微缩智能小车自适应巡航控制系统的设计与实现*

朱秀炜 陆涛**

南宁学院信息工程学院，南宁，530200

摘 要 基于深度学习、图像识别等技术，采用 Scikit-learn+OpenCV 技术，设计与实现一个适用于微缩智能小车的自适应巡航控制系统。该系统应用卷积神经网络算法、车辆控制算法和图像识别算法，实现了在微型道路上的自动驾驶、障碍识别和躲避。在微缩智能车进行的数据采集、系统训练和自动驾驶测试结果表明，通过神经网络及其训练的优化方法能够提升机器学习的效果和速度，设计实现的微缩智能小车决策系统能够较好地与车辆控制相结合，实现道路环境的自动驾驶和障碍躲避。

关键字 微缩智能小车，深度学习，自动驾驶，图像识别，巡航控制

Design and Implementation of Adaptive Cruise Control System for Miniature Intelligent Car

Zhu Xiuwei Lu Tao

School of Information Engineering Nanning University
Nanning 530200,China (961852656@qq.com)

Abstract—Based on deep learning, image recognition and other technologies, a decision-making system and adaptive control system suitable for miniature intelligent vehicle control are designed and implemented by using Scikit_learn and OpenCV development tool. The system uses convolution neural network algorithm, vehicle control algorithm and image recognition algorithm to realize automatic driving, obstacle recognition and avoidance on micro road. The test results of data acquisition, system training and automatic driving test in the miniature intelligent vehicle show that the optimization method of neural network and its training can improve the effect and speed of machine learning. The designed and implemented miniature intelligent vehicle decision-making system can be better combined with vehicle control to realize automatic driving and obstacle avoidance in road environment.

Keyword—Deep learning, Autonomous driving, Image recognition, Miniature intelligent car, cruise control

1 引 言

到目前为止，许多国内外企业都在研究和开发自动驾驶技术，这类技术不仅仅运用在家用轿车，还运用到大型货车、特种车辆、履带坦克、地面武器和智能机器人等，其中的代表的软件架构有机器人操作系统 ROS（Robot Operating System）[1]。Google Driverless Car 项目[2]是基于 Google 公司的 Tensorflow 机器学习技术框架开发的深度学习无人驾驶车辆项目，其目的是打造一个类似 Android 手机系统的开源无人驾驶平台，构建自动驾驶生态圈。目前谷歌的无人驾驶技术已经达到了商用的 L3 级别（高度自动驾驶，需驾驶员陪同），试验中的 L4（超高度自动驾驶）自动驾驶平台测试路程也超过了 2000 英里，虚拟测试路程超过 1000 英里[3]。

近两年，小米、华为、阿里、百度、OPPO 等国内企业纷纷加入到造车俱乐部，其中自动驾驶架构以及算法的研究成为车辆开发的主要内容之一[4]。2020 年，长沙市推出了无人驾驶出租车，这是由百度 Apollo 与一汽红旗联合打造的中国首批 L4 乘用车[3]。在无人汽车试运行的一年多时间里，百度无人车的表现十分出色，在行车速度方面，遇到红绿灯和斑马线提前减速方面做得非常自然。可以说，这些无人驾驶出租车的自动驾驶系统对车辆的控制已经不亚于人类[5]。百度无人车的核心技术在于自动驾驶、线路规划和人机交流，其中就大量地使用深度学习技术进行学习与训练[6]。

*基金资助：本文得到广西高校中青年教师科研基础能力提升项目(2020KY64105)、南宁学院教授培育工程项目(2020JSGC03资助.
**通讯作者：陆涛，副教授，9137339099@qq.com

国内有名的华为公司近几年也深入到自动驾驶领域的研究，2021 年华为推出了自家的自动驾驶算法等架构，并通过自主研发的方式将国外的自动驾驶激光雷达采购成本降低了数倍，打破了国外行业垄断，从而有效地降低了自动驾驶的整车造价，有利于促进自动驾驶技术的进步[7]。英伟达图形设备公司主要从事计算机图形算法、硬件的研发，其人工智能技术已经涵盖了高校研究领域、小型智能机器人领域、工业智能机器人领域以及智慧城市领域[8]。

目前，人工智能技术在智能驾驶、智能控制领域的应用还不是十分成熟。为此，目前国外许多优秀大学和团队也在大力开展智能驾驶和智能机器人领域的学习和研究[2][3][5]。

本文课题将深度学习技术与车辆控制技术相结合，在模型和算法设计的基础，对微缩智能车自动控制系统的开发进行需求分析与架构设计，设计实现了微缩智能车自动控制平台，并在小型模型车辆上完成了基本自动驾驶技术的实验、分析和验证，说明了微缩智能车自动控制系统的技术方案是可行、有效的。

2 系统需求分析

2.1 系统功能设计

本系统是在小型越野遥控车底盘、微型树莓派电脑的基础上设计的小型车辆道路巡航系统，能够实现车辆在道路自动预测与驾驶，执行障碍绕行等功能。系统功能需求如图 1 所示。

手动驾驶功能模块通过用户发送小车控制指令，能够发送实时车辆控制信号，记录用户控制数据，记录摄像头数据，生成训练数据。自动驾驶功能模块能够加载控制参数，通过摄像头输入图像信号来实现神经网络模型对车辆进行预测控制。神经网络训练模块通过加载训练参数、训练数据、神经网络配置实现神经网络模型训练与测试。障碍识别模块通过加载识别类来实时识别障碍物与交通灯，并干预车辆控制。

2.2 开发技术需求

本文系统程序基于 Windows 环境开发，程序开发语言为 Python 3.8 和 C 语言。车辆程序运行环境为小型树莓派（Raspberry Pi）电脑的定制化 Linux 系统，神经网络模型的训练环境为 Windows 系统。深度学习系统的神经网络开发基于 Keras 深度学习库，深度学习开发后端使用了 Python 提供的的机器学习开源框架 Scikit-learn[9]。视觉识别基于 OpenCV 计算机视觉识别技术和机器学习库。用户控制器通信基于无线蓝牙技术，用户控制界面基于 HTML 网页技术。

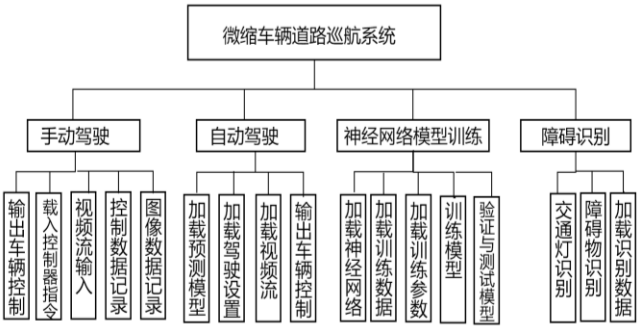

图 1 系统功能需求图

车载树莓派电脑输出的决策命令通过伺服器驱动板（型号 PCA9685）输出具体控制信号给转向舵机和电子调速器，电子调速器用于微缩车辆动力电机控制，转向舵机用于微缩车辆前轮转向控制。系统的视觉输入使用兼容树莓派电脑的可视范围分别为 60°、130°和 160°超广角可调焦距摄像头，车辆平台基于 1:18 比例的微缩越野车。

2.3 系统性能需求

（1） 运行的驾驶主线程序能够维持持续的驾驶回路响应，驾驶线程处理和响应速度应当足够快，驾驶回路需要能够输出高频率的控制信号以确保车辆控制得当，驾驶主线程序应当占用较少资源以保证车载主机性能足够开销。

（2） 训练成型的神经网络模型预测响应应当足够短、决策足够正确以满足车辆实时、持续自动驾驶的需要。

（3）系统神经网络模型的架构类型、学习深度、复杂度以及使用到的参数应当适用于本系统的实现。

（4） 整个系统应当以模块化方式设计多个子系统、使其具备低耦合、高聚合的特点，以实现不同功能模块的组合、调用。

（5） 神经网络的训练方法应该合理高效、用于训练的学习样本数量应当足够多，单个样本容量尽量小，以满足提高训练效果、训练速度的目的。

3 系统概要设计

3.1 微缩车辆平台设计

本系统的硬件平台是在小型越野模型车的底盘上改造而成，车辆为传统的四轮车辆，前轮为转向轮，通过控制舵机以及机械传动相结合实现前轮转向。车辆的驱动动力为有刷电机，通过电子调速器实现电机

正反向转动以及电机转速控制。驾驶程序的运行平台为车载树莓派电脑，该微型电脑的处理器架构为 ARM 架构，微型电脑通过总线接口连接摄像头以及 PCA9685 伺服器控制板，摄像头传感器作为车辆的视觉输入，PCA9685 作为车辆控制输出。微缩智能车所收集到的学习样本等数据需要通过算力较高的 PC 电脑进行处理来获得充分训练的神经网络模型，然后将神经网络模型文件传回树莓派电脑作为车辆自动驾驶的决策模型。车辆的整体架构如图 2 所示。

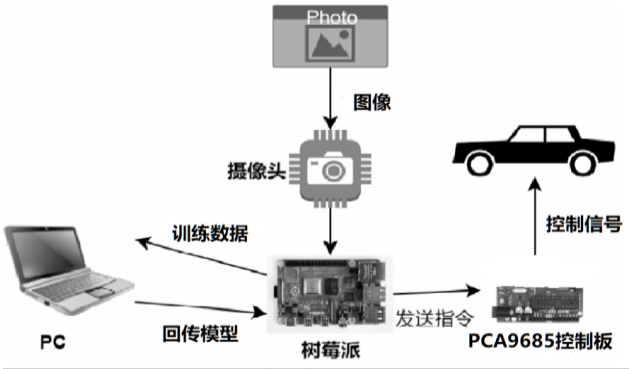

图 2　微缩智能车平台架构与实物图

3.2　系统感知-决策总体架构设计

视觉识别的自动驾驶架构主要分为三种[10]：直接感知架构、间接感知结构、端到端控制架构。本系统是基于计算机视觉自动驾驶方法的端到端控制架构设计。端到端控制架构又称 End-to-End Control，基于机器学习的监督学习方法，端到端控制建立了输入图像到输出控制的映射，将自动车辆控制转换为回归问题或者分类问题去解决。图 3 为微缩小车的感知-决策控制算法架构。

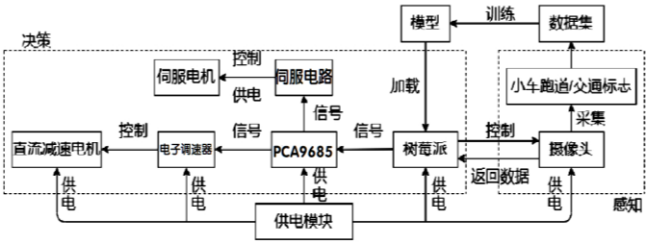

图 3　微缩小车的感知-决策控制算法架构

4　系统功能模块的设计与实现

4.1　神经网络模块

神经网络模型用于小车的控制预测，包括油门、转向角度预测，以及物体识别。相应的神经网络模块、训练模块编写完成后，还需要使用学习样本对神经网络系统进行训练，生成具体的神经网络模型，以实现控制预测。本系统的神经网络系统使用基于 TensorFlow 2 作为后端的 Keras 深度学习库进行编写，模型位于 keras.py 文件。

如图 3，方框内部为系统神经网络的卷积核心层，该层通过 core_cnn_layers()方法定义。本系统设计有两个不同的神经网络类型，第一种为线性输出神经网络，通过 default_n_linear()方法定义完整模型；第二种为分类输出神经网络，通过 default_n_categorical()方法定义完整模型，它们共用 core_cnn_layers()方法定义的卷积核心层。网络的构成都包括输入层、卷积层、平面层、全连接层、输出层、随机损失、激活函数等。类 KerasCategorical()、类 KerasLinear()分别定义了分类输出神经网络、线性输出神经网络使用的完整方法，包括模型初始化、编译、预测、输入输出数值转换、结果返回等，下面通过程序代码分别进行介绍。

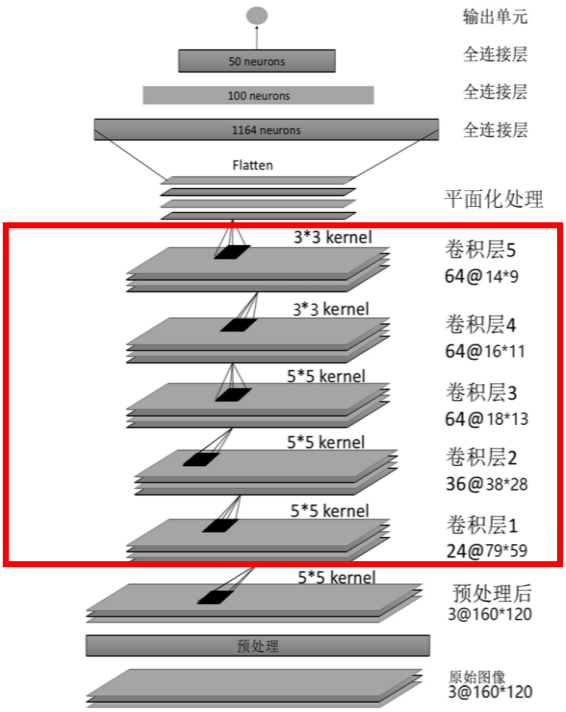

图 4　系统卷积神经网络结构

(1) 卷积层创建

卷积层创建是用 conv2d()方法实现的，其核心代码如下：

```
def conv2d(filters,kernel,strides,layer_num,activation= 'relu'):
    return Convolution2D(filters=filters,
        kernel_size=(kernel, kernel),
        strides=(strides, strides),
        activation=activation,
        name='conv2d_' + str(layer_num))
```

conv2d 方法用于返回一个二维卷积层，二维卷积

层通过 Keras 开发库的 Convolution2D 方法定义。参数 filters 用于定义神经元数量；参数 kernel_size 用于定义卷积核大小；参数 strides 用于定义滑动步长；参数 activation 用于定义激活函数，默认是 relu（ReLU 激活函数）；参数 name 用于定义卷积层层级。

(2) 核心卷积网络层创建

core_cnn_layers()方法定义并返回一个 CNN 卷积核心层，分享给不同类型神经网络，比如线性模型、分类模型。该方法通过事先定义的 conv2d()方法创建 5 个二维卷积层，卷积核数量从上层至下层分别为 24、32、64、64、64 个，第一、二、三层卷积核大小为 5×5，第四、第五层卷积核大小为 3×3，滑动步长为从上层至下层分别为 2、2、2、1、1：每一层二维卷积层都定义一个 Dropout 方法用设定随机损失率，用于神经元的随机失活：在第五层卷积层输出位置使用 Flatten()方法定义一个平面层，用于将输出平面化，再返回完整的卷积层堆输入到下一层级。

核心卷积网络层创建方法 core_cnn_layers()方法核心代码如下：

```
def core_cnn_layers(img_in, drop, l4_stride=1):
    x = img_in
    x = conv2d(24, 5, 2, 1)(x)
    x = Dropout(drop)(x)
    x = conv2d(32, 5, 2, 2)(x)
    x = Dropout(drop)(x)
    x = conv2d(64, 5, 2, 3)(x)
    x = Dropout(drop)(x)
    x = conv2d(64, 3, l4_stride, 4)(x)
    x = Dropout(drop)(x)
    x = conv2d(64, 3, 1, 5)(x)
    x = Dropout(drop)(x)
    x = Flatten(name='flattened')(x)
```

(3) 线性输出模型

default_n_linear()方法可用于定义完整的线性输出卷积神经网络模型。通过 Input()定义输入层；通过事先编写的 core_cnn_layers()方法定义卷积层堆；再通过 Keras 库的 Dense 方法定义两个全连接层：全连接层神经元数量分别为 100、50，激活函数皆为 ReLU 激活函数；每层网络层都会通过 Dropout()方法定义随机损失；最后通过 outputs 定义输出，输出层有两层，outputs[0]和 outputs[1],分别用于转向输出和油门输出，输出预测结果的为线性浮点数值。通过调用该方法可以返回一个线性输出神经网络模型。

default_n_linear()方法的核心代码如下：

```
def default_n_linear(num_outputs, input_shape=(120, 160, 3)):
    drop = 0.2
    img_in = Input(shape=input_shape, name='img_in')
    x = core_cnn_layers(img_in, drop)
    x = Dense(100, activation='relu', name='dense_1')(x)
    x = Dropout(drop)(x)#定义网络层的随机损失
    x = Dense(50, activation='relu', name='dense_2')(x)
    x = Dropout(drop)(x)
    outputs = []
    for i in range(num_outputs):#通过循环输出两个输出层，每层输出一个线性值
        outputs.append(
          Dense(1, activation='linear', name='n_outputs' + str(i))(x))
        model = Model(inputs=[img_in], outputs=outputs)
    return model
```

（4）分类输出模型

default_n_categorical()方法可以定义一个完整的分类输出卷积神经网络。除了输出层以外，结构与 default_n_linear()方法类似，此处不再介绍。输出层为一层二维矩阵结果，由“矩阵[转向输出分类,油门输出分类]”构成，转向输出将车辆转向控制划分为 15 类，每一类对应不同的前轮转向角度；油门输出将车辆电机输出转化为 20 类，其中包括不同速度的前进油门、后退油门以及零油门。两个输出层的激活函数都使用 softmax 激活函数，softmax 激活函数有利于将离散结果转化为分类值。default_n_categorical()方法核心代码如下：

```
def default_categorical(input_shape=(120, 160, 3)):
    drop = 0.2
    img_in = Input(shape=input_shape, name='img_in')
    x = core_cnn_layers(img_in, drop, l4_stride=2)
    x = Dense(100, activation='relu', name="dense_1")(x)
    x = Dropout(drop)(x)
    x = Dense(50, activation='relu', name="dense_2")(x)
    x = Dropout(drop)(x)
    # 分类模型将转向输出分为 15 级，激活函数将预测值转换成概率区间
    angle_out = Dense(15, activation='softmax', name='angle_out')(x)
    # 分类模型将油门输出分为 20 级
    throttle_out = Dense(20, activation='softmax', name='throttle_out')(x)
    model = Model(inputs=[img_in], outputs=[angle_out, throttle_out])
    return model
```

（5）线性输出神经网络使用方法类

类 KerasLinear()定义了完整的线性输出模型的各种方法，_init_()用于初始化一个自动驾驶模型。compile()方法用于编译神经网络模型，inference()方法用于自动驾驶的车辆控制预测，通过将摄像头图像流持续输入到神经网络，获得神经网络的预测值，将预测值输入控制模块实现车辆自动转向以及油门控制；y_transform()和 y_translate()方法用于将训练数据中记录的用户控制值提取并转化成字典集的形式，输入到

神经网络中作为训练标签。 output_shapes()方法用于创建输入输出矩阵的函数图像。

类 KerasLinear()核心代码的操作步骤如下：

Step1：定义线性输出自动驾驶模型，该模型通过输出浮点预测值的形式可输出细腻的转向角度控制和油门控制。其中包括初始化模型、定义编辑模型。

Step2：调用 inference 方法用于输出驾驶预测值，首先输入数据维度转换参数，然后调用 Keras 库的 models.predict 方法获得预测输出，包括转向、油门等数值。

Step3：训练数据记录的控制数据提取。

Step4：将训练数据中的控制数据转化成字典集形式。

Step5：定义数据的输出张量，并返回得到的数值。

类 KerasCategorical()定义了分类输出自动驾驶模型的各种方法，用法与类 KerasLinear()类似，此处不再介绍。

（6）神经网络训练功能

train()方法定义了神经网络的训练方法并返回训练模型以及训练结果、验证结果，包括定义训练生成器 train_gen，验证生成器 val_gen，训练迭代数 epochs、训练集&验证集比例 train_split、保存模型方法 save_best，提前结束训练方法 early_stop 等。train()方法的主要步骤如下：

Step1：定义训练参数，保存路径，训练迭代数等。

Step2：通过模型检查点在每次训练轮结束后保存最佳训练模型。

Step3：定义提前结束训练方法,当训练的验证错误率提高时停止训练。

Step4：调用 Keras 库的 model.fit_generater 方法训练模型并输出训练结果，将结果显示在主控窗口。

4.2 车辆控制模块设计

车辆的控制模块由类 Vehicle 实现，该类下定义了车辆控制线程的各种方法，包括_init_()、add()、remove()、start()、update_parts()、stop()等方法。其中_init_()方法用于初始化，包括划分内存、添加部件、创建车辆控制线程，初始化配置参数等；add()方法用于添加线程部件，包括获取车辆控制输出通道，获取决策输入通道等；remove()方法用于移除车辆线程部件；start()方法用于启动及时响应的车辆驾驶线程；update_parts()方法用于高频率地循环更新车辆驾驶线程组件以获取最新组件数据；stop()方法用于结束驾驶线程。本节主要介绍 start()方法和 update_part()方法。

(1) 车辆控制线程的进程创建方法

车辆控制线程的 start()方法用于开启一个循环的车辆控制主线程。当车辆处于人工收集训练数据模式时，会不断获取摄像头视频流，记录图像以及匹配的人工控制的控制信息（转向控制、油门控制）存入内存中并储存记录，同时根据控制信息给车辆控制板发送实时的 PWM（脉冲宽度控制）信号，控制车辆的舵机（转向控制）、电子调速器（动力电机控制）。

当车辆处于自动驾驶模式时，线程会按照预设定的循环频率不断地获取摄像头输入的视频流保存到内存中，同时持续收听神经网络模块通过图像信息给出预测结果，将结果转换成 PWM 控制指令输出到控制板控制车辆的舵机以及电子调速器。

start()方法开启车辆线程后，线程通过调用 update_parts()方法不断循环线程以实现组件的更新以及车辆的持续驱动。

(2) 车辆控制线程的循环更新方法

update_parts()方法用于车辆驾驶线程的循环更新，由于微缩车辆控制需要高频率地获取摄像头图像流，并将其用作神经网络的预测输入，再将预测结果实时转换成车辆的控制，因此驾驶线程必须不断接收输入数据以及不断发送输出指令，这些组件的循环更新通常是在数毫秒至数十毫秒内完成的，如果数据更新太慢会导致车辆输入与输出出现较大控制延迟，无法正常匹配控制。

(3) 训练脚本程序

训练脚本程序 train.py 用于先前构建好的神经网络的训练，在该程序中定义了 train()方法（与神经网络模块的 train 方法不同，该方法会调用神经网络模块的 train()方法）。本模块的 train()方法中定义了完整的训练方法，包括提取训练样本数据中的图像集以及对应标签集、定义训练参数、迭代训练次数、每轮训练使用的训练样本等训练选项，并输出训练足够的神经网络模型文件以及训练结果、验证结果等数据到控制台。该程序的__main__函数调用方法 train()并执行，并通过可选的程序运行参数指定训练样本地址、模型保存地址、训练参数等。该方法的主要步骤如下：

Step1：指定训练数据配置文件中的图像以及对应控制输出。

Step2：实例化一个线性输出神经网络。

Step3：提取图像集以及输出值，定义训练&验证集比例等。

Step4：执行神经网络模块中的训练函数。

Step5：定义训练模型的名字以及 h5 格式。

Step6：添加训练选项并执行训练，如指定输入路径、模型保存路径、训练参数等。

（4）驾驶脚本程序

该脚本程序用于自动执行小车开车所需的所有操作，根据用户的驾驶命令可以选择手动驾驶（用于测试和收集训练样本），手动驾驶模式会添加网络控制组件，可以通过网页控制车辆以及查看摄像头实时视频流。在自动驾驶模式下会添加自动驾驶模块用于自动

驾驶预测。除此之外还需要添加摄像头、系统时钟、PCA9685 控制器等组件。

驾驶程序核心方法 drive()的主要步骤如下：

Step1：调用 vehicle()的方法实现驾驶控制。
Step2：添加系统计时组件。
Step3：添加树莓派摄像头组件。
Step4：添加网页控制器，输入输出，用于人工控制。
Step5：如果不是手动控制模式，则运行自动驾驶系统。
Step6：通过添加 state_controller 组件来输出控制信号给 PCA9685 控制板控制小车油门和转向。
Step7：添加一个数据集来保存训练样本
Step8：运行车辆主程序。

5 实验与性能分析

实验过程由数据采集、模型训练、自动驾驶测试以及交通灯识别测试四部分组成。数据采集使用人工遥控车辆在跑道上通行来收集训练样本；模型训练阶段针对不同的模型进行训练；自动驾驶测试阶段，将不同神经网络结构、训练方式、训练次数以及驾驶策略的神经网络模型输入微缩智能车进行测试，测试小车在跑道上的通过能力。

实验的训练配置参数设置最大 100 轮训练，每轮随机选择训练样本中的 128 份样本作为训练输入，其中 80%输入样本用作训练，20%输入样本用作验证，训练时使用训练提前结束方法。如果在训练过程中接连五轮训练效果不再提升，那么停止训练进程并且保存训练结果。训练时间依据训练次数、训练方式、神经网络模型的不同有显著差异。

图 5 是训练轮数为五轮的线性输出神经网络模型曲线图，其中上方线条为训练损失曲线，通过训练损失数值可以判断出神经网络模型学习的效果，训练损失越小，模型的学习效果越好。下方曲线为验证损失曲线，在每轮的训练完成后会将一定数量的训练样本输入训练好的神经网络模型来测试预测的正确性。本文实验设置的验证集比例为 0.2，即将输入样本的 20%作为验证样本，通过验证损失可以判断模型是否得到足够的训练。当训练损失与验证损失几乎一致时，证明系统的预测与损失相符，神经网络在该阶段获得了良好的训练效果。

第一次训练使用的神经网络类型为线性输出神经网络，单轮训练样本数量 128 份，设定训练最大轮数为 5 轮。从图 5 可看出，模型前 3 轮训练的训练损失（train）下降很快，第 3 轮到第 5 轮训练的损失下降较前 3 轮减缓，但依旧可以推测出训练 5 轮后训练损失仍有下降空间。其次可以看到验证损失的损失率较训练损失的损失率相差较大，证明模型并没有得到充分的训练。尽管如此，五轮训练的训练损失值依然低于 0.2，说明模型至此已经获得了一定的训练效果。实验中，车辆能够通过跑道中较为笔直的区域，当遇到连续的 S 型弯道以及障碍物时车辆无法正确转弯最终冲出跑道，这说明该模型只能部分通过跑道，训练远远没有达到要求。

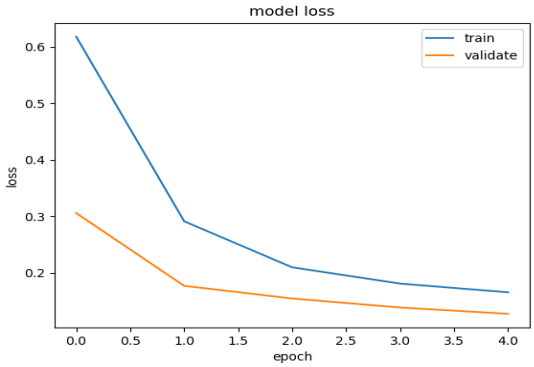

图 5　5 轮 128 样本训练的线性输出模型结果统计图

第二次训练的神经网络模型依旧选用线性输出神经网络模型，训练轮数设定为 100 轮以获得足够训练次数，单次样本数设定为 128 份，在历经 23 轮训练后训练损失与验证损失一致且不再降低，此时训练提前结束。训练结果如图 6 所示，从结果可看出，前 3 轮训练的训练损失曲线下降极快，第 3 轮到第 10 轮训练的训练曲线变缓，第 10 轮训练到训练结束前训练曲线下降极慢，最终训练损失值以及验证损失值均停留在 0.1 左右，说明模型在给定样本下得到了充分的训练。

第三次训练与第二次训练配置类似，不同之处在于将单次训练样本从 128 份调整为 64 份，最终训练和测试结果与第二种训练结果相似，训练结果如图 7 所示。从训练结果曲线图可以看出，此次训练中单轮训练效果相较于第二次训练的单轮效果稍差，主要还是在于单次训练样本较少的原因，但通过训练次数的增加，其结果与第二次训练最终结果相近。

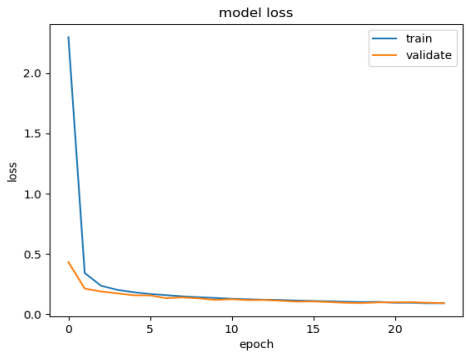

图 6　24 轮 128 样本训练的线性输出模型结果统计图

第四次训练使用分类输出神经网络模型，单轮训练样本数为 128，设定最大训练轮数为 100，最终本次训练提前结束在第 21 轮，此时训练损失最低为 0.2。训练结果如图 8 所示。

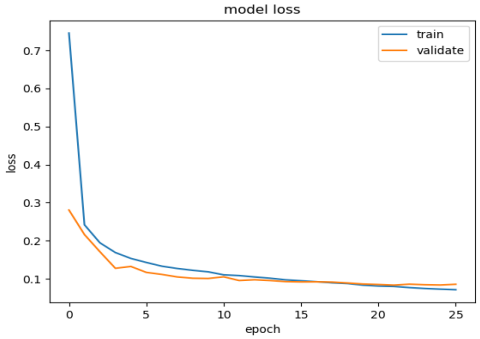

图 7 26 轮 64 样本训练的线性输出模型结果统计图

从实验结果来看，训练与验证的拐角出现在第 9 次附近，在随后的训练中，训练损失持续下降，但验证损失已经趋于不变，在第 16 轮训练后验证损失反而持续上升，此时模型已经趋于过拟合，增加训练不仅不会提升训练效果，反而会较低预测能力，因此提前结束本轮的训练，最终训练损失以及验证损失停留在了 0.2 和 0.5。通过分析训练数据发现，分类训练模型由于分类结果较少，通过大量训练极易导致过拟合，反而影响训练效果。在测试阶段，使用该模型作为决策时，车辆会出现严重的车身抖动，通过赛道时车辆左右摇摆，虽然 10 次测试中多次完成了赛道的通过，但都触及了赛道之外。

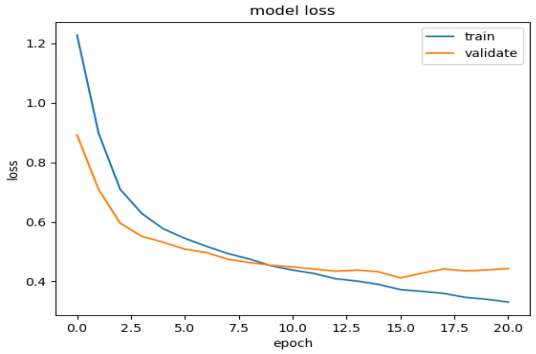

图 8 21 轮 64 样本训练的分类输出模型结果统计图

通过分析以上四次训练测试结果，可以看出线性输出神经网络具有较好的预测精度，且能够输出较为细腻的车辆控制，但线性输出神经网络对硬件性能要求较高，获得最佳训练的模型较长。分类输出神经网络虽然测试与验证损失较线性输出网络模型高，但基本通过了道路测试，但相较于线性输出的网络模型，控制输出较为粗糙，驾驶线路不够平滑。与线性输出神经网络，分类输出神经网络模型可以灵活地调整输出区间规模，通过增加输出区间提高预测精度，减少区间以提高自动驾驶性能，同时其完成最佳模型的训练时长较线性输出神经网络模型短。

实验测试结果可以证明，本文设计微缩智能车的决策系统以及控制系统具有较为良好的自动驾驶效果，但其控制算法以及硬件设计依旧需要改进使其能够在复杂路线中行驶。图像识别结果表明，对于交通灯的红灯和绿灯状态的亮度判断，相关算法能够实现小型交通灯状态的识别，但由于平台限制无法测试真实的驾驶环境效果，因此平台依旧需要进行改进。

6 结束语

本文研究了深度学习技术与车辆自动控制技术，以基于深度学习技术的卷积神经网络模型和“感知-决策-控制”方案为基础，采用 Scikit-learn+OpenCV 技术，对微缩智能车自动控制系统实验平台进行设计与技术实现，包括系统的需求分析、架构设计、硬件设计、软件代码的编写与改进等。在软件方面，设计实现了微缩智能车的决策系统、控制系统、训练系统以及驾驶系统。硬件上进行了运行平台的设计、车辆控制线路的设计，并优化了原有硬件控制。微缩智能车的实验运行结果表明，微缩智能车能够实现微型道路上的自动驾驶、障碍识别和躲避，说明了微缩智能车自动控制系统的设计方案和实现技术是可行、有效的。

参 考 文 献

[1] 李胜. 基于 ROS 的机器人导航系统架构设计[J]. 电脑知识与技术, 2017, (20):171-172, 177.

[2] 张新钰, 高洪波, 赵建辉, 周沫. 基于深度学习的自动驾驶技术综述[J]. 清华大学学报(自然科学版), 2018, 58(04): 438-444.

[3] 张文. 一键呼叫免费试乘 无人驾驶出租车真的来了！[J]. 人民交通, 2020(05):36-37.

[4] 贾传伟. 基于深度学习的智能小车避障优化设计[D]. 青岛科技大学, 2020.

[5] 陈慧, 徐建波. 智能汽车技术发展趋势[J]. 中国集成电路, 2014, 23(11):64-70.

[6] 鲁嘉淇. 浅析基于视觉识别的自动化技术在快递企业中的应用——以顺丰自动分拣机器人为例[J]. 中国战略新兴产业, 2017(12):132-133.

[7] 赫荣亮. 华为自动驾驶技术优势分析[CB/OL]. https://xw.qq.com/amphtml/20210416A02W9C00, 2021

[8] 林嘉文. 英伟达杠上英特尔 谁才是人工智能计算的未来?[J]. 信息与电脑(理论版), 2016(18):29-30.

[9] A Swami, R Jain. Scikit-learn: Machine Learning in Python[J]. Journal of Machine Learning Research, 2013, 12：2825－2830

[10] 白辰甲. 基于计算机视觉和深度学习的自动驾驶方法研究[D]. 哈尔滨：哈尔滨工业大学, 2017