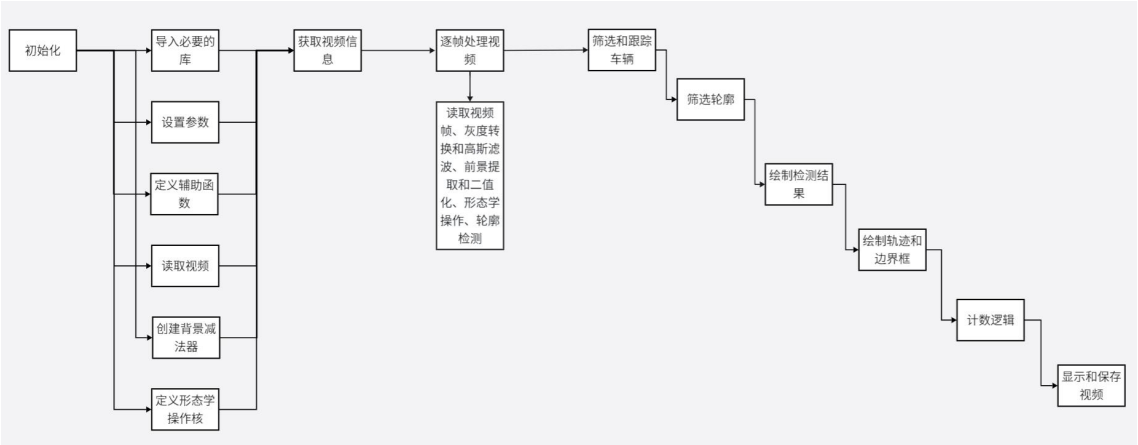
实验一　车辆统计实验

1. 实验流程



实验最终的思路流程：

1. 初始化

导入必要的库：**主要使用了 OpenCV 和 Numpy。**

设置参数：**包括最小检测框的宽度和高度、计数线的位置和偏移量、车辆计数器等。**

定义辅助函数：**例如计算边界框中心点的函数。**

读取视频：**使用 cv2.VideoCapture 从文件中读取视频。**

创建背景减法器：**用于分离前景（车辆）和背景。**

定义形态学操作核：**用于处理二值化图像，去除噪声。**

2. 获取视频信息

获取视频帧率和尺寸：**用于初始化视频写入对象。**

初始化视频写入对象：**使用 cv2.VideoWriter 设置编码格式、输出文件名、帧率和尺寸。**

3. 逐帧处理视频

读取视频帧：**在循环中逐帧读取视频。**

灰度转换和高斯滤波：**将每帧转换为灰度图像并应用高斯滤波去噪。**

前景提取和二值化：**应用背景减法器提取前景，随后二值化处理前景图像。**

形态学操作：**使用腐蚀、膨胀和闭操作去除噪声并增强物体区域。**

轮廓检测：**在处理后的二值化图像中找到轮廓。**

4. 筛选和跟踪车辆

筛选轮廓：**过滤掉小轮廓，并根据最小宽度和高度筛选出有效的车辆边界框。**

跟踪车辆：**通过计算中心点和距离来跟踪车辆，并为每个新检测到的车辆分配唯一 ID。**

5. 绘制检测结果

绘制轨迹和边界框：**画出每辆车的检测框、中心点和移动轨迹。**

计数逻辑：**根据车辆中心点是否经过计数线来更新车辆计数器。**

6. 显示和保存视频

显示视频：**在窗口中显示处理后的每一帧。**

写入视频文件：**将处理后的帧写入输出视频文件。**
处理退出条件：**检测到 ESC 键按下时退出循环。**
7. 释放资源
释放视频捕获和写入对象：**在处理完成后释放所有资源。**
关闭所有窗口：**确保所有 OpenCV 窗口关闭。**

2. 实验结果

最终预测的车辆数为 52 辆，并且每辆车都精确地预测出来，下面是最终代码

```
import cv2
import numpy as np

min_w = 50
min_h = 50
line_coord = 340
offset = 8
carno = 0
vehicle_center_points = {}
vehicle_id = 0

def center(x, y, w, h):
    """计算边界框的中心点"""
    x1 = int(w / 2)
    y1 = int(h / 2)
    cx = x + x1
    cy = y + y1
    return cx, cy

cap = cv2.VideoCapture('./video/vehicle.mp4')  # 读取视频
bg_subtractor = cv2.bgsegm.createBackgroundSubtractorMOG()  # 创建背景减法器
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))  # 创建形态学操作的
核


# 获取视频的帧率和尺寸
fps = cap.get(cv2.CAP_PROP_FPS)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# 初始化 VideoWriter 对象
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, fps, (width, height))
```

```
while True:
    ret, frame = cap.read()  # 读取每一帧
    if not ret:
        break

    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # 转为灰度图
    blur = cv2.GaussianBlur(frame_gray, (3, 3), 5)  # 高斯滤波去噪

    fg_mask = bg_subtractor.apply(blur)  # 应用背景减法以获取前景掩码
    _, mask = cv2.threshold(fg_mask, 180, 255, cv2.THRESH_BINARY)  # 二值化掩
码

    # 形态学操作：腐蚀和膨胀
    erode = cv2.erode(mask, kernel, iterations=1)
    dilate = cv2.dilate(erode, kernel, iterations=3)
    close = cv2.morphologyEx(dilate, cv2.MORPH_CLOSE, kernel)
    # 在前景掩模中找到轮廓
    contours, _ = cv2.findContours(close, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # 筛选轮廓
    boxes = []
    for contour in contours:
        if cv2.contourArea(contour) < 2000:  # 过滤小轮廓
            continue
        x, y, w, h = cv2.boundingRect(contour)  # 获取外接矩形
        if w >= min_w and h >= min_h:  # 检查边界矩形是否满足最小尺寸要求
            boxes.append([x, y, x + w, y + h])  # 添加边界框坐标

    # 跟踪车辆
    new_center_points = {}
    for box in boxes:
        x, y, x2, y2 = box
        cx, cy = center(x, y, x2 - x, y2 - y)  # 计算边界框的中心点
        same_vehicle_detected = False
        for id, pts in vehicle_center_points.items():
            dist = np.sqrt((cx - pts[-1][0]) ** 2 + (cy - pts[-1][1]) ** 2)
            if dist < 50:  # 如果距离小于阈值，则认为是同一辆车
                new_center_points[id] = pts + [(cx, cy)]
                same_vehicle_detected = True
                break
        if not same_vehicle_detected:
            new_center_points[vehicle_id] = [(cx, cy)]
```

```
            vehicle_id += 1

    vehicle_center_points = new_center_points

    # 画出检测框和中心点
    for id, pts in vehicle_center_points.items():
        for i in range(len(pts) - 1):
            cv2.line(frame, pts[i], pts[i + 1], (0, 255, 255), 2)  # 画出轨迹
线
        x, y = pts[-1]
        cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)  # 画出中心点

        # 画出边界矩形
        for box in boxes:
            x1, y1, x2, y2 = box
            if abs(x - ((x1 + x2) // 2)) < 10 and abs(y - ((y1 + y2) // 2)) <
10:
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
                break

    # 计数逻辑
    for id, pts in list(vehicle_center_points.items()):
        if len(pts) >= 2:
            for i in range(len(pts) - 1):
                if (pts[i][1] <= line_coord < pts[i + 1][1]) or (pts[i][1] >=
line_coord > pts[i + 1][1]):
                    carno += 1
                    vehicle_center_points.pop(id)
                    break

    # 显示计数和计数线
    cv2.putText(frame, "Car Count: " + str(carno), (20, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
    cv2.line(frame, (0, line_coord), (frame.shape[1], line_coord), (0, 255, 0),
2)

    cv2.imshow('video', frame)

    # 将处理后的帧写入视频文件
    out.write(frame)
    key = cv2.waitKey(1)
    if key == 27:  # 按下 ESC 键退出
        break
```

```
print("Total car count:", carno)  # 输出总车数

cap.release()
out.release()  # 释放 VideoWriter 对象
cv2.destroyAllWindows()
```

output.avi

这个是效果演示视频

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)基本实验回顾

首先是对基本实验进行流程的回顾

```
#carcount
import cv2
import numpy as np

min_w = 50
min_h = 50
line_coord = 350
offset = 10
carno = 0

def center(x, y, w, h):#计算边框的中心坐标
    x1 = int(w/2)
    y1 = int(h/2)
    cx = x + x1
    cy = y + y1
    return cx, cy

cap = cv2.VideoCapture('./video/vehicle.mp4')#获取整个视频的内容
bgsubmog = cv2.createBackgroundSubtractorMOG2()#加载背景减法器
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))#用于形态学操作

while True:
    ret, frame = cap.read()#读取每一帧
    if ret == True:#读取成功
```

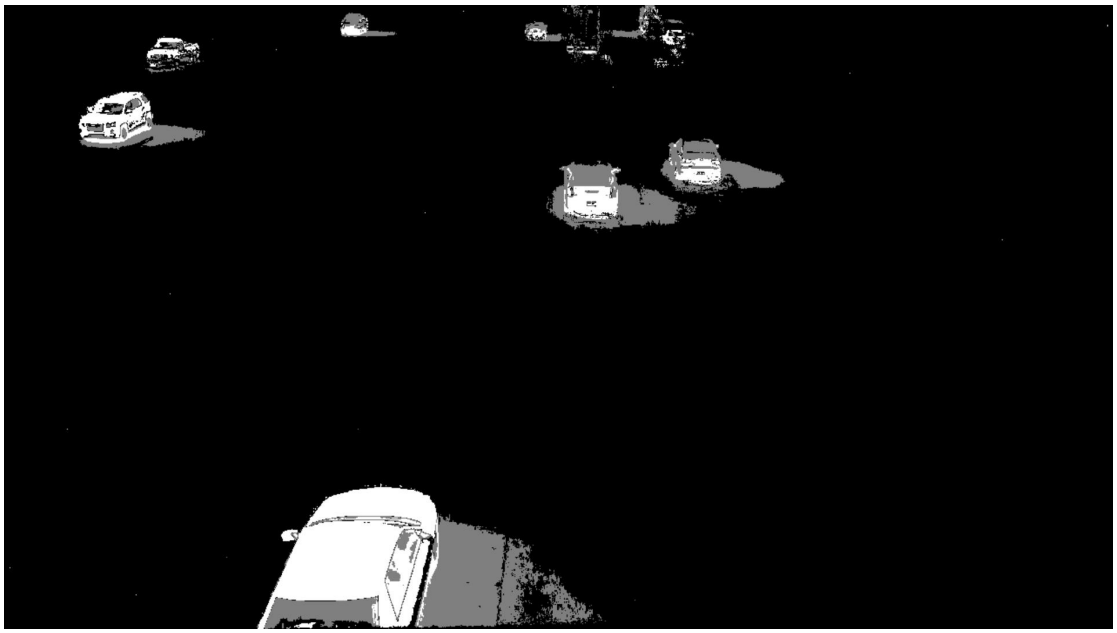
```
25)         frameGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)#将帧转为灰度图
26)         maskOrigin = bgsubmog.apply(frameGray)#应用背景减法以获取前景掩码
27)         #cv2.imshow('mask', maskOrigin)
28)         ret, mask = cv2.threshold(maskOrigin, 130, 255, cv2.THRESH_BINARY)#
    二值化掩码,生成二值化图像
29)         #cv2.imshow('mask', mask)
30)
31)         #以减少噪声并增强物体区域
32)         erode = cv2.erode(mask, kernel)#腐蚀
33)         dilate = cv2.dilate(erode, kernel, iterations=3)#膨胀
34)
35)         #在膨胀后的图像中找到轮廓
36)         contours, hier = cv2.findContours(dilate, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
37)         for (i, c) in enumerate(contours):#遍历轮廓
38)             (x, y, w, h) = cv2.boundingRect(c)#获取轮廓的边界矩形
39)             isValid = (w >= min_w) and (h >= min_h)#检查边界矩形是否满足最
    小尺寸要求
40)             if not isValid:#不有效就筛除掉轮廓
41)                 continue
42)             cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 5)#画出
    边界矩形
43)             cx, cy = center(x, y, w, h)#计算边框的中心坐标
44)             if cy > line_coord - offset and cy < line_coord + offset:#检
    查矩形的中心点是否接近计数线。
45)                 carno += 1
46)
47)         cv2.putText(frame, "Car Count: "+str(carno), (500, 60),
    cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 5)
48)         cv2.line(frame, (0, line_coord), (frame.shape[1], line_coord), (0,
    255, 0), 2)  # Draw line
49)
50)         cv2.imshow('video', frame)
51)
52)     key = cv2.waitKey(100)
53)     if key == 27:#如果按下 Esc 键（ASCII 27），则退出循环。
54)         break
55)
56) print("Total car count:", carno)  # Output total car count after processing
    all frames
57)
58) cap.release()
59) cv2.destroyAllWindows()
```

这是老师在课堂上讲述的方法，效果已经基本实现，但是识别准确度不够好。
1.首先引入相关库，然后定义了识别轮廓的最小宽和高，识别线的位置，识别的偏移量（也就关系着我们的精确度）以及我们的全局变量车辆的数量 carno。
2.接着我们定义了计算边框中心坐标的函数
3.然后初始化我们的 VideoCapture 对象，获取整个视频的内容
4.加载背景减法器(这里采用的是 cv2.createBackgroundSubtractorMOG2()背景减法)
5.创建了形态学操作的结构元素，cv2.MORPH_RECT 指定为矩形，(5, 5)：指定了结构元素的大小，其中较大的核相当于采用了更多的信息，全局性更好，较小的核则具有良好的局部性
6.接着进入循环，进行每一帧内容的操作

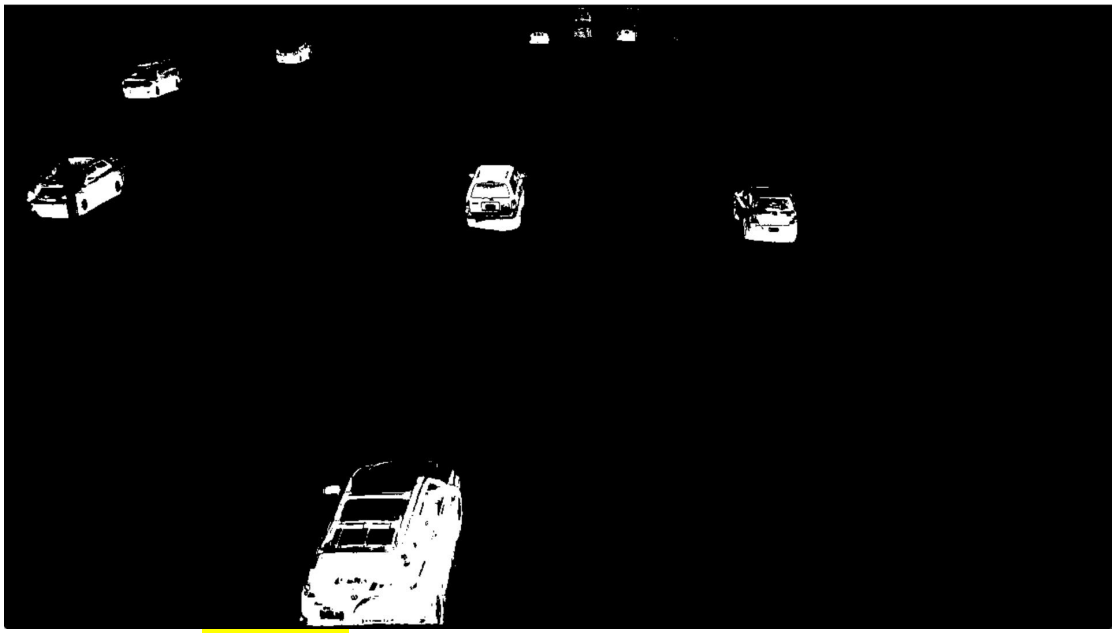
1）cap.read()读取每一帧，如果读取成功则进行图像处理操作

首先将图像灰度化，可以看到我们灰度化后的图像

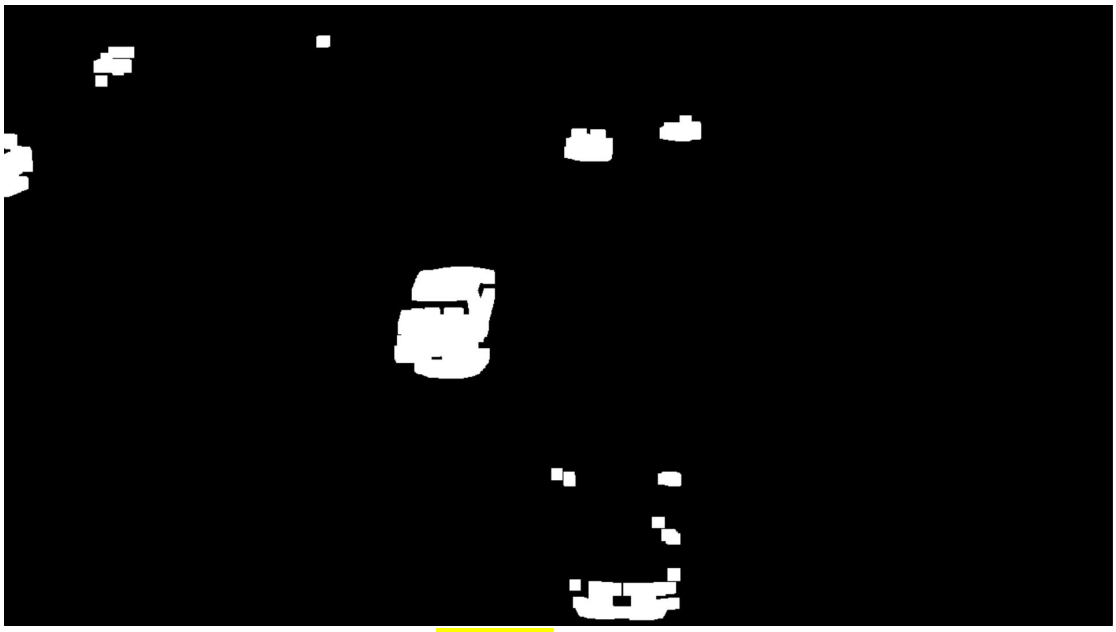
2）然后根据背景减法器获取前景掩码，此时查看我们的前景图像，可以看到有很多的影子，这可能会导致我们在计数时将影子识别为车辆，也就是存在干扰信息



3)接着我们就使用 threshold 进行阈值处理，相当于我们找到了一个阈值（小于阈值的被设置为黑色，大于等于阈值的被设置为白色），相当于我们手动设置了二值化，可以从下面图形看到我们没有了一些影子的干扰项



4)接着使用腐蚀和膨胀，让我们可以识别出整个车辆的大轮廓，而不是错误地识别出其中的一些无效轮廓，可以观察我们的图片



5)接着在图像处理后的图像中查找轮廓（cv2.RETR_EXTERNAL 表示只检测外部轮廓，cv2.CHAIN_APPROX_SIMPLE 表示仅保留轮廓的端点信息，以节省内存空间），接着遍历每一个轮廓，获取轮廓的最大外接矩形的(x,y)以及 w,h，就可以确定一个轮廓框了

6)然后判断边界矩形的尺寸是否满足最小要求，不满足就筛除轮廓
满足的话就画出边界轮廓，然后通过定义的 center()函数求出边界轮廓的中心坐标，如果这个中心坐标距离检测线在 offset 范围内，就将车辆总数+1

7)然后在图像中加入我们目前识别到车辆的数目，记录车辆数目，也同时画出我们的检测线，以及显示出当前识别的车辆数

8)如果按下 Esc 键就退出循环
最终在输出框中输出总共识别的汽车数量。

总的流程已经相当完善了，但是识别准确率还不够。

下面是我的改进与思考

2)改进背景减法器

首先我改进了背景减法器，我先采用的是 cv2.createBackgroundSubtractorKNN()，识别效果有所提高，接着我又使用了 cv2.bgsegm.createBackgroundSubtractorMOG()，这个相比于前面 knn 的背景减法器效果更好，因此最终我采用了 bgsegm 的背景减法器，加载过程没有变化。
此外我还扩大了识别轮廓的面积，便于进行识别操作，筛除小轮廓。

3)高斯滤波和闭运算

图像在识别时出现了漏记的情况，可能是相关环境或者其他车辆的影响，导致我们识别不准确，所以我从图像处理角度进行优化，这里我首先进行了高斯滤波进行去噪，我采用了 3*3 的高斯核，相比于我们的灰度化图像，这个图像更加模糊，但是噪点减少了，我们得到的图像更加平滑，颜色变化相对减小，同时增大了图像的全局性。

接着我又对腐蚀膨胀后进行了闭运算，这步操作相当于与操作，可以消除图像中的小孔洞，平滑对象的边界，连接其中断裂的部分，使我们观察到的是连续的大块。

可以看到其中的车辆的孔洞几乎没有了，只用刚进去识别区域的车辆会有一些不影响识别的小孔洞。

4)修改识别线和识别范围

图像在识别时会出现多次计数的现象，但是我观察到越远的图像识别，识别框越稳定，所以就想到适当地调节识别线，往远处移动，识别有所帮助。同时我也想将识别线倾斜，并且可以限定范围（因为我观察到左边的车辆先稳定，右边的车辆后稳定），所以进行了修改

```
def center(x, y, w, h):  # 计算边框的中心坐标
    x1 = int(w/2)
    y1 = int(h/2)
    cx = x + x1
    cy = y + y1
    return cx, cy
def point_to_line_distance(line_start, line_end, point):
    x1, y1 = line_start
    x2, y2 = line_end
    x0, y0 = point
    distance = np.abs((y2 - y1) * x0 - (x2 - x1) * y0 + x2 * y1 - y2 * x1) /
np.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)
    return distance

line_start = (250, 260)  # 定义斜线的起点坐标
line_end = (1500, 260)    # 定义斜线的终点坐标
```

相当于我计算中心点到直线的距离（运用点到直线的距离公式）但是特别倾斜的线识别效果很糟糕，所以这一步操作在最终的实现中没有加入，当然这一步也是很有效的，可以限定我们识别的区域。

5)非极大值抑制

并且识别过程中识别框经常变化，并且有小的识别框，我采用了非极大值抑制，相当于我对一众识别框进行打分，最终将几个有重叠的小轮廓筛去。

```
# 非极大值抑制

        if len(boxes) > 0:
            boxes = np.array(boxes)
            scores = np.ones(len(boxes))  # 这里我们将所有检测框的得分设为 1，因为我们只是简单地想执行非极大值抑制
            picked_boxes = cv2.dnn.NMSBoxes(boxes, scores, score_threshold=0.5, nms_threshold=0.4)
```

首先检测识别到的边框，转为 numpy 的数组，然后定义对应的得分数组，然后使用 cv2.dnn.NMSBoxes()进行非极大值抑制，可以有效地消除较小的有干扰的边框，但是我项目最后并不是通过识别框来进行，所以最终没有采用这个方法。

6)yolov3 识别

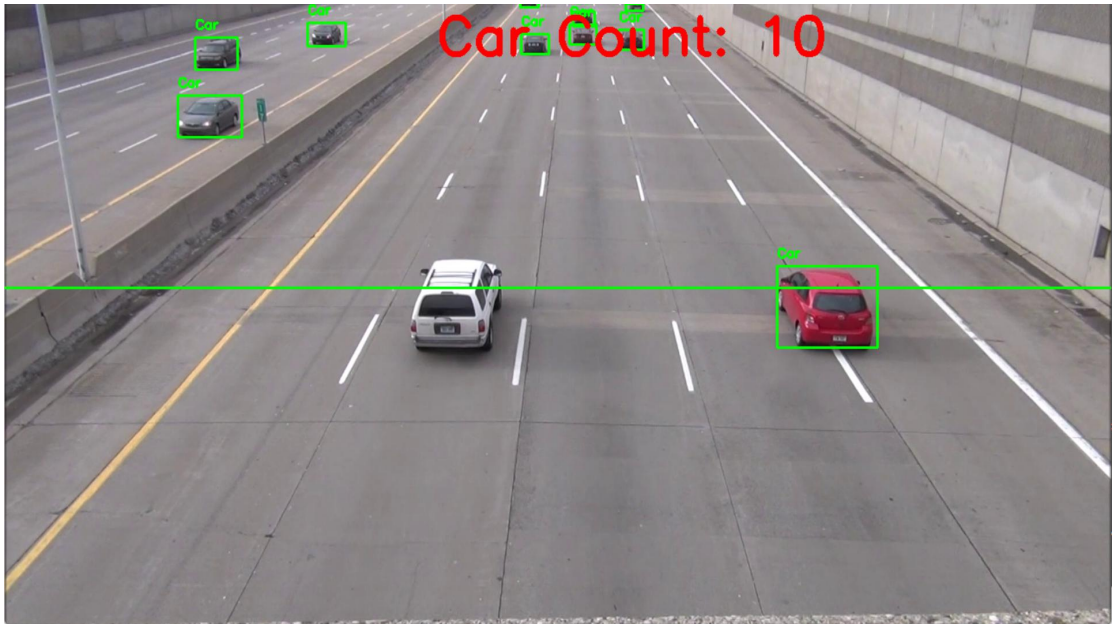
接着就来到最难的思路方面，我陷入了僵局，没有更好的提高准确率的方法，所以我首先想到了课堂上老师不推荐使用的 yolo 算法，我查看了 yolo 算法的使用教程，下载了相关文件，使用了 yolov3 算法，发现识别出的车效果极佳，但是可能是我采用了相对原始的模型，模型识别的效率特别低，并且识别框并不稳定，也是在某些环境下会受到影响，可能是采用 yolo 的版本问题，计数逻辑和之前没有变化，我们识别的效果也不是很好，但是 yolo 算法加上了识别后红色标注，便于我们观察哪些车辆被计数了。

```
import cv2
import numpy as np


# 加载 YOLO
net = cv2.dnn.readNet("./yolov3/yolov3.weights", "./yolov3/yolov3.cfg")
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
classes = []
with open("./yolov3/coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
# 初始化一些变量
line_coord = 330
offset = 10
carno = 0
cap = cv2.VideoCapture('./video/vehicle.mp4')  # 获取整个视频的内容
def center(x, y, w, h):  # 计算边框的中心坐标
    x1 = int(w/2)
    y1 = int(h/2)
    cx = x + x1
```

```
    cy = y + y1
    return cx, cy
while True:
    ret, frame = cap.read()  # 读取每一帧
    if not ret:
        break
    height, width, channels = frame.shape
    # 检测物体
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5 and classes[class_id] == "car":  # 只检测汽车类
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            cx, cy = center(x, y, w, h)
            color = (0, 255, 0)
            if cy > line_coord - offset and cy < line_coord + offset:
                carno += 1
                color = (0, 0, 255)  # 如果通过计数线，用红色标记
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
            cv2.putText(frame, "Car", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    cv2.putText(frame, "Car Count: " + str(carno), (500, 60), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 5)
    cv2.line(frame, (0, line_coord), (frame.shape[1], line_coord), (0, 255, 0), 2)  # 画出计数线
    cv2.imshow('video', frame)
```

```
    key = cv2.waitKey(1)
    if key == 27:  # 如果按下 Esc 键（ASCII 27），则退出循环。
        break
print("Total car count:", carno)  # 输出处理完所有帧后的总车数
cap.release()
cv2.destroyAllWindows()
```



7)思路改进（优化识别）

最后我想起老师课堂上推荐的知乎文章，以及相关计数的介绍，通过将车辆的每一帧所在的点记录，然后连接成线，再到识别线处进行识别，这种效果结合了之前的点的判断，识别效果肯定会有所提高，我就使用了这种方法来实现这个操作，最终的效果经过 offset 和识别线的调节非常好（当然不调节效果也已经相当不错，识别稳定且有效），完全正确，这个在其他视频中，或者现实使用的某一位置，进行特别的标注，可以很精确地实现车辆统计。

```
vehicle_center_points = {}
 boxes = []
    for contour in contours:
        if cv2.contourArea(contour) < 2000:  # 过滤小轮廓
            continue
        x, y, w, h = cv2.boundingRect(contour)  # 获取外接矩形
        if w >= min_w and h >= min_h:  # 检查边界矩形是否满足最小尺寸要求
            boxes.append([x, y, x + w, y + h])  # 添加边界框坐标

    # 跟踪车辆
    new_center_points = {}
```

```
    for box in boxes:
        x, y, x2, y2 = box
        cx, cy = center(x, y, x2 - x, y2 - y)  # 计算边界框的中心点
        same_vehicle_detected = False
        for id, pts in vehicle_center_points.items():
            dist = np.sqrt((cx - pts[-1][0]) ** 2 + (cy - pts[-1][1]) ** 2)
            if dist < 50:  # 如果距离小于阈值，则认为是同一辆车
                new_center_points[id] = pts + [(cx, cy)]
                same_vehicle_detected = True
                break
        if not same_vehicle_detected:
            new_center_points[vehicle_id] = [(cx, cy)]
            vehicle_id += 1

    vehicle_center_points = new_center_points

    # 画出检测框和中心点
    for id, pts in vehicle_center_points.items():
        for i in range(len(pts) - 1):
            cv2.line(frame, pts[i], pts[i + 1], (0, 255, 255), 2)  # 画出轨迹
线
        x, y = pts[-1]
        cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)  # 画出中心点

        # 画出边界矩形
        for box in boxes:
            x1, y1, x2, y2 = box
            if abs(x - ((x1 + x2) // 2)) < 10 and abs(y - ((y1 + y2) // 2)) <
10:
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
                break

    # 计数逻辑
    for id, pts in list(vehicle_center_points.items()):
        if len(pts) >= 2:
            for i in range(len(pts) - 1):
                if (pts[i][1] <= line_coord < pts[i + 1][1]) or (pts[i][1] >=
line_coord > pts[i + 1][1]):
                    carno += 1
                    vehicle_center_points.pop(id)
                    break
```

这种方法的主要思路是筛选出所有符合的轮廓，加入到 boxes 中
然后用 new_center_points（这是一个字典，key 记录车辆唯一的标识符 id，存放着车辆

在每一帧中的中心点坐标）。记录车辆的中心点，然后遍历之前记录的轮廓，计算出所有的中心点，同时使用 same_vehicle_detected 变量来记录是否为同一辆车。
然后遍历 new_center_points 中的内容，如果当前中心点与之前中心点存在距离相差小于 50，那么就认为是同一辆车（为了行车安全，这里车辆大多数距离间隔满足阈值），就会将这个中心点加入到对应车辆 id 对应的列表中，如果没有满足距离要求的，说明是新出现的车辆，就将它新创建一个 id，将中心点加入到新 id 中，最终更新我们的车辆列表。
后续的会画出所有的中心点和当前的检测框，最终到了我们计数逻辑的部分，如果某一 id 对应的列表超过两个点，就可以进行是否经过检测线的判断，如果存在两个点分别在直线的两侧，那么就计数加 1，并且移除这个车辆 id。
这种检测效果表现得很准确，首先直线的标注，可以更好地标记出一辆车，然后根据两个点分别在识别线的两侧，也能提高检测的准确性。

8)处理后视频的保存

最后为了可视化检测内容，加入了视频的获取与写入

```
# 获取视频的帧率和尺寸
fps = cap.get(cv2.CAP_PROP_FPS)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# 初始化 VideoWriter 对象
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, fps, (width, height))
 # 将处理后的帧写入视频文件
    out.write(frame)
```

4 实验总结

4.1 实验中遇到的问题与解决办法

1)背景减法器的改进：最开始的 cv2.createBackgroundSubtractorMOG2（）效果不是很好，通过网上搜索，一步步地优化背景减法器，得到了不错的效果。
2)识别线的修改：我本身发现左部分存在一些车辆，为了避免多余的识别，我需要减小识别线的范围，然后我发现一些车辆存在在左边的车辆识别先稳定，右边的车辆识别后稳定的现象，因此我增加了这一部分。
3)yolov3 算法：不熟悉 yolov3 的使用方法，因此我在网上搜索了大量的资料，并且下载了参数包和物品的识别包，经过调试，可以很好地实现识别，但是神经网络的识别方法太慢了，所以最终没有整合到我们的最终内容。
4）思路改进：这一块还是很有难度的，不知道如何实现计数的优化，查看了老师上课提醒的内容，其中内容代码很多，我看了相关的源码，并结合自己的思考，以及网上的实现过程，

我觉得我的方法和它还是有差别的：

1.它明确地划分出了检测的区域（我当然也可以通过调整识别线来调整检测区域），通过是否在检测区域中，旨在检测区域中画点，而我是在全程画点，如果存在最后两个点正好分别在线的两侧来判断，在计数的一瞬间将之前所有的点清除。

2.并且在数据结构上我的方法更为优越，我使用了字典的数据结构，而它只用二维数组来实现。

3.我的代码简单，但是原理一致，相对情况下我的运行效率更好

4.2 实验收获与不足

收获：

1. 通过实验的实现，熟练地掌握了上课所讲授的背景减法器，以及高斯滤波，图像形态学变换，膨胀，腐蚀，开闭运算（这一块我觉得很有意义），轮廓查找，外接矩形，以及视频帧的读取和保存。
2. 并且自己根据相关内容，自己想出来并实现了一种优化方法，并且真的优化效果很好，并且想到了一些创意的小点，修改识别线来改变识别区域，非极大值抑制，并且尝试了很多种其他的方法，比如 yolo 算法。

不足：

1. 没有尝试在其他视频中使用实验中的实现进行识别，但是应该效果很不错（通过前期多次的画点，确定了一个车辆的存在，然后再在最后两个点是否在识别线的两侧来计数，准确性应该会很好）
2. 并且仍然存在一个车辆的识别框经常性的变化，如果要优化，应该保证识别框的稳定。

其他：有一个优化的思路，但是还没实现，通过多个点确定一辆车的存在，之后通过一些方法，使最开始识别到的图形框大小尽量与车辆绑定（这个”绑定“没有想到很好的实现方法）