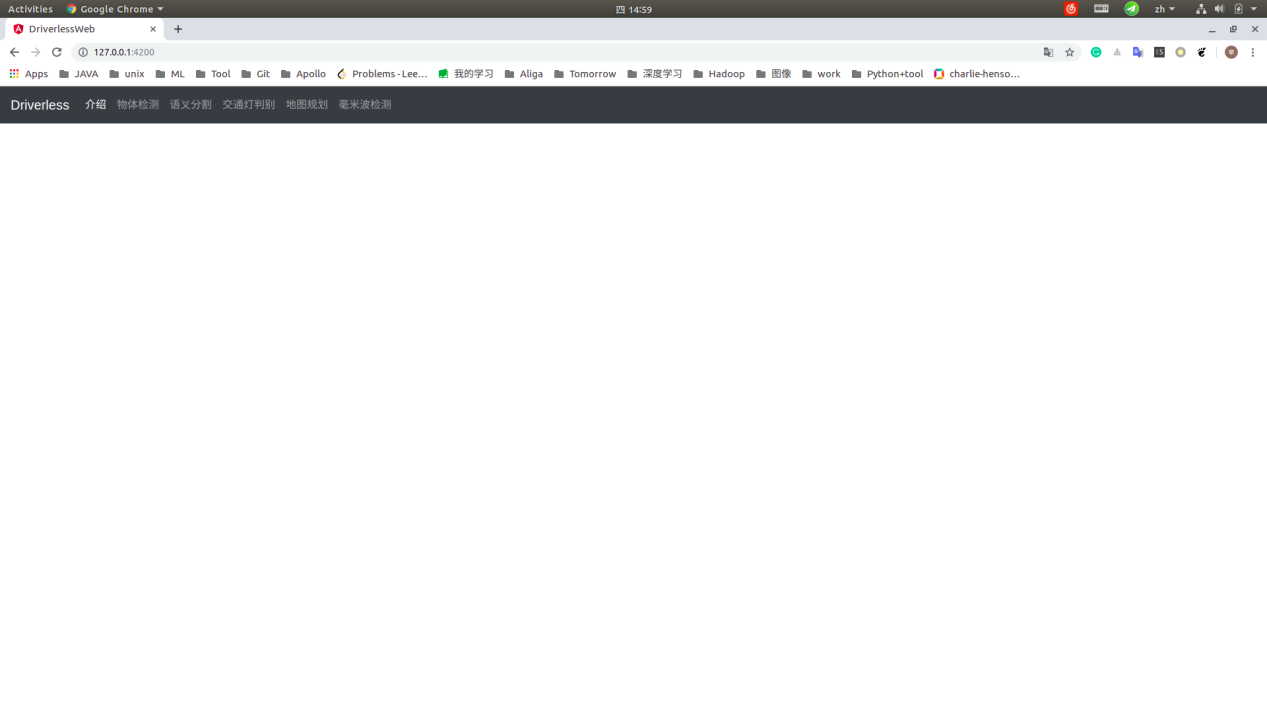
# 物体检测功能接口

## 前端规划

前端([driverless-web](http://git.iscssl.cn/KarlWong/driverless-web.git))大致分为五个页面，分别是物体检测页面、语义分割页面、交通灯判别页面、路径规划页面和毫米波检测页面，后面为展现完整性会增加一个页面将所有个功能放入一个页面进行展示。

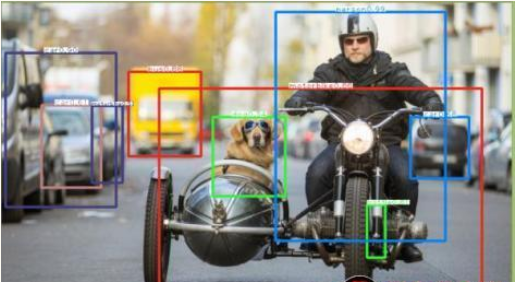
【已完成】Header中有五按钮，分别是物体检测页面、语义分割页面、交通灯判别页面、路径规划页面和毫米波检测页面。



图一 前端Header展示

【未完成】物体检测、语义分割和交通灯页面具有左侧边栏，为复选框，选中之后图像上会展示选中功能的处理后结果。

物体检测页面主要为视频展示，在视频中会对物体的类别和物理位置进行标定，如图二所示。



图二 物体检测结果视频展现

【未完成】所有页面都应该具有右侧边栏，用于数据的实时显示，预测结果正确率等，如当前图片正确与否，总体正确率，某个类别正确率。以具体的功能设计。

## 二、前端接口

showObjectDetection()：点击物体检测按钮后，进入视频界面，对视频使用检测后的结果进行标记展示，如图二所示。

checkboxObjectDetection()：左侧边栏的按钮，选中之后在视频中展示物体检测的结果，未选中则不展示物体检测的结果。

getObjectCategories()：获取物体检测的数据集中存在的类别数量和名称。

showHistoryDetail()：展示预测开始后的各项预测数据。【后期可能进行拆分】

showProgress()：显示当前任务的进度。

## 后端规划

后端分为两个独立的系统，Flask后端([driverless](http://git.iscssl.cn/KarlWong/driverless.git))为web提供数据访问，Model后端是数据进行训练和预测的后端，两者通过数据库进行联系【未讨论】。

### 2.1 Flask后端

需要提供对前端功能的支持。主要有一下接口：

saveObjectDetectionOutput()：对处理结果进行保存。

markData()：使用预测结果对视频数据进行标记。【可能放到前端进行，待讨论】

getObjectCategories()：返回物体检测的数据集中存在的类别数量和名称。

以及调用model后端进行预测。

### 2.2 model后端

数据管理通过，直接通过文件系统，不使用数据库【未讨论】，~/data下的文件管理规范。

由于模型不尽相同，同时运行时应该有先后顺序。【未讨论】

## 一、名词解释

## 1.1 Angular8

Angular是一个模块化开发的前端框架，目前由Google维护。

## 1.2 Bootstarp

Bootstrap，来自 Twitter，是目前最受欢迎的前端框架。Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的，它简洁灵活，使得 Web 开发更加快捷。

## 1.3 Component

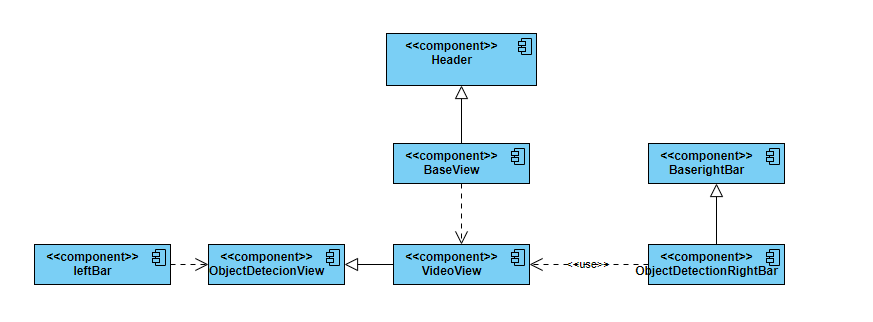
Angular概念，前端部分，每个任务会使用一个或者多个Component进行展示，一个Component就是一个模块，在协作开发时根据自己所开发的功能，写入具体的Compon，从而达到互不影响的效果。

## 1.3 Service

Angular概念，使用Service进行数据传递，使得数据模块和显示模块相互分离。

# 二、物体检测模块

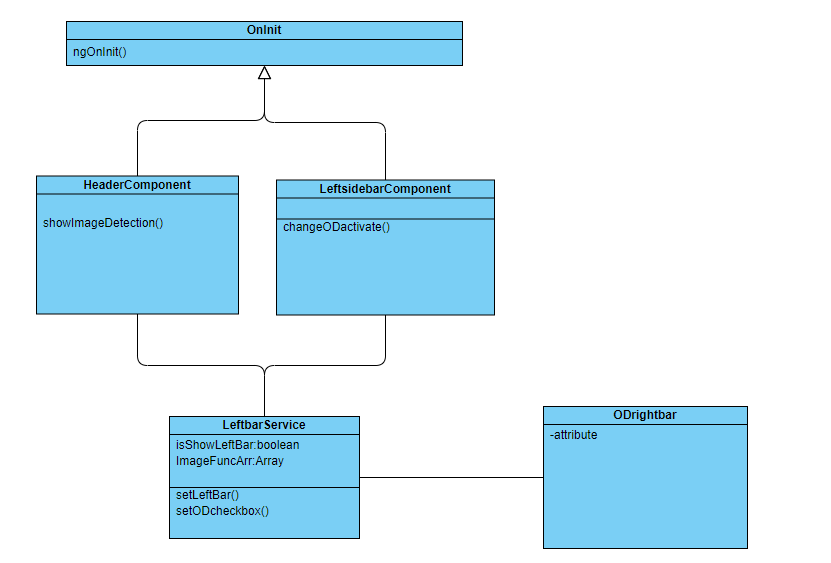
## 2.1 组件图



物体检测模块共有4个组件协同工作，由Header、ObjectDetectionRightBar、ObjectDetectionVIew和VideoView组成，每个组件具有不同作用。其他功能也会使用BaseView、BaseRightBar等基组件。

每个组件有各自的文件夹存放，文件目录中有

## 2.1.1 相关类图（待完善）



## 2.1.2 时序图（待完善）

## 2.1.3 流程图（待完善）

# 程序系统结构

前端组件模块、物体检测模块、语义分割模块、交通灯判别模块、毫米波感应模块、路径规划模块、智能提示模块、及时传输模块、日志模块、关键帧提取模块。

# 2、前端组件模块

## 2.1 程序描述

前端组件模块负责搭建一个可供展示的网页界面，该界面展示经过模型算法处理后的车载摄像头数据、毫米波雷达数据和实时构建车辆行径地图。用户可以根据网页显示实时监控车辆前方道路状态、车辆周围状态、车辆已行驶地图。

## 2.2 功能

前端组件模块下面包括八个小模块：

1. 头部导航栏模块：用户在此处选取需要展示的模块，有主页（一起展示）、图像检测、毫米波检测和地图构建，并且用户可以选择进行实时监控或者运行历史数据。
2. 左侧边栏：此模块属于图像模块的二级菜单，供用户选择视频流需要选择展示的功能，有物体检测，语义分割和交通灯判别。
3. 右侧边栏：此模块是对展示功能的数值描述，如当前进度，当前准确率，各类别准确率，历史数据文件选择。并且在前方存在危险的情况下进行只能提示。
4. 物体检测功能模块：负责在页面中间的车载视频展示中加入物体检测功能作用后的结果。
5. 语义分割功能模块：负责在页面中间的车载视频展示中加入语义分割功能作用后的结果。
6. 交通灯判定模块：负责在页面中间的车载视频展示中加入交通灯判定功能作用后的结果。
7. 毫米波功能模块：负责在页面中间的车辆俯视图中展示车辆周围信息。
8. 地图规划功能模块：负责在页面中间的空白处构建出车辆行驶路径。

## 2.3 流程逻辑

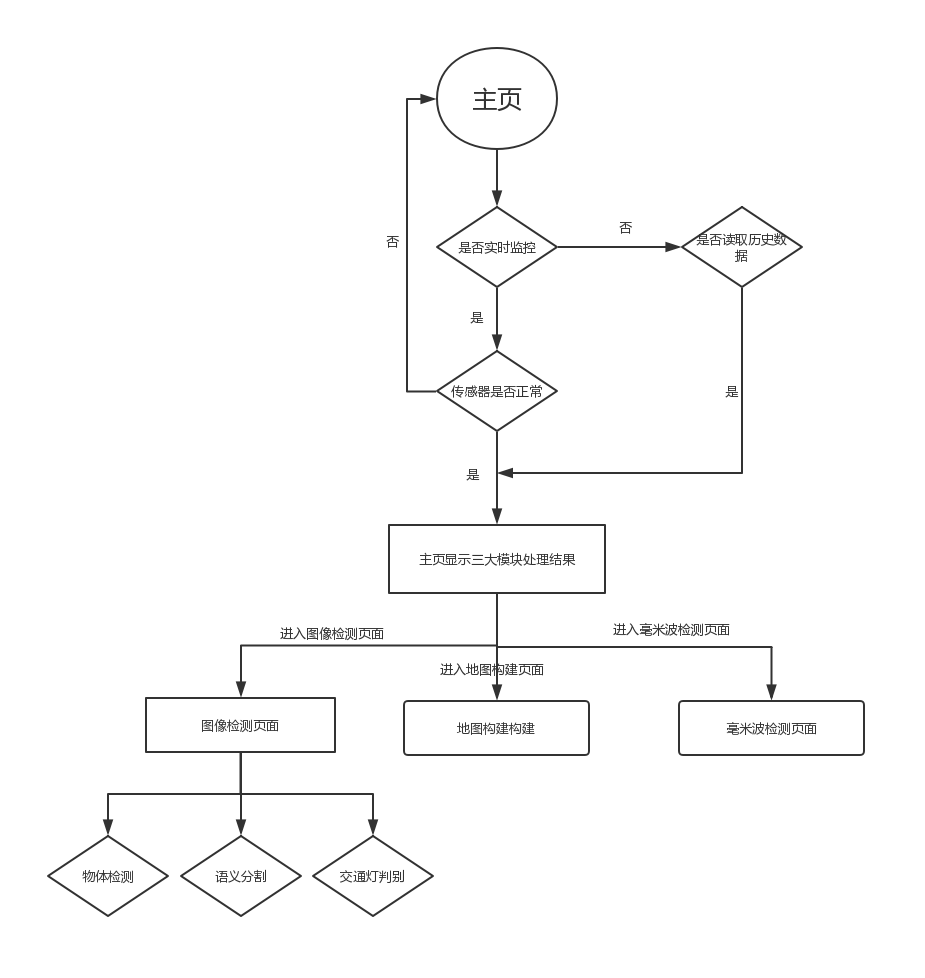


图2-1 网页操作流程

当用户打开网页终端后，操作流程如下：

1. 输入地址后，主页中选择实时监控或者是运行后台中存储好的数据。
2. 选择实时监控之后，会提示等待判断传感器是否连接，之后在主页中同时显示图像检测、毫米波检测和地图构建结果。
3. 选择运行历史数据后，提示选择具体的数据，之后等待数据加载，之后在主页中同时显示图像检测、毫米波检测和地图构建结果。
4. 在主页中可以在头部导航栏模块选择单独展示一个模块的功能。
5. 运行结束，结果保存，提示用户。

## .2.4 输入项

（1）：经过物体检测功能处理后的视频流，当前检测结果。

（2）：经过语义分割功能处理后的视频流，当前检测结果。

（3）：经过交通灯判断功能处理后的视频流，当前检测结果。

（4）：经过算法处理后的毫米波传感器信号。

（5）：构建地图需要的相关信息。

## 2.5 输出项

图像检测模块输出物体检测、语义分割、交通灯检测的结果。

（1）：针对物体检测，对每一个检测出的物体使用方框进行标记，针对不同的种类使用不同颜色进行区别，并且在右侧边栏对检测物体种类、置信度、对危险物体进行红色加强提醒。如果处于运行历史数据状态，则显示当前进度、当前准确率。

地图构建模块会在网页中实时构建车辆行驶过的路径，并且标记出车辆的当前位置。

毫米波雷达模块会对车辆周围的物体进行感应，最终在网页中的毫米波俯视图中标记出来，并且在右侧边栏上对危险物体进行智能提示。

## 2.6 设计方法

采用MVVM设计模式，利用前端框架Angular和CSS模板Semantic-UI进行构建，采用数据-视图分离。Service.ts是针对各个模块提供数据服务的，如leftbar.service.ts和rightbar.service.ts。用户通过访问service对象获取展示数据和实时显示。

## 2.7 接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数名 | 形参说明 | 返回值 | 函数功能 |
| 1 | ShowLeftBar() | 无 | 无 |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |

# 三、物体检测模块

## 3.1 程序描述

物体检测模块分为前端的视频流显示部分、后端深度网络模型和网站后端部分。当用户在导航栏选择了图像检测并且在左侧边栏选择了物体检测功能之后，前端部分发送给后端请求物体检测数据，后端接受到请求后调用模型进行物体检测，后端收集好数据进行回传给前端。

## 3.2 功能

物体检测模块主要对视频流上的物体进行类型判断和位置识别，并且在视频中标注出其具体类别和物理位置。通过右侧边栏的表单对处理结果进行文字描述。

物体检测由三个子模块组成：  
 （1）：前端显示模块，前端显示模块分为视频显示框和右侧边栏信息框，视频显示框针对摄像头传感器采集到的视频数据进行分析，最终显示经过物体检测功能处理后的视频图像；右侧边栏对检测到的物体进行文字描述，如物体类别，物体位置等；对危险距离的物体采用突出提示。

（2）：网站后端模块：针对前端提出的调用物体检测功能或者停用物体检测功能进行接受，并且操作后端深度网络模型进行相应的操作。

（3）：后端深度网络模型：算法层面的模型，针对请求调用相应的网络进行运算。

## 3.3 性能要求

对实时视频流的检测不低于30帧每秒，对历史数据视频流（数据集）的检测，不低于40帧每秒，且准确率高于78%；保证处于危险位置的物体可以完全检测出；在昏暗环境下检测物体具有较强的适应性。

## 3.4 输入项

车载摄像头采集的视频流信息，最终转化为数帧图片进行输入。

## 3.5 输出项

通过物体检测算法后的结果标记数据。

## 3.6 设计方法

（1）模型训练（视频流，numpy格式？？、模型参数、保存方式、传输方式）转换格式函数、格式、模型类。

在物体检测模块加入到前端模块进行实时检测之前，需要对其算法网络模型进行训练，数据集使用到的JPEG格式文件需要通过trans\_array函数进行转换为numpy格式的三维宽高相等的数字型矩阵（3x224x224）;算法模型类class Network通过torch框架进行构建，通过函数\_\_init\_\_对其拥有的每一层网络进行声明，这些网络层是通过conv2D、BatchNorm2D、ReLU、MaxPool（）等基础网络结构进行复合而得到的。模型参数存在于网络结构中，使用浮点矩阵dtype=float32进行存放，函数forward()使之前声明好的网络层有序地连接起来，构成一整张网络；通过train（）函数对构建成功的网络进行训练，在训练期间需要通过CrossEntropyLoss()定义的损失函数进行backward()反向传播从而更新参数。

（2）前端请求包

当前端请求执行物体检测任务的时候，需要向后端传递的请求包应该包括以下几点：运行数据名称run\_data，当此字段为’realtime’时，检查传感器是否正常工作，非正常工作则返回报错包，当此字段为其他字符串时，检查数据库是否存在数据，若无此数据则返回报错包。

（3）模型输出数据

当Numpy格式的图片矩阵传入算法模型，在图像进行预测之后，对模型输出进行读取和实时传输阶段，定义数据结构OD\_data进行存储数据，将所有类别进行编号，并且将预测的类别信息转换成编号放入成员category中,并且将置信度估计放入其成员变量prob中。一张图片中可能存在多个物体，所以一张图片中会生成多个OD\_data。每当生成一张图片的全部预测之后，后端将OD\_data数组上的信息绘制到原始图片。

（4）后端返回数据包

后端针对已经生成的OD\_data数组进行格式化，连同其他功能的模块一起发送给前端。

Forward（）

Back word（）

Lossfunction（）

Train（）

Test（）

Net()

Params

程序流程图

分为实时和历史数据

## 3.7 程序流程图

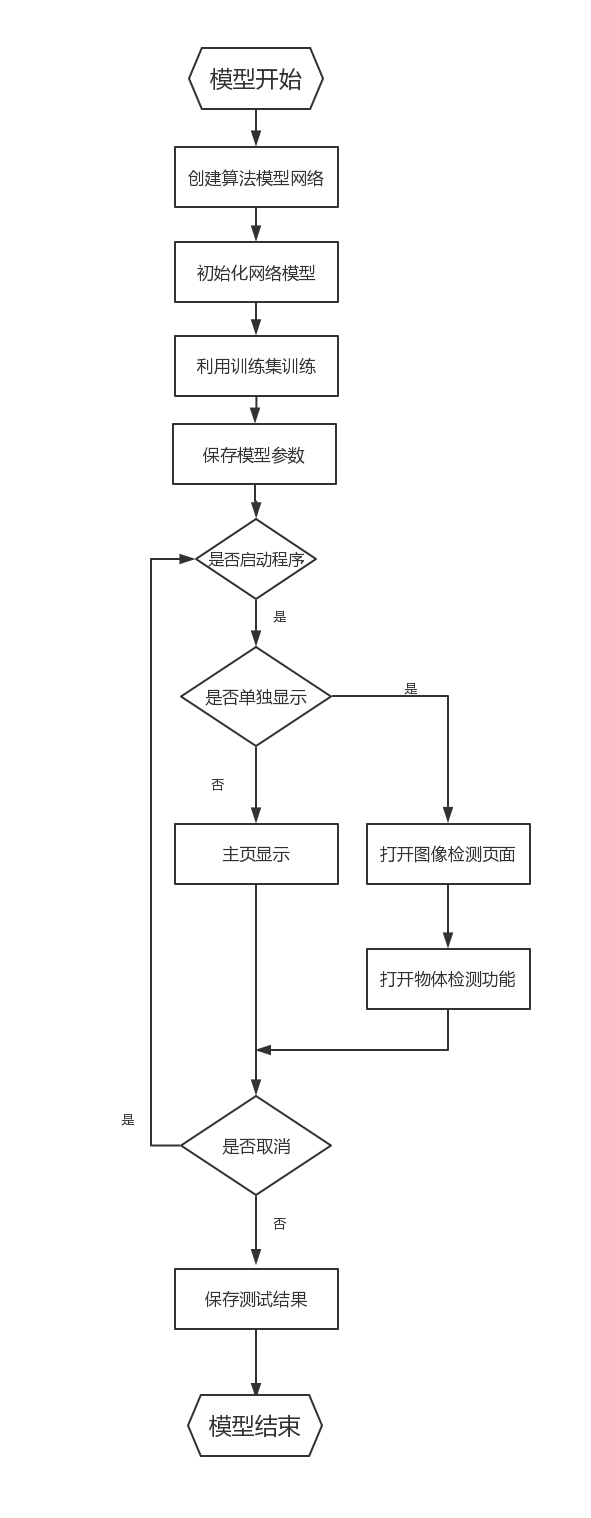


图3-1 物体检测模块流程图

## 3.8 接口

该模块接口分为三个部分，前端接口、算法模型接口，后端传输接口，前端通过show\_OD函数调用后端启动物体检测算法，后端通过collect\_OD\_data进行对模型结果的收集，其中还涉及到了格式转换函数trans\_array，存储函数save\_OD\_params、save\_OD\_data等，具体函数接口如表3-1所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数名 | 形参说明 | 返回值 | 函数功能 |
| 1 | ShowOD(any:status,$event) | Status表明当前网页所属状态，event会记录鼠标或者键盘的所有动作。 | 布尔值，函数是否成功 | 调用隐藏函数hide\_others()对其他功能板块进行隐藏，并显示物体检测页面，调用send\_OD\_request函数。调用check\_sensor函数。 |
| 2 | Hide\_others(any:status) | Status表明当前网页所属状态。 | 布尔值，函数是否成功 | 隐藏其他功能的前端模板。 |
| 3 | Show\_OD\_rightbar  (data\_source) | Data\_source表明数据源是属于历史数据还是实时检测 | 布尔值，函数是否成功 | 显示右侧边栏的信息。 |
| 4 | Check\_sensor(string:cerma) | 字符型数据，表示需要检查的传感器类型 | 成功或者是具体的错误代码。 | 检查传感器是否正常工作。 |
| 5 | Trans\_array(file\_name) | File\_name表明需要进行转换的图片文件，一般为JPEG格式 | 返回numpy.float32类型的3维矩阵 | 将JPEG格式的图片转换为数字矩阵 |
| 6 | Collect\_OD\_data(array:out) | out是算法模型的输出1x1004的数组。 | 返回剔除无用信息后的1x2数组 | 将模型的输出规范化。 |
| 7 | Save\_OD\_params(net) | Net是网络模型类 | 布尔值，是否成功 | 将模型结构、参数持久化。 |
| 8 | Save\_OD\_data(array:1x2) | 规范化后的数组 | 布尔值，是否成功 | 将规范化的输出进行存储。 |
| 9 | Start\_OD\_model(data\_loader) | Data\_loader表明分析数据是属于传感器传输还是历史数据 | 布尔值，OD算法模块是否成功启动。 | 启动后端的物体检查算法模块。 |
| 10 | Net.forward(array) | 三维numpy数组，通过图像转换得到 | Out模型预测结果 | 模型进行前向传播得到预测结果。 |
| 11 | Net.backward() | 类内函数 | 无 | 对模型的训练参数进行反向传播更新。 |
| 12 | Cross\_entropy\_loss(out, real) | Out表示模型输出，real表示真实输出 | 输出out和real的交叉熵 | 对预测输出和真实输出进行比较得到交叉熵。 |

## 3.9 存储分配

传感器实时数据存储:/data/camera

历史数据存储:/data\_set/camera/train 和/data\_set/camera/val

## 3.10 注释设计

对程序的注释：在该模块中各程序的相关信息和对程序功能的说明。

对变量的注释：在该模块中程序用到的主要变量都作出必要的说明。

对函数的注释：在该模块中每个自定义函数的前面都有对该函数的功能说明，参数调用情况，返回值等的说明。

对关键算法的注释：对比较复杂的算法加以专门的说明，以便加强程序的可读性

## 3.11 限制条件

需要显卡内存大于2G，并且项目前端成功启动。

## 3.12 测试计划

测试平台：

操作系统：Ubuntu16.04 LST

前端框架：Angular8.0.4 flask1.0