

190183

基于端到端深度学习的 自动驾驶系统

设计说明书

目录：

- 一、项目摘要
- 二、项目背景
- 三、项目运行环境配置
- 四、数据采集
- 五、核心技术
- 六、模型训练
- 七、模型运行整体运行流程
- 八、项目文件介绍
- 九、运行效果截图
- 十、测试项目
- 十一、未来展望

一、项目摘要

自动驾驶（Self-driving），泛指协助或代替人类驾驶汽车的技术。自动驾驶又称无人驾驶、电脑驾驶或轮式移动机器人，是一种通过电脑系统实现无人驾驶的技术。经过 20 世纪末到本世纪初的数十年发展，自动驾驶技术已日渐走向实用，现今的自动驾驶多用雷达，激光测距仪，摄像头来实现，而我们则在此基础上另辟蹊径，使用纯视觉系统来达成自动驾驶，基于此目的本项目利用卷积神经网络和 tensorflow 高阶 API 训练出一个可以识别路上车辆等物体，以及避让，还能根据路况来自动变速的基础自动驾驶系统。

二、项目背景

现今的自动驾驶技术大多是基于雷达传感器、摄像头等技术，再辅以智能算法，在最短时间内计算行驶路线、躲避障碍物。计算机之于人的优势在于，一方面计算机对于距离的把控和计算能力更为强大，更重要的是可以避免人为的操作失误而带来的交通事故。与此同时自动驾驶让开车之人解放了双手，赋予他们更多的时间在车上去做其他的事情。

自动驾驶在以汽车为职业的工作中优势极为明显，例如公交车和长途货车，在此类时间跨度长，路途较为固定和突发事件较少的行程中，自动驾驶系统可以发挥其最大优势，它可以辅助司机节省体力，并且在减少事故的同时更好的完成行驶任务。

然而常用的自动驾驶系统较为复杂，摄像头，雷达，激光测距仪,虽然这些设备能够保障系统的精确，但是在成本上居高不下，同时各种系统的融合容易导致可靠性不高和优化问题，故我们运用纯视觉自动驾驶系统在路面安全的前提下最大程度的减少成本，并提高系统稳定性。

三、项目运行环境配置

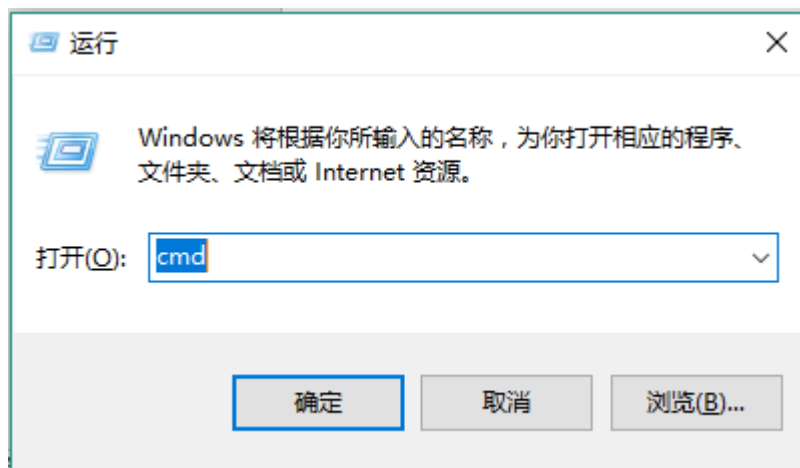
项目需要运行在以下配置的计算机中

CPU	至少 Inter Core I5-9400F
GPU	至少 NVIDIA GeForce GTX 1070
内存	至少 16GB
操作系统	Windows10 64 位

1. 安装 python3.6

在 <https://www.python.org/downloads/release/python-367/> 下载最新版 python3.6 并根据指引安装。在安装最后选择“add PATH”。

安装 python 成功后，打开运行，输入“cmd”，并点击确定。



此时会弹出命令行画面。输入“pip”并点击回车，若出现以下画面，则说明 pip 安装成功。

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.15063]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\lyz\master>pip

Usage:
  pip <command> [options]

Commands:
  install           Install packages.
  download          Download packages.
  uninstall         Uninstall packages.
  freeze            Output installed packages in requirements format.
  list              List installed packages.
  show              Show information about installed packages.
  check             Verify installed packages have compatible dependencies.
  config            Manage local and global configuration.
  search            Search PyPI for packages.
  wheel             Build wheels from your requirements.
  hash              Compute hashes of package archives.
  completion        A helper command used for command completion.
  help              Show help for commands.

General Options:
  -h, --help          Show help.
  --isolated          Run pip in an isolated mode, ignoring environment variables and user configuration.
  -v, --verbose        Give more output. Option is additive, and can be used up to 3 times.
  -V, --version        Show version and exit.
  -q, --quiet          Give less output. Option is additive, and can be used up to 3 times (corresponding to
WARNING, ERROR, and CRITICAL logging levels).
  --log <path>        Path to a verbose appending log.
  --proxy <proxy>      Specify a proxy in the form [user:passwd@]proxy.server:port.
  --retries <retries> Maximum number of retries each connection should attempt (default 5 times).
  --timeout <sec>      Set the socket timeout (default 15 seconds).
  --exists-action <action> Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup,
```

2. 安装所需 pip 包

我们项目中用 tensorflow 作为深度学习算法框架，以及使用 opencv 作为图像处理的主力框架，PIL 作为图像处理的辅助框架。下列包是我们项目用到的第三方 python 库。

包名	版本
Numpy	1. 16. 2
Tensorflow-gpu	1. 12. 0
Keras	2. 2. 4
OpenCV-python	3. 4. 5. 20
Pillow	5. 4. 1
Requests	2. 21. 0
Tqdm	4. 31. 1
Scipy	1. 2. 1

Protobuf	3.6.1
Scikit-learn	0.20.3
Pickle	一般为自带，无需安装

在命令行中输入：pip install 包名 == 版本

例如：pip install numpy==1.16.2

若出现网络错误或网络速度慢的情况，请更换至 pip 清华源 (<https://mirrors.tuna.tsinghua.edu.cn/help/pypi/>)。

若出现权限问题，请使用管理员权限运行命令行。

2. 配置 NVIDIA CUDA 和 cuDNN 环境

CUDA (Compute Unified Device Architecture)，是显卡厂商 NVIDIA 推出的运算平台。CUDA™是一种由 NVIDIA 推出的通用并行计算架构，该架构使 GPU 能够解决复杂的计算问题。

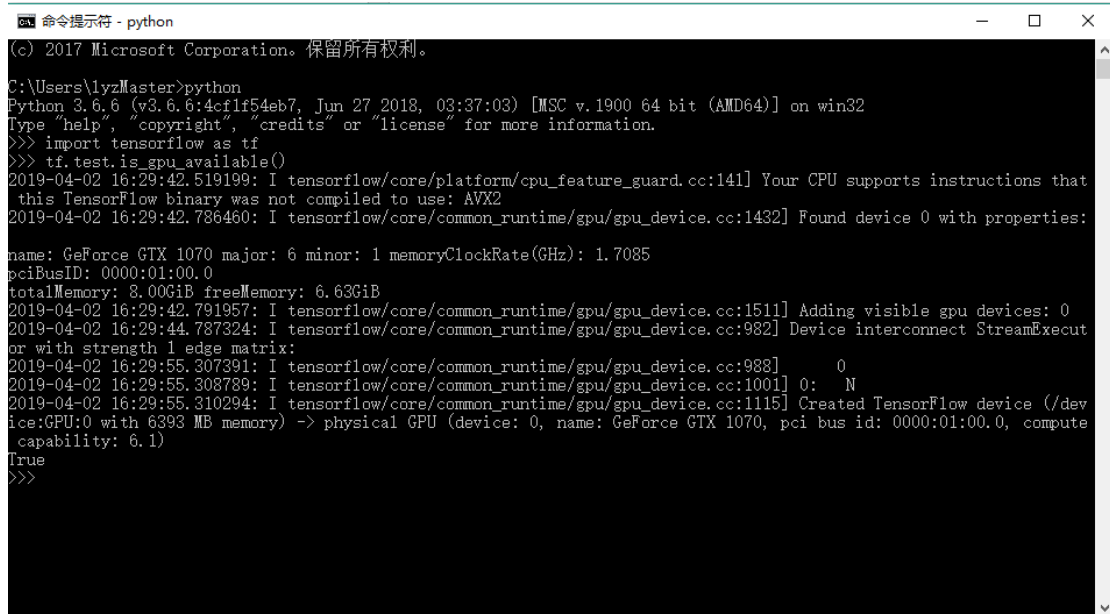
cuDNN 是用于深度神经网络的 GPU 加速库。cuDNN 为标准例程提供高度优化的实现，例如前向和后向卷积，池化，规范化和激活层。

前往官网下载 CUDA Toolkit 9.0 (<https://developer.nvidia.com/cuda-90-download-archive>)，并按默认配置安装。

CUDA 安装成功后，前往官网下载 cuDNN 7.5 (<https://developer.nvidia.com/cudnn>)。将下载完成的文件解压缩，将其中所有文件夹，粘贴至 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0 目录

4. 验证安装是否成功

在命令行中输入 python，并点击回车进入 python 环境



```
(c) 2017 Microsoft Corporation. 保留所有权利。

C:\Users\lyzMaster>python
Python 3.6.6 (v3.6.6:4c1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import tensorflow as tf
>>> tf.test.is_gpu_available()
2019-04-02 16:29:42.519199: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that
this TensorFlow binary was not compiled to use: AVX2
2019-04-02 16:29:42.786460: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: GeForce GTX 1070 major: 6 minor: 1 memoryClockRate(GHz): 1.7085
pciBusID: 0000:01:00.0
totalMemory: 8.00GiB freeMemory: 6.63GiB
2019-04-02 16:29:42.791957: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2019-04-02 16:29:44.787324: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecut
or with strength 1 edge matrix:
2019-04-02 16:29:55.307391: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988] 0
2019-04-02 16:29:55.308789: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0: N
2019-04-02 16:29:55.310294: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/dev
ice:GPU:0 with 6393 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1070, pci bus id: 0000:01:00.0, compute
capability: 6.1)
True
>>>
```

出现以上内容即为 CUDA 环境配置成功。

5. 安装 pyCharm

我们编辑调试代码时,所使用的IDE为pyCharm,这是目前最为常见且最为强大的python IDE。

前往 <https://www.jetbrains.com/pycharm/> 下载最新社区版安装文件,并按照默认配置安装。

6. 安装模拟器 (可选)

这一步,我们将安装模拟器,这是一款由 ROCKSTAR 公司出品的模拟城市以及驾驶游戏 Grand Theft Auto V (简称为 GTAV)。这款游戏的画面逼真,许多无人驾驶团队都将该游戏作为数据采集和测试模型的工具。使用游戏训练模型采集数据,可大大降低成本。我们也将这款游戏作为我们的模拟器。

我们使用的为 1.0.231.0 NON-STEAM 版本的 GTAV。由于, GTAV 本身不作为模拟器而设计,所以无法直接读取游戏中的数据或使用程序与 GTAV 通讯。所以我们使用插件 DeepGTAV 作为程序与模拟器通讯的桥梁,这款插件基于 ScriptHook 开发。所以使用新版本 GTAV 可能会出现插件无法读取的问题。

GTAV 是一款需要付费的商业软件,所以由于版权问题无法进行传播。

我们考虑到以上问题,从 GTAV 中提取出了一段具有代表性的路段,可使用这段数据进行模型测试,而无需安装 GTAV。(video_drive.py 即为该程序)

四、数据采集

1. 数据采集系统整体流程

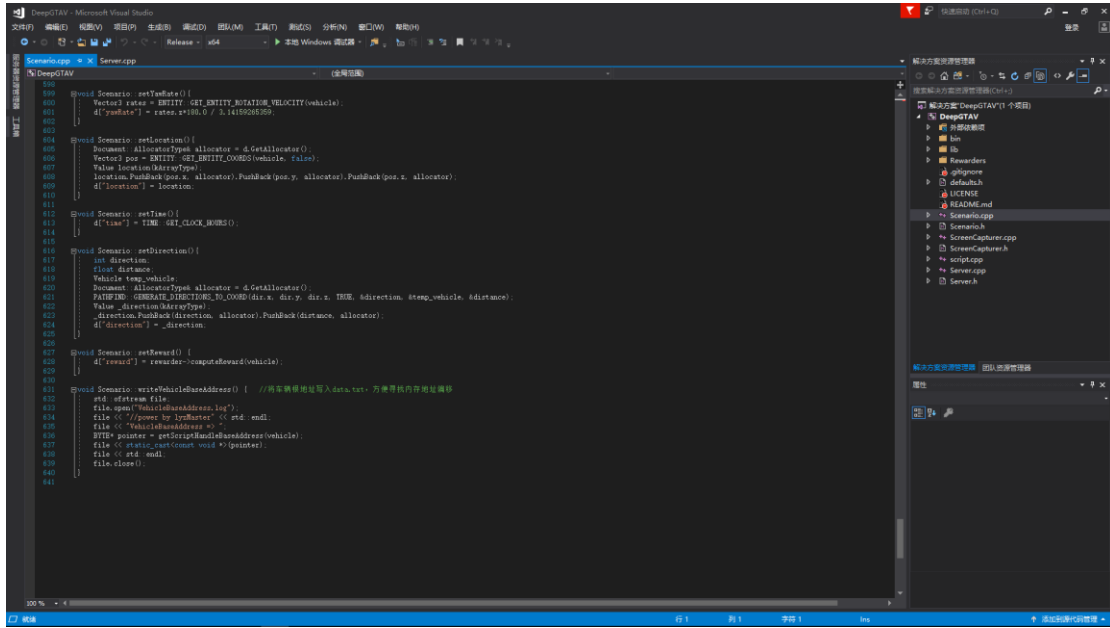


2. 准备部分

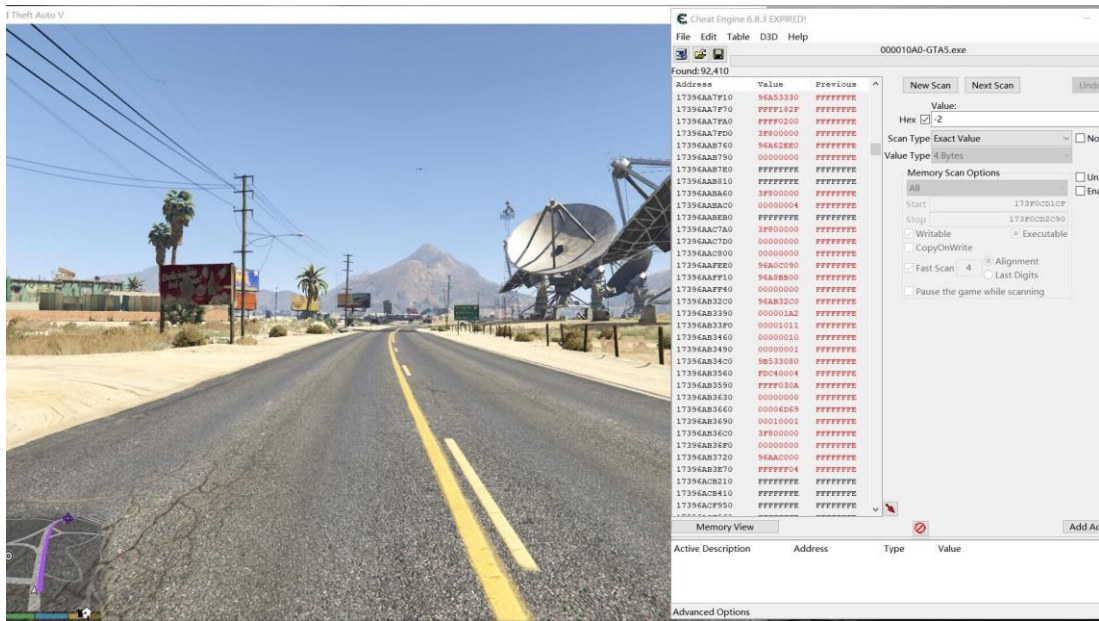
首先在数据采集过程中我们运用到专业插件辅助我们整个采集过程，保证我们的采集数据精准可靠：DeepGTAV v2 Revision 是基于 Grand Thief Auto V(后文简称 GTAV)所衍生出的数据采集插件，这个插件基于开源项目 DeepGTAV，我们对其中参数进行修改，并为了方便调试，加入了将车辆在内存中的首地址存储在 log 中功能。本项目正是运用此插件完成我们的数据采集工作。

1. 前期准备工作过程如下：首先我们所需确认 *steering*, *throttle*, *brake* 在内存中与 vehicle 首地址的偏移量，在此过程中可以使用 *bin/Release* 预编译文件尝试，并使用 python 程序与位于 8000 端口的 DeepGTAV 进行通讯。
1. 在数据采集前我们重点要寻找车辆首地址偏移量，这不仅方便日后的调试同时便于他人的再开发，针对于此我们将提供的预编译文件拷贝到 GTAV 根目录，运行时将对 8000 端口发送 start 指令，此时，我们预编译好的 DeepGTAV 会将采集数据车辆的首地址写入 log 文件（VehicleBaseAddress.log），同时使用 Cheat Engine 从 GTAV 根目录下的 VehicleBaseAddress.log 文件里的地址开始，到此地址后 0xfa0 位，对[-2,2]的 float 值进行搜索，同时进行游戏，观察地址中值的变化情况，预编译文件偏移量为：throttle (0x8A4)，brake (0x8A8)，steering (0x89C)。内存地址的偏移量与模拟器版本有关，若内存偏移量与预编译文件偏移不同可使用 visual studio 打开 DeepGTAV.vcxproj 修改代码并编译。

使用 visual studio 打开项目

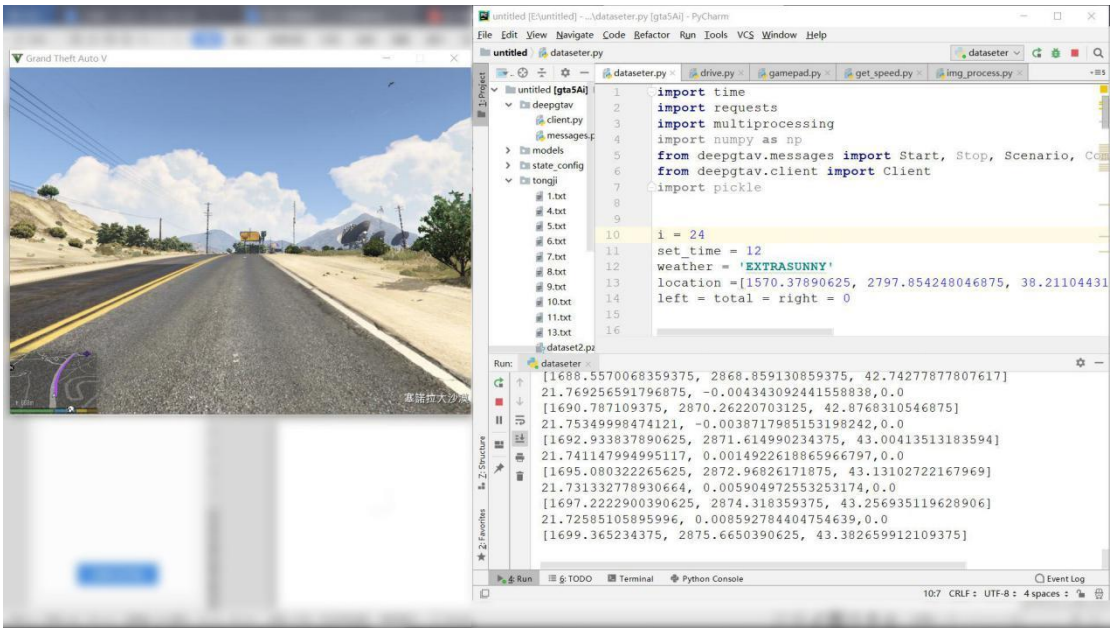


车辆首地址偏移量的具体操作过程



3. 开始采集

在准备工作完成之后，运行 dataset_collection.py 并且运行模拟器，便可以正式采集数据，在采集文件时，我们可以修改四种参数值：i, set-time, weather, location 来分别改变我们的编号，模拟器内开始时间，行驶过程中的天气情况，出发地点，使我们采集的数据更加多样化和多元化，使其更具真实性。



为了能够更好更具效率的控制模拟器，我们用手机登陆网址 (<https://www.pixeldesert.com/compare>)来完成远程控制，远程遥控器由 Bootstrap 前端框架和 mysql+serverlet+tomcat8.0 后台组成。具体的展示情况如下（下图为控制模拟器关闭与开启）：



使用远程遥控器的好处在于可以灵活的控制模拟器，使得模拟器可以在我们想要的时刻开始运行，同时模拟器在运行过程中，我们无法操作其他程序，所以当我们遇到程序错误和意外情况时可以即时的控制模拟器，从而提高我们数据采集的精确度和可靠性。

在采集数据的过程中，为了能够更好的贴近真实情况，我们模拟了不同时段的路况包括高架，城市公路，乡间小道。

数据集截图样例如下：





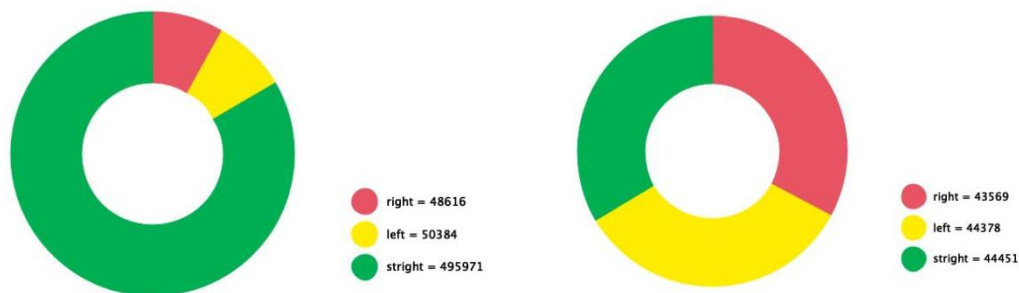


我们自动驾驶系统在上述路段中的表现都较为稳定，其中在乡村路段要比城市路段行驶效果好。夜间的驾驶情况较白天较差。

4. 数据清理

在采集数据的过程中难免会产生错误和冗余数据，我们将对初步数据进行清理和优化。我们清理数据时设定了数个键位来控制模拟器：space 表示向前，delete 表示删除，q 表示退出。

我们总共花费了采集了向右 48616 张，向左 50384 张，向前 495971 张照片，总计 325.37GB，所耗时约一周时间，总时长 21-28 小时。经过清理，筛选出向右 43569 张，向左 44378 张，向前 44451 张，总计 72.39GB,整理耗时约两周，总时长 42-56 小时。数据采集频率为 10hz，即每秒采集 10 张。



初步的采集数据中有大量的直线路程，我们将直线路程照片缩减，使得其与左右转弯照片数量接近，让整个数据集的比例更趋于合理，更能表现纯视觉自动驾驶的优势。

下面为我们所采集数据集中，每组数据的示例：



brake: 0.0

throttle: 0.02060

steering: -0.01557

speed: 22.18734 (mph)



brake: 0.0

throttle: 0.18563

steering: -0.21097

speed: 21.81074 (mph)


为了整理数据集，我们设置了统计文件，统计文件在数据清理后可自动生成，其可以方便我们整理数据集，使得我们的数据集的管理更加具有效率。

五、 核心技术

1. 模型选择

对于图像识别问题，CNN(卷积神经网络)无疑是最好的选择。CNN 彻底改变了传统模式识别。在广泛采用 CNN 之前，大多数模式识别任务使用的是手工提取的特征，然后使用分类器进行分类。CNN 的突破在于，它可以从训练样本中自动学习特征。CNN 在图像识别任务中尤其强大，因为卷积操作捕获图像的 2D 特性。此外，通过使用卷积核扫描整个图像，与 Full-connected 网络相比，需要学习相对较少的参数。

我们的卷积神经网络模型，可以通过一张汽车第一人称视角的图像，来判断油门、刹车以及转向。

$$f(\text{img}, \text{speed}) = \text{throttle}, \text{brake}, \text{steering}$$


2. 模型介绍

i. 转向控制神经网络：

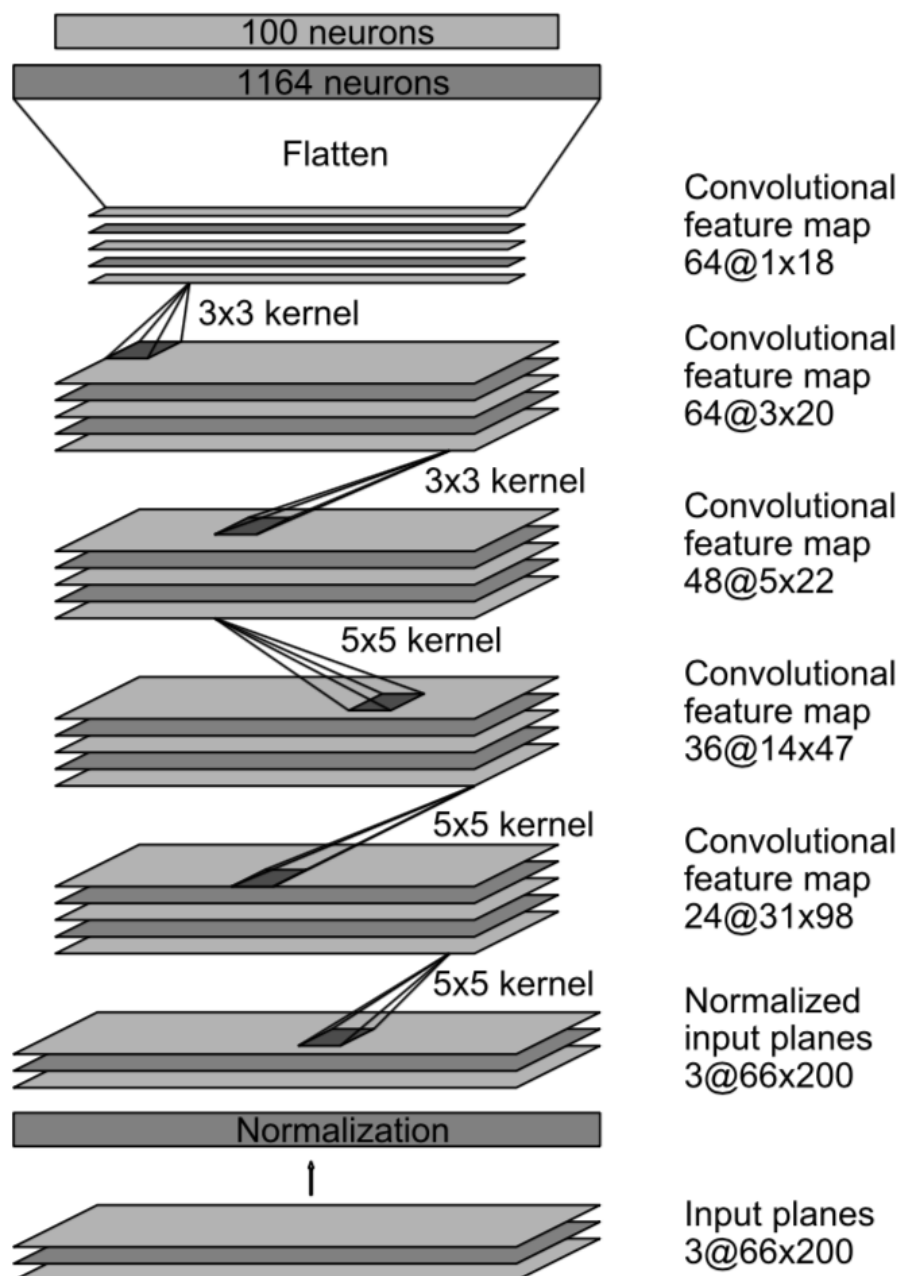
在控制汽车转向的神经网络中，我们将图像分为两个部分，分别为：主画面部分和地图路径部分。主画面部分是将模拟器中的汽车调至第一人称视角所采集的图像，这部分图像在真实的自动驾驶中相当于汽车摄像头采集到的图像；地图路径部分为从模拟器图像中提取出的汽车运行的路径信息，相当于真实自动驾驶的导航地图，告诉车辆应该向什么方向行驶。



图像分划示例

<1>主画面部分神经网络

在主画面部分的卷积神经网络中，我们的输入层输入图像采用了 YUV 色彩空间而不是 RGB 色彩空间，因为 YUV 色彩空间与 RGB 色彩空间相比，有着传输占带宽较小的优点。神经网络的构造和计算流程如下：



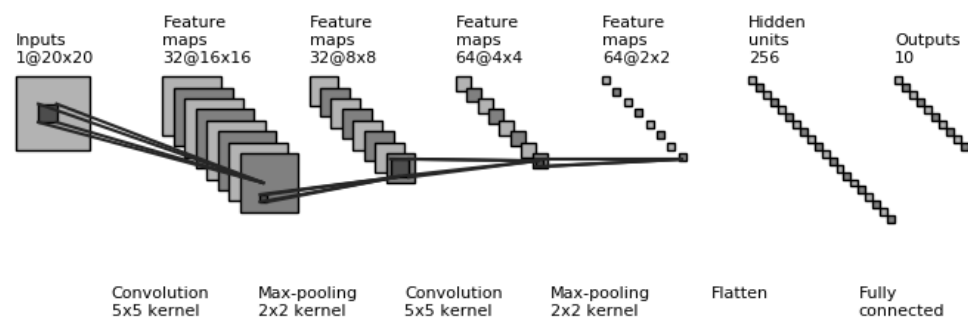
1. 我们将主画面部分的 ROI 提取出来，此时图像为 RGB 色彩空间，接着我们使用 OpenCV 将 RGB 色彩空间转化为 YUV 色彩空间并将主画面 resize 为 66*200 的图像。
2. 再将图像的每个像素点正则化，使得结果的均值为 0，方差为 1。这样做可以加快神经网络拟合速率，提高模型精度，防止梯度弥散或梯度爆炸而造成的神经网络无法训练。
3. 将正则化后维度为 3 维、大小为 66*200 的像素点集与 24 个 5*5 的卷积核进行

- 步长为 2、无填充的卷积运算，得到维度为 24 维、大小为 31*98 的张量。
4. 将上一步得到的维度为 24 维、大小为 31*98 的张量与 36 个 5*5 的卷积核进行步长为 2、无填充的卷积运算，得到维度为 36 维、大小为 14*47 的张量。
 5. 将上一步得到的维度为 36 维、大小为 14*47 的张量与 48 个 5*5 的卷积核进行步长为 2、无填充的卷积运算，得到维度为 48 维、大小为 5*22 的张量。
 6. 将上一步得到的维度为 48 维、大小为 5*22 的张量与 64 个 3*3 的卷积核进行步长为 2、无填充的卷积运算，得到维度为 64 维、大小为 3*20 的张量。
 7. 将上一步得到的维度为 64 维、大小为 3*20 的张量与 64 个 3*3 的卷积核进行步长为 2、无填充的卷积运算，得到维度为 64 维、大小为 1*18 的张量。
 8. 将上一步得到的维度为 64 维、大小为 1*18 的张量拉伸成为一个 1164*1 的向量。
 9. 将上一步得到的 1164*1 的向量与 100*1 的向量进行全连接操作。
 10. 最后得到一个 100*1 的向量。

以上所有层都以 ReLU 作为激活函数，因为训练神经网络时 ReLU 收敛速度会比 sigmoid 或 tanh 快很多。

<2>地图路径部分神经网络

由于在模拟器中，无法像真实世界一样获得详细的地图以及路线信息，所以我们只能通过模拟器左下角的地图来获得路线信息，以保证汽车沿正确的路线行驶。我们搭建的卷积神经网络构造和计算流程如下：

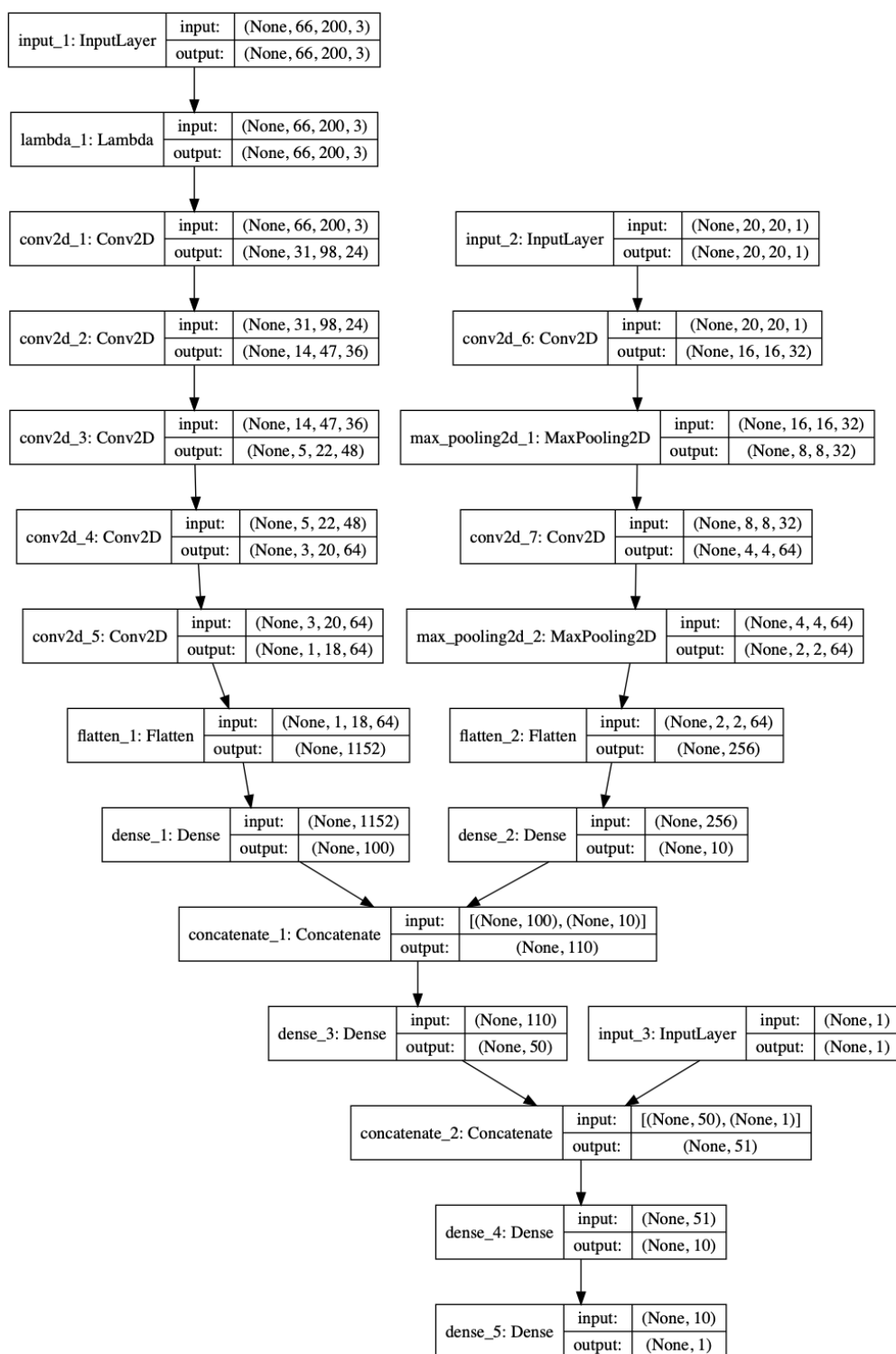


1. 将左下角的地图压缩至 20*20 大小的 RGB 图像，并使用 OpenCV 将图像 变为灰度图，图像由三维的 RGB 图变为了一维的灰度图。
2. 将维度为 1 维、大小为 20*20 的灰度图的像素点集与 32 个 5*5 的卷积核进行步长为 1、无填充的卷积运算，得到维度为 32 维、大小为 16*16 的张量。
3. 将上一步得到的维度为 32 维、大小为 16*16 的张量进行 2*2 的最大池化运算得到维度为 32 维、大小为 8*8 的张量。
4. 将上一步得到的维度为 32 维、大小为 8*8 的张量与 64 个 5*5 的卷积核进行步长为 1、无填充的卷积运算，得到维度为 64 维、大小为 4*4 的张量。
5. 将上一步得到的维度为 64 维、大小为 4*4 的张量进行 2*2 的最大池化运算得到维度为 64 维、大小为 2*2 的张量。
6. 将上一步得到的维度为 64 维、大小为 2*2 的张量拉伸成为一个 256*1 的向量。
7. 将上一步得到的 256*1 的向量与 10*1 的向量进行全连接操作。
8. 最后得到一个 10*1 的向量

以上所有层都以 ReLU 作为激活函数。

<3>转向控制神经网络整体模型：

我们将上述两部分模型融合在了一起，并加入了速度参数进行判断。在模型的输入中加入速度参数意义是，转向时，如果车速很大，那么如果同时方向盘转很大角度，车辆就会翻车，而速度很小时不会出现这样的情况。转向控制的神经网络构造和计算流程如下：



1. 将上述两个神经网络的输出：一个 100×1 的向量、一个 10×1 的向量融合为一个 110×1 的向量。
2. 将上一步得到的 110×1 的向量与 50×1 的向量进行全连接操作。
3. 将速度数值以一个 1×1 的向量输入神经网络。
4. 将 50×1 的向量、 1×1 的向量融合为一个 51×1 的向量。
5. 将上一步得到的 51×1 的向量与 10×1 的向量进行全连接操作。
6. 将上一步得到的 10×1 的向量与 1×1 的向量进行全连接操作，这一层用 sigmoid 函数作为激活函数，因为这一层需要进行 $[-1, 1]$ 的回归预测。
7. 得到的 1×1 的向量便是转角。

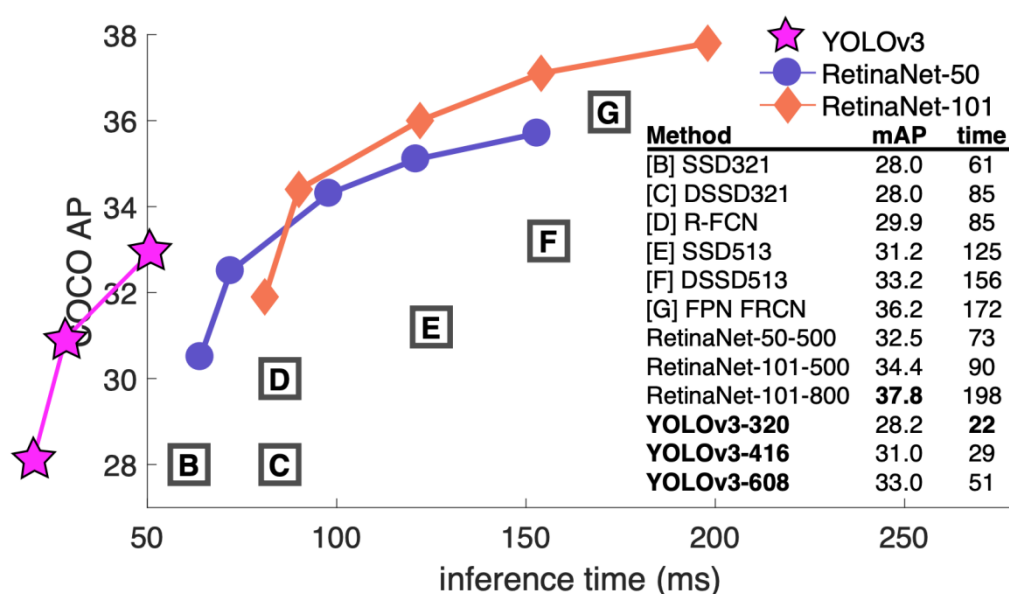
除了第六步以外，以上所有层都以 ReLU 作为激活函数。转角为一个 $[-1, 1]$ 之间的浮点数，负数意味着向左，正数意味着向右。浮点数代表了相对于最大转角转向的百分比，在模拟器中，车辆的向左向右的最大转角为 72 度。

ii. 物体检测神经网络

在自动驾驶中，我们需要检测到周围的车辆，以实现车辆进行闪避。我们对比了 ssd_mobilenet、ssd_resnet、faster_rcnn 以及 YOLO。我们对比了这四种神经网络的预训练模型的性能，包括准确度、速度等指标，最终我们选择了最先进的 YOLO v3 神经网络。

YOLO v3 具有先验检测 (Prior detection) 系统，它将分类器或定位器重新用于执行检测任务。他们将模型应用于图像的多个位置和尺度。而那些评分较高的区域就可以视为检测结果。YOLO v3 将一个单神经网络应用于整张图像，该网络将图像划分为不同的区域，因而预测每一块区域的边界框和概率，这些边界框会通过预测的概率加权。它在测试时会查看整个图像，所以它的预测利用了图像中的全局信息。与需要数千张单一目标图像的 R-CNN 不同，它通过单一网络评估进行预测。所以 YOLO v3 的速度非常快。

YOLO v3 处理一张图像仅需 30ms 左右的时间。这意味着，即使我们的车辆采集到的图像为 30Hz，神经网络也能逐帧处理判断。



YOLO V3 与其他物体检测神经网络的对比

我们使用了 YOLO v3 的预训练模型，并对预训模型进行迁移学习，使得模型更加适合我们的使用场景。

	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1	
	Convolutional	64	3×3	
	Residual			128×128
	Convolutional	128	$3 \times 3 / 2$	64×64
2x	Convolutional	64	1×1	
	Convolutional	128	3×3	
	Residual			64×64
	Convolutional	256	$3 \times 3 / 2$	32×32
8x	Convolutional	128	1×1	
	Convolutional	256	3×3	
	Residual			32×32
	Convolutional	512	$3 \times 3 / 2$	16×16
8x	Convolutional	256	1×1	
	Convolutional	512	3×3	
	Residual			16×16
	Convolutional	1024	$3 \times 3 / 2$	8×8
4x	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

由于我们的系统使用纯图像识别的方式进行自动驾驶，所以相对于目前大多数较为成熟的使用激光雷达检测车辆距离的方案，我们只能通过图像对车辆的距离进行判断。

我们使用检测出的车辆的边界框的相对大小（边界框大小与图像大小的比值）来判断其他车辆与我们车辆之间的相对距离。

```
relative = round(((1 - re_width)**4), 2)
```

其中，**4 是使得宽度变化反映在 relative 上的效果更加强烈。

一旦距离小于阈值，车辆将会减速，以防止与前方车辆相撞。这时图像中的边界框会变成紫色，以示警告。



六、模型训练

我们使用了 TensorFlow 和 Keras 搭建神经网络，使用 tensorboard 可视化我们的训练过程，方便即使调试模型。

我们使用 Adam 优化器，使用平均方差作为损失函数。

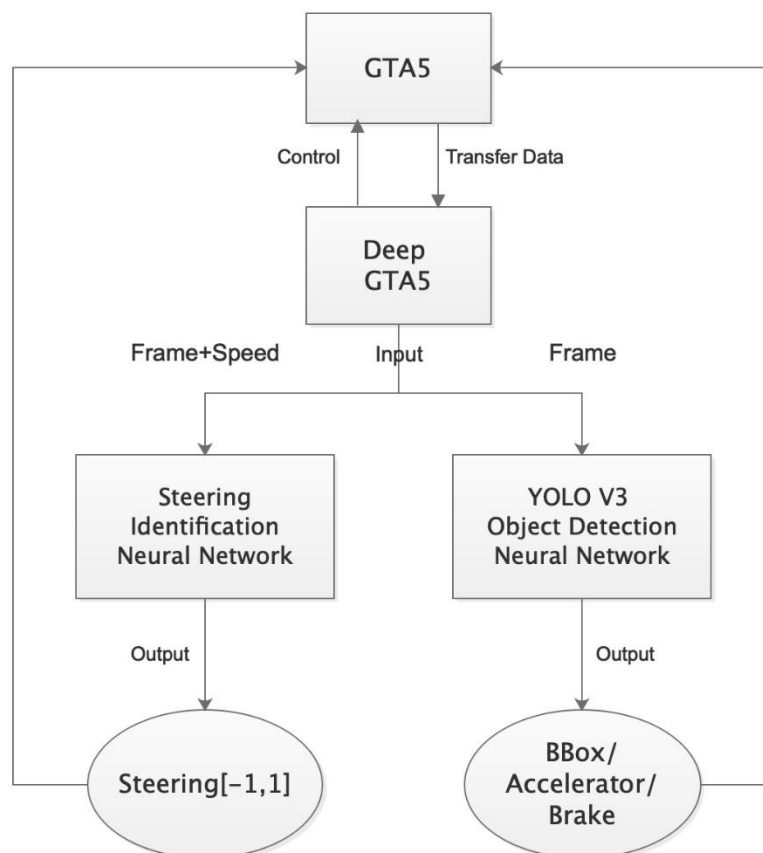
我们使用了上述模型和数据集对所搭建神经网络进行训练。我们将数据集的 80% 作为训练集，20% 作为测试集。

我们编写了 data_loader.py 程序从采集整理好的数据集中读取数据。由于神经网络为多输入结构，我们将 X 数据的每一项先存为 numpy 数组，然后将所有项存为 list，传给 train.py 去训练神经网络。

我们在一块 NVIDIA GTX1070 GPU 上训练了 48 个小时，期间，共训练了 100 个 Epochs。

训练结束后使用测试集进行测试，准确率达到 74.6%。效果较为可观。

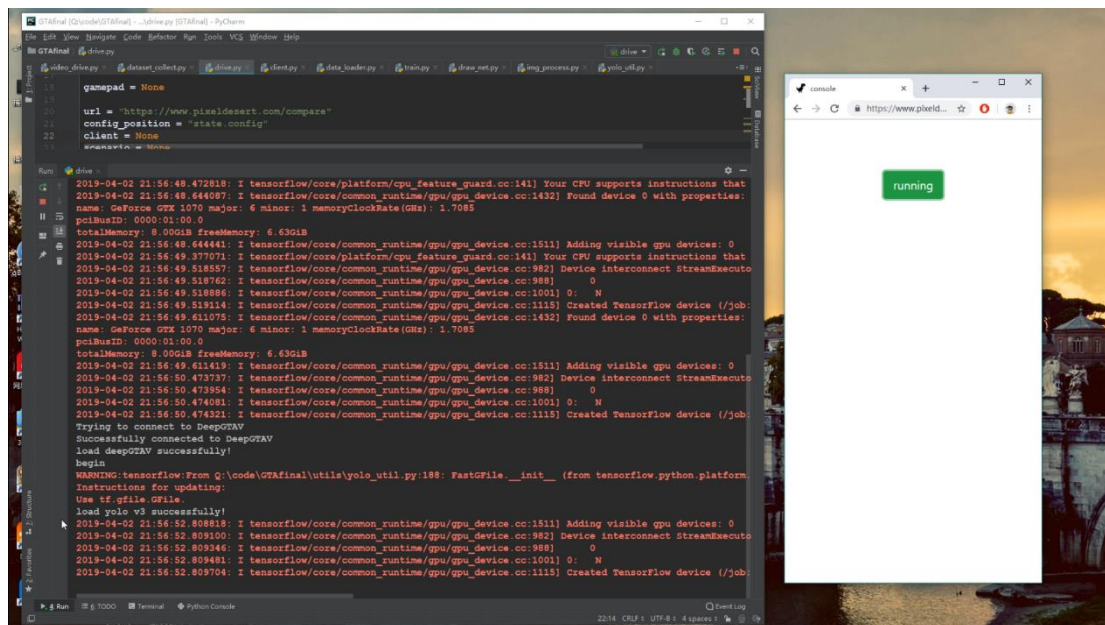
七、模型运行整体运行流程

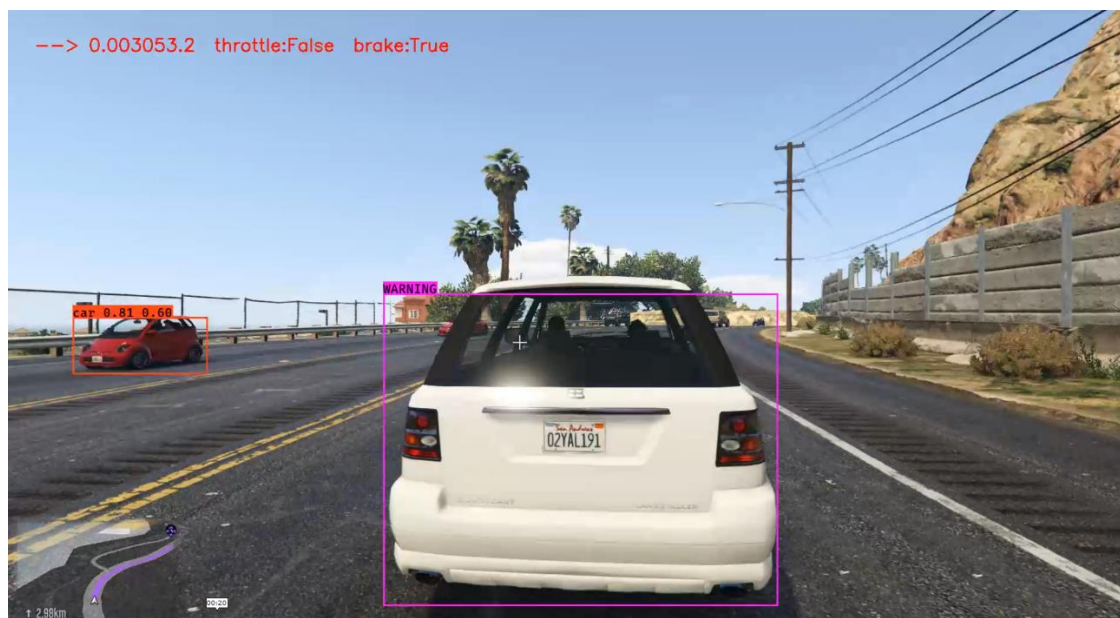


八、项目文件介绍



九、运行效果截图











十、测试项目

1. 使用我们提供的测试数据集（推荐）

正如上文模拟器安装段落所说问题，我们建议使用我们提供的数据集进行项目的测试。运行 **video_drive.py** 文件，会加载模型。等待模型加载完毕：

```
Using TensorFlow backend.
2019-04-03 14:00:44.418218: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2019-04-03 14:00:44.616935: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: GeForce GTX 1070 major: 6 minor: 1 memoryClockRate(GHz): 1.7685
pciBusID: 0000:01:00:0
TotalMemory: 8.00GiB freeMemory: 6.63GiB
2019-04-03 14:00:44.617279: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2019-04-03 14:00:49.362310: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-04-03 14:00:49.362498: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988] 0
2019-04-03 14:00:49.362610: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0: N
2019-04-03 14:00:49.362830: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 6393 MB memory) ->
WARNING:tensorflow:From Q:\code\GTAfinal\utils\yolo_util.py:188: FastGFile._init_ (from tensorflow.python.platform.gfile) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.gfile.GFile.
load yolo v3 successfully!
2019-04-03 14:00:51.627301: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2019-04-03 14:00:51.627514: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-04-03 14:00:51.627712: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988] 0
2019-04-03 14:00:51.627837: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0: N
2019-04-03 14:00:51.628019: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 6393 MB memory) ->
load main_model successfully!
```

加载成功后，会自动打开 openCV 窗口。点击空格计算下一张图像，点击 Q 退出程序。

2. 使用 GTAV 模拟器

将 DeepGTAV-Revision 目录下的 bin/Release 拷贝至 GTAV 根目录。

（若 GTAV 版本不为 1.0.231.0 NON-STEAM 需按照数据采集准备部分说明进行偏移量的查找以及插件的编译）。

运行 **drive.py**，会提示等待指令。

打开远程遥控器界面 (<https://www.pixeldesert.com/console>)。点击开始。

```
waiting for instructions...
waiting for instructions...
waiting for instructions...
waiting for instructions...
waiting for instructions...
waiting for instructions...
$$$$$$$$$$$$$$$$$$$$ after 3s will begin $$$$$$$$$$$$$$$$$$
3
2
1
=====start now!=====
```

程序会自动与 DeepGTAV 通讯。通讯成功会成功。程序此时会自动加载模型。

```
Trying to connect to DeepGTAV  
Successfully connected to DeepGTAV  
load deepGTAV successfully!  
begin
```

加载模型成功后, 切入 GTAV 中即可。

十一、未来展望

1. 项目近期发展方向

目前, 在车辆较多, 路况较为复杂的路况下形式状况不太理想。我们打算结合递归神经网络与卷积神经网络, 将每一帧图像之间产生关系, 而不是每一张图像之间都是一个独立的个体。

2. 最终目标

虽然自动驾驶系统至今已经有数十年的发展历程, 并且已向实用化发展, 但是其大多使用传感器例如雷达, 激光测距来完成行车路程, 一方面大量的设备堆砌虽说能完成自动化驾驶, 但是可靠性和性能优化方面一直以来都是问题, 另一方面大量传感器带来的成本问题也同样严峻, 所以我们所探索的端到端的纯视觉自动驾驶对于上述问题都有良好的解决, 我们希望通过模型在模拟器中的训练, 在提高模型精确度的同时, 将模型逐步迁移至现实世界, 做出真实可用的自动驾驶汽车。

参考文献：

- [1]《End to End learning for self-driving cars》. Mariusz Bojarski
- [2]《Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?》. Matthew Johnson-Roberson
- [3]《BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling》 Fisher Yu
- [4]《YOLOv3: An Incremental Improvement》 Joseph Redmon