安阳工学院

ANYANG INSTITUTE OF TECHNOLOGY

本 科 毕 业 论 文

基于深度学习的自动驾驶小车的研究与实现

Research and Realization of Automatic Driving Car Based on Depth Learning

学院名称：计算机科学与信息工程学院

专业班级：网络工程 13-1

学生姓名：申恒恒

学　　号：13031110141

指导教师姓名：闫怀平

指导教师职称：讲师

毕业设计（论文）原创性声明和使用授权说明

原创性声明

本人郑重承诺：所呈交的毕业设计（论文），是我个人在指导教师的指导下进行的研究工作及取得的成果。尽我所知，除文中特别加以标注和致谢的地方外，不包含其他人或组织已经发表或公布过的研究成果，也不包含我为获得安阳工学院及其它教育机构的学位或学历而使用过的材料。对本研究提供过帮助和做出过贡献的个人或集体，均已在文中作了明确的说明并表示了谢意。

作 者 签 名：_____ 日 期：_____

指导教师签名：_____ 日 期：_____

使用授权说明

本人完全了解安阳工学院关于收集、保存、使用毕业设计（论文）的规定，即：按照学校要求提交毕业设计（论文）的印刷本和电子版本；学校有权保存毕业设计（论文）的印刷本和电子版，并提供目录检索与阅览服务；学校可以采用影印、缩印、数字化或其它复制手段保存论文；在不以赢利为目的前提下，学校可以公布论文的部分或全部内容。

作者签名：_____ 日 期：_____

目　录

引　言 7

第 1 章　需求分析 8

1.1 系统分析 8

1.1.1 功能分析 8

1.1.2 硬件需求 8

1.2 系统架构 9

第 2 章　整体设计 10

2.1　目标 10

2.2　系统的设计 10

2.2.1 输入单元 10

2.2.2 处理单元 10

2.2.3 控制单元 11

第 3 章　详细设计 13

3.1 TCP 服务器 13

3.2 神经网络模型 13

3.2.1 多层感知机模型（ANN_MLP） 14

3.2.2 卷积神经网络模型（CNN） 16

3.3 目标检测 19

第 4 章　具体实现 21

4.1 数据收集与处理 21

4.1.1 视频流网络传输 22

4.1.2 样本集的生成 22

4.2 模型的设计、实现、训练和测试 25

4.2.1 多层感知机的实现 25

4.2.2 卷积神经网络的实现 26

4.3 结论 29
第 5 章 系统测试 30
5.1 模块测试 30
5.1.1 实时视频传输 30
5.1.2 实时识别目标 30
5.1.3 基于小车观测的图像来预测方向 31
5.2 整体测试 32
5.2.1 收集数据 32
5.2.1 训练模型 32
5.2.3 最终测试 32
结 论 34
致 谢 35
参考文献 36
附录 37

基于深度学习的自动驾驶小车的研究与实现

摘要：自动驾驶是室外移动机器人在交通领域最重要的应用之一。从传统上改变交通工具的控制模式和开发难度， 但是大大地提高了交通系统的效率。自动驾驶小车就是基于自动驾驶汽车模型开发的一个小型的自动驾驶模型。首先，讨论目前系统的需求，即需求分析；其次，将阐述自动驾驶的最新进展和相关研究，并且将会对神经网络尤其在机器学习以及深度学习的应用进行描述，包括图像处理、MLP 模型、ANN 算法、CNN 模型、BP 算法等若干关键技术。接下来，将沿着需求分析进行展开，逐步完善和实现该需求；最后将对模型的结果进行分析，并得出相应的结论。

关键词：自动驾驶　机器学习　深度学习　MLP 模型　CNN 模型

Research and Realization of Automatic Driving Car Based on Depth Learning

Abstract： Automatic driving is one of the most important applications of outdoor mobile robots in traffic field. It is difficult to change the control mode and development of the vehicle, but greatly improves the efficiency of the transportation system. The auto-driving car is a small driving model developed based on the auto-driving model. Firstly, discussing the current system demand, i.e. demand analysis, and secondly, the newest progress and related research of automatic driving will be expounded, and the application of neural network especially in machine learning and deep learning is described, including some key technologies such as MLP, CNN and BP algorithm. Next, we will proceed along the demand analysis to gradually improve and realize the demand, and finally analyze the results of the model and draw the corresponding conclusions.

Key words： Auto-driving ; Machine Learning ; Deep Learning ; MLP Model ; CNN Model

引　言

自动驾驶小车又称之为智能车，是目前唯一可以代替传统的汽车的一种交通工具，它从根本上改变传统的汽车控制方式，从原本的“人-车-路”闭环控制方式[1]到“车-路”控制方式。这样的处理方式大大地提高了交通工具的效率和安全，但这也增加了汽车本身运算和负载能力，它使用了各种传感器来感知道路，车辆、障碍物等这些十分重要的信息，然后通过后台的强有力的运算资源，来控制车辆的行驶速度和转向，从而使得小车安全的在道路上行驶。

目前我正在做的自动驾驶小车模型是基于视觉感知来实现的。通过改装普通遥控小车的控制模式，从原来的手工遥控控制到自主驾驶，其后使用了大量的数据处理和机器学习技术，其中包括了视频网络传输、图像处理、摄像机标定、单目视觉测距、机器学习/深度学习算法等等，并且在树莓派硬件的支持下实现的。本文认为自主驾驶小车不再是一个只有依靠大公司等背景下开发的产物。并且值得说明的是本项目主要参考了国外视频网站YouTube上几个有关自动驾驶遥控车的例子，分别是OpenCV Python Neural Network Autonomous RC Car[2]、MATLAB Neural Network Autonomous Car[3]、How to Build Your Own Self Driving Toy Car[4]等。

第 1 章　需求分析

1.1 系统分析

1.1.1 功能分析

由于目前智能小车的定位偏向于自主驾驶，因此智能小车主要实现了以下几个功能：自主控制驾驶，避障，行人检测，交通信号灯检测，交通标志牌检测，警告语音提示（目前正在做）以及上述检测过后的行为决策等功能。

1.1.2 整体流程

在此之前需要声明的是在基本测试过程中，主要使用了笔记本和树莓派来做，但后期的测试以及现实应用，将会将所有功能和程序集成在树莓派上。一下会从下向上地阐述有关智能小车的各个模块。

首先小车主体由玩具遥控汽车（RC Car）改装而成，改装的主要部件是其控制部分（遥控器）。然后利用了树莓派先天的优异的硬件 GPIO 引脚控制编程环境，进行快速的底层改造和实现。

接下来，就是对装载在小车上的树莓派进行环境配置，主要包括以下主要工具：

1）为了更便利且更安全地向外提供访问接口，将小车自身配置成热点（AP）；

2）结合上述的功能分析可知，由于需要树莓派提供视频信息，距离，声音等数据信息，因此对传感器的选择是必不可少的环节；

3）为了能够快速的进行开发，在这里树莓派和 PC 上使用的均是 Python 环境，对图像数据的处理均是利用了 OpenCV（3.1）环境。如果单机下，还需要树莓派装有 Tensorflow 和 Keras 等深度学习框架。

然后，对 PC 机上对环境与树莓派相似，由于树莓派的性能相比 PC 性能较低，因此，需要将树莓派采集的图像数据传输到 PC 机，然后再做相应的机器学习建模。模型一旦保存下来，就可以就可以在相同的机器学习框架环境下进行加载和预测。

最后，利用 OpenCV 的级联分类器，训练和生成相关的特定二分类器。结合各个模块之间的联系，进行组合和优化环节，最终完成一部智能小车

1.1.2 硬件需求

通过上述分析，主要需要利用以下材料：

1）树莓派 3 B+

2）PC 8GRAM GEFORCE GTX 4G

3）树莓派摄像头模块

4）无线网卡（外配，做 AP）

4）声音模块（可能有）

5）超声波模块

6）电源和若干杜邦线

1.2 系统架构

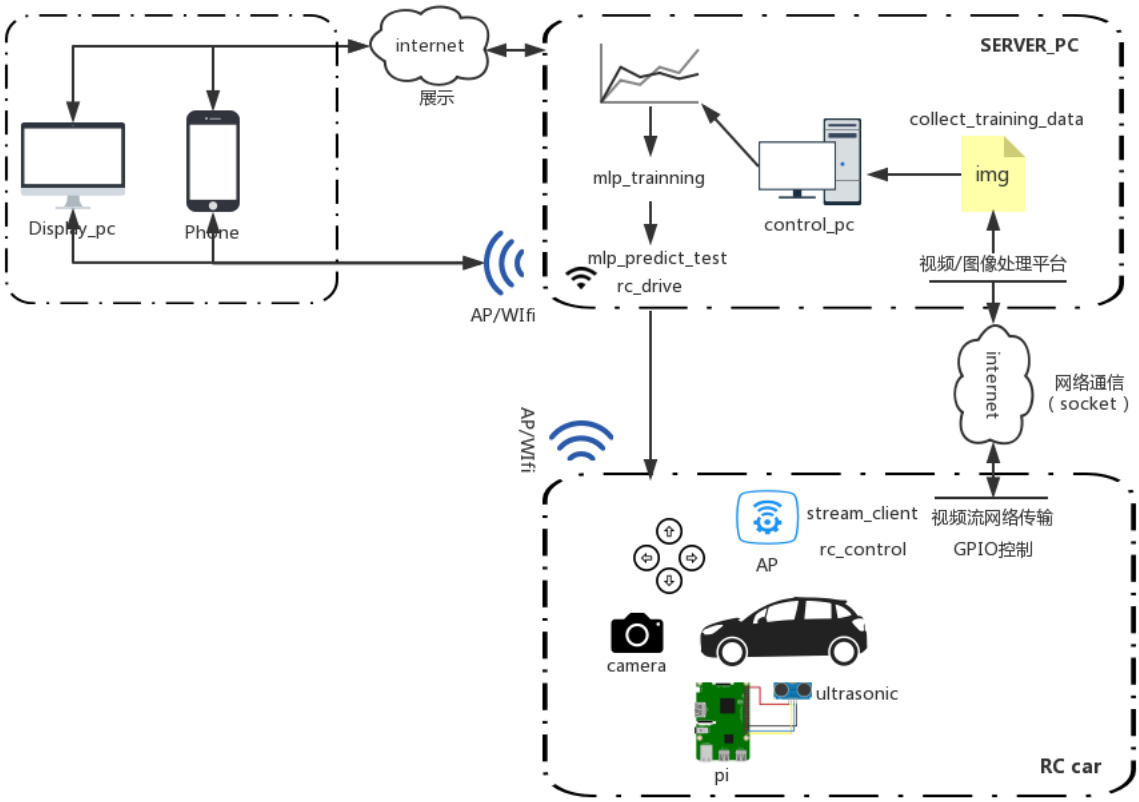

图 1.2 整体架构图

树莓派数据主要来源于各个传感器所检测到的数据，在树莓派上经过简单的处理然后通过 SocketServer 与 PC 通信，树莓派与 PC 通信主要依靠于热点共享，PC 机将树莓派所接收到的数据进行深层次的加工处理，然后喂给机器学习算法进行训练，然后经过验证数据集和测试数据集等评价模型的好坏，当准确度达到一定的指标时，停止训练，将模型保存然后加载到树莓派或 PC 上，利用单机或双机环境下，进行实时的对图像进行决策，当树莓派接收到相关的决策指令，然后控制小车进行移动。

第 2 章　整体设计

2.1　目标

在车道上自动驾驶，交通标志牌和交通信号灯的识别，前方障碍物（比如前方小车，行人）的检测，并控制小车避免与其碰撞或追尾.

2.2　系统的设计

该系统由三个子系统所组成，其中包括了输入单元（摄像头，超声波传感器）、处理器单元（PC 的 CPU 和 GPU，树莓派的计算资源）和小车的控制单元。

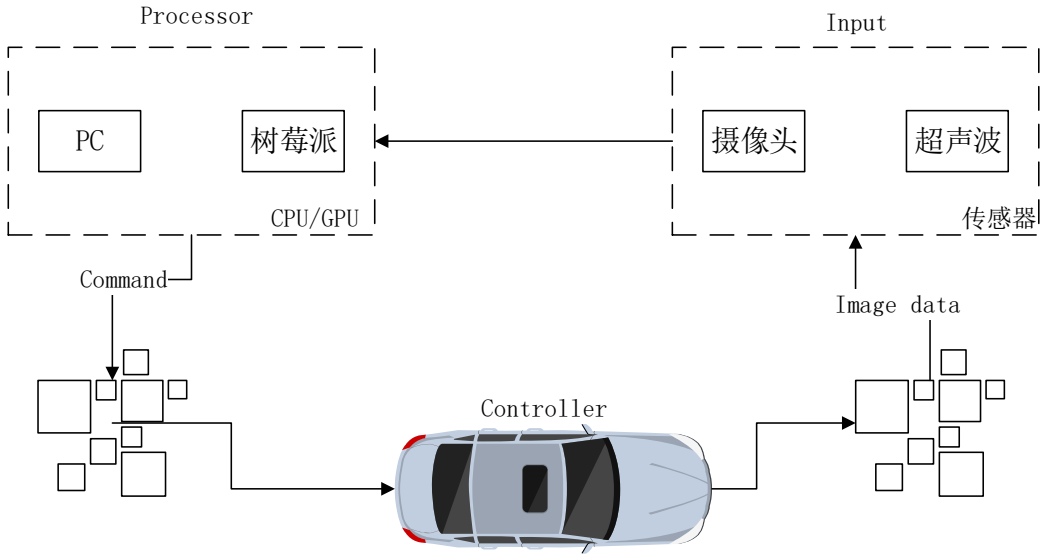

图 2.2 该系统有三个子系统构成，分别是是输入单元、处理器单元和控制单元

2.2.1 输入单元

树莓派 3 B+板且附有一个树莓派摄像机模块，和一个用来采集距离数据的 HC SR04 超声波传感器。树莓派运行三个客户端脚本，一个是用来传输视频流数据的，另外一个是通过网络连接（这里使用树莓派做热点）将超声波传感器感知的距离数据通过 TCP/IP 网络传输到本地计算机，还有一个是用于控制树莓派 GPIO 引脚的，驱动小车移动的。为了实现低延迟的视频流。在这里视频分辨率设置为 320×240 大小。

2.2.2 处理单元

PC 机在本地执行多个任务，分别是接收来自树莓派的数据、训练机器学习算法和做出预测([前进，后退，左转，右转])，目标检测（交通标志和交通信号灯），测量与目标的距离（利用单目视觉技术），并将指令发送给树莓派，然后树莓派依据设定的 GPIO 引脚来

驱动小车执行指定的动作。（后续将逐步展开具体描述）

2.2.3 控制单元

该项目中使用的小车有一个开/关控制器。当按下按钮时，相关芯片引脚和地之间的电阻为零。因此，使用树莓派的 GPIO 引脚来模拟按钮按下操作。选择树莓派的四个 GPIO 针脚分别连接控制器四个芯片引脚上，分别对应于前进、后退、左转和右转的动作。一方面 GPIO 引脚发送 LOW 信号显示控制器芯片的接地引脚；另一方面发送 HIGH 信号表明芯片引脚之间的电阻和地面保持不变。GPIO 通过若干线与小车的控制板相连。计算机通过 socket 向树莓派发送指令，然后树莓派读取命令，通过 GPIO 写出 LOW 或 HIGH 信号，模拟通过按下按钮来驾驶小车。

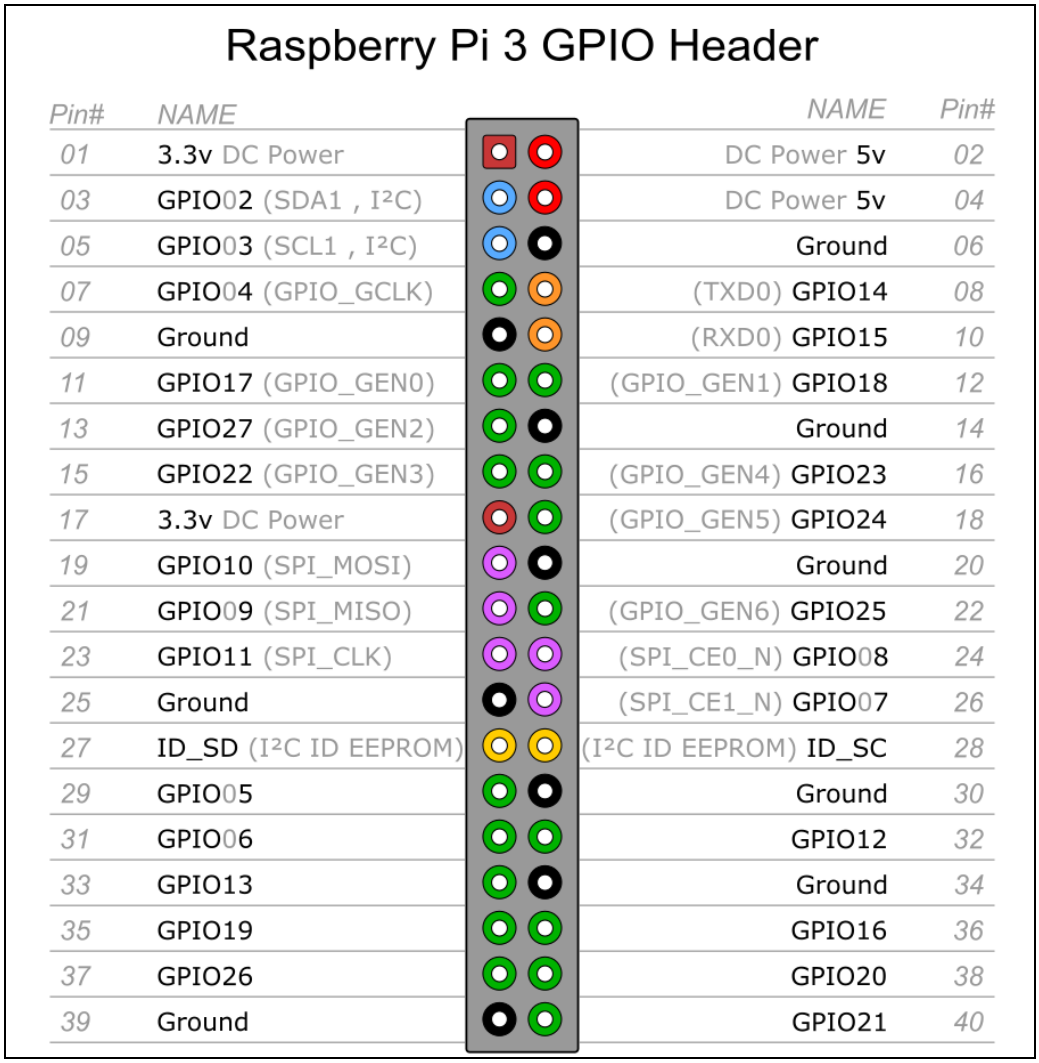

图 2.2.3 树莓派 3 每个 GPIO 引脚的功能[1]

本项目中引脚的使用情况：

[1] 图片来源于 www.element14.com/RaspberryPi

小车控制和超声波控制			
功能	引脚	功能	引脚
前进	11	后退	7
左转	13	右转	15
ECHO	16	TRIG	18
VCC	4	GND	14

表 2.2.3 引脚使用情况

第 3 章　详细设计

3.1 TCP 服务器

运行在计算机端的多线程 TCP 服务器，负责接受来自树莓派的视频流数据和超声波感知的数据，其中每一帧图像被转化为灰度图像并存储在 numpy 数组中。

3.2 神经网络模型

使用神经网络的优点是，一旦神经网络训练完毕，它只需要加载训练完成后生成的模型文件即可，因此它的预测是非常快的[5]。为了更好地训练模型，在这里特别说明一下图像处理的一个关键技巧，能大大地减少了训练数据的代价。在这里选取了捕捉到的图像的下半部分作为训练数据，其中该部分数据在计算机视觉中常被称作 ROI，即感兴趣区域。因此图像数据维度的减小将大大地减少了训练的难度和时间。

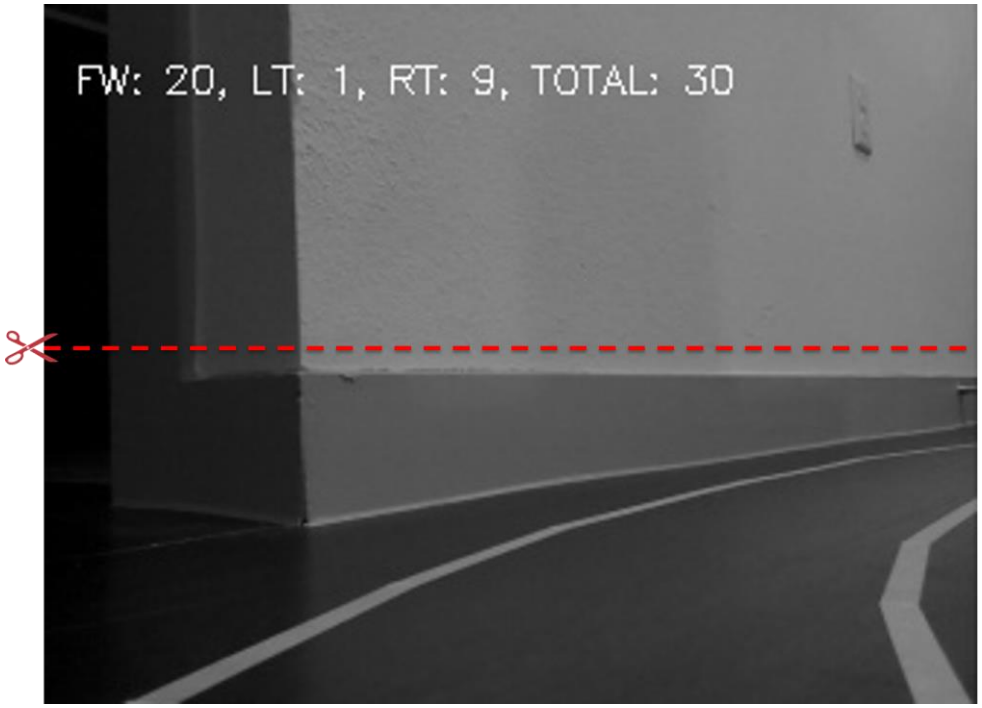

图 3.2（1） 摄像头模块捕捉到的原始图像（320px*240px）

图 3.2（2） 由于图像的上半部分与小车驾驶的任务无关，因此剪掉图像的上半部分，此时的图像大小为 320px*120px

3.2.1 多层感知机模型（ANN_MLP）

深度前馈网络，也叫作前馈神经网络或者多层感知机（MLP），是典型的深度学习模型。多层感知机仅仅是一个将一组输入值映射到输出值的数学函数。该函数由许多较简单的函数复合而成。不同数学函数的每一次应用都为输入提供了新的表示。

由于图像数据是小车移动的最终决定因素,即两者存在着较强的相关性,但并非一定是线性关系,因此从直观上来看,可以通过某种统计工具和数学手段找出这种关系. 而神经网络模型更像一个黑箱具有十分强大的非线性映射功能[6],因此对于图像数据与小车移动之间的复杂关系显得十分有效.

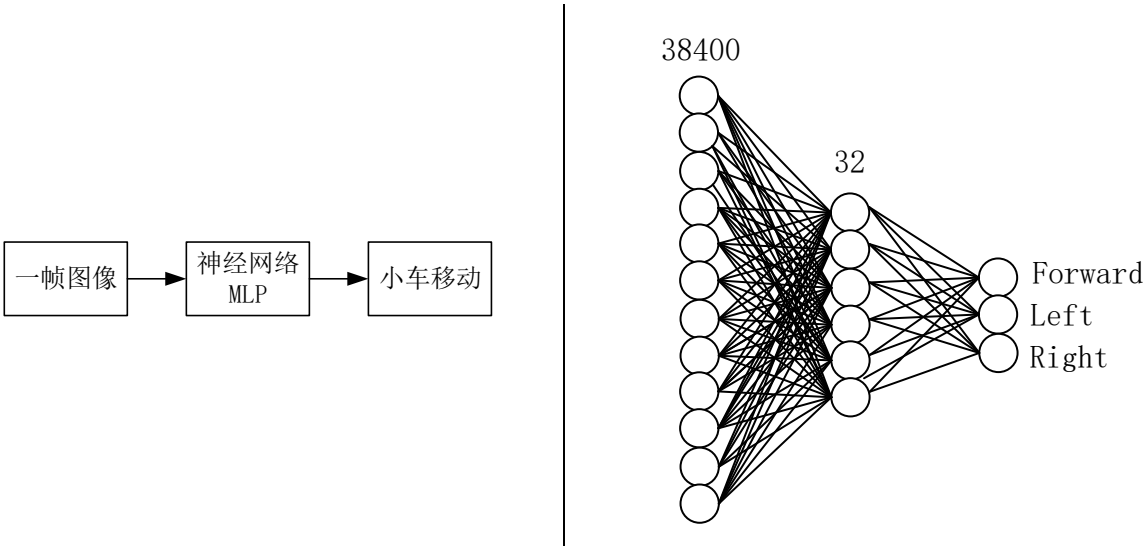

图 3.2.1（左）抽象地表示了预测模型的基本思想 图 3.2.1（右）描述了三层 MLP 神经网络结构,即输入层、隐含层及输出层,其中输入的节点数目等于 pixels,输出节点数目等于所要预测的小车动作的个数，这里是 3 个.

在输入层有 38,400(320 * 120) 个节点，在隐含层的数目是选择是随意选择的，在这里指定隐含层有 32 个神经元节点。在输出层有三个节点，每个节点对应于驱动小车的

控制指令，分别是前进、左转、右转，(在此项目，剔除掉后退指令)。

在本项目使用了 OpenCV 的机器学习工具箱中的 cv2.ml.ANN_MLP_create() 函数，建立多层感知机模型，然后利用 setLayerSizes(np.int32([38400, 32, 3])) 函数指定 MLP 网络结构，即 Input layer、hidden layer 和 output layer 的神经元个数分别是 38400，32 和 3。setTrainMethod(cv2.ml.ANN_MLP_BACKPROP|cv2.ml.ANN_MLP_UPDATE_WEIGHTS) 设置利用反向传播算法来更新参数；setActivationFunction(cv2.ml.ANN_MLP_SIGMOID_SYM) 设置 Sigmoid 函数为激活函数；setTermCriteria (cv2.TERM_CRITERIA_COUNT, 100, 0.001) 函数用于设置优化的参数。train(np.array(train_data, dtype=np.float32), cv2.ml.ROW_SAMPLE,np.array(train_labels, dtype=np.float32)) 开始训练模型，训练数据为（train_data，train_label）.

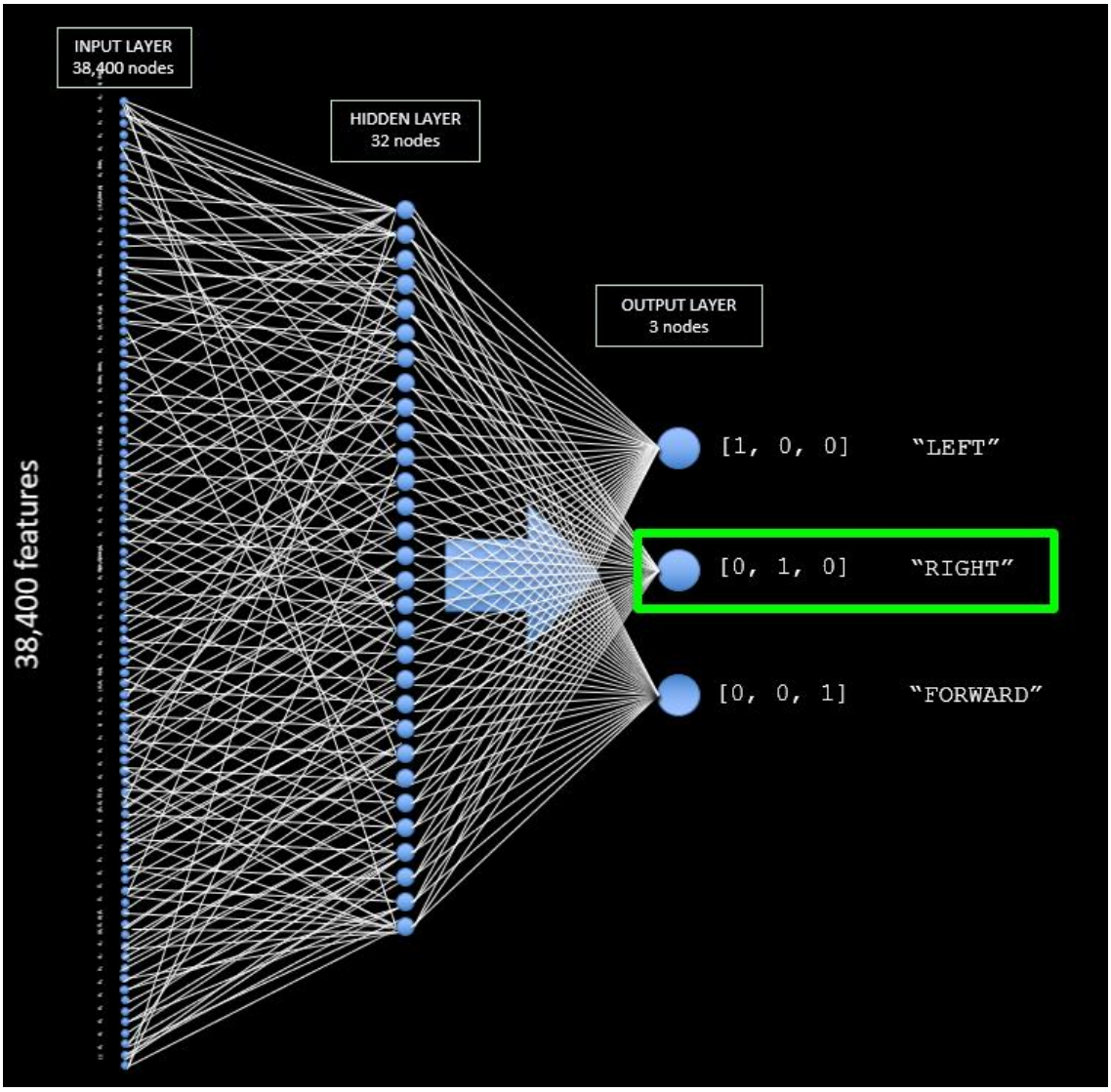

图 3.2.1（2） 将图像数据 Flatten 后，喂给 MLP 模型，然后进行预测，输出层输出的是概率，概率高的为最后的预测结果

图 3.2.1（3）显示了训练数据的收集过程。首先把每帧图像裁剪（图 3.2）后并转换为一个 numpy 数组。然后标记为相应的 label。最后，所有标记完成的图像数据和相应标签合并后保存成 npz 格式的数据文件。接下来使用反向传播算法训练数据。训练完成后，将生成的神经网络节点的权重被保存为 xml 文件。若要对新的图像数据生成预测，直接加载 xml 文件构建的模型与训练数据生成的神经网络具有相同的网络结构，若要进行预测，直接加载模型文件即可。

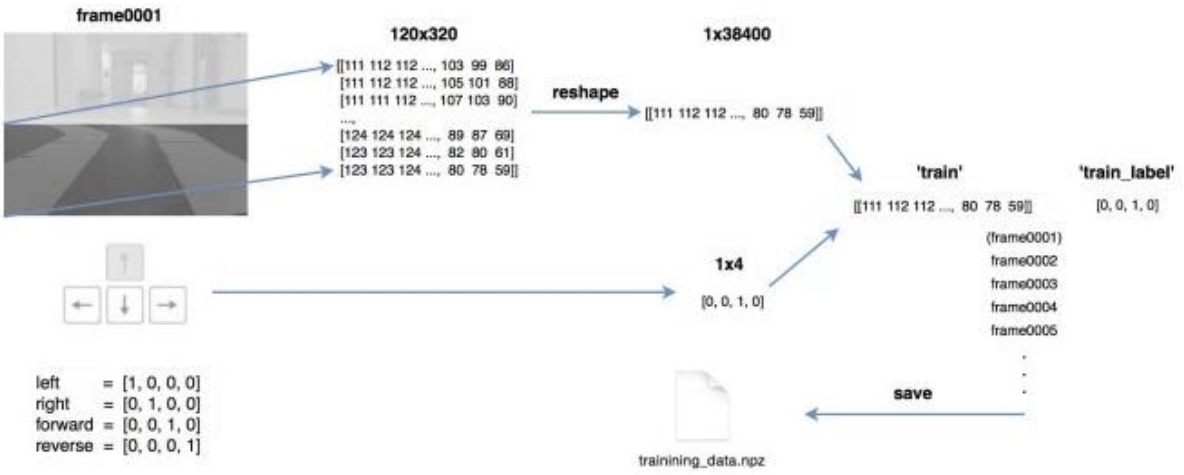

图 3.2.1（3） 图像处理步骤[2]

3.2.2 卷积神经网络模型（CNN）

卷积网络，也叫做卷积神经网络（convolutional neural network, CNN），是一种专门用来处理具有类似网格结构的数据的神经网络。例如时间序列数据（可以认为是在时间轴上有规律地采样形成的一维网格）和图像数据（可以看作是二维的像素网格）。卷积网络在诸多应用领域都表现优异。“卷积神经网络” 一词表明该网络使用了卷积（convolution）这种数学运算。卷积是一种特殊的线性运算。卷积网络是指那些至少在网络的一层中使用卷积运算来替代一般的矩阵乘法运算的神经网络.[7]

深度学习以卷积神经网络（CNN，Convolutional Neural Network）为代表，相比早先的浅学习，它不但可以从局部到全局提取不同层次的特征参数，还可以利用卷积的微分性质通过改变卷积核在更高阶上提取特征参数，是抽象认知能力的提升，而不仅仅是神经网络的宽度——神经元数目的增加[8]。

深度学习让计算机通过较简单概念构建复杂的概念。图 3.2.2(1) 展示了深度学习系统如何通过组合较简单的概念（例如转角和轮廓[9]，它们转而由边线定义）来表示图像

[2] 图片来源于 https://zhengludwig.wordpress.com/projects/self-driving-rc-car/

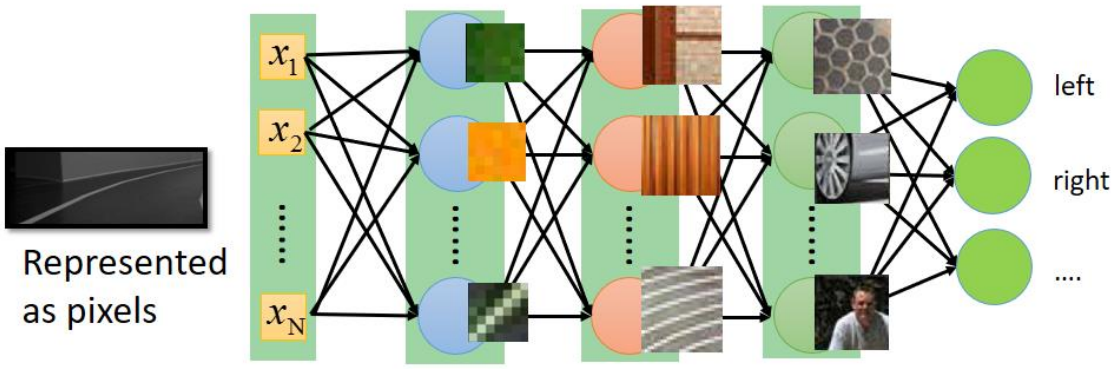

图 3.2.2（1） 深度学习模型的示意图。计算机难以理解原始感观输入数据的含义，如表示为像素值集合的图像。将一组像素映射到对象标识的函数非常复杂。如果直接处理，学习或评估此映射似乎是不可能的。深度学习将所需的复杂映射分解为一系列嵌套的简单映射（每个由模型的不同层描述）来解决这一难题。输入展示在可见层（visible layer），这样命名的原因是因为它包含我们能观察到的变量。然后是一系列从图像中提取越来越多抽象特征的隐藏层（hidden layer）。因为它们的值不在数据中给出，所以将这些层称为“隐藏”；模型必须确定哪些概念有利于解释观察数据中的关系。这里的图像是每个隐藏单元表示的特征的可视化。给定像素，第一层可以轻易地通过比较相邻像素的亮度来识别边缘。有了第一隐藏层描述的边缘，第二隐藏层可以容易地搜索可识别为角和扩展轮廓的边集合,即纹理。给定第二隐藏层中关于角和轮廓的图像描述，第三隐藏层可以找到轮廓和角的特定集合来检测特定对象的整个部分。最后，根据图像描述中包含的对象部分，可以做出相应的预测[7]。

常规神经网络对于大尺寸图像效果不尽人意。在 CIFAR-10[3]中，图像的尺寸是 32x32x3（宽高均为 32 像素，3 个颜色通道），因此，对应的的常规神经网络的第一个隐层中，每一个单独的全连接神经元就有 32x32x3=3072 个权重。这个数量看起来还可以接受，但是很显然这个全连接的结构不适用于更大尺寸的图像。举例说来，一个尺寸为 320x120x1 的小车捕捉到的图像，会让神经元包含 320x120x3=38,400 个权重值。而网络中肯定不止一个神经元，那么参数的量就会快速增加！显然，这种全连接方式效率低下，大量的参数也很快会导致网络过拟合。而卷积神经网络可以避免参数过多，从而消除过拟合的风险。

一个简单的卷积神经网络是由各种层按照顺序（Sequential）排列组成，网络中的每个层使用一个可以微分的函数来激活数据从一个层传递到另一个层。卷积神经网络主要由三种类型的层构成：卷积层，汇聚（Pooling）层和全连接层（全连接层和常规神经网络

[3] 有关 CIFAR-10 数据集详见：https://www.cs.toronto.edu/~kriz/cifar.html

中的一样），也常用到的技巧是增加 Dropout 层来实现避免过拟合的情况。通过这些层的叠加，就可以构建一个完整的卷积神经网络。比如图 3.2.2（2）所描述就是一个典型的卷积神经网络结构。

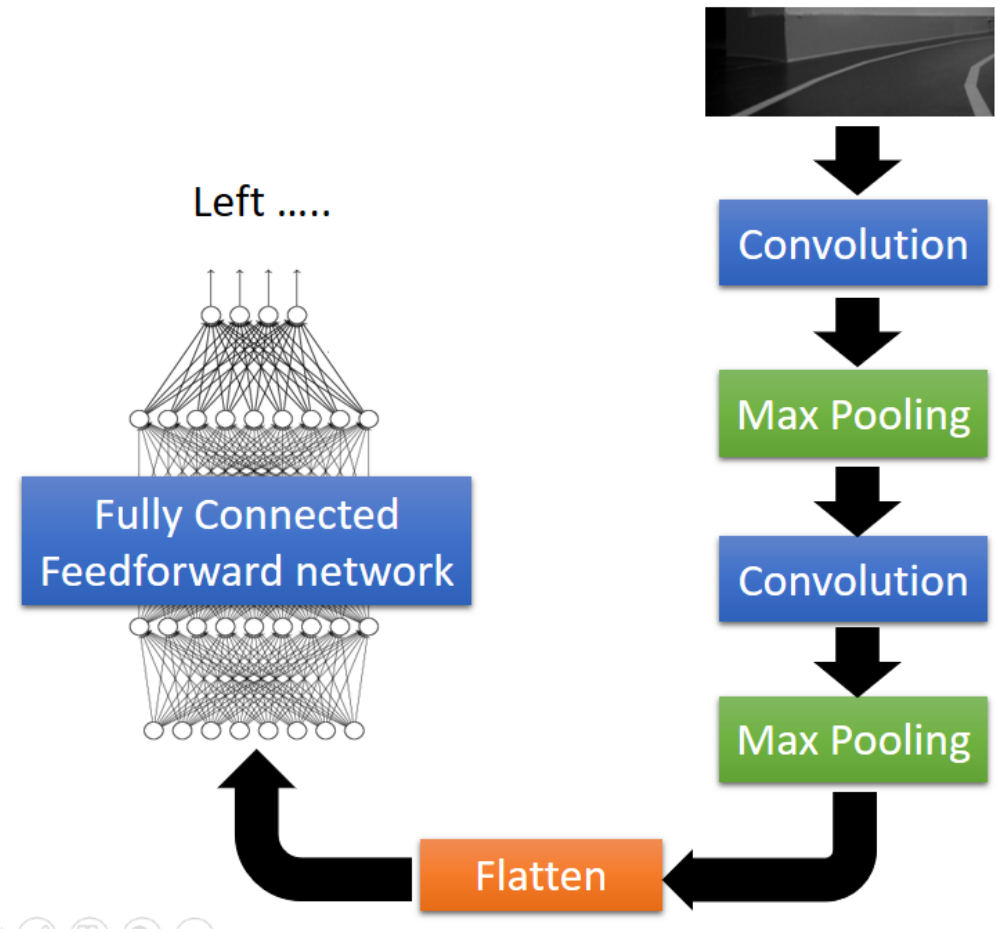

图 3.2.2（2） 小车预测模型的 CNN 基本结构，其中[Conv]->[MaxPool]可以重复很多次

在图 3.2.2（2）中可以看到，不同于传统的神经网络结构，它往往涉及到高级运算，卷积运算。另外在处理图像数据时，他不需要在输入层进行将其展开，而是经过卷积运算过后再进行 Flatten（展开）的，因为前面的卷积操作有效的减少了参数个数，进而避免在全连接层过拟合。

卷积神经网络通常是由三种层构成：卷积层，汇聚层（除非特别说明，一般就是最大值汇聚 MaxPooling layer）和全连接层（Fully-Connection layer 简称 FC）。ReLU 激活函数也应该算是是一层，因为它逐元素地进行激活函数操作，但不含参数。接下来将探讨在卷积神经网络中这些层通常是如何组合在一起的。

卷积神经网络最常见的形式就是将一些卷积层和 ReLU 层放在一起，其后紧跟汇聚层，然后重复如此直到图像在空间上被缩小到一个足够小的尺寸，在某个地方过渡成成全连接

层也较为常见。最后的全连接层得到输出，比如分类评分等。最常见的卷积神经网络结构如下：

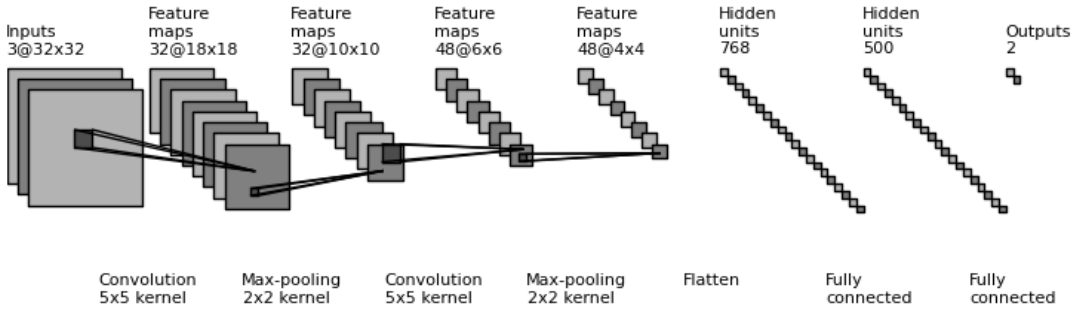

图 3.2.2（4） 小车预测模型的 CNN 基本结构，其中[Conv]->[MaxPool]可以重复很多次

卷积层主要是深度地来提取特征的，而池化层是保留重要特征的，而全连接层就是普通的神经网络层，而卷积神经网络设计的神奇之处是对图像高维数据非常有利，它以复杂地网络结构来减少参数的规模。并且网络学习的不单单是全连接层的权重，还有卷积核的权重。

3.3 目标检测

这个项目采用了基于形状(目标的形状)检测的方法，并且使用基于特征的级联分类器，主要用于目标检测。由于每个目标对象需要其自己的分类器，并在训练过程中和检测过程中是相同，所以这个项目只集中于交通标志和交通信号灯的检测。OpenCV 提供了训练器以及特征探测器。正样本图像主要使用了一些采集好的图像，也即是要检测的目标对象，并将图像进行适当的裁剪，使得正样本中只有目标图像可见。负样本图像即没有目标对象的图像。比如，交通红绿灯正样本包含相同数量的红色交通灯和绿色交通灯图像。同样的，负样本数据集使用了用交通标志 STOP 和黄色交通灯来进行训练。[10]

下面显示了在该项目中使用一些正样本和负样本图像。

图 3.3（1） 正样本和负样本数据的选择

类型	正样本的数目	负样本的数目	样本大小（像素）
STOP 标志	20	400	25*25
交通信号灯	26	400	25*45

图 3.3（2） 正样本和负样本数据规格

为了识别不同的状态的交通信号灯（Red，Green），所以在应用级联分类器后，找到大致的交通灯，接下来需要进行必要的图像处理才可以进行交通信号灯的识别。下面的流程图总结了交通灯识别的过程。

图 3.3（2） 交通灯识别流程

首先，利用训练完成的级联分类器用来检测交通红绿灯的状态。边框内的图像被认为是 ROI 。其次，将高斯模糊（高斯低通滤波器）应用在 ROI 中，用来减少图像噪音。接下来，在 ROI 在找到亮度最强的点。最后，通过基于 ROI 中最亮的点的位置就可以确定交通红绿灯的状态（红色或绿色）

第 4 章　具体实现

本章具体实现部分，主要阐述深度学习的主要步骤和相关的代码实现，主要有数据收集及处理，模型的选择和设计，训练和测试，分析结论及相关的改进与优化。

图 4 会展示传统上的机器学习的完整流程，在这里也会按照图 4 描述的来展开。

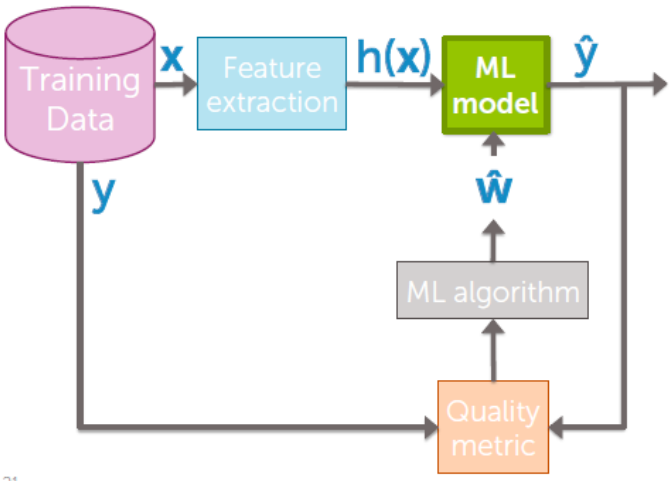

图 4 机器学习流程图

4.1 数据收集与处理

数据收集与处理是基于 C/S 架构下的采集，通过网络传输，将树莓派摄像机模块采集到的数据发送到 PC 机上处理，由于是监督式学习，所以图像的标记是在 PC 上利用 pygame 模块进行标记且对小车进行控制来实现的。

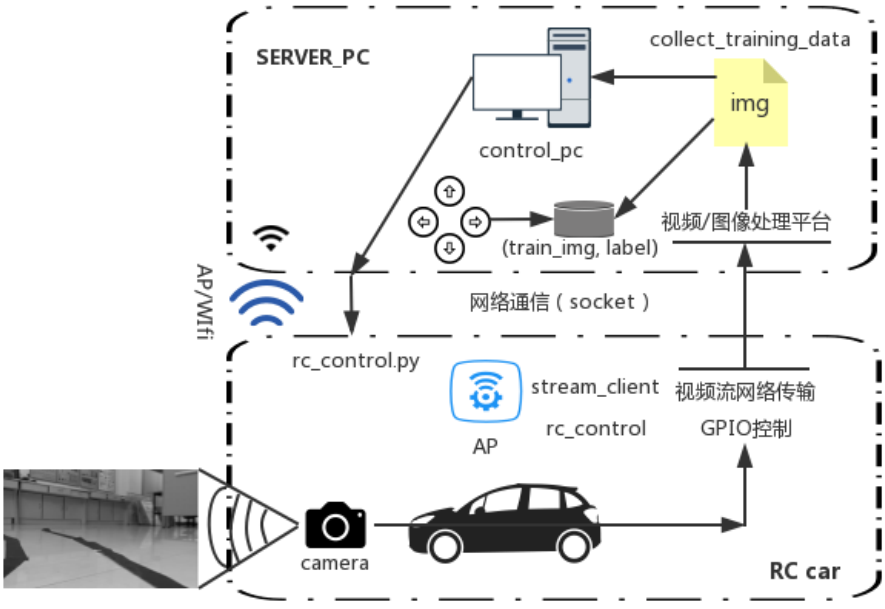

图 4.1 数据采集及处理框架图

4.1.1 视频流网络传输

主要采用了 PiCameramo 模块来做的视频流网络传输[11]，树莓派作为客户端来采集数据，PC 作为服务器端，接收来自客户端传输的视频流，并把在 PC 机上的指令作为 label，构成一个样本，从来经过每一趟的数据采集，就构成了一个子样本集。将其组合起来就是样本数据。

数据流传输客户端关键代码如下：

```
try:
    with PiCamera() as camera:
        camera.resolution = (320, 240)       # pi camera resolution
        camera.framerate = 10                 # 10 frames/sec
        sleep(2)                              # give 2 secs for camera to warm-up
        start = time()
        stream = BytesIO()

        # send jpeg format video stream
        for foo in camera.capture_continuous(stream, 'jpeg', use_video_port = True):
            connection.write(struct.pack('<L', stream.tell()))
            connection.flush()
            stream.seek(0)
            connection.write(stream.read())
            if time.time() - start > 6000:
                break
            stream.seek(0)
            stream.truncate()
    connection.write(struct.pack('<L', 0))
finally:
    connection.close()
    client_socket.close()
```

将摄像头捕捉到的视频流写入内存（BytesIO），然后通过 socket 传输经过特定格式编码后的视频流数据，服务器端接受时同样地通过相应的编码来读入相关的内存中，进而组织成相应可处理的数据结构。

4.1.2 样本集的生成

该部分的工作主要是在 PC 上完成的，为了更好地能够实时准确的采集到数据，在 PC 上主要创建了两个服务器，正如图 4.1.2 所示，server 1 用来与小车的驱动有关，控制小车移动，并且在 PC 机上生成 label，而 server 2 接受来自客户端的图像数据，并且生成 feature，那么（feature，label）就是一个样本数据，通过不断地驱动小车移动，经过一个回合地人工采集，就能形成一个子数据集，由于为了模型的准确率，在这里小车我们

采集了 3 组子数据集.

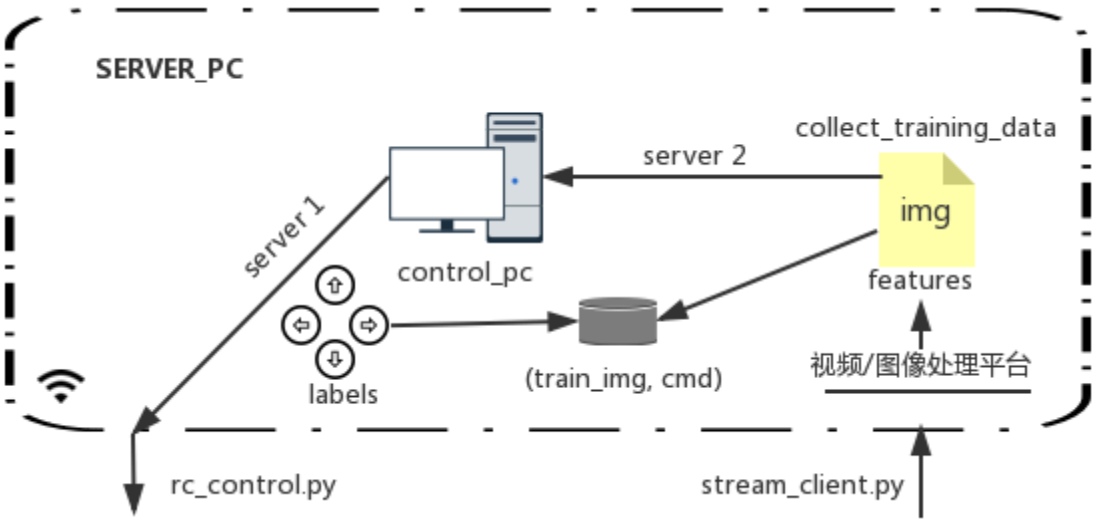

图 4.1.2 PC 机采集数据的流程图，其中（train_img, cmd）对应于（features,label）

关键代码如下：

```
try:
    stream_bytes = ''
    frame = 1
    while self.send_inst:
        stream_bytes += self.connection.read(1024)
        first = stream_bytes.find('\xff\xd8')
        last = stream_bytes.find('\xff\xd9')
        if first != -1 and last != -1:
            jpg = stream_bytes[first:last + 2]
            stream_bytes = stream_bytes[last + 2:]
            image = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8), 0)
```

接受来自客户端的图像数据，并将按照特定的格式进行解码。

```
            # select lower half of the image
            roi = image[120:240, :]
            # save streamed images
            cv2.imwrite('training_images/frame{:>05}.jpg'.format(frame), image)

            cv2.imshow('roi_image', roi)
            cv2.imshow('image', image)
```

在 3.2 节描述过 roi（image 的下半部分）是感兴趣区域，image 是接收到的原始数据，因此在这里只需将 roi 作为样本数据部分。因此下面的数据大小就是 120×320=38400,（1，38400）中的 1 是指这里的图像是单通道。

```
            # reshape the roi image into one row array
            temp_array = roi.reshape(1, 38400).astype(np.float32)
```

```
                frame += 1
                total_frame += 1

                # get input from human driver
                for event in pygame.event.get():
                    if event.type == KEYDOWN:
                        key_input = pygame.key.get_pressed()

                        if key_input[pygame.K_d]:
                            print("Forward Right")
                            self.conn2.sendall('turnrightO')
                            image_array = np.vstack((image_array, temp_array))
                            label_array = np.vstack((label_array, self.k[1]))
                            saved_frame += 1

                        elif key_input[pygame.K_a]:
                            print("Forward Left")
                            self.conn2.sendall('turnleftO')
                            image_array = np.vstack((image_array, temp_array))
                            label_array = np.vstack((label_array, self.k[0]))
                            saved_frame += 1

                        elif key_input[pygame.K_UP]:
                            print("Forward")
                            self.conn2.sendall('upO')
                            saved_frame += 1
                            image_array = np.vstack((image_array, temp_array))
                            label_array = np.vstack((label_array, self.k[2]))

                        elif key_input[pygame.K_x] or key_input[pygame.K_q]:
                            print 'exit'
                            self.conn2.sendall('clean')
                            self.send_inst = False
                            break

                    elif event.type == pygame.KEYUP:
                        print '0'
        # save training images and labels
        train = image_array[1:, :]
        train_labels = label_array[1:, :]

        # save training data as a numpy file
        np.savez('training_data/test017.npz', train=train, train_labels=train_labels)
```

```
        e2 = cv2.getTickCount()
        # calculate streaming duration
        time0 = (e2 - e1) / cv2.getTickFrequency()
        print 'Streaming duration:', time0

        print(train.shape)
        print(train_labels.shape)
        print 'Total frame:', total_frame
        print 'Saved frame:', saved_frame
        print 'Dropped frame', total_frame - saved_frame

    finally:
        self.connection.close()
        self.server_socket.close()
        self.gpio_socket.close()
        self.conn2.close()
```

在这里是根据传输过来的视频信息，来执行相应的驱动小车移动的指令，每个指令相应的对应着不同的标记，比如 left 的标记为 0，right 的标记为 1，up 的标记为 2 等。这些标记与上一帧图像组织成一个样本。并且将每个指令通过 socket 发送给树莓派端，树莓派上的 rc_control 会根据接收到的指令驱动小车向前移动.最后将采集到的样本数据保存成一个大的样本集

4.2 模型的设计、实现、训练和测试

4.2.1 多层感知机的实现

关键代码：

```
# load training data
image_array = np.zeros((1, 38400))
label_array = np.zeros((1, 4), 'float')
# label_array = np.zeros((1, 3), 'float')
training_data = glob.glob('training_data/*.npz')

for single_npz in training_data:
    with np.load(single_npz) as data:
        print data.files
        train_temp = data['train']
        train_labels_temp = data['train_labels']
        print train_temp.shape
        print train_labels_temp.shape
    image_array = np.vstack((image_array, train_temp))
    label_array = np.vstack((label_array, train_labels_temp))
```

```
train = image_array[1:, :]
train_labels = label_array[1:, :]
```

这部分是加载训练数据，将多个子样本集数据全部加载到内存中去。

```
# create MLP
layer_sizes = np.int32([38400, 32, 4])
# layer_sizes = np.int32([38400, 32, 3])
model = cv2.ANN_MLP()
model.create(layer_sizes)
criteria = (cv2.TERM_CRITERIA_COUNT | cv2.TERM_CRITERIA_EPS, 500, 0.0001)
criteria2 = (cv2.TERM_CRITERIA_COUNT, 100, 0.001)
params = dict(term_crit = criteria,
                        train_method = cv2.ANN_MLP_TRAIN_PARAMS_BACKPROP,
                        bp_dw_scale = 0.001,
                        bp_moment_scale = 0.0 )
num_iter = model.train(train, train_labels, None, params = params)

model.save('mlp_xml/mlp_v1.xml')
```

使用 OpenCV 中机器学习工具包中的 ANN_MLP 来搭建 MLP 模型，该模型的架构是 3 层，每一层的大小分别为 38400，32，4. 设置训练算法为反向传播算法，并且设置学习速率为 0.001，以及迭代次数为 500. 训练完成后将模型保存，下次直接加载该模型即可预测。

4.2.2 卷积神经网络的实现

关键代码如下：

```
# Load training data .npz to get label_array, unpacking what's in the saved .npz files.
training_data = glob.glob('training_data/*.npz')  # Finds filename matching specified path or pattern.
image_array = None
data_array = None
for single_npz in training_data:
    with np.load(single_npz) as data:
        train_image_temp = data['train']
        print('train data shape: ', train_image_temp.shape)
        train_labels_temp = data['train_labels']                '
        print ('Original labels shape:', train_labels_temp.shape)
    label_array = np.array([label for label in tqdm(train_labels_temp)], dtype=np.float64)
    image_array = np.array([data for data in tqdm(train_image_temp)], dtype=np.float64)
    image_array = np.reshape(image_array, (len(image_array), 120, 320, 1))
    print(label_array.shape)
    print(image_array.shape)
```

加载训练数据.npz 以获取 label_array，以及打开保存的.npz 文件中的内容。改变 image_array 数组的形状，从（1，38400）改变为（None，120，320，1），卷积神经网络不像 MLP 神经网络那样，要求数据必须是一维数据，卷积神经网络另一个最大的特点就是

最大的保留了图像的原始信息，通过 Conv->MaxPooling->Conv->MaxPooling 层来最大限度地保留重要信息（特征）,然后将池化的数据 Flatten，接下来接上两个全连接层（普通的神经网络层），最后最初预测（基于概率）；在深度学习中，最后一层的激活函数通常为 softmax() 函数。其中

$$\text{softmax}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \quad (4\text{-}1)$$

最后，卷积神经网络的架构为：

卷积层->池化层->卷积层->池化层->全连接层->Dropout 层->全连接层->输出层

```
model = Sequential()
print( 'Training...')

model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape=(120 ,320, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(32, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(64, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(30, init='uniform'))
model.add(Dropout(0.2))
model.add(Activation('relu'))

model.add(Dense(4, init='uniform'))
model.add(Activation('softmax'))
```

深度学习框架结构如图 4.2.2 所示：

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 120, 320, 32)	320	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 120, 320, 32)	0	convolution2d_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 60, 160, 32)	0	activation_1[0][0]
convolution2d_2 (Convolution2D)	(None, 58, 158, 32)	9248	maxpooling2d_1[0][0]
activation_2 (Activation)	(None, 58, 158, 32)	0	convolution2d_2[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 29, 79, 32)	0	activation_2[0][0]
convolution2d_3 (Convolution2D)	(None, 27, 77, 64)	18496	maxpooling2d_2[0][0]
activation_3 (Activation)	(None, 27, 77, 64)	0	convolution2d_3[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 13, 38, 64)	0	activation_3[0][0]
flatten_1 (Flatten)	(None, 31616)	0	maxpooling2d_3[0][0]
dense_1 (Dense)	(None, 30)	948510	flatten_1[0][0]
dropout_1 (Dropout)	(None, 30)	0	dense_1[0][0]
activation_4 (Activation)	(None, 30)	0	dropout_1[0][0]
dense_2 (Dense)	(None, 4)	124	activation_4[0][0]
activation_5 (Activation)	(None, 4)	0	dense_2[0][0]

Total params: 976,698
Trainable params: 976,698
Non-trainable params: 0

图 4.2.2 深度学习框架图和参数规模，卷积神经网络大大地减少了参数的个数

```
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])
```

模型使用随机梯度下降算法优化，其中 SGD 的学习率为 0.01，并且使用 categorical_crossentropy 来计算分类误差，

```
# Fit the model
model.fit(X_train, y_train,
          nb_epoch=30,
          batch_size=100)
```

利用搭建好的深度学习模型来拟合训练数据。

```
# Evalute trained model on TEST set
print ('Evaluation of model on test holdout set:')
score = model.evaluate(X_test, y_test, batch_size=1000)
loss = score[0]
```

```
accuracy = score[1]
print( '')
print ('Loss score: ', loss)
print ('Accuracy score: ', accuracy)
```

利用训练好的深度学习模型评估测试数据。

```
# Save model as h5
timestr = time.strftime('%Y%m%d_%H%M%S')
filename_timestr = 'nn_{}.h5'.format(timestr)
model.save('nn_h5/nn_{}.h5'.format(timestr))

# Save parameters to json file
json_string = model.to_json()
with open('./logs/nn_params_json/nn_{}.json'.format(timestr), 'w') as new_json:
    json.dump(json_string, new_json)

# Save training results to csv log
row_params = [str(training_data)[-33:-2], filename_timestr, loss, accuracy]
with open('./logs/log_conv_training.csv','a') as log:
    log_writer = csv.writer(log)
    log_writer.writerow(row_params)
```

将训练好的模型以文件的方式保存，文件主要保存了所有的参数。

4.3 结论

如果根据参数和训练时间来评估两个模型的好坏，那么卷积神经网络的设计相对来说，是非常地适合地。如果利用准确率评价模型的话，相比说，越简单的网络结构，准确率越高。

所以，在实际的上线测试中，简单的模型往往比复杂结构的模型更有利。

第 5 章　系统测试

5.1 模块测试

5.1.1 实时视频传输

树莓派（OS：Debian IP：172.14.1.1）：

```
python stream_client.py
```

PC 机 （OS：Windows10 IP:172.14.1.126）：

```
python stream_server.py
```

在 PC 机，测试结果如图 5.1.1

图 5.1.1 视频传输成功

5.1.2 实时识别目标

树莓派（OS：Debian IP：172.14.1.1）：

```
python stream_client.py
```

PC 机 （OS：Windows10 IP:172.14.1.126）：

```
python stream_detect.py
```

在 PC 机，共分为两组测试：

1. 测试交通路标，测试结果如图 5.1.2（1）
2. 测试信号（红绿两种状态）灯，测试结果如图 5.1.2（2）

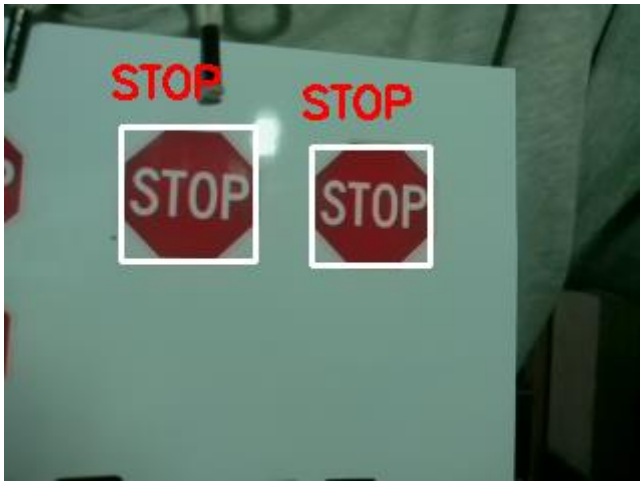

图 5.1.2（1） STOP 路标识别成功

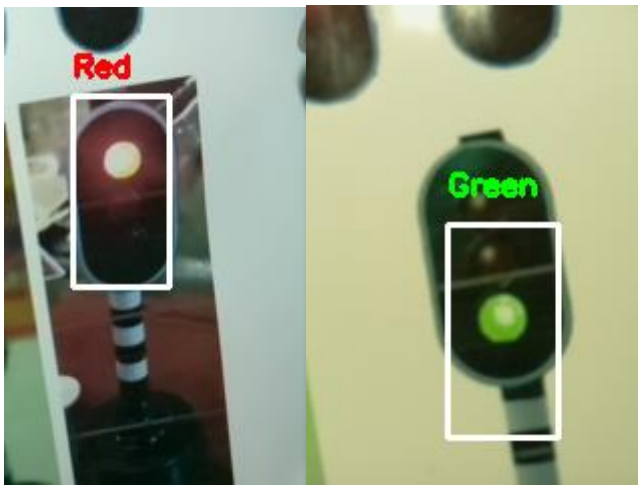

图 5.1.2（2） 交通信号灯识别成功

5.1.3 基于小车观测的图像来预测方向

PC 机单机测试，环境：Jupyter NoteBook。

主要有以下步骤：

1. 加载训练好的模型
2. “喂”给模型一张树莓派捕捉到的图像
3. 模型针对该图像做出预测（LEFT，RIGHT，UP）

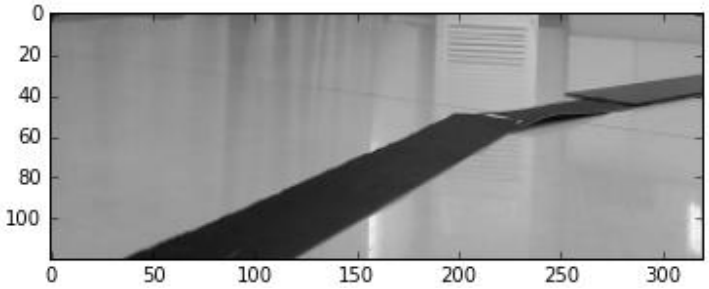

图 5.1.3（1） 将该图像喂给模型，按照常识可知，小车目前应该向右转

```
In [83]: ret, resp = model.predict(test)
         prediction = resp.argmax(-1)

         print 'Prediction:', prediction

         Prediction: [1]
```

图 5.1.3（1） 预测的结果是 LEFT，4.1.2 设置的[LEFT:0, RIGHT:1, UP:2]

5.2 整体测试

5.2.1 收集数据

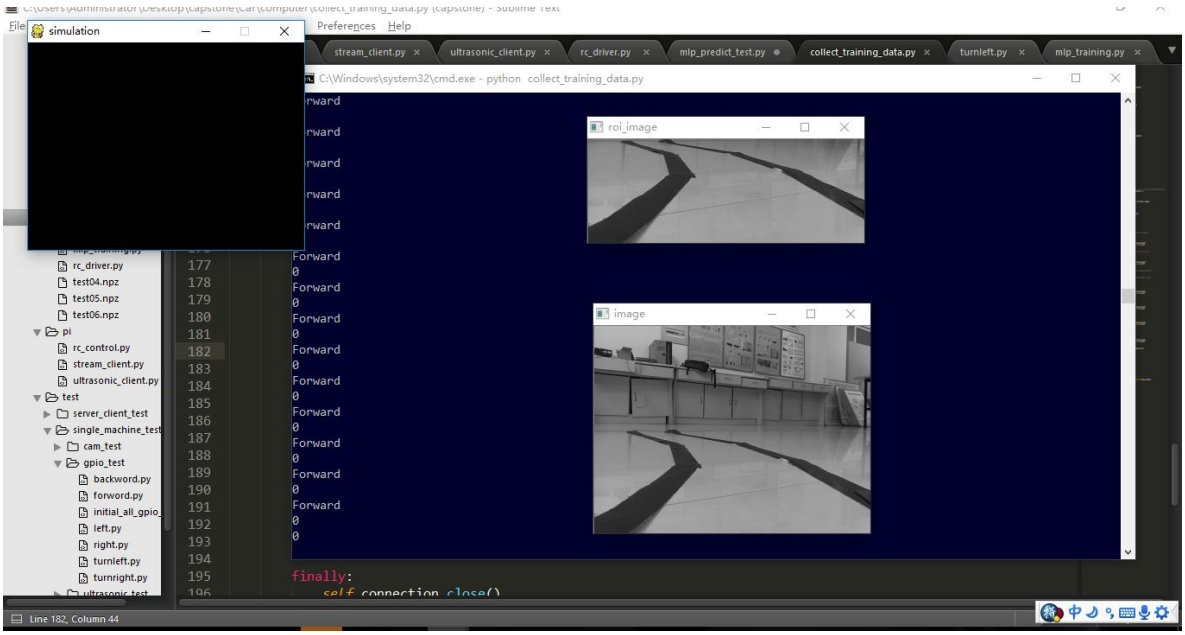

图 5.2.1 收集数据

5.2.1 训练模型

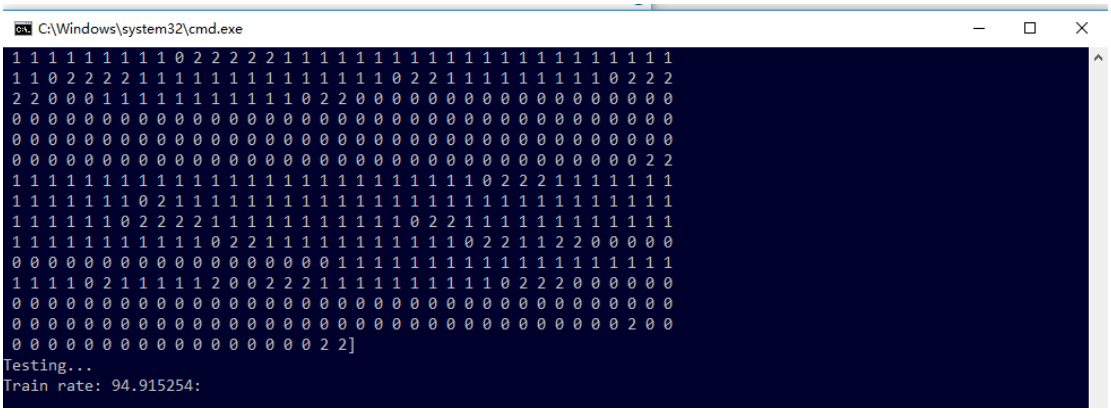

图 5.2.1 训练结果

5.2.3 最终测试

结　论

本项目主要利用深度学习来实现小车智能控制，并且实现了从传统的“人-车-路”闭环控制方式到目前的“车-路”控制方式的转变。本文深度地分析了两种深度学习模型-多层感知机模型和卷积神经网络模型，它们其中一个作为深度学习最典型的模型，另一个作为深度学习最重要的模型之一。样本的规模、参数的数量对模型的复杂度和准确率都有很大的影响。

由于卷积神经网络在增加网络结构复杂度的同时减少参数数量，从而达到优化网络的目的，并且卷积神经网络模型对图像，声音等非结构化的高维数据非常适合，因为在数字信号处理和数字图像处理中经常卷积运算提取特征，比如边缘信息，轮廓等等。相应的，卷积神经网络的基础组成就是由卷积层组成的，而深度学习学习的就是卷积核的各个参数。另外池化操作（Pooling）操作，往往是减少向下传递的参数地，逐步地递归，从而达到减少整个网络的参数的目的；又因为针对分类应用场景，往往需要将经过 Conv->MaxPooling->Conv->MaxPooling 之后的数据 Flatten 操作，然后再接上一个传统的神经网络，做出预测。

然而，在该项目中，由于数据维度和机器自身的短板（软件自身限制，内存限制）的原因，使得可以利用的数据量十分有限，因此卷积神经网络模型并不是最佳选择。另外通过实验证明，简单的网络结构不仅在准确度上优于复杂的网络结构，并且降低了模型的过拟合的风险。因此本项目中，实现了两种深度学习模型，将实验结果做出合理的分析且对两种模型做出合理的解析，最终选择简单结构的多层感知机作为上线测试模型，并且测试结果也达到了预期的要求。

另外，实验结果也证明，小车自身的硬件（左右转向不平衡）、轨道设计不标准等等对模型都造成了不必要的干扰，且树莓派自身的性能已经对于该项目已经绰绰有余，因此针对本项目接下来的工作将是针对小车做优化，降低在不必要的“噪音”。针对小车，我还想做智能城市自动化系统中的自动垃圾回收与处理智能小车系统。

致 谢

××××××××××××××××××××××××××××××××××××××
××
××××××××××

参考文献

[1] 杨明. 无人自动驾驶车辆研究综述与展望[J]. 哈尔滨工业大学学报. 2006, 38(8).

[2] Zheng W. OpenCV Python Neural Network Autonomous RC Car[Z]. 2013: 2015.

[3] Hadik A. MATLAB Neural Network Autonomous Car[Z]. 2013: 2016.

[4] Zotti R. How to Build Your Own Self Driving Toy Car[Z]. 2016: 2016.

[5] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library[M]. " O'Reilly Media, Inc.", 2008.

[6] 后锐，张毕西. 基于MLP神经网络的区域物流需求预测方法及其应用[J]. 系统工程理论与实践. 2005(12): 43-47.

[7] Bengio Y, Goodfellow I J, Courville A. Deep learning[J]. Nature. 2015, 521: 436-444.

[8] 李德毅，郑思仪. 轮式机器人的实践与展望[J]. 科技导报. 2015, 33(23): 52-54.

[9] Zeiler M D, Fergus R. Visualizing and understanding convolutional networks[C]. Springer, 2014.

[10] http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html[Z].: 2017.

[11] http://picamera.readthedocs.io/en/release-1.13/recipes1.html#capturing-to-a-network-stream[Z].

附录

所有代码及材料在：https://github.com/rh01/raspi-driving-car

部分记录文件：http://shenhengheng.xyz/blog