像素作为特征的。这要感谢群里的"如果有一天"同学，他坚持认为全部像素的输入是最低级的做法，认为用特征提取后的效果会好多。我问大概能到多少准确率，当时的准确率有 92%，我以为已经很高了，结果他说能到 99%。在半信半疑中我尝试了，果真如他所说，结合了 rbf 核与新特征训练的模型达到的 precise 在 99%左右，而且 recall 也有 98%，这真是个令人咋舌并且非常惊喜的成绩。

“如果有一天”建议用的是 SFIT 特征提取或者 HOG 特征提取，由于时间原因，这两者我没有实现，但是把函数留在了那里。留待以后有时间完成。在这个过程中，我充分体会到了开源的力量，如果不是把软件开源，如果不是有这么多优秀的大家一起讨论，这样的思路与改善是不可能出现的。

4.接口函数

由于有 SIFT 以及 HOG 等特征没有实现，而且未来有可能会有更多有效的特征函数出现。因此我把特征函数抽象为借口。使用回调函数的思路实现。所有回调函数的代码都在 feature.cpp 中，开发者可以实现自己的回调函数，并把它赋值给 EasyPR 中的某个函数指针，从而实现自定义的特征提取。也许你们会有更多更好的特征的想法与创意。

关于特征其实有更多的思考，原始的 SVM 模型的输入是图像的全部像素，正如人类在小时候通过图像识别各种事物的过程。后来 SVM 模型的输入是经过抽取的特 征。正如随着人类接触的事物越来越多，会发现单凭图像越来越难区分一些非常相似的东西，于是学会了总结特征。例如太阳就是圆的，黄色，在天空等，可以凭借 这些特征就进行区分和判断。

从本质上说，特征是区分事物的关键特性。这些特性，一定是从某些维度去看待的。例如，苹果和梨子，一个是绿色，一个是黄色，这就是颜色的维度；鱼和鸟，一个在水里，一个在空中，这是位置的区分，也就是空间的维度。**特征，是许多维度中最有区分意义的维度。**传统数据仓库中的 OLAP，也称为多维分析，提供了人类从多个维度观察，比较的能力。通过人类的观察比较，从多个维度中挑选出来的维度，就是要分析目标的特征。从这点来看，机器学习与多维分析有了关联。多维分析提供了选择特征的能力。而机器学习可以根据这些特征进行建模。

机器学习界也有很多算法，专门是用来从数据中抽取特征信息的。例如传统的 PCA(主成分分析)算法以及最近流行的深度学习中的 AutoEncoder(自动编码机)技术。这些算法的主要功能就是在数据中学习出最能够明显区分数据的特征，从而提升后续的机器学习分类算法的效果。

说一个特征学习的案例。作者买车时，经常会把大众的两款车--迈腾与帕萨特给弄混，因为两者实在太像了。大家可以到网上去搜一下这两车的图片。如果不依赖后排的文字，光靠外形实在难以将两车区分开来(虽然从生产商来说，前者是一汽大众生产的，产地在长春，后者是上海大众生产的，产地在上海。两个不同的公司，南北两个地方，相差了十万八千里)。后来我通过仔细观察，终于发现了一个明显区分两辆车的特征，后来我再也没有认错过。这个特征就是：迈腾的前脸有四条银杠，而帕萨特只有三条，迈腾比帕萨特多一条银杠。可以这么说，就是这么一条银杠，分割了北和南两个地方生产的汽车。

图 15 一条银杠，分割了“北”和“南”

在这里区分的过程，我是通过不断学习与研究才发现了这些区分的特征，这充分说明了事物的特征也是可以被学习的。如果让机器学习中的特征选择方法 PCA 和 AutoEncoder 来分析的话，按理来说它们也应该找出这条银杠，否则它们就无法做到对这两类最有效的分类与判断。如果没有找到的话，证明我们目前的特征选择算法还有很多的改进空间（与这个案例类似的还有大众的另两款车，高尔夫和 Polo。它们两

的区分也是用同样的道理。相比迈腾和帕萨特，高尔夫和 Polo 价格差别的更大，所以区分的特征也更有价值）。

5.自动化

最后我想简单谈一下 EasyPR1.1 新增的自动化训练功能与命令行。大家可能看到第二部分介绍 SVM 训练时我将过程分成了 5 个步骤。事实上，这些步骤中的很多过程是可以模块化的。一开始的时候我写一些不相关的代码函数，帮我处理各种需要解决的问题，例如数据的分组，打标签等等。但后来，我把思路理清后，我觉得这几个步骤中很多的代码都可以通用。于是我把一些步骤模块化出来，形成通用的函数，并写了一个命令行界面去调用它们。在你运行 EasyPR1.1 版后，在你看到的第一个命令行界面选择“3.SVM 训练过程”，你就可以看到这些全部的命令。

图 16　svm 训练命令行

这里的命令主要有 6 个部分。第一个部分是最可能需要修改代码的地方，因为每个人的原始数据(raw data)都是不一样的，因此你需要在 data_prepare.cpp 中找到这个函数，改写成适应你格式的代码。接下来的第二个部分以后的功能基本都可以复用。例如自动贴标签(注意贴完以后要人工核对一下)。

第三个到第六部分功能类似。如果你的数据还没分组，那么你执行 3 以后，系统自动帮你分组，然后训练，再测试验证。第四个命令行省略了分组过程。第五个命令行部分省略训练过程。第六个命令行省略了前面所有过程，只做最后模型的测试部分。

让我们回顾一下 SVM 调优的五个思路。第一部分是 rbf 核，也就是模型选择层次，根据你的实际环境选择最合适的模型。第二部分是参数调优，也就是参数优化层次，这部分的参数最好通过一个验证集来确认，也可以使用 opencv 自带的 train_auto 函数。第三部分是特征抽取部分，也就是特征甄选们，要能选择出最能反映数据本质区别的特征来。在这方面，pca 以及深度学习技术中的 autoencoder 可能都会有所帮助。第四部分是通用接口部分，为了给优化留下空间，需要抽象出接口，方便后续的改进与对比。第五部分是自动化部分，为了节省时间，将大量可以自动化处理的功能模块化出来，然后提供一些方便的操作界面。前三部分是从机器学习的效果来提高，后两部分是从软件工程的层面去优化。

总结起来，就是**模型，参数，特征，接口，模块**五个层面。通过这五个层面，可以有效的提高机器学习模型训练的效果与速度，从而降低机器学习工程实施的难度与提升相关的效率。当需要对机器学习模型进行调优的时候，我们可以从这五个层面去考虑。