代码备份：

import numpy as np  
import cv2 as cv  
import matplotlib.pyplot as plt  
  
# 灰度图转换  
def grayscale(image):  
 return cv.cvtColor(image, cv.COLOR\_RGB2GRAY)  
  
# 高斯滤波  
# size为高斯核大小，即高斯滤波器的尺寸；0是高斯标准差，一般默认为0  
def gaussian\_blur(image, kernel\_size):  
 return cv.GaussianBlur(image, (kernel\_size, kernel\_size), 0)  
  
# Canny边缘检测  
# low\_threshold低阈值； high\_threshold高阈值  
def canny(edge\_image, low\_threshold, high\_threshold):  
 return cv.Canny(image, low\_threshold, high\_threshold)  
  
# 生成感兴趣区域  
# 步骤：1.生成一个与原image大小一致的的mask矩阵，初始化为全0  
# 2.对照原图在mask上构建感兴趣区域  
# 3.利用opencv中cv.fillpoly()函数对所限定的多边形轮廓进行填充，填充为1，即全白。  
# 4.利用opencv中cv.bitwise()函数与canny边缘检测后的图像按位与，保留原图相中对应感兴趣区域内的白色像素值，剔除黑色像素值  
  
def region\_of\_interest(image, vertices):  
 mask = np.zeros\_like(image) # 生成维度与image大小一致的mask矩阵，并为其初始化为全0,即构建出全黑的mask图像  
  
 # 填充顶点vertices中间区域  
 if len(image.shape) > 2: # image.shape是一个1\*3的矩阵，len()方法，返回的是矩阵中列元素的个数  
 channel\_count = image.shape[2]  
 ignore\_mask\_color = (255,) \* channel\_count  
 else:  
 ignore\_mask\_color = 255  
  
# cv.fillPoly()用于一个被多边形轮廓所限定的区域内进行填充。mask：图像；vertices：顶点坐标的小数点位数；ignore\_mask\_color:多边形的颜色  
 cv.fillPoly(mask, vertices, ignore\_mask\_color)  
 # cv.imshow('maskRIO',mask)  
# cv.bitwise\_and()函数是对二进制数据进行与操作，即对图像每个像素值进行二进制与操作  
# 利用掩膜进行与操作，即掩膜图像白色区域是对需要处理图像像素的保留，黑色区域是对需要处理图像像素的剔除。  
 masked\_image = cv.bitwise\_and(image, mask)  
 return masked\_image  
  
# 原图像与车道线图像按照a:b比例融合  
def weighted\_img(img, initial\_img, a=0.8, b=1., c=0.):  
 return cv.addWeighted(initial\_img, a, img, b, c) # cv.addWeighted()函数是将两张相同大小，相同类型的图片融合  
  
# def reset\_global\_vars():  
#  
# global SET\_LFLAG  
# global SET\_RFLAG  
# global LAST\_LSLOPE  
# global LAST\_RSLOPE  
# global LAST\_LEFT  
# global LAST\_RIGHT  
#  
# SET\_RFLAG = 0  
# SET\_LFLAG = 0  
# LAST\_LSLOPE = 0  
# LAST\_RSLOPE = 0  
# LAST\_RIGHT = [0, 0, 0]  
# LAST\_LEFT = [0, 0, 0]  
  
  
# 绘制车道线  
def draw\_lines(image, lines, color=[0,255,0], thickness=2):  
  
 right\_y\_set = []  
 right\_x\_set = []  
 right\_slope\_set = []  
  
 left\_y\_set = []  
 left\_x\_set = []  
 left\_slope\_set = []  
  
 slope\_min = .35 # 斜率低阈值  
 slope\_max = .85 # 斜率高阈值  
 middle\_x = image.shape[1] / 2 # 图像中线x坐标  
 max\_y = image.shape[0] # 最大y坐标  
  
 for line in lines:  
 for x1, y1, x2, y2 in line:  
 fit = np.polyfit((x1, x2), (y1, y2), 1) # 一次多项式拟合相当于线性拟合，即拟合成直线  
 slope = fit[0] # 斜率  
  
 if slope\_min < np.absolute(slope) <= slope\_max:  
  
 # 将斜率大于0且线段X坐标在图像中线右边的点存为右边车道线  
 if slope > 0 and x1 > middle\_x and x2 > middle\_x:  
 right\_y\_set.append(y1)  
 right\_y\_set.append(y2)  
 right\_x\_set.append(x1)  
 right\_x\_set.append(x2)  
 right\_slope\_set.append(slope)  
  
 # 将斜率小于0且线段X坐标在图像中线左边的点存为左边车道线  
 elif slope < 0 and x1 < middle\_x and x2 < middle\_x:  
 left\_y\_set.append(y1)  
 left\_y\_set.append(y2)  
 left\_x\_set.append(x1)  
 left\_x\_set.append(x2)  
 left\_slope\_set.append(slope)  
  
 # 绘制左车道线  
 if left\_y\_set:  
 lindex = left\_y\_set.index(min(left\_y\_set)) # 图像中的最高点，对应坐标系中y的最小值  
 left\_x\_top = left\_x\_set[lindex]  
 left\_y\_top = left\_y\_set[lindex]  
 lslope = np.median(left\_slope\_set) # 计算斜率的平均值  
  
 # 根据斜率计算车道线与图片下方交点作为起点  
 left\_x\_bottom = int(left\_x\_top + (max\_y - left\_y\_top) / lslope)  
  
 # 绘制线段  
 cv.line(image, (left\_x\_bottom, max\_y), (left\_x\_top, left\_y\_top), color, thickness)  
  
 # 绘制右车道线  
 if right\_y\_set:  
 rindex = right\_y\_set.index(min(right\_y\_set)) # 最高点  
 right\_x\_top = right\_x\_set[rindex]  
 right\_y\_top = right\_y\_set[rindex]  
 rslope = np.median(right\_slope\_set)  
  
 # 根据斜率计算车道线与图片下方交点作为起点  
 right\_x\_bottom = int(right\_x\_top + (max\_y - right\_y\_top) / rslope)  
  
 # 绘制线段  
 cv.line(image, (right\_x\_top, right\_y\_top), (right\_x\_bottom, max\_y), color, thickness)  
  
def hough\_lines(img, rho, theta, threshold, min\_line\_len, max\_line\_gap):  
  
 # rho：线段以像素为单位的距离精度  
 # theta : 像素以弧度为单位的角度精度(np.pi/180较为合适)  
 # threshold : 霍夫平面累加的阈值  
 # minLineLength : 线段最小长度(像素级)  
 # maxLineGap : 最大允许断裂长度  
 lines = cv.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min\_line\_len, maxLineGap=max\_line\_gap)  
 return lines  
  
def process\_image(image):  
  
 kernel\_size = 5 # 高斯滤波器大小size  
 canny\_low\_threshold = 75 # canny边缘检测低阈值  
 canny\_high\_threshold = canny\_low\_threshold \* 3 # canny边缘检测高阈值  
  
 rho = 1 # 霍夫像素单位  
 theta = np.pi / 180 # 霍夫角度移动步长  
 hof\_threshold = 20 # 霍夫平面累加阈值threshold  
 min\_line\_len = 30 # 线段最小长度  
 max\_line\_gap = 60 # 最大允许断裂长度  
  
 alpha = 0.8 # 原图像权重  
 beta = 1. # 车道线图像权重  
 lambda\_ = 0.  
  
 imshape = image.shape # 获取图像大小，返回的值是H,W,C（H是图像的高，W是图像的宽，C表示图像有几层）  
 # print(imshape) #(459, 867, 3) 代表的意思就是该图高459，宽867，有bgr三层  
  
 # 灰度图转换  
 gray = grayscale(image)  
 cv.imshow('gray\_image', gray)  
  
 # 高斯滤波  
 blur\_gray = gaussian\_blur(gray, kernel\_size)  
 cv.imshow('blur\_gray\_image', blur\_gray)  
  
 # Canny边缘检测  
 edge\_image = canny(blur\_gray, canny\_low\_threshold, canny\_high\_threshold)  
 cv.imshow('edge\_image', edge\_image)  
  
 # 绘制感兴趣区域  
 vertices = np.array([[(0, imshape[0]), (9 \* imshape[1] / 20, 11 \* imshape[0] / 18),  
 (11 \* imshape[1] / 20, 11 \* imshape[0] / 18), (imshape[1], imshape[0])]], dtype=np.int32)  
 masked\_edges = region\_of\_interest(edge\_image, vertices)  
 cv.imshow('masked\_edges\_image', masked\_edges)  
  
 # 基于霍夫变换的直线检测  
 lines = hough\_lines(masked\_edges, rho, theta, hof\_threshold, min\_line\_len, max\_line\_gap)  
 line\_image = np.zeros\_like(image) # 构造一个与image矩阵大小维度一致的矩阵，并初始化为全0（纯黑）  
  
 # 绘制车道线线段  
 draw\_lines(line\_image, lines, thickness=10)  
 cv.imshow('line\_image',line\_image)  
 # 图像融合  
 final\_image = weighted\_img(image, line\_image, alpha, beta, lambda\_)  
 # cv.imshow('final\_image',final\_image)  
 return final\_image  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # cap = cv.VideoCapture("./test\_videos/solidYellowLeft.mp4")  
 # while(cap.isOpened()):  
 # \_, frame = cap.read()  
 # processed = process\_image(frame)  
 # cv.imshow("image", processed)  
 # cv.waitKey(1)  
  
 image = cv.imread('D:/lane-detection-cv/test-image/333.PNG')  
# 显示原始图像  
 cv.imshow('original\_image', image)  
# 显示图像融入的最终效果图  
 final\_image = process\_image(image)  
 cv.imshow('image',final\_image)  
 cv.waitKey(0)