

# 目 录

摘 要 .....	I
Abstract.....	II
引 言 .....	1
第 1 章 绪论 .....	2
1.1 课题内容 .....	2
1.2 课题背景 .....	2
1.3 课题意义 .....	3
1.4 本章小结 .....	3
第 2 章 需求分析 .....	4
2.1 系统分析 .....	4
2.1.1 功能分析 .....	4
2.1.2 硬件需求 .....	5
2.1.3 软件需求 .....	5
2.2 系统架构 .....	5
2.3 本章小结 .....	6
第 3 章 整体设计 .....	7
3.1 目标 .....	7
3.2 系统的设计 .....	7
3.2.1 输入单元 .....	7
3.2.2 处理单元 .....	8
3.2.3 控制单元 .....	8
3.3 核心算法理论背景 .....	9
3.3.1 多层感知机模型（ANN_MLP） .....	9
3.3.2 卷积神经网络模型（CNN） .....	9
3.4 本章小结 .....	13

第 4 章 详细设计 .....	14
4.1 样本数据的收集 .....	14
4.1.1 图像数据的收集 .....	14
4.1.2 标记数据的收集及样本数据生成 .....	15
4.1.3 数据格式说明 .....	17
4.2 模型的训练与评估 .....	18
4.2.1 多层感知机模型（ANN_MLP）的设计 .....	19
4.2.2 卷积神经网络模型（CNN）的设计 .....	22
4.3 道路标识的检测与识别 .....	25
4.4 本章小结 .....	26
第 5 章 具体实现 .....	27
5.1 数据收集与处理 .....	27
5.1.1 视频流网络传输 .....	28
5.1.2 样本集的生成 .....	29
5.2 模型的设计、实现、训练和测试 .....	32
5.2.1 多层感知机的实现 .....	32
5.2.2 卷积神经网络的实现 .....	34
5.3 本章小结 .....	38
第 6 章 系统测试 .....	39
6.1 模块测试 .....	39
6.1.1 实时视频传输 .....	39
6.1.2 实时识别目标 .....	39
6.1.3 基于小车观测的图像来预测方向 .....	40
6.2 整体测试 .....	41
6.2.1 收集数据 .....	41
6.2.2 训练模型 .....	42
6.2.3 小车实时检测并驾驶 .....	42
6.3 本章小结 .....	42

结 论 .....	43
致 谢 .....	44
参考文献 .....	45

# 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。该技术可改变传统的交通工具的控制模式，从而加强了交通系统的效率。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。本论文主要从需求，设计再到实现等步骤进行阐述，关键对使用的两种不同的算法进行对比研究。该项目主要使用端到端模式实现自动驾驶，计算机主要负责核心计算任务，树莓派负责数据的采集和驱动小车。另外利用了深度学习算法对图像数据进行建模，利用算法发现众多图像之间的相似或相同的模式，利用模式感知环境进行学习。本项目核心创新点是利用 CNN，MLP 等深度学习算法完成了复杂的模式识别任务和利用数据驱动小车自动驾驶。

**关键词：**自动驾驶；机器学习；深度学习；MLP 模型；CNN 模型；监督学习

# Research and Realization of Automatic Driving Car Based on Deep Learning

**Abstract:** The Autonomous Car is one of the most important applications of outdoor mobile robots in the field of transportation. The technology can improve the efficiency of transportation system by changing the traditional control mode of transportation. The Autonomous RC Car is based on the self-driving car model developed by a small automatic model. This paper mainly describes from system requirements, system design to system implementation, and critical comparative study using two different algorithms. The project will use end-to-end model for automatic driving, The computer is mainly responsible for the core computing tasks, raspberry PI is responsible for data collection and driving car. In addition, the deep learning algorithm is used to model the image data, and using algorithms to find similar or identical patterns between images, using pattern-aware environments for learning. The core innovation of this project is to use CNN, MLP and other deep learning algorithms to complete the complex pattern recognition tasks and the use of data-driven car Auto-Driving.

**Key words:** Autonomous Car ; Machine Learning ; Deep Learning ; MLP Model ; CNN Model ; Supervised Learning

## 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性，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。

本文中描述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动驾驶，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。需要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、MATLAB Neural Network Autonomous Car、How to Build Your Own Self Driving Toy Car 等。

## 第 1 章 绪论

目前当今信息时代的主体是移动互联网时代，而在这个时代中由于数据的不断膨胀，由此产生大数据时代，在信息时代的年代，人工智能、机器学习等一些高端技术蓬勃发展。而自动驾驶技术本身就是对传统交通运输行业的颠覆，从根本上改变了汽车的驾驶方式。所使用的技术不尽相同。有的基于传感器和激光雷达感知的，有的基于视觉辅助驾驶的。本课题研究的就是基于视觉感知来控制小车自动驾驶。

### 1.1 课题内容

课题主要研究和实现了自动驾驶车基于视觉和深度学习决策的算法。由于自动驾驶车从传统上改变了交通工具的控制模式，所以有力地提高了交通工具的效率。另外课题使用的研究方法和研究过程是传统机器学习建模的方法。主要剖析了基于计算机视觉和深度学习模型的自动驾驶车的控制模式和具体的驾驶模型。

### 1.2 课题背景

自动驾驶汽车，是现代的智能联网机器人。自动驾驶开启了交通运输行业的新时代。以百度、谷歌为代表的互联网企业，从人工智能的角度切入无人驾驶产业，将无人驾驶汽车看作一个智能的机器人系统，基于无人驾驶技术的汽车，实质上就是一台移动的智能联网机器人，可以实现真正的智能化和共享化。另外只有自动驾驶汽车运用了人工智能等技术才能真正意义上自动驾驶。

无人驾驶技术可抽象为“环境探测-自动决策-控制响应”，其发展主要依赖于三方面技术的成熟：智能感知技术是前提，智能决策和控制技术是核心，高精度地图及智能交通设施等是重要支撑。智能识别及决策技术就想智能汽车的中枢神经，是自动驾驶技术成熟的核心及瓶颈。深度学习让每一个新上路的新“驾驶脑”都像“老司机”那样，拥有丰富的驾驶经验。

深度学习是近些年在大数据的背景下产生的一门学科。自深度学习出现以来，它已成为很多领域，尤其是在计算机视觉和语音识别中，成为各种领先系统的一部分。在通用的用于检验的数据集，例如语音识别中的 TIMIT 和图像识别中的 ImageNet, Cifar10 上的实验证明，深度学习能够提高识别的精度。与此同时，神经网络也受到了其他更加简单归类模型的挑战，支持向量机等模型在 20 世纪 90 年代到 21 世纪初成为过流行的机器学习算法。

基于此，本次课题主要利用深度学习技术来对视觉感知到的信息进行加工和提取主要信息，并且对环境进行预测小车的行为。

### 1.3 课题意义

本课题主要创新点在于基于现有的数据模型来对小车进行建模，实现了数据驱动驾驶小车。在目前大数据和人工智能勃勃生机的时代，这使得在传统编程和传统的驾驶模型上进行改造，合理地对数据建模，并在数据中发现“模式”，另外本项目是一个跨学科交叉项目，对后期的研究和改进具有创新意义。

### 1.4 本章小结

本章主要讲述了课题研究的主题内容，并对相应用到的技术进行背景阐述。并从自身出发，说明了课题对学习目标和未来的研究有什么样的价值和意义。



## 第 2 章 需求分析

目前深度学习在图像理解、语音识别、自动驾驶等等领域发挥了很强大的作用。尤其在自动驾驶领域，近些年来，由于三维激光雷达、毫米波雷达、图像识别等新型探测器的成熟使得汽车工业由传统制造业主导转向人工智能主导。智能控制技术就好比智能汽车的运动中枢神经，是自动驾驶技术发展的核心。那么智能控制技术的核心就是深度学习算法。

### 2.1 系统分析

#### 2.1.1 功能分析

由于目前智能小车的定位偏向于自动驾驶，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，交通标志牌检测以及上述检测过后的行为决策等功能。

以下是主要的流程与关键步骤设计：

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（Version: 2.4.8）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，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。

### 2.1.2 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）
- (5) 超声波模块
- (6) 电源和若干杜邦线

### 2.1.3 软件需求

通过上述分析，软件准备分为两部分准备：

PC 机上主要应该具备的软件及模组：Python2.7、Keras、OpenCV 等，树莓派上与 PC 具备相同的环境，额外需要 PiCamera 模块来捕捉和传输视频流。

## 2.2 系统架构

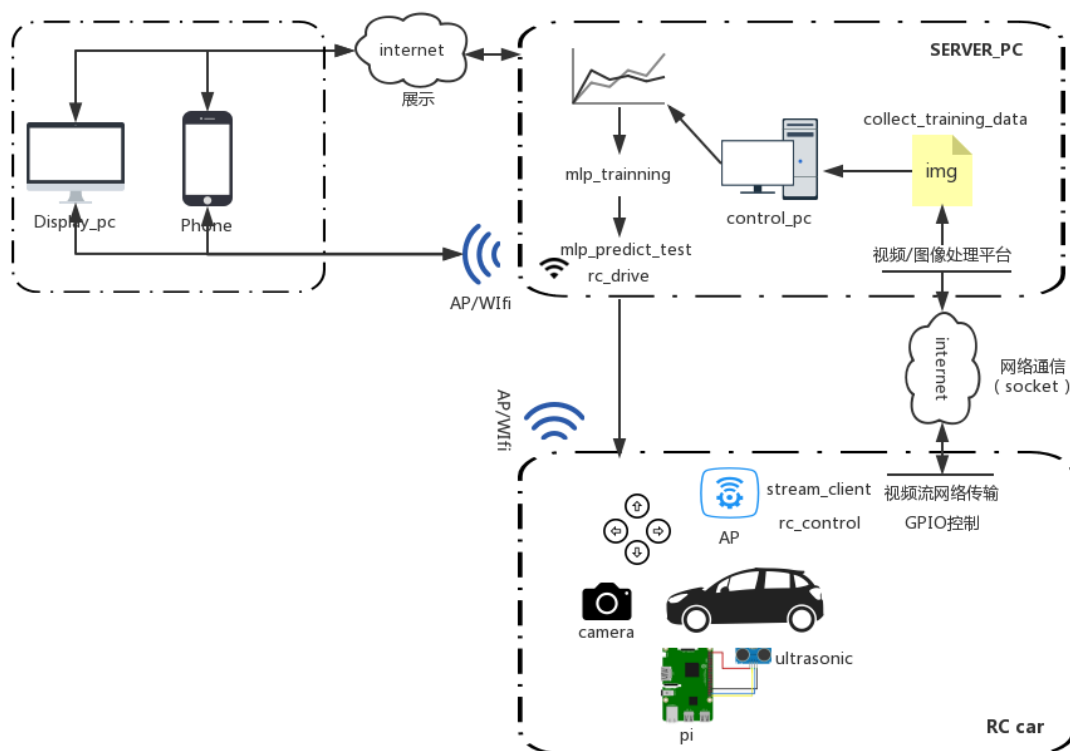


图 2-1 整体架构图

如图 2-1 所示，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。

## 2.3 本章小结

本章主要介绍了系统分析、在系统分析中又具体地描述了应该具备的功能和相关软硬件的需求。最后针对以上的描述给出工程架构图。

## 第 3 章 整体设计

整体设计是从用户的角度来看待软件系统的，主要从产品设计的眼光来设计的。主要将系统划分几个模块，逐个模块简略的介绍和相关算法的理论基础。

### 3.1 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。

### 3.2 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。

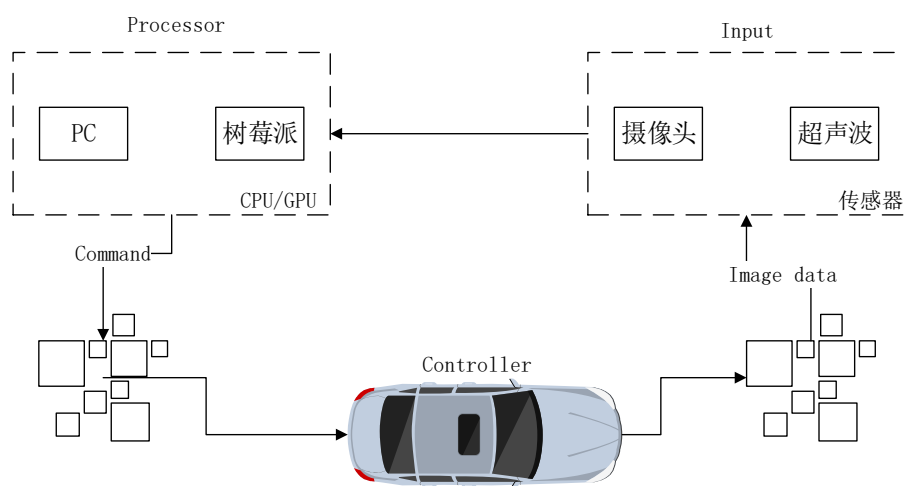


图 3-1 系统组成

如图 3-1 所示，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。

#### 3.2.1 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  $320 \times 240$  大小。

3.2.2 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）

3.2.3 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。树莓派 3 B+的 GPIO 引脚具体功能如图 3-2 所示。图 3-2 来源于文献[6]。




















Raspberry Pi 3 GPIO Header				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

图 3-2 树莓派 3 每个 GPIO 引脚的功能

本项目中引脚的使用情况，见表 3-1。

表 3-1 引脚使用情况

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

### 3.3 核心算法理论背景

该项目主要采用了两个深度学习模型进行实验，分别是多层感知机模型和卷积神经网络模型，并且对比在实验中的结果进行综合评估，并选择合适的模型放在小车上进行预测实时图像数据。下面针对两种深度学习的模型及其核心理论知识展开描述。

#### 3.3.1 多层感知机模型（ANN\_MLP）

深度前馈网络，也叫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。该函数由一些简单的函数复合而成，它可能是多项式函数也可能是超越函数。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，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。对于小车在 MLP 模型的基本思想体现如图 3-3 所示。

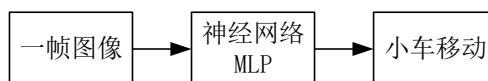


图 3-3 预测模型的基本思想

#### 3.3.2 卷积神经网络模型（CNN）

卷积网络，也叫做卷积神经网络，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。例如图像数据（二维网格数据）。卷积网络在诸多应用领域都表现优异，比如大

规模图像识别，人脸识别等等。“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。

深度学习以卷积神经网络为代表，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。

深度学习让计算机通过较简单概念构建复杂的概念。图 3-4 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，它们转而由边线定义）来表示图像。

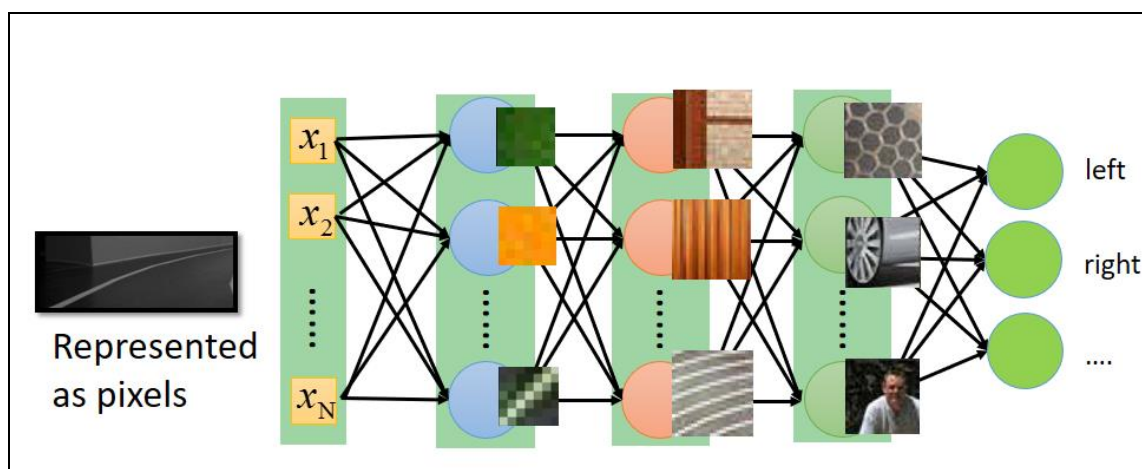


图 3-4 深度学习模型的示意图

在图 3-4 中  $x_1, x_2, \dots, x_N$  表示图像里面的像素点元素，有三层隐含层，输出层分别表示小车要预测的方向。

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。

传统的神经网络对于大尺寸图像处理效果并不是很理想。在 CIFAR-10 中，图像的尺寸是  $32 \times 32 \times 3$ （宽高均为 32 像素，3 个颜色通道），因此，对应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  $32 \times 32 \times 3 = 3072$  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  $320 \times 120 \times 3$  的小车捕捉到的图像，会让神经元包含  $320 \times 120 \times 3 = 38,400$  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中的隐含层一样），也常用到的技巧是增加 Dropout 层来减少传递到下一层神经元的个数以达到避免过拟合的情况。

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 3-5 所描述就是在项目中所使用的卷积神经网络架构。图 3-5 来源于台湾大学李宏毅老师的 Machine Learning 课件，具体见文献[12]。

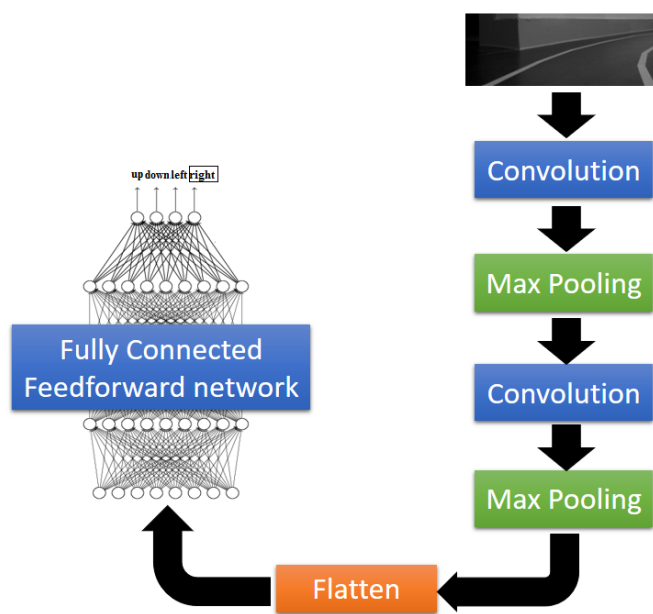


图 3-5 小车预测模型的 CNN 基本结构



在图 3-5 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层 MaxPooling layer）和全连接层（Fully-Connection layer 简称 FC）。ReLU 激活函数也应该算是一层，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图 3-6 所示。

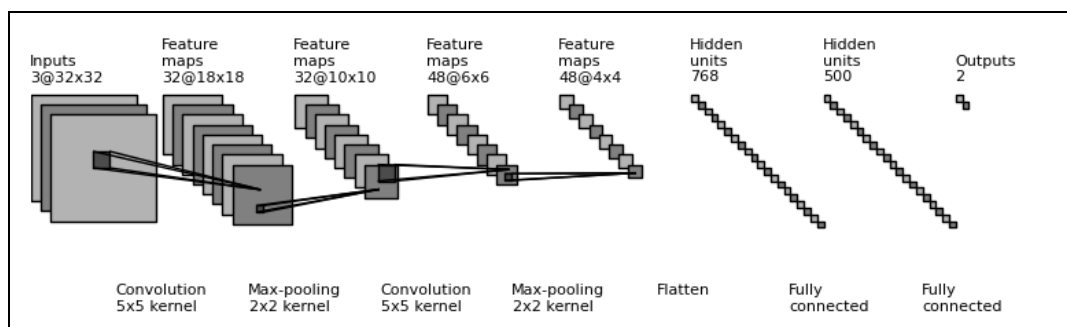


图 3-6 常见的卷积神经网络结构

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。

小结：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。

### 3.4 本章小结

本章主要介绍了目标、系统的设计和核心算法的理论背景，其中系统的设计宏观的描述了几个简单的功能组件，核心的流程及建模的细节将在第 4 章展开。

## 第 4 章 详细设计

详细设计是软件系统的核心，它主要从算法，模型，和数据收集方面进行完整的设计。

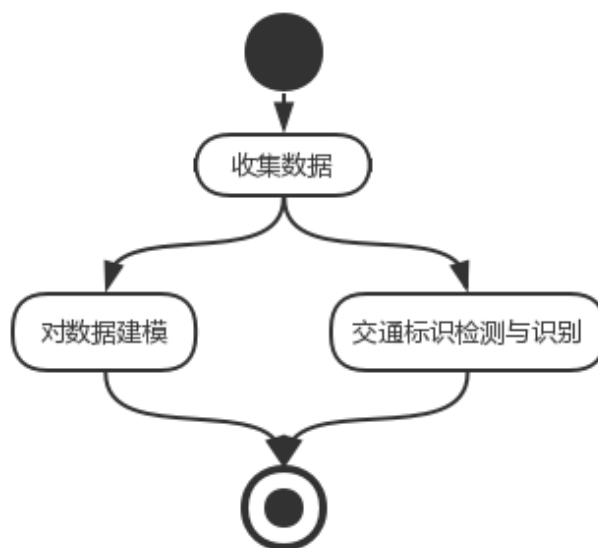


图 4-1 详细设计流程图

本章将按照图 4-1 所示的流程进行展开，首先凡事开头难，将具体描述收集数据部分的具体细节，另外数据有了之后，需要结合需求和数据的格式进行对数据建立模型，在本章采用两种算法对数据建模，对建模方法进行具体描述。作为智能车重要的交通标识检测与识别的模块，将会简要地进行设计与实现。

对数据进行建模和交通标识的检测与识别这两个部分为小车进行自动驾驶和模式识别的重要的算法支撑。

### 4.1 样本数据的收集

#### 4.1.1 图像数据的收集

图像数据的收集的流程按照训练数据样本的要求，在这里主要分为两部分：图像数据 `img_data` 的收集，指令数据 `label_data` 的收集并将图像数据和指令数据结合并生成样本数据。图 4-2 是图像数据的收集部分，箭头表示为数据走向。

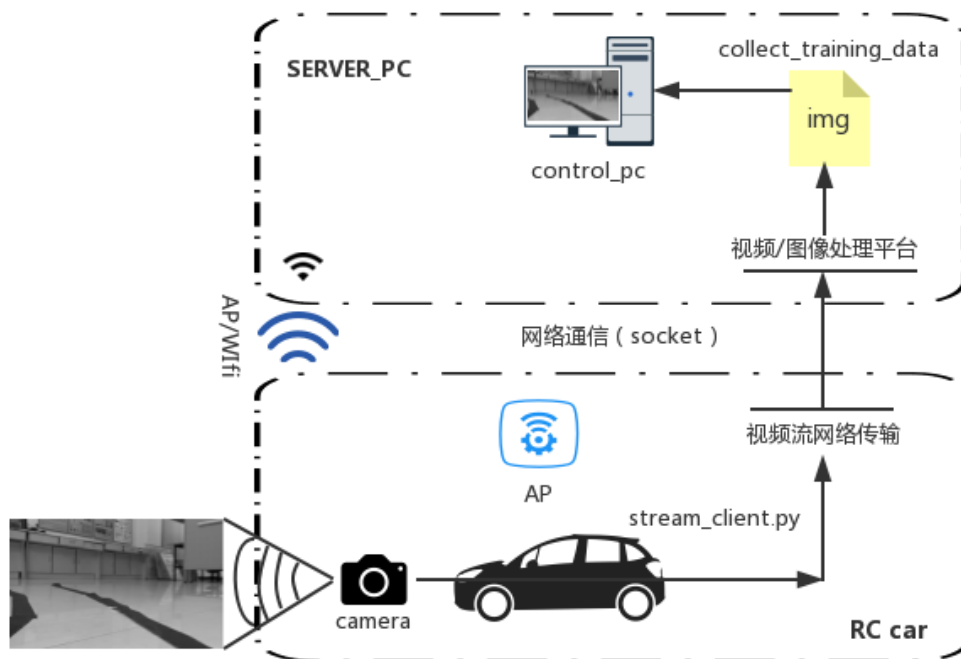


图 4-2 图像数据的收集

图像数据的收集过程:

- (1) 服务器端为 PC (因为 PC 性能较好且稳定), 客户端为树莓派;
- (2) 树莓派控制摄像头拍摄视频流数据, 并将每一帧数据通过网络传送给 PC, 在树莓派中控制摄像头拍摄及传输的脚本为 `stream_client.py`;
- (3) PC 机接收数据, 并将图像数据处理为灰度图像, 并将每一帧图像写入到本地硬盘上。另外将传送过来的图像数据展示在 PC 桌面上。

#### 4.1.2 标记数据的收集及样本数据生成

由于是监督式学习, 因此需要一位专家来教小车学习驾驶, 当图像数据展示在桌面上后, 此时需要根据反馈过来的图像进行主观上的决策, 即利用人类的经验进行监督学习, 比如图 4-2 所示的一样, PC 上反馈过来的图像数据, 按照人类的经验, 应该是向前走, 即 `forward` 指令。为了更加方便的与小车底层控制交互, 在这里主要使用了 `pygame` 模块里面的键盘事件控制, 为了后期训练的方便, 将指令编码为 `[left, right, forward, backward] -> [0, 1, 2, 3]`。

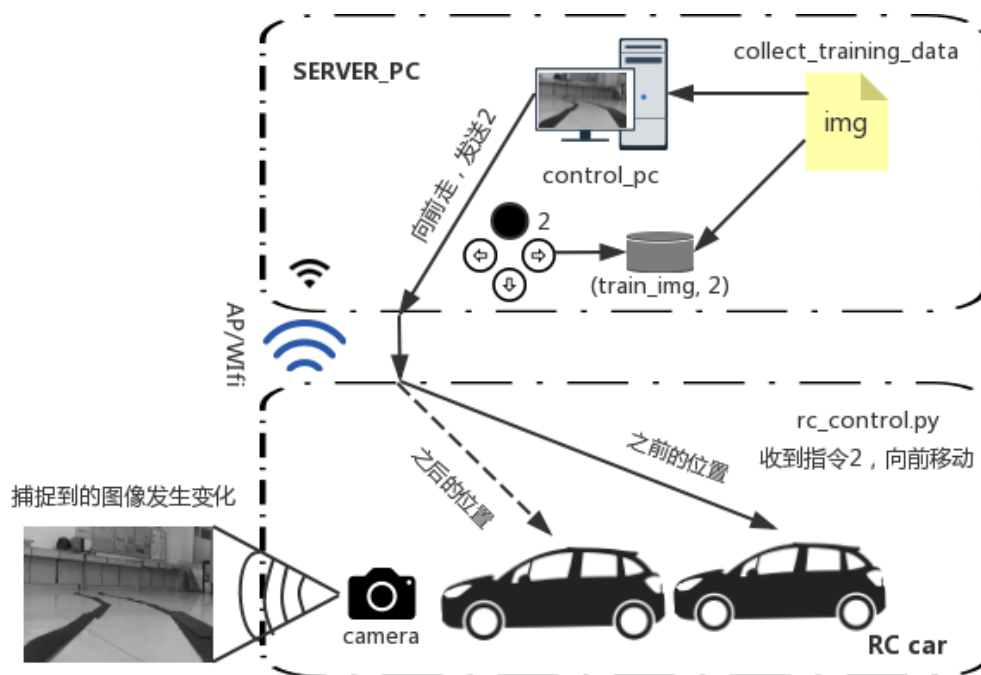


图 4-3 标记数据的收集及样本数据的生成

标记数据的收集和样本数据的生成过程：

(1) 当人看到桌面显示的图像后，会按照主观的判断，做出决策，在图 4-3 中人按照自己的判断，按下上键，上键对应数字 2，并将“2”通过另外一个端口发送给树莓派；

(2) 发送之后，PC 将会把图像数据和数字 2 合并成如下格式的数据： $(\text{train\_data}, \text{train\_label})$ ，即 PC 机上显示的图像为训练数据，而人类通过该图像做出的决策相对应的数字是训练标签；

(3) 树莓派的脚本 `rc_control.py` 接收到 PC 发送的指令 2 后，会执行控制硬件驱动小车向前行驶；

(4) 那么行驶后，下一帧的图像数据将发生改变。

以此类推，收集一个回合之后，将所有的样本数据生成一个大的样本数据集，并将数据集按照 `numpy` 支持的数据格式写入到硬盘中去。

总结：图 4-4 显示了训练数据的收集的过程。首先将二维图像数据进展成一维数据，接下来将指令数据和上面的图像数据进行合并成样本数据，然后形成样本集数据，然后将数据集打包成 `numpy` 多维数组格式的数据存储在本地。图 4-4 引用自文献[22]。

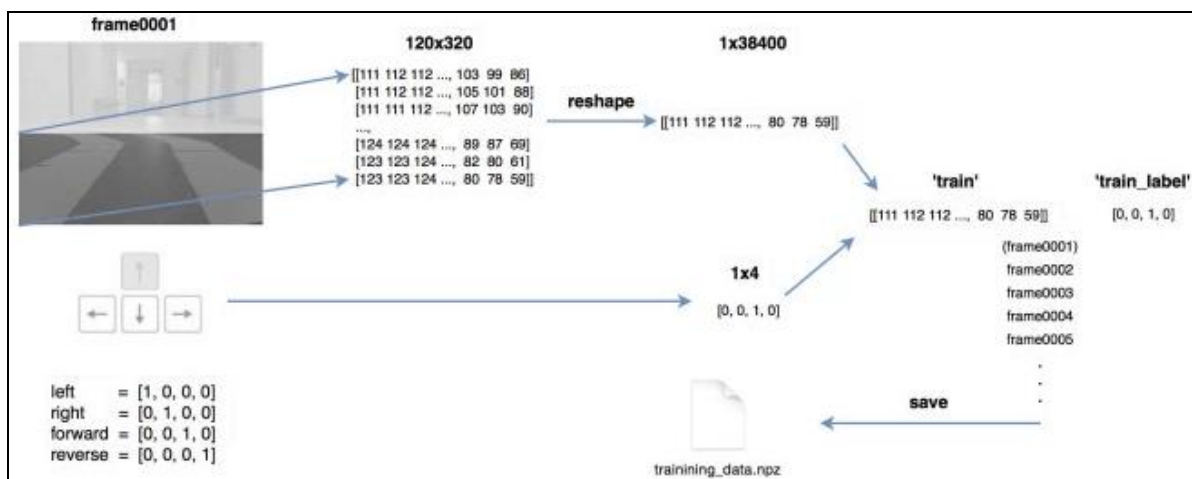


图 4-4 图像处理步骤

### 4.1.3 数据格式说明

(1) 利用 One Hot encoding 方法对样本标记进行处理，比如项目中的 0, 1, 2, 3 分别利用 One Hot Encoding 表示表 4-1 所示。

表 4-1 One-Hot 编码

Label	One-Hot Encoding
0	[1, 0, 0, 0]
1	[0, 1, 0, 0]
2	[0, 0, 1, 0]
3	[0, 0, 0, 1]

(2) 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。因为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 4-5 描述的为树莓派摄像头捕捉到的原始图像，由于在图像中和现实生活给出的经验得，图 4-5 的上半部分是冗余的图像信息，对图像处理和模型的训练没有用处，因此需要裁剪掉上半部分图像，得到训练所使用到的训练图像，如图 4-6 所示。

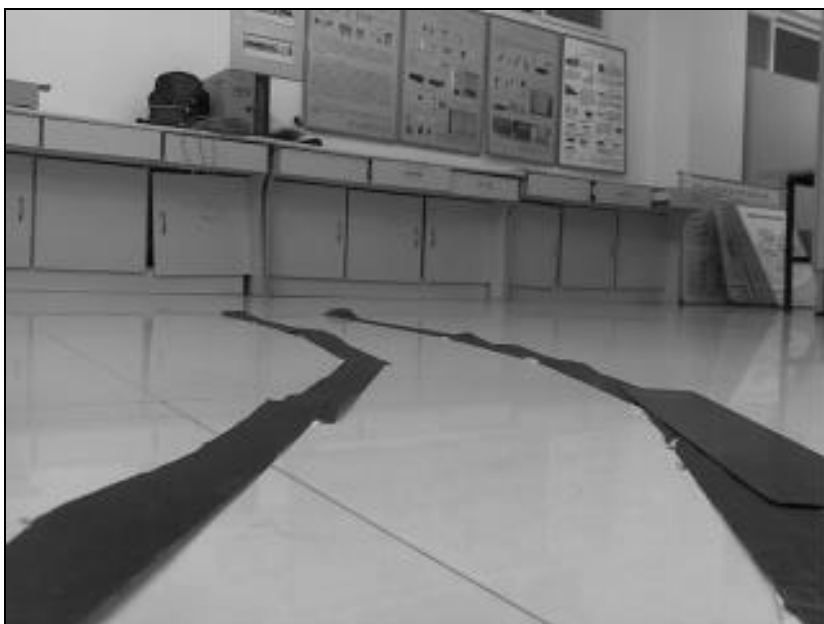


图 4-5 摄像头模块捕捉到的原始图像（320px\*240px）



图 4-6 裁剪后的图像，ROI 区域

由于在不同的神经网络模型下，数据格式要求是不同的，但为了后期处理的方便，在这里将二维的图像数据（单通道）展开成一维数据，即深度学习中的扁平化（**Flatten**）的概念。

所以一条样本数据格式为（38400，[\_, \_, \_, \_]）。

## 4.2 模型的训练与评估

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。

在本节将按照图 4-7 流程图一样来描述模型的训练，评估及决策。

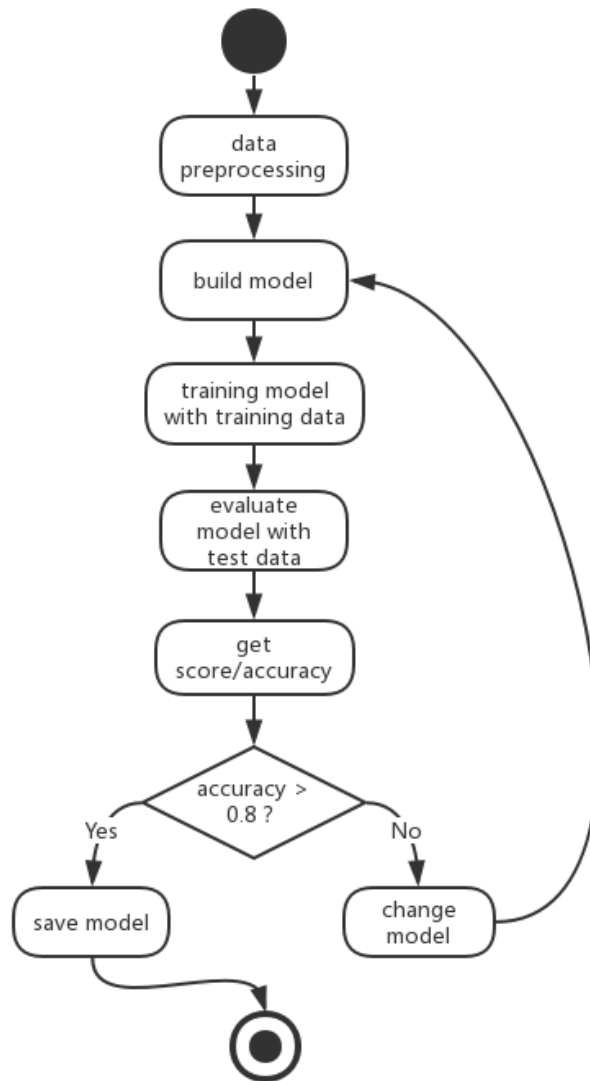


图 4-7 建模的流程图

#### 4.2.1 多层感知机模型（ANN\_MLP）的设计

##### （1）样本数据的预处理

将 4.1 节收集到的数据进行加工，将训练数据即图像数据存放在 `train` 中，将训练数据的标签存放在 `train_label` 中，`train` 的数据格式为 `(None, 38400)`，`train_label` 数据格式为 `(None, 4)`，并且将数据集写入到内存中，以便训练模型使用。

##### （2）建立 MLP 模型

设计要点：

在 MLP 建模时，模型的结构如图 4-8 所示：



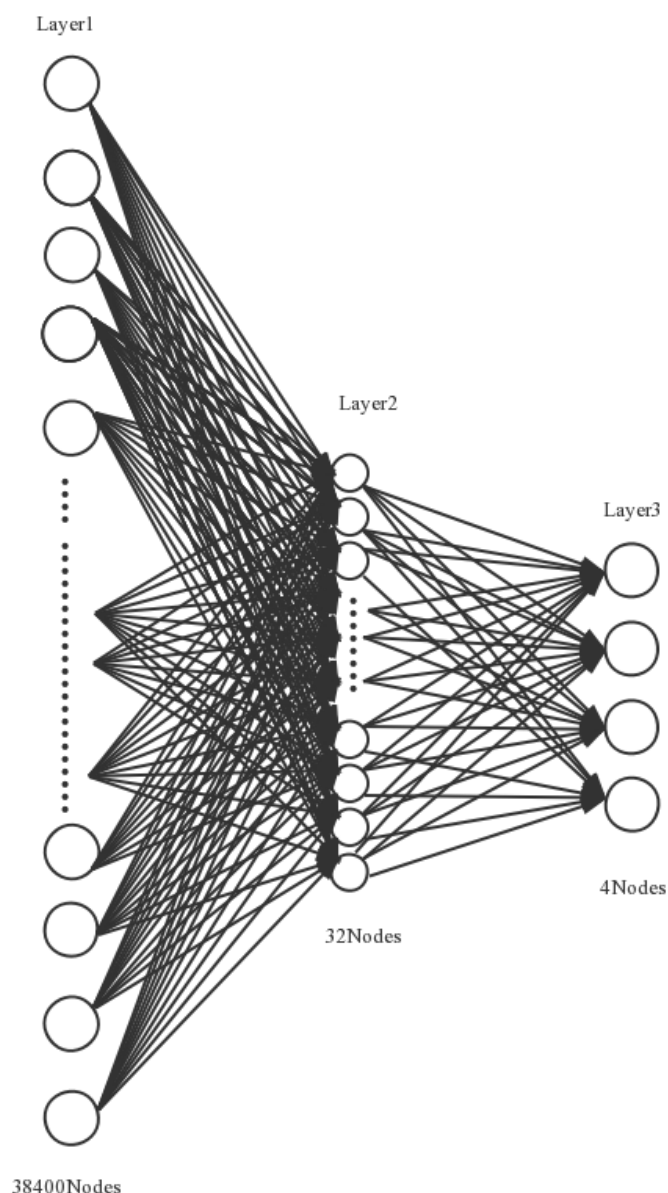


图 4-8 MLP 模型结构图

如图 4-8 所示在输入层有 38,400( $320 * 120$ ) 个节点，隐含层的数目和每个隐含层神经元的个数是没有严格限制的，在这里指定隐含层有 32 个神经元节点。在输出层有 4 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、右转和后退指令。

本项目中的 MLP 模型主要使用了 OpenCV 的机器学习工具箱中的 `cv2.ml.ANN_MLP_create()` 方法进行创建的。下面会描述有关该方法的具体使用说明。

```
model = cv2.ml.ANN_MLP_create()
```

`cv2.ml.ANN_MLP_create()` 是继承了机器学习工具箱机器学习类的一个子类，调用该方法将会实例化一个 MLP 模型类。通过调用 `cv2.ml.ANN_MLP_create()` 将会创建一个 MLP 模型。

创建之后，需要设置它的拓扑结构。

```
model.setLayerSizes(numpy.array([38400, 32, 4],dtype=numpy.uint8))  
  
model.setTrainMethod(cv2.ml.ANN_MLP_BACKPROP)  
  
model.setActivationFunction(cv2.ml.ANN_MLP_SIGMOID_SYM)
```

`setLayerSizes` 方法接受 Layer Size 作为参数，其中 Layer Size 需要转化为 numpy 的数组格式。其中第一个元素为输入层的大小，最后一个元素是输出层的大小，中间的所有元素是每一个隐含层的大小，这里的大小指的是神经元的个数。

`setTrainMethod` 方法设置模型训练的方法，训练方法设置为弹性 BP 算法（该算法是 BP 算法的一个升级版本）来更新参数。一般情况下，这里有两种选择：**BACKPROP** 和 **PROP**。这两个选择都是反向传播算法，该算法在分类时将对神经网络的权重进行调整。

`setActivationFunction` 方法设置每一个神经元的激活函数为 Sigmoid 函数；

```
model.setTermCriteria (cv2.TERM_CRITERIA_COUNT, 100, 0.001)
```

`setTermCriteria` 方法设置训练的终止条件。

设置完成之后，有了前期的数据之后就可以进行训练。

```
model.train(np.array(train_data, dtype=np.float32), cv2.ml.ROW_SAMPLE,np.array(train_labels, dtype=  
np.float32))
```

`train` 方法将对训练数据和训练数据标签进行有监督的学习。

### （3）模型的评估

按照样本数据收集的方法收集相应的测试数据，按照本节的方法进行预处理成测试数据。

```
model.predict(test_data)  
  
prediction = resp.argmax(-1)  
  
  
true_labels = train_labels.argmax(-1)  
  
  
train_rate = np.mean(prediction == true_labels)
```

`predict` 方法结合上述的代码进行对模型进行打分。利用准确度来评测模型好坏。

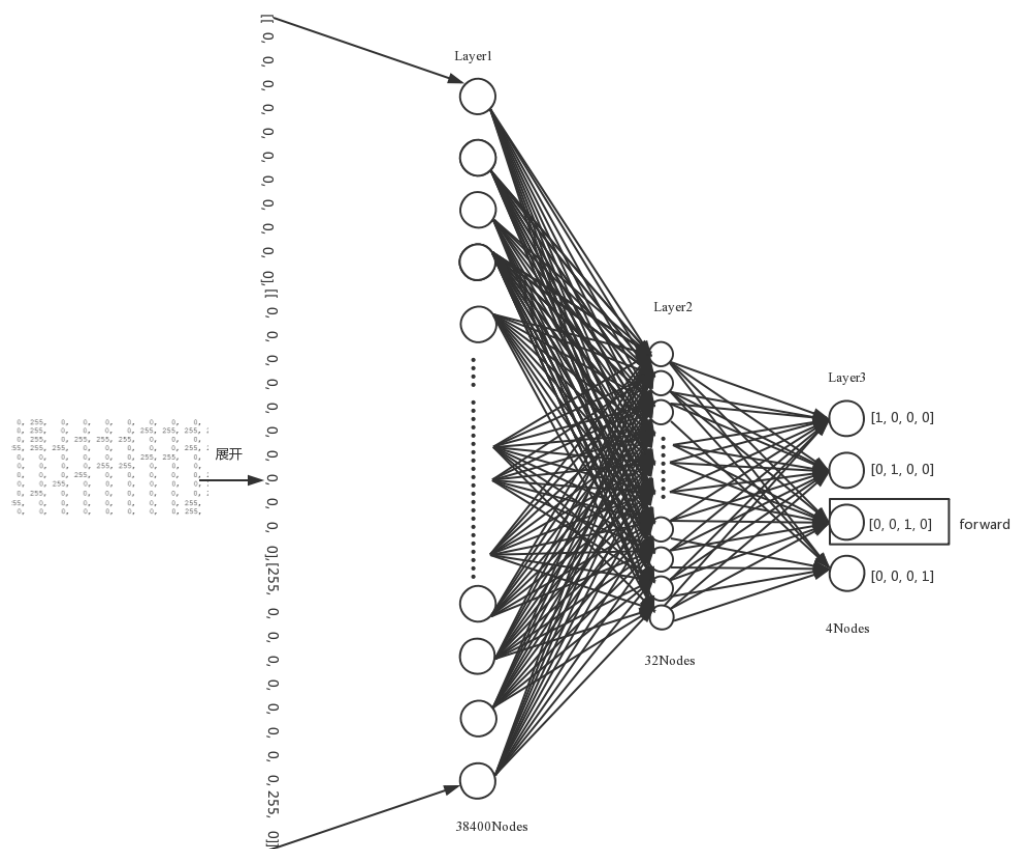


图 4-9 实际工程 MLP 流程图

如图 4-9 所示，将图像数据展开后，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。

#### (4) 模型的保存

```
model.save('mlp.xml')
```

当给模型打分超过 80%后，就可以对模型进行保存，保存到 mlp.xml 文件中。

下次若要对新的图像数据进行预测，直接加载模型文件即可，模型文件主要存储了模型的权重信息，它与训练数据生成的神经网络具有相同的网络结构，因此如果下次要进行预测，直接加载模型文件即可。

### 4.2.2 卷积神经网络模型（CNN）的设计

#### (1) 数据的预处理过程

预处理过程与 MLP 模型处理类似。

#### (2) 模型的建立

建立 CNN 模型时，在这里利用了 Keras 强大的深度神经网络框架进行实验的，数据格式与 MLP 稍有不同的是，由于 CNN 支持二维数据的输入，因此在 MLP 模型的基础上，

只需要将数据重新设置二维即可。

由于 Keras 相比较其他优秀的深度学习框架而言，适合初学者并且使用了 Tensorflow 和 Theano 作为引擎，因此大大的优化了训练的速度和建模的难度。因此选择了 Keras 来创建 CNN 模型。

Keara 作为入门级的深度学习框架，主要原因是，一旦设计好 CNN 的框架，就可以利用 Keras 的序贯模型来建立 CNN 网络，序贯模型是多个网络层的线性堆叠，也就是“一条路走到黑”。如图 4-10 所示的 CNN 结构。

序贯模型有两种建模方式，一种是通过一次性传递一个列表，列表描述了网络的结构，另一种是通过.add()方法来搭建模型。本项目中为了很清楚的描述结构，使用.add()方法将一个层的 layer 加入到模型中。

模型需要知道输入数据的 shape，因此，Sequential 的第一层需要接受一个关于输入数据 shape 的参数，后面的各个层则可以自动的推导出中间数据的 shape，因此不需要为每个层都指定这个参数。

### （3）模型的编译

在训练模型之前，需要通过 compile 来对学习过程进行配置。compile 接收三个参数：

a. 优化器 optimizer：该参数可指定为已预定义的优化器名，如 rmsprop、adagrad，或一个 Optimizer 类的对象，在实验中使用随机梯度下降 SGD 算法来优化。

b. 损失函数 loss：该参数为模型试图最小化的目标函数，它可为预定义的损失函数名，如 categorical\_crossentropy 和 mse，也可以为一个损失函数，由于是分类问题，因此在本项目中使用 categorical\_crossentropy 作为损失函数；

c. 指标列表 metrics：对分类问题，一般将该列表设置为 metrics=['accuracy']。指标可以是一个预定义指标的名字，也可以是一个用户定制的函数。指标函数应该返回单个张量，或一个完成 metric\_name -> metric\_value 映射的字典，在本项目中设置列表为 metrics=['accuracy']，在后期的训练中将为每一轮训练计算 accuracy。

### （4）训练

Keras 以 Numpy 数组作为输入数据和标签的数据类型。训练模型一般使用 fit 函数。fit 函数接受以下参数。

a. 输入数据。如果模型只有一个输入，那么 x 的类型是 numpy array，如果模型有多个输入，那么 x 的类型应当为 list，list 的元素是对应于各个输入的 numpy array；

b. 标签，numpy array；

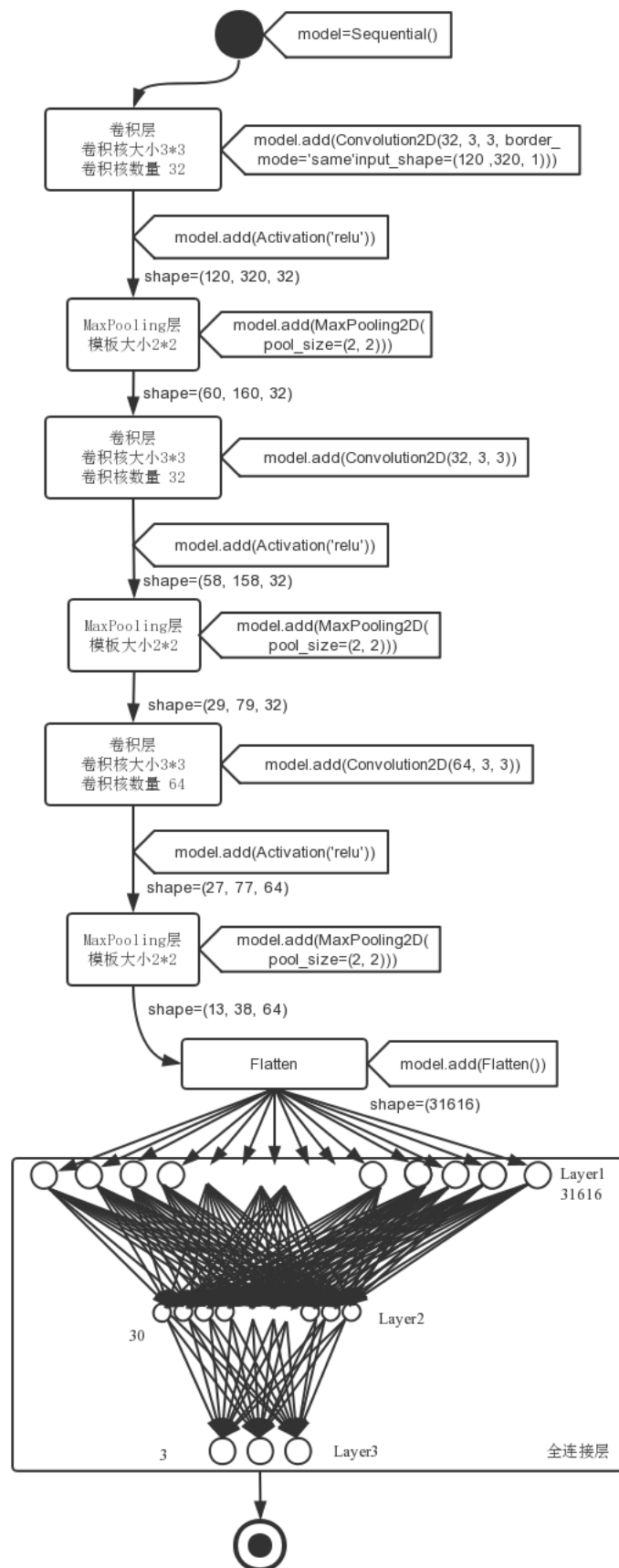


图 4-10 项目使用的 CNN 架构及相关实现

- c. 每一批的训练数据的样本大小 **batch\_size**: 整数, 指定进行梯度下降时每个 **batch** 包含的样本数。训练时一个 **batch** 的样本会被计算一次梯度下降, 使目标函数优化一步;
- d. 迭代次数: 整数, 训练的轮数, 每个 **epoch** 会把训练集迭代一遍;
- e. 是否打印日志信息 **verbose**: 日志显示, 0 为不在标准输出流输出日志信息, 1 为输出进度条记录, 2 为每个 **epoch** 输出一行记录。默认是 1。

#### (5) 评估模型

Keras 以 Numpy 数组作为输入数据和标签的数据类, 评估模型往往使用 **evaluate** 函数, **evaluate** 函数接受以下参数:

- a. 输入数据 **x**, 与 **fit** 一样, 是 **numpy array** 或 **numpy array** 的 **list**;
- b. 标签 **y**, **numpy array**;
- c. 每一批的训练数据的样本大小 **batch\_size**: 整数, 含义同 **fit** 的同名参数;
- d. 是否打印日志信息 **verbose**: 含义同 **fit** 的同名参数, 但只能取 0 或 1;

### 4.3 道路标识的检测与识别

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, 主要训练算法为 **Boosting** 算法和随机森林算法。正样本图像主要使用了一些采集好的图像, 也并且进行裁剪出图像中的要检测的目标对象, 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, 交通红绿灯负样本包含相同数量的黄色交通灯图像。

图 4-11 显示了在该项目中使用一些正样本和负样本图像。本项目中训练级联分类器所使用的正样本和负样本数据规格如表 4-2 所示。

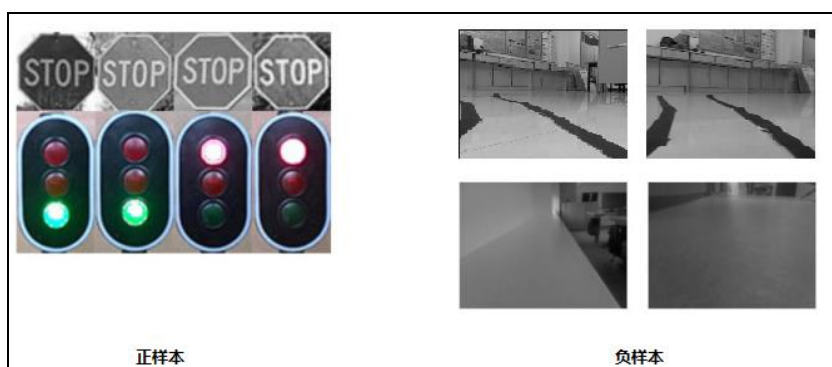


图 4-11 正样本和负样本数据的选择

表 4-2 正样本和负样本数据规格

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

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。流程图4-12总结了交通灯识别的过程。图 4-12 引用自文献[22]。



图 4-12 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 中找到亮度最强的点。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）。

#### 4.4 本章小结

本章主要针对样本数据的收集和深度学习模型的设计、训练和评估进行设计和相关技术的阐述。这些为系统的具体实现提供了工程支持。

## 第 5 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 5-1 会展示传统的机器学习的完整流程，在这里也会按照图 5-1 描述的来展开。

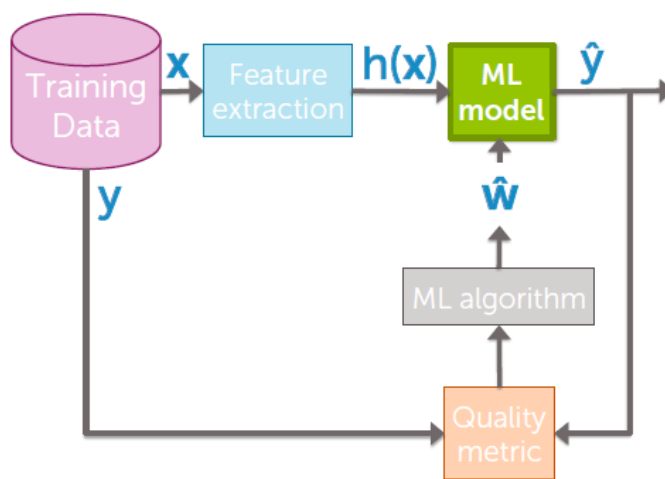


图 5-1 机器学习流程图

### 5.1 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。具体的数据收集如图 5-2 所示。

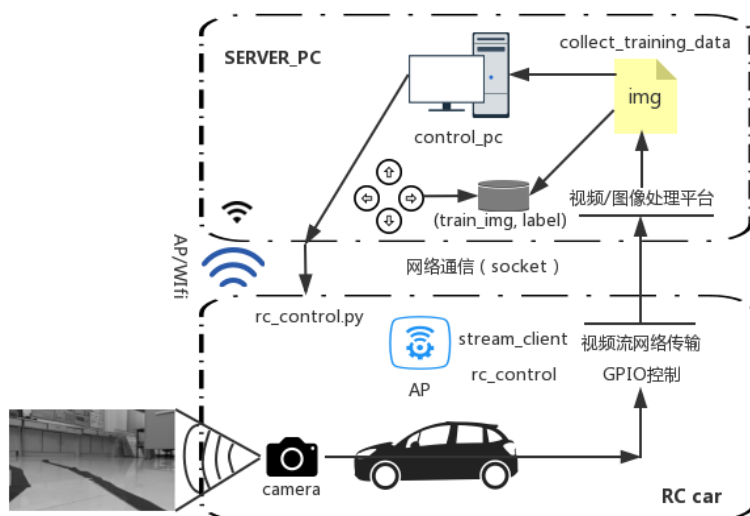


图 5-2 数据采集及处理框架图



### 5.1.1 视频流网络传输

主要采用了 **PiCamera** 模块来做的视频流网络传输，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，这样经过一个来回的数据采集，就构成了一个样本集。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。

### 5.1.2 样本集的生成

该部分的工作主要是在 PC 上完成的，为了能够更好地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 5-3 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 组数据。

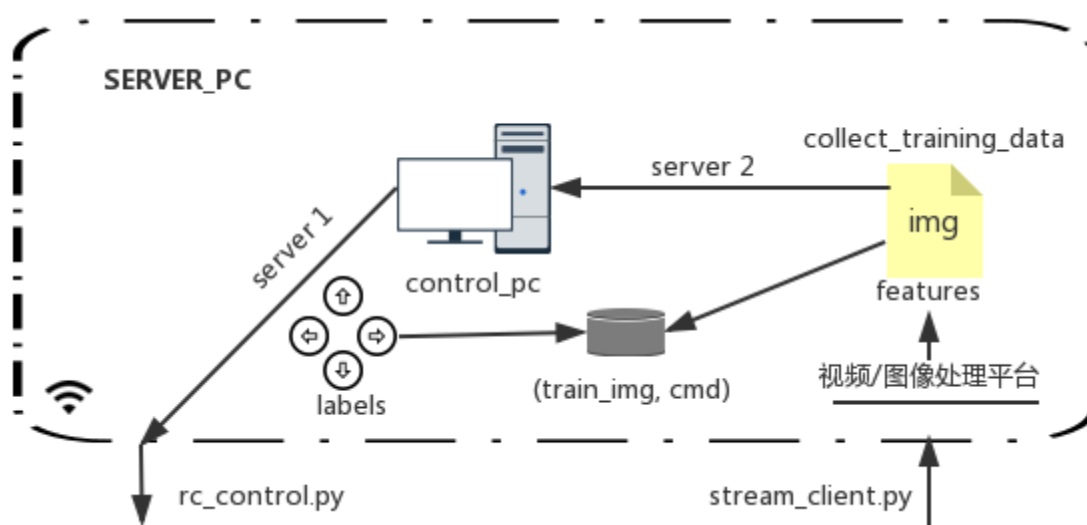


图 5-3 PC 机采集数据的流程图

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 4.2 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  $120 \times 320 = 38400$ ，（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。

## 5.2 模型的设计、实现、训练和测试

### 5.2.1 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。

### 5.2.2 卷积神经网络的实现

实现部分需要注意的是：此时只实现了三种决策的分类任务，分别是[left, right, forward]。

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），最后给出预测（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

$$\text{softmax}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \quad (5-1)$$

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'))

model.summary()
```

深度学习框架结构如图 5-4 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 5-4 深度学习框架结构

图 5-4 描述了深度学习框架图和参数规模，神经网络的参数个数为  $38400 \times 32 \times 4 = 4,195,200$ ，由图 5-4 知 CNN 的参数个数为 976,698，因此与神经网络相比，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。

```
# Evaluate 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。

小结：虽然 MLP 模型参数相比 CNN 参数较多，但 MLP 的网络结构简单且准确度达到 93.2%，而 CNN 的网络结构较复杂且准确度仅为 88%，因此在实际测试中采用了 MLP 作为测试使用的模型。

### 5.3 本章小结

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 6 章 系统测试

系统测试主要分为两部分进行测试，模块测试又称为单元测试，针对特定的功能进行测试。测试又分为单机测试和联机测试，另一部分为整体测试，是针对程序的完整的过程进行有序测试。

### 6.1 模块测试

#### 6.1.1 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 6-1 所示。



图 6-1 视频传输成功

#### 6.1.2 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 6-2 所示

(2) 测试信号（红绿两种状态）灯，测试结果如图 6-3 所示

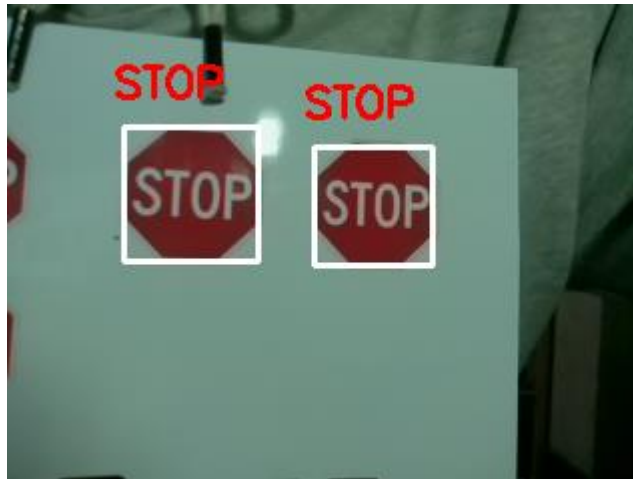


图 6-2 STOP 路标识别成功

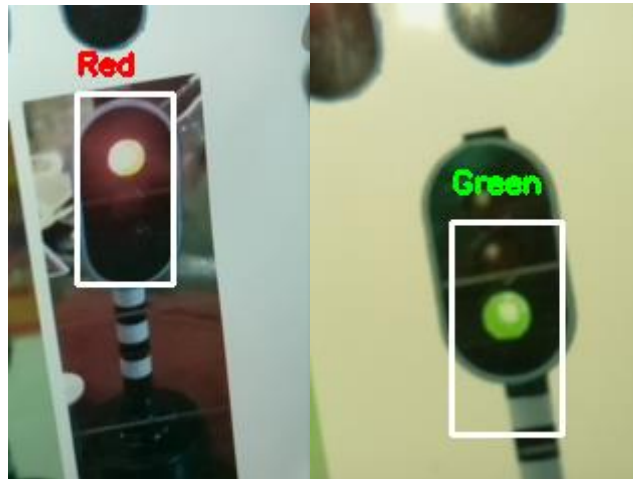


图 6-3 交通信号灯识别成功

### 6.1.3 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）。

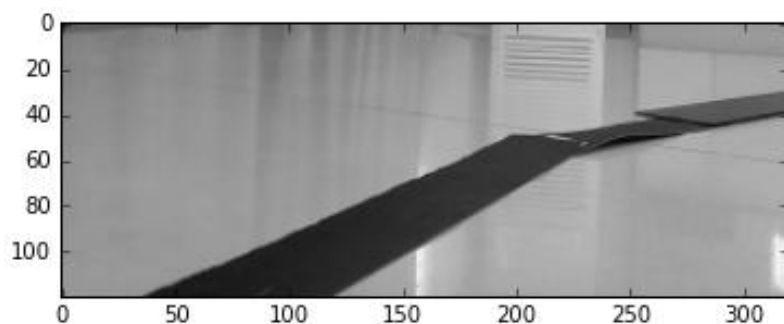


图 6-4 基于小车观测的图像来预测方向

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 6-5 模型给出预测的结果

根据 4.1.2 节设置的[LEFT:0, RIGHT:1, UP:2]可知模型预测的结果是 RIGHT，如图 6-5 所示。

## 6.2 整体测试

### 6.2.1 收集数据

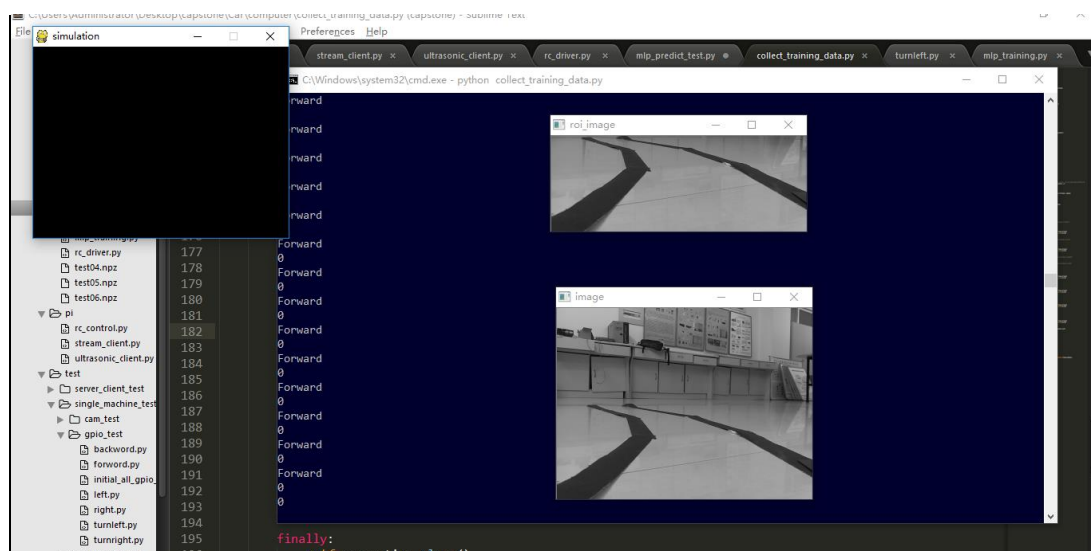


图 6-6 收集数据

收集数据时，PC 与树莓派的交互截图如图 6-6 所示。

6.2.2 训练模型

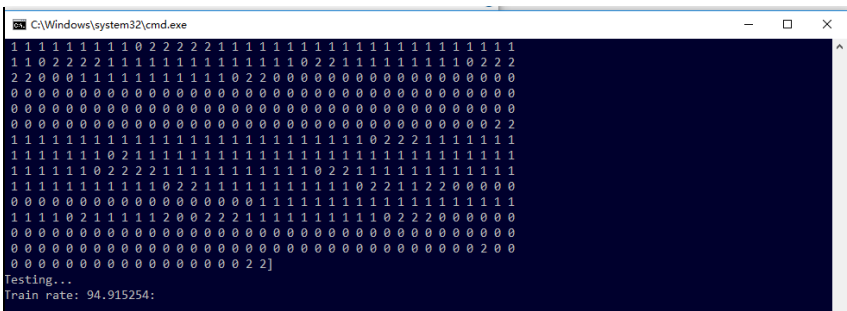


图 6-7 训练结果

主要在 PC 上实现图像数据的训练，给出的训练结果如图 6-7 所示。

6.2.3 小车实时检测并驾驶

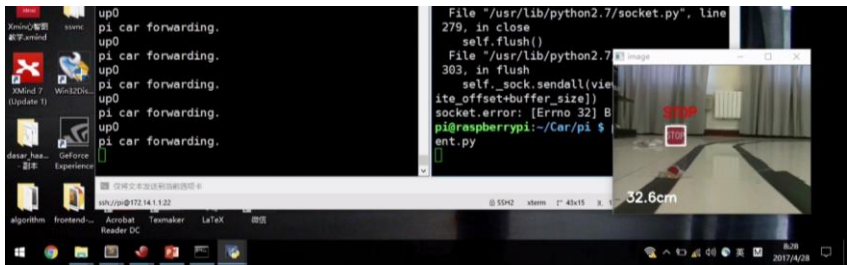


图 6-8 小车检测到 Stop 标志

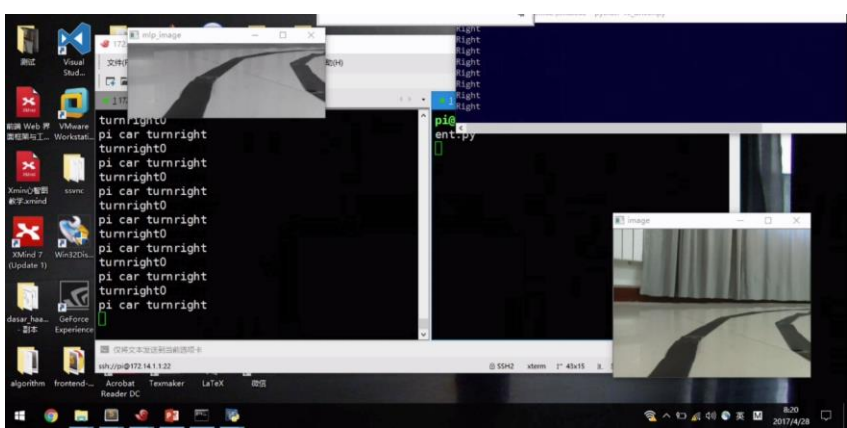


图 6-9 小车在自动驾驶

图 6-8 描述了小车在自动驾驶过程中实时监测到 STOP 标志，图 6-9 描述了小车在利用视觉进行自动驾驶。

6.3 本章小结

本章主要有模块测试和整体测试，在模块测试中使用了 CNN 模型，在整体测试中利用了 MLP 模型。根据整体测试的结果分析，该模型给出了预期的效果。

## 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。

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。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，树莓派自身的性能已经对于该项目已经绰绰有余，但是由于小车自身的硬件（左右转向不平衡）、轨道设计不标准，视频传输不稳定等因素都对小车的测试造成了不必要的干扰，因此针对本项目接下来的工作将是针对这些因素对软件及硬件做优化。针对小车，下一步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，该系统主要基于视觉感知和分类算法。



## 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] Jiangwei C, Lisheng J, Lie G, et al. Study on method of detecting preceding vehicle based on monocular camera[C]. IEEE, 2004.
- [3] Zheng W. OpenCV Python Neural Network Autonomous RC Car[Z]. 2013: 2015.
- [4] Hadik A. MATLAB Neural Network Autonomous Car[Z]. 2013: 2016.
- [5] Zotti R. How to Build Your Own Self Driving Toy Car[Z]. 2016: 2016.
- [6] [www.element14.com/RaspberryPi](http://www.element14.com/RaspberryPi)[Z]. 2017: 2017.
- [7] 贾慧星, 章毓晋. 车辆辅助驾驶系统中基于计算机视觉的行人检测研究综述[J]. 自动化学报. 2007, 33(1): 84-90.
- [8] 刘颖璐. 图像识别中的特征表达方法研究[J]. 2015.
- [9] Srinivasan D, Ng W S, Liew A C. Neural-network-based signature recognition for harmonic source identification[J]. IEEE Transactions on Power Delivery. 2006, 21(1): 398-405.
- [10] 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.
- [11] Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision[C]. 2016.
- [12] 李宏毅. [http://speech.ee.ntu.edu.tw/~tlkagk/courses\\_ML16.html](http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML16.html)[Z]. 2016: 2017.
- [13] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]. 2015.
- [14] Bengio Y, Goodfellow I J, Courville A. Deep learning[J]. Nature. 2015, 521: 436-444.
- [15] 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.
- [16] Zeiler M D, Fergus R. Visualizing and understanding convolutional networks[C]. Springer, 2014.
- [17] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]. 2012.
- [18] 李君宝, 杨文慧, 许剑清, 等. 基于深度卷积网络的 SAR 图像目标检测识别[J].
- [19] 周凯龙. 基于深度学习的图像识别应用研究[D]. 北京工业大学, 2016.
- [20] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library[M]. " O'Reilly Media, Inc.", 2008.

- [21] <https://keras.io/getting-started/sequential-model-guide/>[Z]. 2016: 2017.
- [22] W Z. <https://zhengludwig.wordpress.com/projects/self-driving-rc-car/>[Z]. 2013: 2016.
- [23] [http://docs.opencv.org/2.4/modules/objdetect/doc/cascade\\_classification.html](http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html)[Z].: 2017.
- [24] <http://picamera.readthedocs.io/en/release-1.13/recipes1.html#capturing-to-a-network-stream>[Z].