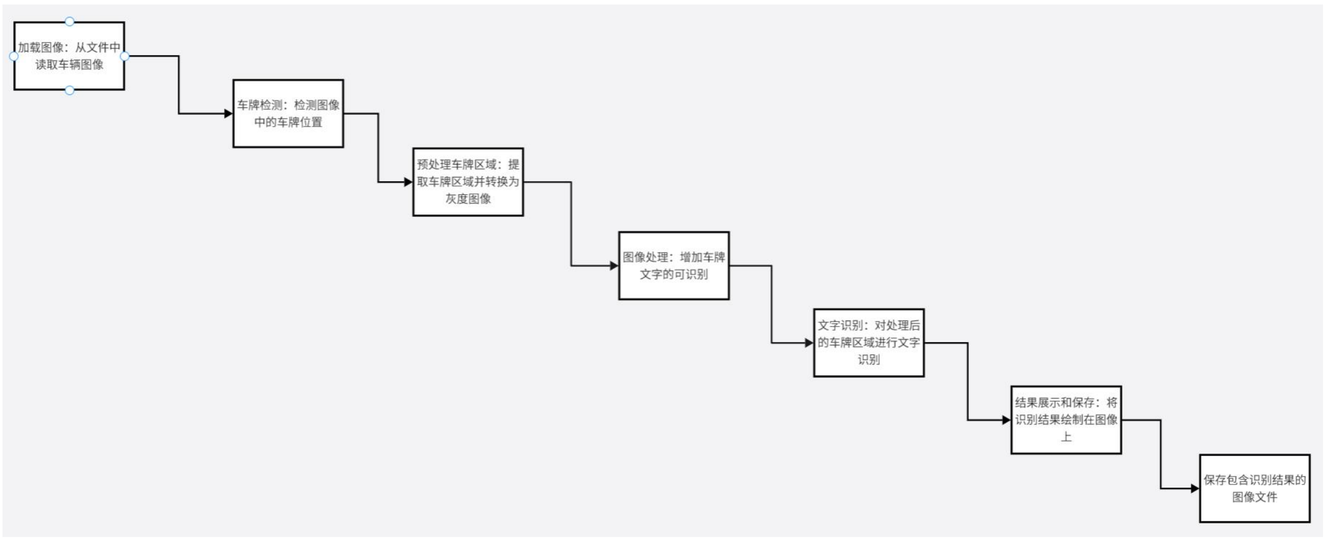
实验三车辆检测实验

1. 实验流程


加载图像：从文件中读取车辆图像
车牌检测：检测图像中的车牌位置
预处理车牌区域：提取车牌区域并转换为灰度图像
图像处理：增加车牌文字的可识别
文字识别：对处理后的车牌区域进行文字识别
结果展示和保存：将识别结果绘制在图像上
保存包含识别结果的图像文件

1.加载图像：从文件中读取车辆图像。
2.车牌检测：使用级联分类器检测图像中的车牌位置。
3.预处理车牌区域：提取车牌区域并转换为灰度图像。
根据颜色判断是否需要反转二值化处理。
4.文字识别：使用 PaddleOCR 对处理后的车牌区域进行文字识别。
5.结果展示和保存：将识别结果绘制在图像上。
保存包含识别结果的图像文件。

2. 实验结果

最终实现的代码如下，使用了 yolov5 算法框定识别车牌区域，使用 LPRNet 进行文字识别，并且可以动态地进行识别过程

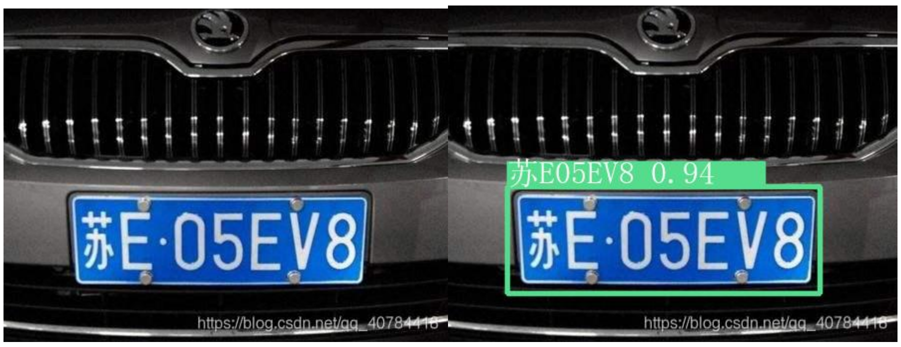
```
import argparse


import torch.backends.cudnn as cudnn

from models.experimental import *

from utils.datasets import *

from utils.utils import *

from models.LPRNet import *

def detect(save_img=False):

    classify, out, source, det_weights, rec_weights, view_img, save_txt, imgsz = \

        opt.classify, opt.output, opt.source, opt.det_weights, opt.rec_weights,  opt.view_img, opt.save_txt,
opt.img_size

    webcam = source == '0' or source.startswith('rtsp') or source.startswith('http') or source.endswith('.txt')
```

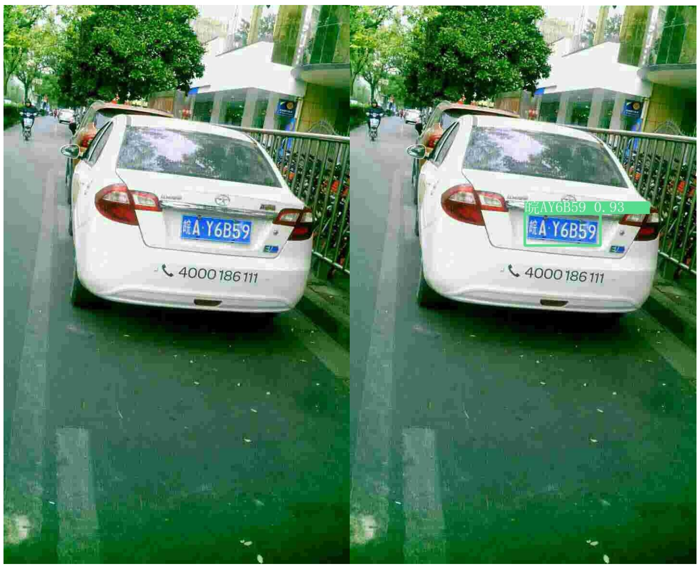
```
# Initialize
device = torch_utils.select_device(opt.device)
if os.path.exists(out):
    shutil.rmtree(out)  # delete rec_result folder
os.makedirs(out)  # make new rec_result folder
half = device.type != 'cpu'  # half precision only supported on CUDA
# Load yolov5 model
model = attempt_load(det_weights, map_location=device)  # load FP32 model
print("load det pretrained model successful!")
imgsz = check_img_size(imgsz, s=model.stride.max())  # check img_size
if half:
    model.half()  # to FP16
# Second-stage classifier  也就是 rec 字符识别
if classify:
    modelc = LPRNet(lpr_max_len=8, phase=False, class_num=len(CHARS), dropout_rate=0).to(device)
    modelc.load_state_dict(torch.load(rec_weights, map_location=torch.device('cpu')))
    print("load rec pretrained model successful!")
    modelc.to(device).eval()
# Set Dataloader
vid_path, vid_writer = None, None
if webcam:
    view_img = True
    cudnn.benchmark = True  # set True to speed up constant image size demo
    dataset = LoadStreams(source, img_size=imgsz)
else:
    save_img = True
    dataset = LoadImages(source, img_size=imgsz)
```

```
# Get names and colors
names = model.module.names if hasattr(model, 'module') else model.names
colors = [[random.randint(0, 255) for _ in range(3)] for _ in range(len(names))]
# Run demo
t0 = time.time()
img = torch.zeros((1, 3, imgsz, imgsz), device=device)  # init img
_ = model(img.half() if half else img) if device.type != 'cpu' else None  # run once
for path, img, im0s, vid_cap in dataset:
    img = torch.from_numpy(img).to(device)
    img = img.half() if half else img.float()  # uint8 to fp16/32
    img /= 255.0  # 0 - 255 to 0.0 - 1.0
    if img.ndimension() == 3:
        img = img.unsqueeze(0)
    # Inference
    t1 = torch_utils.time_synchronized()
```

```
        pred = model(img, augment=opt.augment)[0]
        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes,
agnostic=opt.agnostic_nms)
        # Apply Classifier
        if classify:
            pred, plat_num = apply_classifier(pred, modelc, img, im0s)
        t2 = torch_utils.time_synchronized()
        # Process detections
        for i, det in enumerate(pred):  # detections per image
            if webcam:  # batch_size >= 1
                p, s, im0 = path[i], '%g: ' % i, im0s[i].copy()
            else:
                p, s, im0 = path, '', im0s
            save_path = str(Path(out) / Path(p).name)
            txt_path = str(Path(out) / Path(p).stem) + ('_%g' % dataset.frame if dataset.mode == 'video' else
'')
            s += '%gx%g ' % img.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            if det is not None and len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()
                # Print results
                for c in det[:, 5].unique():
                    n = (det[:, 5] == c).sum()  # detections per class
                    s += '%g %ss, ' % (n, names[int(c)])  # add to string
                # Write results
                for de, lic_plat in zip(det, plat_num):
                    # xyxy,conf,cls,lic_plat=de[:4],de[4],de[5],de[6:]
                    *xyxy, conf, cls=de
                    if conf > 0.5:  # 置信度过滤
                        if save_txt:  # Write to file
                            xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  #
normalized xywh
                            with open(txt_path + '.txt', 'a') as f:
                                f.write(('%g ' * 5 + '\n') % (cls, xywh))  # label format
                    if save_img or view_img:  # Add bbox to image
                        # label = '%s %.2f' % (names[int(cls)], conf)
                        lb = ""
                        for a,i in enumerate(lic_plat):
                            # if a ==0:
                            #     continue
                            lb += CHARS[int(i)]
                        label = '%s %.2f' % (lb, conf)
                        im0 = plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=3)
            # Print time (demo + NMS)
            print('%sDone. (%.3fs)' % (s, t2 - t1))
            # Stream results
            if view_img:
                cv2.imshow(p, im0)
                if cv2.waitKey(1) == ord('q'):  # q to quit
                    raise StopIteration
            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'images':
                    cv2.imwrite(save_path, im0)
                else:
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.release()  # release previous video writer
                        fourcc = 'mp4v'  # rec_result video codec
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*fourcc), fps, (w, h))
                    vid_writer.write(im0)
    if save_txt or save_img:
        print('Results saved to %s' % os.getcwd() + os.sep + out)
        if platform == 'darwin':  # MacOS
            os.system('open ' + save_path)
    print('Done. (%.3fs)' % (time.time() - t0))
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--classify', nargs='+', type=str, default=True, help='True rec')
    parser.add_argument('--det-weights', nargs='+', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/weights/yolov5_best.pt',
help='model.pt path(s)')
    parser.add_argument('--rec-weights', nargs='+', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/weights/lprnet_best.pth',
help='model.pt path(s)')
    parser.add_argument('--source', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/demo/images',
help='source')  # file/folder, 0 for webcam
    parser.add_argument('--output', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/demo/rec_result',
help='rec_result folder')  # rec_result folder
    parser.add_argument('--img-size', type=int, default=640, help='demo size (pixels)')
```

```python
parser.add_argument('--conf-thres', type=float, default=0.4, help='object confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.5, help='IOU threshold for NMS')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--view-img', action='store_true', help='display results')
parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
parser.add_argument('--classes', nargs='+', type=int, help='filter by class')
parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
parser.add_argument('--augment', action='store_true', help='augmented demo')
parser.add_argument('--update', action='store_true', help='update all models')
opt = parser.parse_args()
print(opt)
with torch.no_grad():
    if opt.update:  # update all models (to fix SourceChangeWarning)
        for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt', 'yolov3-spp.pt']:
            detect()
            create_pretrained(opt.weights, opt.weights)
    else:
        detect()
```


苏E05EV8 0.94
https://blog.csdn.net/qq_40784418
https://blog.csdn.net/qq_40784418



最后看到成功地识别出了对应的车牌区域，并且识别出了文字，苏 E05EV8 和皖 AY6B59

其中 yolov5 实现了车牌区域的框定，效果非常好。此外我还设置了极大值抑制地方法来筛除不是车牌区域的其他文字部分，如果置信度过低，直接省略，可以很好地只识别出车牌部分，而略去非车牌区域

整体的实现思路：
1.模型加载：
加载车牌检测模型（YOLOv5）。
如果需要字符识别，加载 LPRNet 模型。
2.数据加载：根据输入源是摄像头、视频还是图片，设置不同的数据加载方式。
3.检测与识别：
　　1）图像预处理：
　　将输入图像转换为模型所需的尺寸（imgsz），通常使用缩放或裁剪方法。将图像数据类型从 uint8 转换为浮点数，范围从[0, 255]变为[0.0, 1.0]。
　　2）模型推理：
　　将预处理后的图像数据输入到 YOLOv5 模型中进行车牌检测。模型输出包括检测框的坐标、置信度和类别概率。
　　3）非极大值抑制（NMS）：
　　对模型输出的多个检测框进行处理，去除重叠的检测框。NMS 通过保留置信度最高的检测框来减少冗余。
　　4）应用分类器（如果设置了字符识别）：
　　对于检测到的每个车牌区域，使用 LPRNet 模型进行字符识别。LPRNet 模型将车牌区域作为输入，输出字符类别的概率。
　　5）坐标转换：
　　将模型输出的检测框坐标从模型输入尺寸转换为原始图像尺寸，以便在原始图像上进行标记。
　　6）绘制检测框：
　　在原始图像上绘制检测框，并标注车牌类别和置信度。
　　7）字符识别：
　　对于每个检测到的车牌区域，使用 LPRNet 模型进行字符识别。模型将输出每个字符的概率分布。
　　8)结果整合：
　　将识别出的字符按照概率排序，选择概率最高的字符组成车牌号码。
　　9)后处理：
　　对识别结果进行后处理，如根据车牌号码的格式进行校验和修正。
4.结果保存：
将识别结果保存到指定的输出目录。
如果是视频输入，将处理后的视频帧写入视频文件。
5.性能评估：计算整个检测和识别过程的时间，并在控制台输出。
6.用户交互：如果是实时摄像头输入，使用 OpenCV 显示结果，并允许用户通过按键退出。
7.清理资源：如果是视频处理，释放视频写入器资源。

3.实验结果分析

对基本实验和进行优化处理后的结果进行分析和思考。

1）基本实验

```python
import cv2
import numpy as np
import pytesseract
#调用级联器
plate_cascade = cv2.CascadeClassifier('./haarcascades/haarcascade_russian_plate_number.xml')

img=cv2.imread('./image/carnum2.png')
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plates=plate_cascade.detectMultiScale(gray,1.1,5)
for (x,y,w,h) in plates:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi=gray[y:y+h,x:x+w]
    #roi=cv2.resize(roi,(20,20))
    ret,roibinary=cv2.threshold(roi,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    cv2.imshow('roibinary',roibinary)
    # cv2.imshow('roi',roi)
    #print(pytesseract.image_to_string(roibinary,lang='chi_sim+eng',config='--psm 8 --oem 3'))
    print(pytesseract.image_to_string(roibinary,lang='chi_sim+eng',config='--psm 7 --oem 3 -c
tessedit_char_whitelist=0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ京津沪渝黑吉辽冀晋鲁豫陕陇宁甘青台蒙新藏桂粤湘
赣闽川黔滇皖鄂苏浙琼港澳'))
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

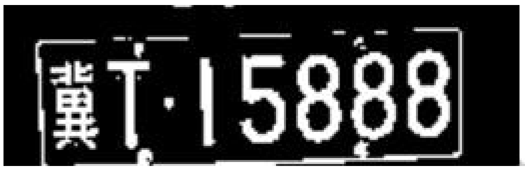
根据课程上所讲的内容，首先对加载的图片进行灰度化，然后使用级联器对车牌轮廓进行绘制，对识别区域进行 otsu 方法的自动查找阈值的二值化，然后使用 pytesseract 对车牌号内容进行识别，但是识别效果不好

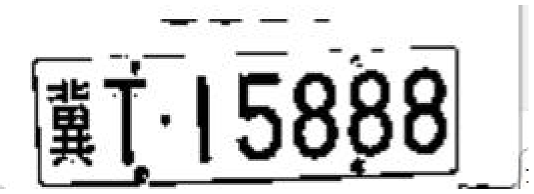
下面是我验收时使用的代码，进行了图像处理以及

```python
import cv2
import numpy as np
from paddleocr import PaddleOCR, draw_ocr
from PIL import Image

def is_green_plate(roi):
    """检查 ROI 区域是否为绿色车牌"""
    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    green_lower = np.array([35, 100, 100])
    green_upper = np.array([85, 255, 255])
    mask = cv2.inRange(hsv_roi, green_lower, green_upper)
    green_ratio = cv2.countNonZero(mask) / (roi.shape[0] * roi.shape[1])
    return green_ratio > 0.5

# 调用级联器
plate_cascade = 
cv2.CascadeClassifier('./haarcascades/haarcascade_russian_plate_number.
xml')

# 读取图像
img = cv2.imread('./image/carnum.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 使用级联器检测车牌
plates = plate_cascade.detectMultiScale(gray, 1.1, 5)

# 对每个检测到的车牌进行处理
for (x, y, w, h) in plates:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

    roi_color = img[y:y + h, x:x + w]
    roi_gray = gray[y:y + h, x:x + w]

    ret, roibinary = cv2.threshold(roi_gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
    cv2.imshow('roibinary', roibinary)

    # 检查车牌颜色，绿色车牌不进行反转
    if is_green_plate(roi_color):
        processed_img = roibinary
    else:
        processed_img = cv2.bitwise_not(roibinary)



    # 显示处理后的图像
    cv2.imshow('opened_img', processed_img)

    # 将处理后的图像保存为临时文件
    temp_img_path = './image/temp_roi.png'
    cv2.imwrite(temp_img_path, processed_img)

    # 使用 PaddleOCR 进行 OCR 识别
    ocr = PaddleOCR(use_angle_cls=True, lang="ch")
    result = ocr.ocr(temp_img_path, cls=True)

    # 打印 OCR 识别结果
    for idx in range(len(result)):
        res = result[idx]
        for line in res:
            print(line)

    # 绘制 OCR 结果
    result = result[0]
    image = Image.open(temp_img_path).convert('RGB')
    boxes = [line[0] for line in result]
    txts = [line[1][0] for line in result]
    scores = [line[1][1] for line in result]
    im_show = draw_ocr(image, boxes, txts, scores,
font_path='./fonts/simfang.ttf')
    im_show = Image.fromarray(im_show)
    im_show.save('result_roi.jpg')

# 显示结果
cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

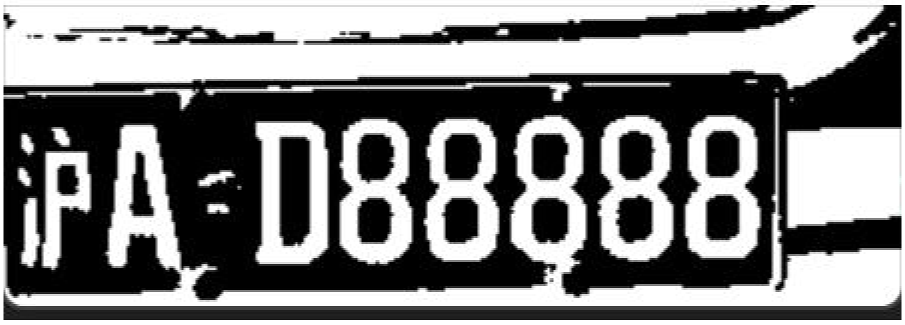
2）图像处理

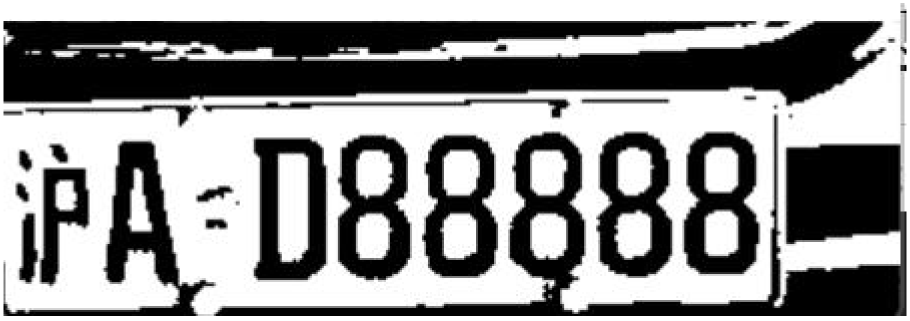
上课所讲内容中对车牌识别并不准确，因此我做了一些处理，首先我对二值化的图像进行反转，使得图像中的文字部分更加清晰，而且绿色车牌的颜色较浅，所以不需要进行反转





```
def is_green_plate(roi):
    """检查 ROI 区域是否为绿色车牌"""
    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    green_lower = np.array([35, 100, 100])
    green_upper = np.array([85, 255, 255])
    mask = cv2.inRange(hsv_roi, green_lower, green_upper)
    green_ratio = cv2.countNonZero(mask) / (roi.shape[0] * roi.shape[1])
    return green_ratio > 0.5

# 检查车牌颜色，绿色车牌不进行反转
    if is_green_plate(roi_color):
        processed_img = roibinary
    else:
        processed_img = cv2.bitwise_not(roibinary)
```





发现翻转后的图像具有很多的噪点，这些属于外部噪点，因此我们可以采用开运算

3）paddleOCR 识别

但是 tesseract 的识别效果仍然很差，所以我下载了 paddleocr 的内容进行识别，发现可以很好地识别出轮廓中的文本内容

```
from paddleocr import PaddleOCR, draw_ocr

# Paddleocr 目前支持的多语言语种可以通过修改 lang 参数进行切换
# 例如`ch`, `en`, `fr`, `german`, `korean`, `japan`
ocr = PaddleOCR(use_angle_cls=True, lang="ch")  # need to run only once to download and load model into memory
img_path = './image/carnum.png'
result = ocr.ocr(img_path, cls=True)
for idx in range(len(result)):
    res = result[idx]
    for line in res:
        print(line)
```

```
# 显示结果
from PIL import Image
result = result[0]
image = Image.open(img_path).convert('RGB')
boxes = [line[0] for line in result]
txts = [line[1][0] for line in result]
scores = [line[1][1] for line in result]
im_show = draw_ocr(image, boxes, txts, scores, font_path='./fonts/simfang.ttf')
im_show = Image.fromarray(im_show)
im_show.save('result.jpg')
```

4）角点检测和透视变换进行车牌摆正

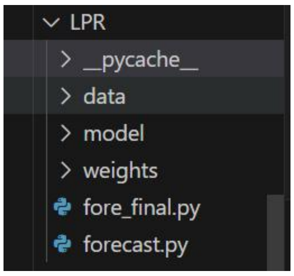
此外我还进行了角点检测和透视变换的操作，想要将车牌进行摆正，但是效果并不是很好

```
def order_points(pts):#确定四个顶点函数
    rect = np.zeros((4, 2), dtype="float32")
    s = pts.sum(axis=1)
    rect[0] = pts[np.argmin(s)]
    rect[2] = pts[np.argmax(s)]

    diff = np.diff(pts, axis=1)
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]

    return rect
def four_point_transform(image, pts):#将区域透视变换为一个矩形视图
    rect = order_points(pts)#得到四个顶点
    (tl, tr, br, bl) = rect

    widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
    widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
    maxWidth = max(int(widthA), int(widthB))

    heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
    heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
    maxHeight = max(int(heightA), int(heightB))

    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]], dtype="float32")
```

```python
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

    return warped

def scan_image(image_path):#图像扫描
    image = cv2.imread(image_path)
    orig = image.copy()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)#转为灰度图 可以简化后续处理
    gray = cv2.GaussianBlur(gray, (5, 5), 0)#使用高斯模糊 平滑图像 去除噪声
#图像膨胀+边缘检测
    element = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    gray = cv2.dilate(gray, element)

    edged = cv2.Canny(gray, 30, 120, 3)

    cnts, _ = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)#找到轮廓，用 cnts 保存
    cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]#找到最大的五个轮廓

    screenCnt = None
    for c in cnts:
        peri = cv2.arcLength(c, True)#计算轮廓的周长
        approx = cv2.approxPolyDP(c, 0.02 * peri, True)#多边形逼近，得到近似的多边形

        if len(approx) == 4:#如果逼近多边形有四个顶点，则认为找到了目标轮廓，就退出
            screenCnt = approx
            break

    if screenCnt is None:
        raise Exception("No contour of four points found")#如果没有就输出

    warped = four_point_transform(orig, screenCnt.reshape(4, 2))#透视变换变换成四边形图形
    cv2.imshow("Scanned", warped)
    warped_gray = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)#转为灰度图
    cv2.imshow("Scanned_gray", warped_gray)
    _, scanned = cv2.threshold(warped_gray, 150, 255, cv2.THRESH_BINARY)#灰度图二值化
    cv2.imshow("Scanned_binary", scanned)

    height, width = scanned.shape
    print(f"Scanned image size: {width}x{height}")
    if height>width:
    # 逆时针旋转 90 度
        scanned_final = cv2.rotate(scanned,
cv2.ROTATE_90_COUNTERCLOCKWISE)
        cv2.imshow("Scanned_final", scanned_final)
    else:
        scanned_final = scanned
        cv2.imshow("Scanned_final", scanned_final)
```

5）使用 LPRNET 进行训练模型，来直接识别模型

因为模型训练需要使用 CUDA 进行加速，所以我将它放置到服务器上进行运行

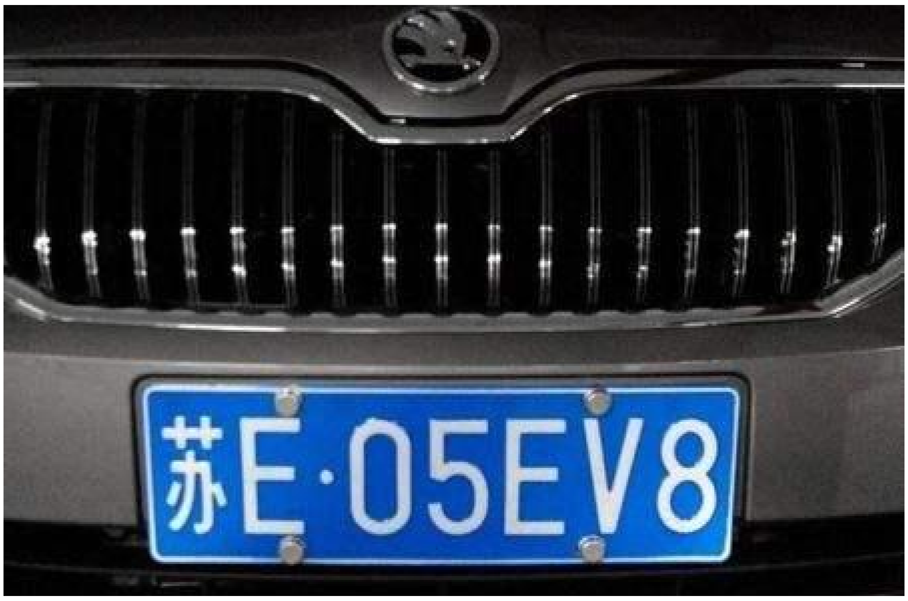


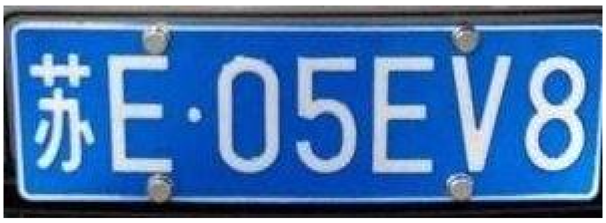
其中/data 中是我们的数据集以及加载数据到模型中的一系列操作
/model 中有 LPRNet.py，是 LPRNet 网络的构建，也是我们训练模型的基本框架
/weights 中是我们的网络参数
forecast.py 是我们预测的过程，我们将它整合成一个在 fore_final.py 中使用的一个 predict()函数

```
import cv2
import hyperlpr3 as lpr3
from forecast import predict  # 根据你的实际文件结构导入 predict 函数
import os


# 使用 HyperLPR v3 识别车牌区域
def detect_and_recognize_plate(image):
    # 实例化 LicensePlateCatcher 对象
    catcher = lpr3.LicensePlateCatcher(detect_level=lpr3.DETECT_LEVEL_HIGH)
    # 获取车牌识别结果
    plate_infos = catcher(image)
    results = []
    for plate_info in plate_infos:
        plate, confidence, plate_type, box = plate_info
        if confidence > 0.5:  # 设置置信度阈值
            x_min, y_min, x_max, y_max = box
            plate_image = image[y_min:y_max, x_min:x_max]

            # 将车牌图片调整为目标大小 94x24
            plate_image_resized = cv2.resize(plate_image, (94, 24))

            # 保存调整大小后的图片
            rec_path = './outputs/plate_image_resized.png'  # 设置正确的输出路径和文件名
            cv2.imwrite(rec_path, plate_image_resized)
            results.append((plate, confidence, plate_image_resized))

    return results
# 读取输入图像
img_path = './image/carnum3.png'
image = cv2.imread(img_path)
# 检测和识别车牌
results = detect_and_recognize_plate(image)
```

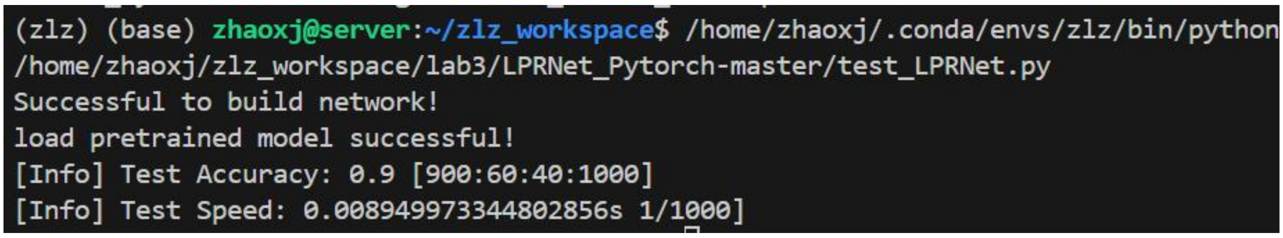
```
output_dir = './outputs/'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
# 显示识别结果
for idx, (plate_text, confidence, plate_image) in enumerate(results):
    print(f"Plate {idx + 1}: {plate_text}, Confidence: {confidence}")
```

这里使用了 HyperLPR 的库，用来进行车牌的定位与分割，成功得到了效果很好的分割图



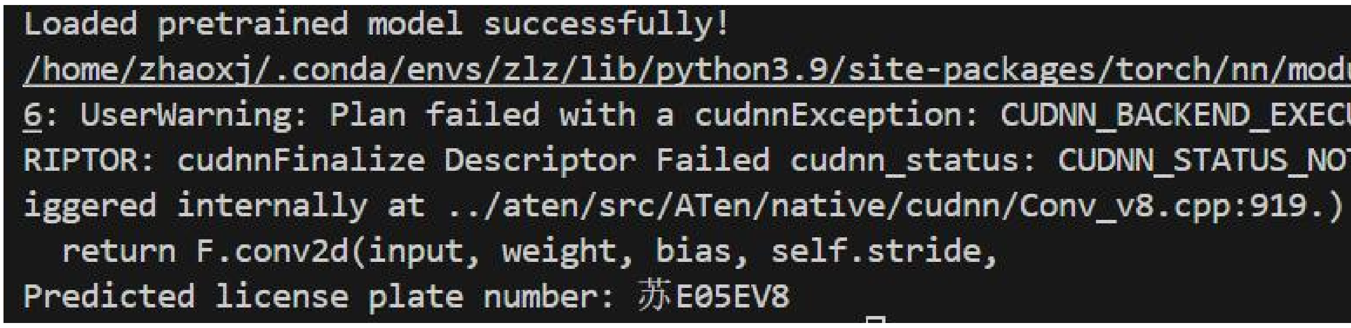


可以看到分割的效果整体不错
接着我使用 LPRNet 来进行车牌识别，LPRNet 直接针对车牌进行识别，LPRNet 的效果很好，



```
(zlz) (base) zhaoxj@server:~/zlz_workspace$ /home/zhaoxj/.conda/envs/zlz/bin/python
/home/zhaoxj/zlz_workspace/lab3/LPRNet_Pytorch-master/test_LPRNet.py
Successful to build network!
load pretrained model successful!
[Info] Test Accuracy: 0.9 [900:60:40:1000]
[Info] Test Speed: 0.008949973344802856s 1/1000]
```

整体的 accuracy 是 0.9



```
Loaded pretrained model successfully!
/home/zhaoxj/.conda/envs/zlz/lib/python3.9/site-packages/torch/nn/modu
6: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECU
RIPTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NO
iggered internally at ../aten/src/ATen/native/cudnn/Conv_v8.cpp:919.)
  return F.conv2d(input, weight, bias, self.stride,
Predicted license plate number: 苏E05EV8
```

最后看到成功地识别出了苏 E05EV8

6）使用 yolov5 模型识别车牌区域

我们在上面使用的是 HyperLPR 进行车牌区域的识别，相当于直接使用了库中的函数，但是为了更加地“独立自主”，我们尝试使用 yolov5 算法进行识别

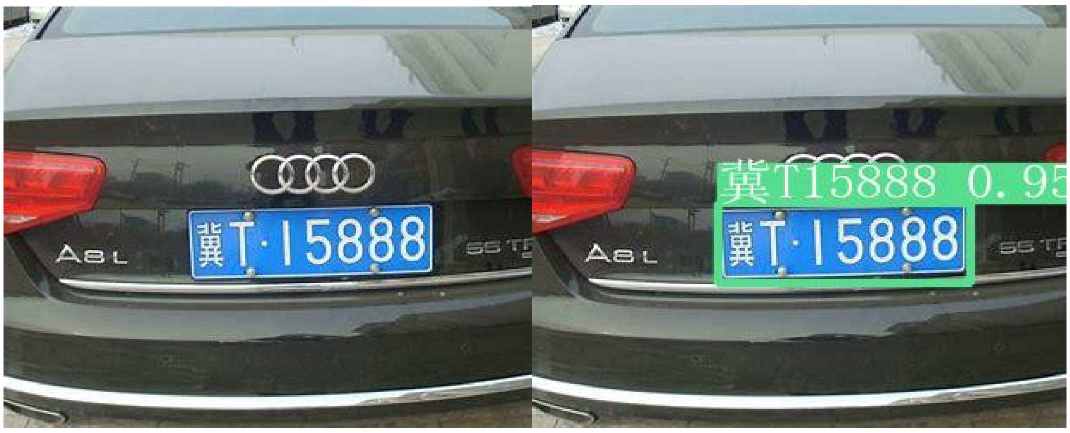
```python
import argparse
import torch.backends.cudnn as cudnn
from models.experimental import *
from utils.datasets import *
from utils.utils import *
from models.LPRNet import *
def detect(save_img=False):
    classify, out, source, det_weights, rec_weights, view_img, save_txt, imgsz = \
        opt.classify, opt.output, opt.source, opt.det_weights, opt.rec_weights,  opt.view_img, opt.save_txt,
opt.img_size
    webcam = source == '0' or source.startswith('rtsp') or source.startswith('http') or source.endswith('.txt')
    # Initialize
    device = torch_utils.select_device(opt.device)
    if os.path.exists(out):
        shutil.rmtree(out)  # delete rec_result folder
    os.makedirs(out)  # make new rec_result folder
    half = device.type != 'cpu'  # half precision only supported on CUDA
    # Load yolov5 model
    model = attempt_load(det_weights, map_location=device)  # load FP32 model
    print("load det pretrained model successful!")
    imgsz = check_img_size(imgsz, s=model.stride.max())  # check img_size
    if half:
        model.half()  # to FP16
    # Second-stage classifier  也就是 rec 字符识别
    if classify:
        modelc = LPRNet(lpr_max_len=8, phase=False, class_num=len(CHARS), dropout_rate=0).to(device)
        modelc.load_state_dict(torch.load(rec_weights, map_location=torch.device('cpu')))
        print("load rec pretrained model successful!")
        modelc.to(device).eval()
    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = True
        cudnn.benchmark = True  # set True to speed up constant image size demo
        dataset = LoadStreams(source, img_size=imgsz)
    else:
        save_img = True
        dataset = LoadImages(source, img_size=imgsz)
    # Get names and colors
    names = model.module.names if hasattr(model, 'module') else model.names
    colors = [[random.randint(0, 255) for _ in range(3)] for _ in range(len(names))]
    # Run demo
    t0 = time.time()
    img = torch.zeros((1, 3, imgsz, imgsz), device=device)  # init img
    _ = model(img.half() if half else img) if device.type != 'cpu' else None  # run once
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)
        # Inference
        t1 = torch_utils.time_synchronized()
        pred = model(img, augment=opt.augment)[0]
        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes,
agnostic=opt.agnostic_nms)
        # Apply Classifier
        if classify:
            pred, plat_num = apply_classifier(pred, modelc, img, im0s)
        t2 = torch_utils.time_synchronized()
        # Process detections
        for i, det in enumerate(pred):  # detections per image
            if webcam:  # batch_size >= 1
                p, s, im0 = path[i], '%g: ' % i, im0s[i].copy()
            else:
                p, s, im0 = path, '', im0s
            save_path = str(Path(out) / Path(p).name)
            txt_path = str(Path(out) / Path(p).stem) + ('_%g' % dataset.frame if dataset.mode == 'video' else
'')
            s += '%gx%g ' % img.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            if det is not None and len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()
                # Print results
                for c in det[:, 5].unique():
                    n = (det[:, 5] == c).sum()  # detections per class
                    s += '%g %ss, ' % (n, names[int(c)])  # add to string
                # Write results
                for de, lic_plat in zip(det, plat_num):
                    # xyxy,conf,cls,lic_plat=de[:4],de[4],de[5],de[6:]
                    *xyxy, conf, cls=de
```

```
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized
xywh
                        with open(txt_path + '.txt', 'a') as f:
```

```python
                            f.write(('%g ' * 5 + '\n') % (cls, xywh))  # label format
                    if save_img or view_img:  # Add bbox to image
                        # label = '%s %.2f' % (names[int(cls)], conf)
                        lb = ""
                        for a,i in enumerate(lic_plat):
                            # if a ==0:
                            #     continue
                            lb += CHARS[int(i)]
                        label = '%s %.2f' % (lb, conf)
                        im0 = plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=3)
            # Print time (demo + NMS)
            print('%sDone. (%.3fs)' % (s, t2 - t1))
            # Stream results
            if view_img:
                cv2.imshow(p, im0)
                if cv2.waitKey(1) == ord('q'):  # q to quit
                    raise StopIteration
            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'images':
                    cv2.imwrite(save_path, im0)
                else:
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.release()  # release previous video writer
                        fourcc = 'mp4v'  # rec_result video codec
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*fourcc), fps, (w, h))
                    vid_writer.write(im0)
    if save_txt or save_img:
        print('Results saved to %s' % os.getcwd() + os.sep + out)
        if platform == 'darwin':  # MacOS
            os.system('open ' + save_path)
    print('Done. (%.3fs)' % (time.time() - t0))
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--classify', nargs='+', type=str, default=True, help='True rec')
    parser.add_argument('--det-weights', nargs='+', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/weights/yolov5_best.pt',
help='model.pt path(s)')
```

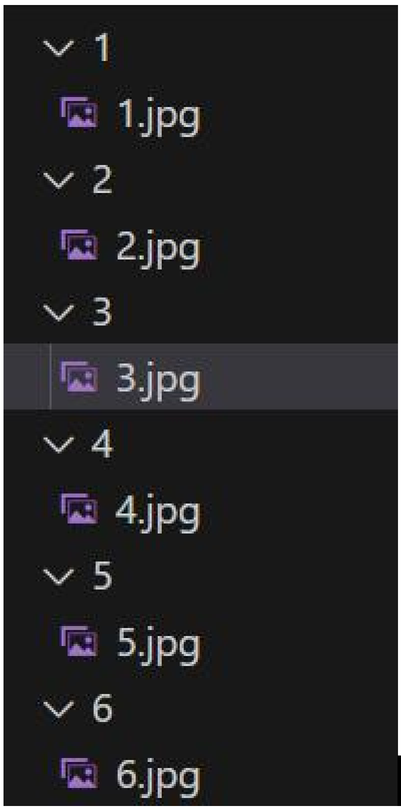
```
    parser.add_argument('--rec-weights', nargs='+', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/weights/lprnet_best.pth',
help='model.pt path(s)')
    parser.add_argument('--source', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/demo/images',
help='source')  # file/folder, 0 for webcam
    parser.add_argument('--output', type=str,
default=r'/home/zhaoxj/zlz_workspace/YOLOv5-LPRNet-Licence-Recognition-master/demo/rec_result',
help='rec_result folder')  # rec_result folder
    parser.add_argument('--img-size', type=int, default=640, help='demo size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.4, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.5, help='IOU threshold for NMS')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented demo')
    parser.add_argument('--update', action='store_true', help='update all models')
    opt = parser.parse_args()
    print(opt)
    with torch.no_grad():
        if opt.update:  # update all models (to fix SourceChangeWarning)
            for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt', 'yolov3-spp.pt']:
                detect()
                create_pretrained(opt.weights, opt.weights)
        else:
            detect()
```

我们分别加载 yolov5 模型和 LPRNet 模型,通过 yolov5 模型框定车牌区域，然后再使用 LRPRNet 进行车牌识别,最终看到的效果很不错，成功使用了 yolov5 模型

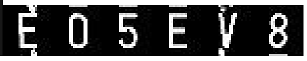


7）车牌定位与文字分割

```python
import matplotlib.pyplot as plt
import numpy as np
import torch
import cv2
import os


def find_card(I):
    # 识别出车牌区域并返回该区域的图像
    [y, x, z] = I.shape
    # y 取值范围分析
    Blue_y = np.zeros((y, 1))
    for i in range(y):
        for j in range(x):
            # 蓝色 rgb 范围
            temp = I[i, j, :]
            if (I[i, j, 2] <= 30) and (I[i, j, 0] >= 119):
                Blue_y[i][0] += 1
    MaxY = np.argmax(Blue_y)
    PY1 = MaxY
    while (Blue_y[PY1, 0] >= 5) and (PY1 > 0):
        PY1 -= 1
    PY2 = MaxY
    while (Blue_y[PY2, 0] >= 5) and (PY2 < y - 1):
        PY2 += 1
    # x 取值
    Blue_x = np.zeros((1, x))
    for i in range(x):
        for j in range(PY1, PY2):
            if (I[j, i, 2] <= 30) and (I[j, i, 0] >= 119):
                Blue_x[0][i] += 1
    PX1 = 0
    while (Blue_x[0, PX1] < 3) and (PX1 < x - 1):
        PX1 += 1
    PX2 = x - 1
    while (Blue_x[0, PX2] < 3) and (PX2 > PX1):
        PX2 -= 1
    # 对车牌区域的修正
    PX1 -= 2
    PX2 += 2
    return I[PY1:PY2, PX1 - 2: PX2, :]
def divide(I):
    [y, x, z] = I.shape
    White_x = np.zeros((x, 1))
    for i in range(x):
        for j in range(y):
            if I[j, i, 1] > 176:
                White_x[i][0] += 1
    return White_x
def divide_each_character(I):
    [y, x, z] = I.shape
    White_x = np.zeros((x, 1))
    for i in range(x):
        for j in range(y):
            if I[j, i, 1] > 176:
                White_x[i][0] += 1
    res = []
    length = 0
    for i in range(White_x.shape[0]):
        # 使用超参数经验分割
        t = I.shape[1] / 297
        num = White_x[i]
        if num > 8:
            length += 1
        elif length > 20 * t:
            res.append([i - length - 2, i + 2])
            length = 0
        else:
            length = 0
    return res
if __name__ == '__main__':
    I = cv2.imread('./image/carnum3.png')
    Plate = find_card(I)
    # White_x = divide(Plate)
    plt.imshow(Plate)
    plt.show()
    # plt.plot(np.arange(Plate.shape[1]), White_x)
    res = divide_each_character(Plate)
    plate_save_path = './image/'
    for t in range(len(res)):
        plt.subplot(1, 7, t + 1)
        temp = res[t]
        save_img = cv2.cvtColor(Plate[:, temp[0]:temp[1], :],cv2.COLOR_BGR2GRAY)
        ma = max(save_img.shape[0], save_img.shape[1])
        mi = min(save_img.shape[0], save_img.shape[1])
        ans = np.zeros(shape=(ma, ma, 3),dtype=np.uint8)
        start =int(ma/2-mi/2)
        for i in range(mi):
            for j in range(ma):
                if save_img[j,i] > 125:
                    for k in range(3):
                        ans[j,start+i,k]=255
        ans=cv2.merge([ans[:,:,0],ans[:,:,1],ans[:,:,2]])
        ans=cv2.resize(ans,(25,25))
        dir_name=plate_save_path+str(t)
        os.mkdir(dir_name)
        cv2.imwrite(dir_name+'/'+str(t)+'.jpg',ans)
        plt.imshow(ans)
    plt.show()
```



```
∨ 1
    1.jpg
∨ 2
    2.jpg
∨ 3
    3.jpg
∨ 4
    4.jpg
∨ 5
    5.jpg
∨ 6
    6.jpg
```



成功进行了分割，但是对于某些车牌，文字分割效果很差

8）进行图像处理并且对汉字和英文数字分别匹配

```
# 导入所需模块
import cv2
from matplotlib import pyplot as plt
import os
import numpy as np
```

```python
# plt 显示彩色图片
def plt_show0(img):
#cv2 与 plt 的图像通道不同：cv2 为[b,g,r];plt 为[r, g, b]
    b,g,r = cv2.split(img)
    img = cv2.merge([r, g, b])
    plt.imshow(img)
    plt.show()

# plt 显示灰度图片
def plt_show(img):
    plt.imshow(img,cmap='gray')
    plt.show()
```

```python
# 图像去噪灰度处理
def gray_guss(image):
    if len(image.shape) == 3 and image.shape[2] == 3:  # 确保图像是彩色的，且有三个通道
        image = cv2.GaussianBlur(image, (3, 3), 0)
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        # 如果图像已经是灰度的，直接使用
        gray_image = image
    return gray_image
# 读取待检测图片
origin_image = cv2.imread('./image/carnum3.png')
# 复制一张图片，在复制图上进行图像操作，保留原图
image = origin_image.copy()
# 图像去噪灰度处理
gray_image = gray_guss(image)
# x 方向上的边缘检测（增强边缘信息）
Sobel_x = cv2.Sobel(gray_image, cv2.CV_16S, 1, 0)
absX = cv2.convertScaleAbs(Sobel_x)
image = absX
```

```python
# 图像阈值化操作——获得二值化图
ret, image = cv2.threshold(image, 0, 255, cv2.THRESH_OTSU)
# 显示灰度图像
plt_show(image)
# 形态学（从图像中提取对表达和描绘区域形状有意义的图像分量）——闭操作
kernelX = cv2.getStructuringElement(cv2.MORPH_RECT, (30, 10))
image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernelX,iterations = 1)
# 显示灰度图像
plt_show(image)
# 腐蚀（erode）和膨胀（dilate）
kernelX = cv2.getStructuringElement(cv2.MORPH_RECT, (50, 1))
```

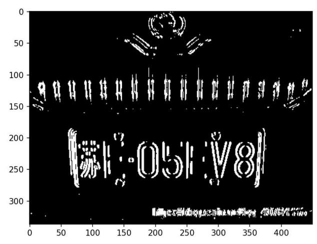
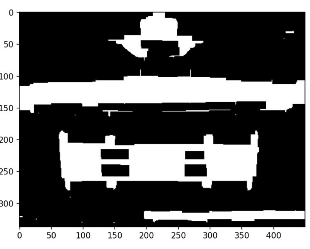
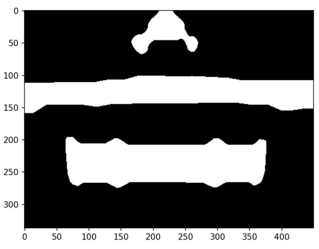
```
kernelY = cv2.getStructuringElement(cv2.MORPH_RECT, (1, 20))
#x 方向进行闭操作（抑制暗细节）
image = cv2.dilate(image, kernelX)
image = cv2.erode(image, kernelX)
#y 方向的开操作
image = cv2.erode(image, kernelY)
image = cv2.dilate(image, kernelY)
# 中值滤波（去噪）
image = cv2.medianBlur(image, 21)
# 显示灰度图像
plt_show(image)
# 获得轮廓
contours, hierarchy = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for item in contours:
    rect = cv2.boundingRect(item)
    x = rect[0]
    y = rect[1]
    weight = rect[2]
    height = rect[3]
    # 根据轮廓的形状特点，确定车牌的轮廓位置并截取图像
    if (weight > (height * 3.5)) and (weight < (height * 4)):
        image = origin_image[y:y + height, x:x + weight]
        plt_show0(image)
#车牌字符分割
# 图像去噪灰度处理
gray_image = gray_guss(image)
# 图像阈值化操作——获得二值化图
ret, image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_OTSU)
plt_show(image)
#膨胀操作，使“苏”字膨胀为一个近似的整体，为分割做准备
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
image = cv2.dilate(image, kernel)
plt_show(image)
# 查找轮廓
contours, hierarchy = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
words = []
word_images = []
#对所有轮廓逐一操作
for item in contours:
    word = []
    rect = cv2.boundingRect(item)
    x = rect[0]
    y = rect[1]
    weight = rect[2]
```

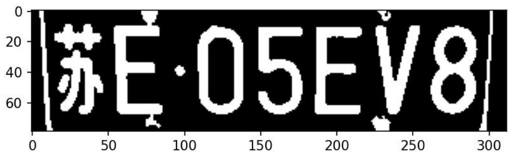
```python
    height = rect[3]
    word.append(x)
    word.append(y)
    word.append(weight)
    word.append(height)
    words.append(word)
# 排序，车牌号有顺序。words 是一个嵌套列表
words = sorted(words,key=lambda s:s[0],reverse=False)
i = 0
#word 中存放轮廓的起始点和宽高
for word in words:
    # 筛选字符的轮廓
    if (word[3] > (word[2] * 1.5)) and (word[3] < (word[2] * 3.5)) and (word[2] > 25):
        i = i+1
        splite_image = image[word[1]:word[1] + word[3], word[0]:word[0] + word[2]]
        word_images.append(splite_image)
        print(i)
print(words)
for i,j in enumerate(word_images):
    plt.subplot(1,7,i+1)
    plt.imshow(word_images[i],cmap='gray')
plt.show()
#模版匹配
# 准备模板(template[0-9]为数字模板；)
template = ['0','1','2','3','4','5','6','7','8','9',
            'A','B','C','D','E','F','G','H','J','K','L','M','N','P','Q','R','S','T','U','V','W','X','Y','Z',
            '藏','川','鄂','甘','赣','贵','桂','黑','沪','吉','冀','津','晋','京','辽','鲁','蒙','闽','宁',
            '青','琼','陕','苏','皖','湘','新','渝','豫','粤','云','浙']
# 读取一个文件夹下的所有图片，输入参数是文件名，返回模板文件地址列表
def read_directory(directory_name):
    referImg_list = []
    for filename in os.listdir(directory_name):
        referImg_list.append(os.path.join(directory_name, filename))  # 使用 os.path.join()构建文件路径
    return referImg_list
# 获得中文模板列表（只匹配车牌的第一个字符）
def get_chinese_words_list():
    chinese_words_list = []
    for i in range(34,64):
        # 将模板存放在字典中
        c_word = read_directory('./refer1/' + template[i])  # 使用 os.path.join()构建文件路径
        chinese_words_list.append(c_word)
    return chinese_words_list
chinese_words_list = get_chinese_words_list()
# 获得英文模板列表（只匹配车牌的第二个字符）
```

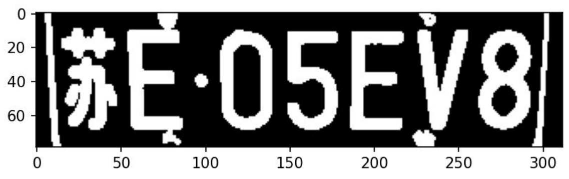
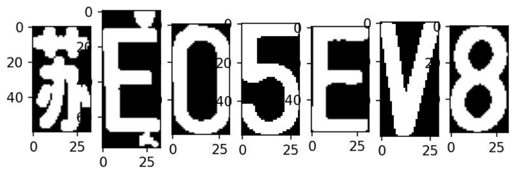
```python
def get_eng_words_list():
    eng_words_list = []
    for i in range(10,34):
        e_word = read_directory('./refer1/'+ template[i])
        eng_words_list.append(e_word)
    return eng_words_list
eng_words_list = get_eng_words_list()
# 获得英文和数字模板列表（匹配车牌后面的字符）
def get_eng_num_words_list():
    eng_num_words_list = []
    for i in range(0,34):
        word = read_directory('./refer1/'+ template[i])
        eng_num_words_list.append(word)
    return eng_num_words_list
eng_num_words_list = get_eng_num_words_list()
# 读取一个模板地址与图片进行匹配，返回得分
def template_score(template,image):
    #将模板进行格式转换
    template_img=cv2.imdecode(np.fromfile(template,dtype=np.uint8),1)
    template_img = cv2.cvtColor(template_img, cv2.COLOR_RGB2GRAY)
    #模板图像阈值化处理——获得黑白图
    ret, template_img = cv2.threshold(template_img, 0, 255, cv2.THRESH_OTSU)
#     height, width = template_img.shape
#     image_ = image.copy()
#     image_ = cv2.resize(image_, (width, height))
    image_ = image.copy()
    #获得待检测图片的尺寸
    height, width = image_.shape
    # 将模板 resize 至与图像一样大小
    template_img = cv2.resize(template_img, (width, height))
    # 模板匹配，返回匹配得分
    result = cv2.matchTemplate(image_, template_img, cv2.TM_CCOEFF)
    return result[0][0]
# 对分割得到的字符逐一匹配
def template_matching(word_images):
    results = []
    for index,word_image in enumerate(word_images):
        if index==0:
            best_score = []
            for chinese_words in chinese_words_list:
                score = []
                for chinese_word in chinese_words:
                    result = template_score(chinese_word,word_image)
                    score.append(result)
                best_score.append(max(score))
            i = best_score.index(max(best_score))
            # print(template[34+i])
            r = template[34+i]
            results.append(r)
            continue
        if index==1:
            best_score = []
            for eng_word_list in eng_words_list:
                score = []
                for eng_word in eng_word_list:
                    result = template_score(eng_word,word_image)
                    score.append(result)
                best_score.append(max(score))
            i = best_score.index(max(best_score))
            # print(template[10+i])
            r = template[10+i]
            results.append(r)
            continue
        else:
            best_score = []
            for eng_num_word_list in eng_num_words_list:
                score = []
                for eng_num_word in eng_num_word_list:
                    result = template_score(eng_num_word,word_image)
                    score.append(result)
                best_score.append(max(score))
            i = best_score.index(max(best_score))
            # print(template[i])
            r = template[i]
            results.append(r)
            continue
    return results
```

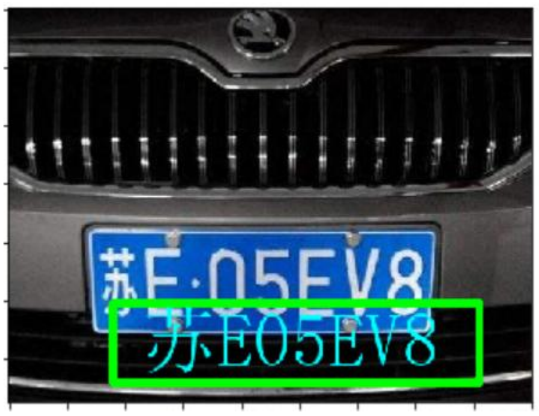
```python
word_images_ = word_images.copy()
# 调用函数获得结果
result = template_matching(word_images_)
print(result)
# "".join(result)函数将列表转换为拼接好的字符串，方便结果显示
print( "".join(result))
from PIL import ImageFont, ImageDraw, Image
height,weight = origin_image.shape[0:2]
print(height)
print(weight)
```

```
image_1 = origin_image.copy()
cv2.rectangle(image_1, (int(0.2*weight), int(0.75*height)), (int(weight*0.9), int(height*0.95)), (0, 255, 0), 5)
#设置需要显示的字体
fontpath = "font/simsun.ttc"
font = ImageFont.truetype(fontpath,64)
img_pil = Image.fromarray(image_1)
draw = ImageDraw.Draw(img_pil)
#绘制文字信息
draw.text((int(0.2*weight)+25, int(0.75*height)),  "".join(result), font = font, fill = (255, 255, 0))
bk_img = np.array(img_pil)
print(result)
print( "".join(result))
plt_show0(bk_img)
```





讲解一下实现的过程
1. 图像预处理：
读取原始图像，并对其进行去噪和灰度化处理。

使用高斯模糊和 Sobel 算子进行边缘检测，增强图像中的边缘信息。
2.二值化和形态学操作：
应用阈值化方法（大津阈值法）将图像转换为二值图像。
使用形态学操作，如闭操作和开操作，来消除噪声和连接断裂的区域。
3.中值滤波：
应用中值滤波进一步去除图像中的噪点。
4.轮廓检测：
寻找图像中的轮廓，通常用于识别可能的车牌区域。
5.车牌定位：
根据轮廓的几何特性（如宽度和高度的比例），筛选出可能是车牌的区域，并从原始图像中截取这些区域。
6.字符分割：
对截取的车牌区域进行进一步的图像处理，如阈值化和膨胀操作，以便于字符的分割。
7.字符识别：
对分割后的每个字符图像，使用模板匹配方法进行识别。这涉及到读取模板图像，将它们与待识别的字符图像进行比较，并找出最佳匹配。
8.模板匹配：
对于车牌的每个字符，使用不同的模板列表进行匹配。这包括中文字符、英文字符和数字。
9.结果整合：
将识别出的字符组合成完整的车牌号码，并显示或输出结果。

可以看到分割的效果很好，在分割的基础上对每个字符进行匹配，可以很好地识别出对应的字符，当然这里可以改进成用 CNN 训练字符识别模型，更为准确

4 实验总结

4.1 实验中遇到的问题与解决办法

1）对于车牌轮廓的识别，级联器有时表现的不好，因此我在后面使用了 HyperLPR 进行车牌区域的识别，也使用了 yolov5 算法的识别。
2）对于车牌号文字和数字的识别，pytesseract 效果不好，因此在后面使用了 paddleOCR 以及 LPRNet 进行识别。
3)而且在收集车牌数据的时候，发现有部分车牌的区域比较歪斜，因此采取了角点检测和透视变换的方法，想要将车牌成功地拉到四个角。

4.2 实验收获与不足

收获：
1.通过实验的实现，掌握了车牌识别的流程，并且熟练地掌握了上课所讲授的级联器，以及图像形态学方法，角点检测和透视变换以及 ocr 识别方法
2.自己在改进的过程中使用了很多方法，使用了 paddleOCR 的方法，之后也自己训练了 LPRNet，最终的识别效果很不错，也使用了 HyperLPR 进行车牌区域的识别，此外，我还使用了 yolov5 算法，进行车牌区域的识别。
不足：

1.自己开始的时候想要通过将识别出的车牌区域进行图像处理,使用图像形态学变换等操作加强车牌的可识别度,这样可以在 tesseractOCR 的基础上进行识别,但是我在我的较好识别的图像上采用形态学变化以及高斯滤波等方法，效果很差，甚至不如原始的图像，因此最终没有采用这种方法。但是对于一些可辨识度很差的图片，经过一些图像处理，可能更好识别。但是自己通过 yolo 算法解决了这个问题，通过训练可以精确识别出车牌区域，不需要再进行图像处理来定位车牌区域。

2.在实现文字分割后，没有训练 cnn 进行文字识别，只是用匹配得分的方法，导致精确度不够，有时识别出来不准确