**摘　要**

高速公路车辆类型自动检测是智能交通系统中的一个重要组成部分。本文使用当前实时性最佳的机器学习框架Darknet实现了车辆类型的识别系统。该系统基于Darknet框架，使用YOLOv3目标检测算法，实现196种不同厂商及型号的汽车位置检测与分类。该系统不仅可以用于道路关卡的车流量分析，还可以协助人类认识各类汽车。

**关键词**：机器学习，车辆检测，智慧交通，Darknet，YOLO

**ABSTRACT**

The automatic detection of expressway vehicle type is an important part of its. In this paper, we use Darknet, the best real-time machine learning framework, to implement the vehicle type recognition system. This system is based on the framework of Darknet, and uses the algorithm of yolov3 target detection to detect and classify the position of 196 different manufacturers and models. The system can not only be used in traffic flow analysis of road level, but also help people to know all kinds of cars.

**KEY WORDS**：Machine Learning, Vehicle Detection, Intelligent Transportation, Darknet, Yolo

**目 录**

[第一章 绪论 1](#_Toc40809110)

[1.1 课题背景 1](#_Toc40809111)

[1.2 研究与应用现状 1](#_Toc40809112)

[1.3课题的意义和主要内容 3](#_Toc40809113)

[1.4 论文的组织结构 3](#_Toc40809114)

[第二章 机器学习与机器学习框架概述 4](#_Toc40809115)

[2.1 机器学习介绍 4](#_Toc40809116)

[2.1.1 机器学习的概念 4](#_Toc40809117)

[2.1.2 深度学习的介绍 4](#_Toc40809118)

[2.1.3 机器学习的应用与未来 6](#_Toc40809119)

[2.2 TensorFlow介绍 7](#_Toc40809120)

[2.2.1 TensorFlow概述 7](#_Toc40809121)

[2.2.2 TensorFlow的发展 7](#_Toc40809122)

[2.2.3 TensorFlow2.0的优点 7](#_Toc40809123)

[2.3 YOLO与darknet 7](#_Toc40809124)

[2.3.1 YOLO简介 7](#_Toc40809125)

[2.3.2 YOLO原理 7](#_Toc40809126)

[2.3.3 YOLO的优点 14](#_Toc40809127)

[2.3.4 darknet简介 14](#_Toc40809128)

[2.3.5 AlexeyAB/darknet 15](#_Toc40809129)

[2.4 本章小结 16](#_Toc40809130)

[第三章 系统的总体设计和平台搭建 17](#_Toc40809131)

[3.1 系统的总体设计框架 17](#_Toc40809132)

[3.2 车型识别系统的需求分析 18](#_Toc40809133)

[3.2.1 系统定义 18](#_Toc40809134)

[3.2.2 功能需求 18](#_Toc40809135)

[3.2.1 性能需求 19](#_Toc40809136)

[3.3 开发环境平台搭建 20](#_Toc40809137)

[3.3.1 Anaconda环境搭建 20](#_Toc40809138)

[3.3.2 安装TensorFlow-GPU 21](#_Toc40809139)

[3.3.3 安装CUDA和CUDNN 21](#_Toc40809140)

[3.3.4 安装OpenCV 22](#_Toc40809141)

[3.3.4 编译darknet 22](#_Toc40809142)

[3.4 本章小结 26](#_Toc40809143)

[第四章 车型识别系统软件设计 26](#_Toc40809144)

[4.1 VDS1.0-alpha 26](#_Toc40809145)

[4.1.1 简介 26](#_Toc40809146)

[4.1.2 数据预处理 27](#_Toc40809147)

[4.1.3 神经网络结构探索 30](#_Toc40809148)

[4.2 VDS1.0 31](#_Toc40809149)

[4.2.1 简介 31](#_Toc40809150)

[4.2.2 设计思路 31](#_Toc40809151)

[4.2.3 做出的改进 33](#_Toc40809152)

[4.3 VDS2.0 33](#_Toc40809153)

[4.3.1 简介 33](#_Toc40809154)

[4.3.2 设计思路 33](#_Toc40809155)

[4.4 本章小结 39](#_Toc40809156)

[第五章 车型识别系统的验证测试 40](#_Toc40809157)

[5.1 VDS1.0 40](#_Toc40809158)

[5.1.1功能点VDS-1.0-001 40](#_Toc40809159)

[5.1.2功能点VDS-1.0-002 40](#_Toc40809160)

[5.1.3功能点VDS-1.0-003 41](#_Toc40809161)

[5.2 VDS2.0 42](#_Toc40809162)

[5.2.1功能点VDS-1.0-001 42](#_Toc40809163)

[5.2.2功能点VDS-1.0-002 42](#_Toc40809164)

[5.2.3功能点VDS-1.0-003 42](#_Toc40809165)

[5.2.4功能点VDS-2.0-001 42](#_Toc40809166)

[5.2.5功能点VDS-2.0-002 42](#_Toc40809167)

[5.3 本章小结 43](#_Toc40809168)

[结论与展望 46](#_Toc40809169)

[致 谢 47](#_Toc40809170)

[参考文献 48](#_Toc40809171)

[附录一 VDS1.0-alpha源码 48](#_Toc40809172)

[附录二 VDS1.0 源码 61](#_Toc40809173)

[附录三 VDS 2.0源码 66](#_Toc40809174)

# 第一章 绪论

## 1.1 课题背景

随着近年来计算机计算能力的提高与数据量的积累，世界正在从IT时代转变为DT时代。DT，即Data Technology的缩写，即数据技术,是对数据进行存储、清洗、加工、分析、挖掘，从数据中发掘规律的技术。而最近如雨后春笋般涌现的人工智能本质上就是数据技术，而机器学习又是人工智能尤为重要的一个分支，它是人工智能的核心，是使计算机拥有智能的根本途径。而深度学习是机器学习的一个重要组成部分，由于大量数据与高性能计算机这两个条件得以满足，深度学习变得越来越热门，它的应用也越来越广泛。随着卷积神经网络的面世，深度学习被广泛应用于计算机视觉领域。计算机视觉，通俗来讲，就是让计算机像人的眼睛一样具有感知世界的能力。它在人脸识别、行为动作识别、智能机器人、自动驾驶、图片文字提取、自动化农业等领域都有所应用。

## 1.2 研究与应用现状

近年来，深度学习广泛应用于图像分类（ResNet50），目标检测（Mask R-CNN），图像分割（UNET），图像恢复等领域。通俗一点讲，比如我们可以用神经网络对遥感图像进行分类，可以对人脸和车辆进行识别和检测，可以恢复损坏的照片，可以给照片自动着色，甚至直接从影像恢复三维信息。

卷积神经网络（CNN）起源于人类对大脑视觉皮层的研究，从上个世纪80年代开始被用于图像识别。典型的卷积神经网络架构堆叠几个卷积层和激活层然后是一个池化层，接着再堆叠几个卷积层、激活层，然后再一个池化层……以此类推，最后是一个全连接层。这种架构被提出之后，有人相继提出了一些更佳的变体，Top-5错误率从开始的26%降到了3%。经典的CNN架构介绍如表1-1所示：

表1-1 经典CNN架构介绍

|  |  |  |  |
| --- | --- | --- | --- |
|  | 提出时间 | Top-5错误率 | 重点技术 |
| LeNet-5 | 1998 | -- | 使用tanh激活函数；  池化层的每个神经元计算输入值，将结果乘以可学习系数并添加偏执参数；  稀疏连接。 |
| AlexNet | 2012 | 17% | 本地相应归一化（LRN） |
| GoogleNet | 2014 | 5% | 初始化模块 |
| ResNet | 2015 | 3.6% | 快捷链接、残差学习 |

CNN结构越来越复杂，于是谷歌提出了Nasnet来自动用Reinforcement Learning 去search一个优化的结构。Nas是目前CV界一个主流的方向，自动寻找出最好的结构，以及给定参数数量/运算量下最好的结构，是当时图像识别的发展方向。然而，2019年2月，瑞士学者[Kaicheng Yu](https://arxiv.org/search/cs?searchtype=author&query=Yu%2C+K), [Christian Sciuto](https://arxiv.org/search/cs?searchtype=author&query=Sciuto%2C+C), [Martin Jaggi](https://arxiv.org/search/cs?searchtype=author&query=Jaggi%2C+M), [Claudiu Musat](https://arxiv.org/search/cs?searchtype=author&query=Musat%2C+C), [Mathieu Salzmann](https://arxiv.org/search/cs?searchtype=author&query=Salzmann%2C+M)等人发表了一篇题为Evaluating the Search Phase of Neural Architecture Search [1]的论文。论文对Nas和Random方法进行评估得出结论：平均而言，随机策略优于最先进的NAS算法。

为了减少开发人员的工作量，深度学习开发框架开始出现。而众多开发框架中最热门、使用人数最多的当属谷歌的TensorFlow。它拥有详细的说明文档和大量开发者，支持GPU和分布式训练，但是和其他框架（如Keras）相比，它的运算速度差强人意。

我国的旷视公司从2015年开始，研发属于自己的深度学习框架Brain++。对比国外框架它主要有以下优点：

1）动静合一。静态图容易部署，但是它没有办法实时调试，开发人员将浪费大量时间和精力调试；动态图可以即时调试，但是部署起来很困难。而Brain++可以在程序中随时进行动态图与静态图的转换，将二者的优点结合。

2）灵活高效。使用高性能算子和高效的内存优化策略。内置算法选择，智能适配设备。

3）训练推理一体。从训练到推理，无需模型转化，精度损失最小化。

Brain++于2020年3月25日正式宣布开源，但是它目前只能在Linux系统上进行开发，如果使用Window系统的Linux虚拟机，将不支持GPU计算。

机器学习目前仍存在一些问题需要解决。如它现在虽然能对具体的问题予以解决，但是没有人类“举一反三”的能力，也就是说它的普适性差；再比如，对复杂问题难以进行有效的逻辑推理；再如，虽然它可以进行简单自然语言处理。但是无法像人一样理解一整段话，甚至整篇文章的中心思想与深刻含义；再如，它无法像人类一样自动过滤掉噪声干扰，举例来说，即使一张图片打了轻微马赛克，人类仍能够将画面脑补全，从而精确识别，但是机器做不到，机器会把打了马赛克的图片和没打马赛克的图片识别为完全不同的物品；再比如，人类视觉有相对坐标系，把一张图片翻转，平移，镜像处理后，人类仍然能够认出图片中的物体，而目前机器学习不具备这种通过坐标变换进行泛化的能力。

## 1.3课题的意义和主要内容

本系统基于大量的车型数据训练得到车辆分类模型，将摄像头捕捉到的车辆图片输入到模型当中，即可通过模型预测车辆的型号、价格等信息。可应用于智能车辆卡口、拍照识车、二手车交易等。

本系统可分为图像捕捉、图像识别模型训练、车辆价格预测模型训练三部分。图像捕捉意在捕捉静止或运动的图像，将其传给图像识别模型。将其与图像识别模块分开也可以实现文件读取方式传入图像，增加系统灵活性。图像识别模型训练模块作用为训练出识别车辆的厂家与型号的模型。

## 1.4 论文的组织结构

全文的内容做了如下安排：

第一章：绪论。主要介绍了车型识别系统的课题研究背景，国内外研究现状和发展前景，主要的研究内容和论文的组织形式。

第二章：机器学习与机器学习框架概述。主要介绍什么是机器学习，机器学习与深度学习，机器学习的应用与未来，tensorflow的概念与发展，YOLO与darknet的原理与优点。

第三章：主要内容为本系统的设计框架与需求分析以及开发环境的搭建。

第四章：本章详细介绍了本系统三个不同版本的软件设计思路与部分具体实现。

第五章：本章主要内容为系统各模块的调试和测试结果展示。

第六章：本章对全文进行了简要总结，并展望其在未来得到发展方向与愿景

# 第二章 机器学习与机器学习框架概述

本章的主要内容分为机器学习的介绍与TensorFlow的概述两个部分。第一部分介绍了机器学习技术的发展与应用。第二部分描述了TensorFlow的概念和特点，以及目前的应用。

## 2.1 机器学习介绍

### 2.1.1 机器学习的概念

首先给出维基百科的解释： A branch of artificial intelligence in which computer generates rules underlying or based on a raw data that has been fed into it[2]。

机器学习是人工智能的一个分支，其中计算机根据输入的原始数据生成规则。换句话说，计算机从输入的数据中得到可加以应用的一般规律。从实践的意义上来说，机器学习是一种通过利用数据，训练出模型，然后使用模型预测的一种方法。也就是说机器学习不需要制定具体的模型，而是让计算机根据庞大的数据量自己训练模型。

根据训练数据是否含有标签，机器学习可分为监督学习和非监督学习。给机器的训练数据拥有标记或标签的学习方式是监督学习。监督学习主要处理回归问题与分类问题。其主要代表算法有：线性回归、逻辑回归、K邻近算法、决策树、随机森林、SVM支持向量机。而训练数据不含标签的即非监督学习。非监督学习的经典代表是聚类。

### 2.1.2 深度学习的介绍

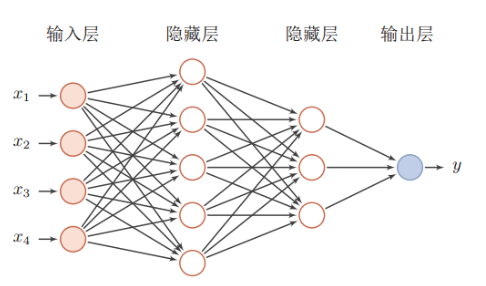
#### 2.1.2.1 神经网络

首先介绍神经网络的基本结构—神经元。在计算机领域，神经元指的是诸如以下样式的数学表达式：

y = f (w0+w1x1+w2x2+…+wnxn)

其中x1~xn为输入，y为输出，w1~wn为线性函数的权重，f为非线性函数，这里称为激活函数。

神经网络就是由诸多此类神经元组成的网络，其结构如图2-1所示：



**图2-1 神经网络的基本机构**

按照神经元接受信号的先后顺序，神经网络分为不同的层，如图2-1中的神经网络有4层，分别是输入层，隐藏层，隐藏层，输出层。这里只是对神经网络的大概结构做一个基本介绍，各层的具体细节不予赘述。

接下来介绍卷积神经网络。卷积神经网络的基本组成结构：

数据输入曾（Input）

卷积层（Conv）

池化层（Pool）

全连接层（FC）

在卷积层和池化层之后通常有一层激活层，以解决线性不可分问题。

卷积神经网络的特别之处是其具有卷积层。卷积层即含有卷积操作的一组神经元。卷积就是利用卷积核，对图像进行逐行（或者跨行，根据参数而定）扫描，最终提取出图像中的特征的操作。至于卷积是为什么能提取出图像特征，本文不予赘述，有兴趣的请参考博客“[为什么CNN能自动提取图像特征](https://www.cnblogs.com/ymd12103410/p/9874878.html)”[3]。

引入池化层的主要目的是保留主要特征，降低下一层参数的数量从而减少运算量。如果输入是图像的话，池化层的作用相当于压缩图像。池化层分为平均池化层和最大池化层两种。下面举例说明这两种池化操作：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 2 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 4 | 1 | 0 |
| 1 | 2 | 4 | 3 |

**图2-2 4\*4像素矩阵**

假设图2-2为一张图片的像素矩阵，过滤器大小为2\*2，步长为2。

对其最大池化：

将2\*2的过滤器以步长为2在矩阵中平移，将过滤器覆盖的四个像素求最大值作为最大池化的结果。如第一步选择图2-3中加黄色底色的四个数字，其最大值为6，第二步选择绿色底色的四个数字，其最大值为8，以此类推。最终结果如图2-4所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 2 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 4 | 1 | 0 |
| 1 | 2 | 4 | 3 |

**图2-3 池化操作示意图**

|  |  |
| --- | --- |
| **6** | **8** |
| **4** | **4** |

**图2-4 池化结果**

从中可以看出进行池化后，图片像素由4\*4被压缩到2\*2，这正是池化层能够减少运算量的原因。平均池化与最大池化类似，只不过是由取最大值改为取平均值。

全连接的含义为两层之间所有神经元都有权重连接，也就是跟传统的神经网络神经元的连接方式是一样的。

#### 2.1.2.2 深度学习

深度学习是采用神经网络的机器学习框架的机器学习。也就是说，深度学习是以神经网络为学习策略的机器学习，这意味着深度学习是机器学习的一个组成部分。

通常来讲，训练数据足够多的前提下，隐藏层的深度越深、神经元的数量越多，模型就能发现数据中更复杂的规律。正因如此，深度学习的感知能力远强于传统的机器学习方法。有意思的是，即便一个深度学习模型已经训练得非常完善，可以非常好地拟合数据，但设计该模型的工程师也很难说出为什么这些神经元要以这样的结构排列，某一层或者某个神经元对结果有什么影响。

### 2.1.3 机器学习的应用与未来

首先，在应用方面，随着数字化的普及，机器学习在将来可以应用到教育领域。如今，绝大多数高校的学生的学习和生活，都留下了“数字轨迹“。学生的考试成绩、图书馆的借阅记录、学生上课的点名打卡记录、学生一卡通的消费记录、学校Wifi基站的连接记录、学生支付宝和微信的消费记录……无一不是数字轨迹的体现。而机器学习完全可以依赖这些轨迹数据预测学生的考试成绩、分析学生的贫困程度（进行贫困生认定），从而督促学生学习，优化课程安排，做到因材施教。此外，还可以进行智能化助学金、奖学金评定 。

再者，在基础理论方面，正如前文提到的，目前深度学习还处于“黑匣子“状态。这让人不免有些担心，因为我们不知道计算机到底学到了什么，这会不会产生对人类有危害的智慧也是个未知数。在未来，可解释的机器学习模型可能会被提出。正如牛顿所说，真理永远是简单的。目前的机器学习模型过于复杂，复杂到令人难以解释。未来机器学习模型会与物理学、数学上的公式一样，简单而富有美感。

## 2.2 TensorFlow介绍

### 2.2.1 TensorFlow概述

TensorFlow是谷歌于2015年11月9日正式开源的计算框架。TensorFlow计算框架可以很好地支持深度学习的各种算法，但它的应用也不限于深度学习。

### 2.2.2 TensorFlow的发展

TensorFlow的前身是出现于2011年谷歌的DistBelief。 2015年，谷歌针对DistBelief进行改进，并改名为TensorFlow，版本号为0.8。来年，也就是2016年，谷歌发布了0.9版本，该版本添加了iOS系统的支持。2017年，谷歌先后公布了 TensorFlow 1.0.0-alpha、TensorFlow 1.0.0-rc0。2019年，谷歌发布了支持Keras接口的TensorFlow2.0，简化了API，大大减小了入门难度。

### 2.2.3 TensorFlow2.0的优点

相比于Tensorflow1.0，它具有以下优点：

1. 调试简单
2. API相对清晰
3. 易上手

## 2.3 YOLO与darknet

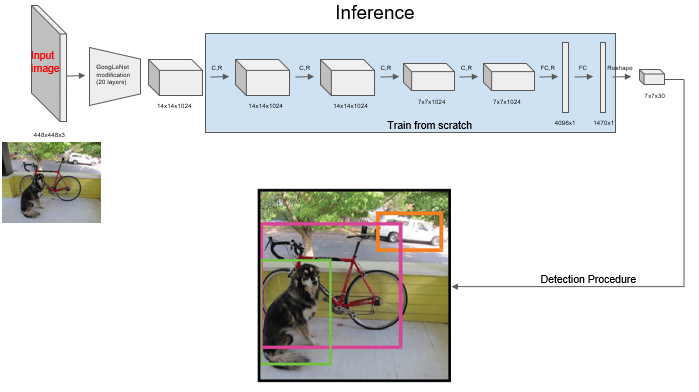
### 2.3.1 YOLO简介

YOLO[4]，被解释为You Only Look Once: Unified, Real-Time Object Detection。它是一种一体化的实时目标检测与分类算法模型。它只通过一次卷积就能实现端到端的预测，而它的运算速度，是保证实时性的关键。

### 2.3.2 YOLO原理

#### 2.3.2.1 YOLO v1

至今，YOLO已经发展到YOLO v4。为了更清晰的弄清YOLO的原理，我们先看YOLO v1模型是如何工作的。如图2-5所示，YOLO一次性读入整张图片，经过一个20层的GoogleNet（当然，这里采用的是简化版的GoogleNet），然后经过四个带RELU激活函数的卷积层，接着是两个全连接层，最后将维度reshape为7\*7\*30。



**图2-5YOLO v1模型结构**

**对**模型有了大致了解之后，我们进一步探索它的原理。如图2-6所示，输入图片被分割为S=7的S\*S窗格单元（grid cell），每个窗格单元关联两个大小形状不同的边界框（Bounding Box），每个边界框有五个参数，分别是中心点的坐标x,y，边框的宽w和高h,以及置信度(confidence)，每个窗格单元有一组条件类别概率，其数量等于类别数C。接下来的内容可能较难理解：当窗格单元中有物体时，窗格可被编码为一个

S\*S(B\*5+C) （1）

S\*S : 窗格单元的数目

B : 每个窗格单元负责的边界框的数量

C : 分类数

的张量。之所以B乘以5是因为每个边界框有5个参数，它包含了物体的位置信息。而公式中的C包含了物体的分类信息。在形式上，我们将置信度confidence定义为

​  （2）

要理解公式（2），首先要解释一下什么是IOU。IOU的定义如图2-7所示 ，即真实结果与预测结果的交集与并集之比。

如果该网格中不存在目标的中心，则置信度分数应是零(因为Pr(Object)=0)。否则，我们希望置信度分数等于预测框与真实值之间联合部分的交集（IOU）（即Pr(Object)应趋向1，因而Confidence=IoU）。

综上所述，YOLO基于GoogleNet架构，一次性读入整张图片，利用窗格单元与边界框一次性完成位置和分类特征的提取。

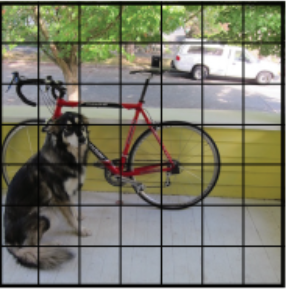
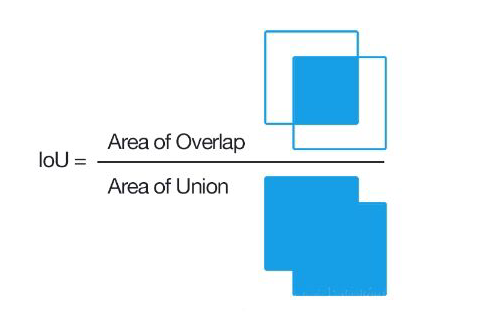
 

图2-6窗格单元（左）和边界框（右）

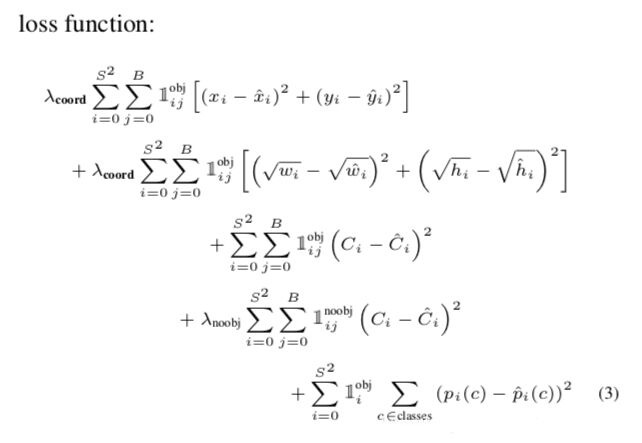


**图2-7 IOU图解**

众所周知，机器学习算法都要定义损失函数，也就是常说的loss，YOLO网络同时要考虑位置的预测和类别的预测，那么，它的loss该如何定义呢？

Loss = λcoord ×坐标预测误差 + （含object的box confidence预测误差 + λnoobj ×不含object的box confidence预测误差） + 类别预测误差

详见公式（3）。



公式（3）中，前两行代表位置误差，其中第一行是针对中心点坐标，第二行是针对宽高；三四两行为IOU误差其中第三行为正类误差，第四行为反类误差；最后一行为分类误差。

最后介绍类别置信度的概念。类别置信度 = 条件类别概率 \* 置信度。数学公式如下：

（4）

即每个网格预测的分类信息乘以每个边界框预测的置信度信息。这得到了每个边界框某个类别的置信度分数。类别置信度包含了某类在框中出现的概率以及边界框与物体的拟合程度。通俗来讲，就是某一类别c完美落在边界框b中的概率，或者说该边界框b中的物体完全是某类c的概率。

虽然每个格子可以预测 B 个边界框，但是最终只选择 IOU 最高的边界框作为物体检测输出，即每个格子最多只预测出一个物体。当物体占画面比例较小，如图像中包含畜群或鸟群时，每个格子包含多个物体，但却只能检测出其中一个。这是 YOLO 方法的一个缺陷。

得到每个窗格单元的置信度向量之后，我们以对图2-2中的狗的预测为例，分析预测的原理。第一步，我们设置一个类别置信度阈值thresh，将所有类别置信度小于thresh的向量类别置信度设为0，以减小低概率类别对算力的浪费。第二步，按照类别狗的类别置信度，对置信度向量进行从大到小的排序。第三步，对该类进行极大值抑制（NMS），防止同一物体被多次框出。在这里解释一下NMS：首先从所有的预测框中找到置信度最大的那个边界框（已经排好序放到最前面了），然后挨个计算其与剩余边界框的IOU，如果IOU值大于一定阈值（IOU过高意味着重合度过高），那么就将该类别置信度值置为0，把该bbox剔除；然后对剩余的预测框重复上述过程，直到处理完所有的检测框。最后，所有边界框的置信度向量中都会有一个分类置信度最大值（边界框中无物体的分类置信度最大值为零），这个值对应的分类即该边界框中物体的类。

总结一下YOLO v1的特点：

1）对靠的近的物体以及体积小的物体检测效果不好；

2）损失函数受定位误差影响较大；

3）模型简单，速度快；

4）端到端训练。

#### 2.3.2.2 YOLO v2

YOLO v2在YOLO v1的基础上做出了一些改进。

第一，它在每一个卷积层之前进行批量归一化操作，提高了网络的泛化能力与收敛速度，而且模型中可以去掉Dropout层而不出现过拟合。

第二，目前业界标准，卷积神经网络都要在ImageNet上进行预训练，使得卷积网络对物体更敏感。在预训练的基础上对网络进行改进与进一步训练。从 Alexnet 开始，大多数的分类器都运行在小于 256×256 的图片上。而YOLO v2将图片分辨率从224\*224提高到448\*448 ，这远远高于ImageNet的预训练分辨率。

第三，YOLO v1包含有全连接层，从而能直接预测 Bounding Boxes 的坐标值。 而Faster R-CNN 只用卷积层与 Region Proposal Network 来预测 Anchor Box 偏移值offset与置信度（前景得分），而不是直接预测坐标值。作者发现通过预测偏移量的方式而不是坐标值能够简化问题，让神经网络学习起来更加容易。所以 YOLO v2 去掉了全连接层，使用 Anchor Boxes 来预测 Bounding Boxes。作者去掉了网络中一个池化层和一个全连接层，这让卷积层的输出能有更高的分辨率，所以将原来448\*448的分辨率减小到416\*416，以加快模型的速度。由于图片中的物体都倾向于出现在图片的中心位置，特别是那种比较大的物体，所以有一个单独位于物体中心的位置用于预测这些物体。使用 Anchor Box 会让精确度稍微下降，但用了它能让 YOLOv2 能预测出大于一千个框，同时 召回率达到88%，mAP(mean Average Precision) 达到 69.2%。

第四，作者在训练集的边界框上进行了K-means聚类，以对anchor进行更好的初始化。从表2-1中可以看出，使用5个K-mean聚类选择的anchor的平均IOU比使用9个原始anchor还要高出0.9%，而对9个anchor使用K-mean的话，平均IOU明显提高，高达67.2%。

表2-1 使用K-mean与原始anchor的比较

|  |  |  |
| --- | --- | --- |
| Box Generation | # | AVG IOU |
| Cluster SSE | 5 | 58.7 |
| Cluster IOU | 5 | 61.0 |
| Anchor Box | 9 | 60.9 |
| Cluster IOU | 9 | 67.3 |

第五，YOLO v2在26\*26的特征层上新增了一层passthrough层（又叫reorg layer），该层结合了GoogleNet初始化层将不同层水平叠加在一起的思想和ResNet的identity mapping堆叠高分辨率特征与低分辨率特征的方式，将26\*26\*512的特征层reshape为13\*13\*2048，这样一来，它就可以与最后一层13\*13\*1024的特征层叠加在一起，生成一个13\*13\*3072的特征图，该特征图同时包含高分辨率和低分辨率的特征，将预测精度提高了1%。

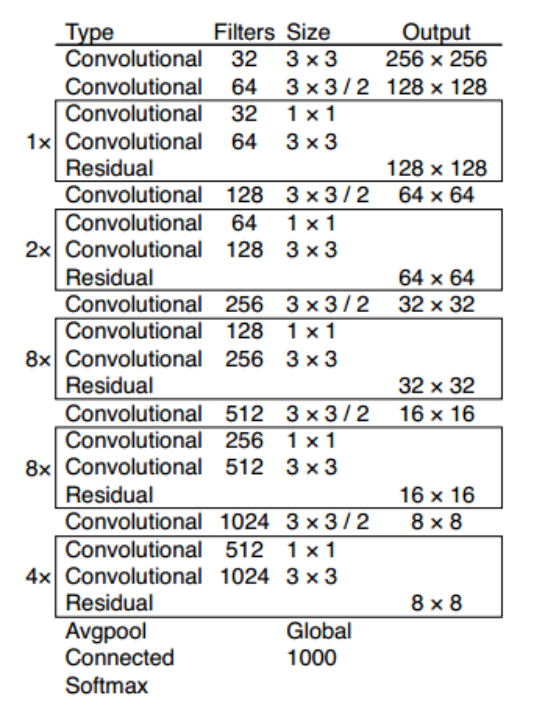
第六，和YOLO v1不同的是，YOLO v2每迭代几次就会修改网络结构，修改最后的检测层。每10个批次，网络会随机选择一个新的分辨率。需要注意的是，图片分辨率需要是32的整数倍（分辨率也不能过大，最大为608\*608）。当分辨率高时，预测的精度也会随之提高，但是速度会有所减缓，相反，当分辨率低时，精度会随之降低，但是速度会相应提高。表2-2是YOLO不同分辨率的网络和其他框架在voc2007/2012 数据集下精度mAP与速度FPS之间的关系对照表，你可以根据表2-2在精度与速度之间权衡。

表2-2 不同框架mAP与FPS对照表

|  |  |  |  |
| --- | --- | --- | --- |
| Dection Framework | Train | mAP | FPS |
| Fast R-CNN[5] | 2007+2012 | 70.0 | 0.5 |
| Faster R-CNN VGG-16[6] | 2007+2012 | 73.2 | 7 |
| Faster R-CNN ResNet [6] | 2007+2012 | 76.4 | 5 |
| YOLO [4] | 2007+2012 | 63.4 | 45 |
| SSD300[7] | 2007+2012 | 74.3 | 46 |
| SSD500[7] | 2007+2012 | 76.8 | 19 |
| YOLOv2 288\*288 | 2007+2012 | 69.0 | 91 |
| YOLOv2 352\*352 | 2007+2012 | 73.7 | 81 |
| YOLOv2 416\*416 | 2007+2012 | 76.8 | 67 |
| YOLOv2 480\*480 | 2007+2012 | 77.8 | 59 |
| YOLOv2 544\*544 | 2007+2012 | 78.6 | 40 |

#### 2.3.2.3 YOLO v3

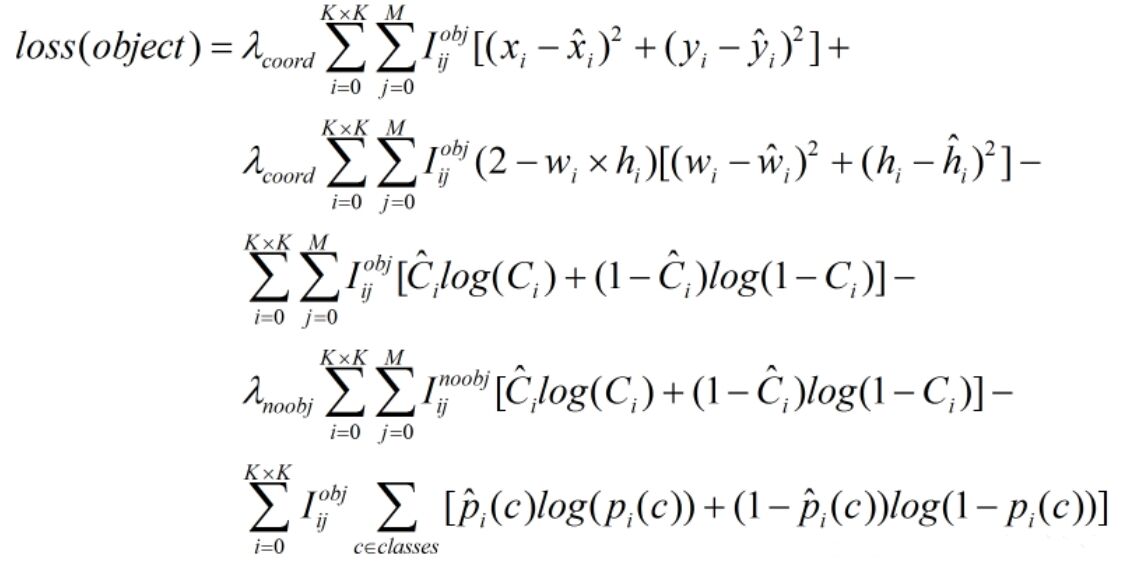
YOLO v3的网络结构变得更深，它由原来的19层网络一下子跃迁到53层，其结构如图2-8所示。从图中可以发现，网络结构中取消了池化层，而是以步长为2的卷积层代替。5个这样的卷积层实现对输入分辨率缩小32倍的效果。当然，过深的网络深度必然会导致模型难以收敛。对此YOLO v3使用了残差结构和类FPN结构进行优化。



**图2-8 YOLO v3采用的网络结构**

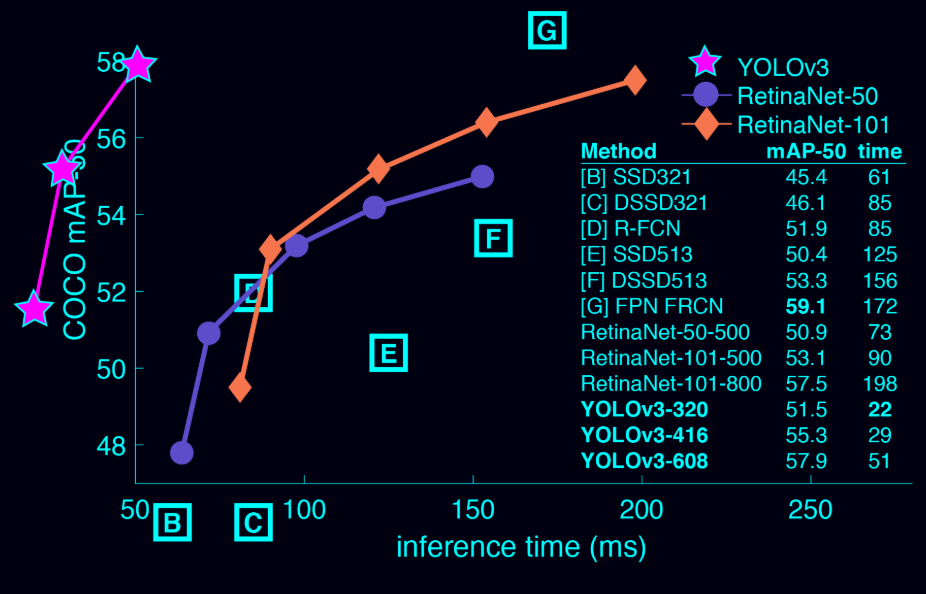
除了anchor和边界框数目的增多以外，YOLO v3最主要的改进是损失函数的变化，新的损失函数见公式（4）。YOLO v3中不再使用MSE作为分类损失的计算而是改用交叉熵损失函数，这是因为YOLO v3为了支持多标签多分类，取消使用softmax而是使用logstic预测每个边界框的分数，如果先验边界框与真实框的重叠度比之前的任何其他边界框都要好，则该值应该为1， 如果先验边界框不是最好的，但确实与真实对象的重叠超过某个阈值(这里是0.5)，那么就忽略这次预测。YOLO v3只为每个真实对象分配一个边界框，如果先验边界框与真实对象不吻合，则不会产生坐标或类别预测损失，只会产生物体预测损失，也就是说，YOLO v3不再需要对每一个边界框计算分类置信度，而是只对存在目标可能性最大的那个边界框进行计算，如此一来，可以去掉不必要anchor，可以减少计算量。而YOLO v3之所以要进行多标签预测，是为了弥补二代版本只能对相互独立的类进行区分（例如区分狗和汽车）而无法同时检测出相互包含的类（如狗和哈士奇）的缺陷。也就是说，无法给目标同时预测狗和哈士奇两个标签，而实际上目标可能同时属于狗和哈士奇两个类。

（4）



### 2.3.3 YOLO的优点

图2-9为YOLO的官网给出的数据。除了图2-9之外，YOLO官方还给出了其他对照表格以彰显其性能的优越，但是它们都没有图2-9直观，所以在这里就不再引用那些对比表格。从图2-9中可以看出YOLO v3无论是在精度还是速度上，都远远超过SSD、RetinaNet等其他框架。这也是YOLO v3最大的优点和实时性准确性的保证



**图2-9 YOLOv3与其他网络的对比**

### 2.3.4 darknet简介

Darknet的作者对darknet给出的定义是：

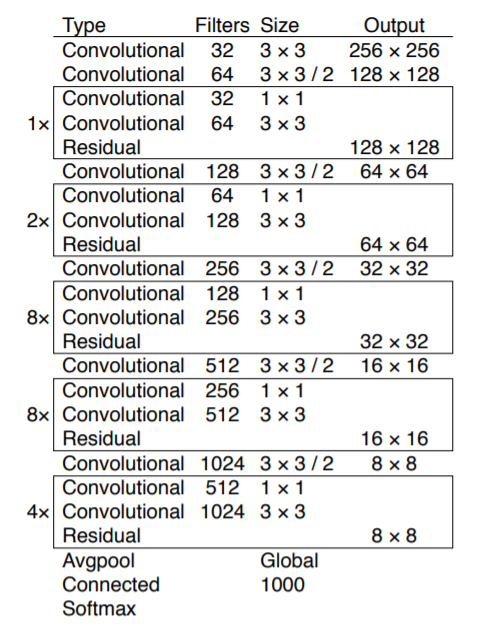
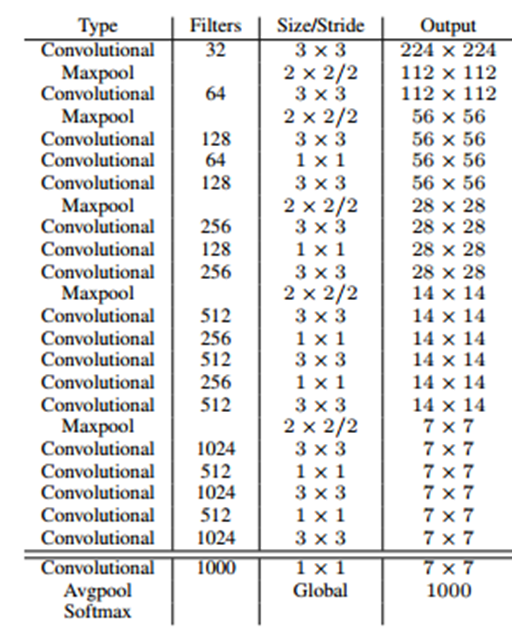
Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.[4]

Darknet是一个较为轻型的完全基于C与CUDA的开源深度学习框架，其主要特点就是容易安装，没有任何依赖项（OpenCV都可以不用），移植性非常好，支持CPU与GPU两种计算方式。

和Tensorflow、Keras等主流深度学习框架不同，darknet是一个非常小众化的框架，甚至没有社区，完全靠作者维护。这也导致了它的功能没有主流框架那么多。但是，正如老子在《道德经》中所说“有无相生，难易相成，长短相形，高下相倾，音声相和，前后相随”，因为它的小众化，使得它的源码结构明晰，接口友好。它完全由C语言编写，没有外部依赖项，作者甚至连OpenCV都设计了可替代的函数。不仅如此，它易于安装，配置简单，移植性极高，还有清晰的python接口。

图2-10是darknet框架的基本结构。Darknet采用3×3的卷积，像Network-in-Network NIN一样，采用全局平均池化，在3×3卷积直接用1×1的过滤器压缩特征表达。用了BN加速收敛，稳定训练，正则化模型。因为有19个卷积层和5个最大池化，所以叫Darknet-19。处理一张图片只需要55.8亿次运算，这个网络包含19个卷积层和5个max pooling层，而在YOLO v1中采用的GooleNet，包含24个卷积层和2个全连接层，因此Darknet-19整体上卷积操作比YOLO v1中用的GoogleNet要少，这是计算量减少的关键。最后用average pooling层代替全连接层进行预测。这个网络在ImageNet上取得了top-5的91.2%的准确率。

Darknet-53的基本结构在前文已有介绍，这里再简单重申一下。它没有池化层，而是用步长为二的卷积层代替。算上这些卷积层，它共有53个卷积层，所以命名为Darknet-53。



**图2-10 darknet19（左）与darknet53（右）**

### 2.3.5 AlexeyAB/darknet

AlexeyAB版本的Darknet是在官方Darknet基础上进行了一些修改，新增对Windows系统的支持，添加了一些新算法，新特性，新Backbone（神经网络模型），是最流行的目标检测开源项目之一。

AlexeyAB版本的改进：

添加了训练循环模型，以精确检测视频功能；

添加了数据扩充；

增加了使用CPU-RAM进行GPU处理训练的能力，以增加mini\_batch\_size和准确性；

通过将2层融合到1个卷积+批量归一化层中，将神经网络性能提高了**7％;**

定义CUDNN\_HALF后，GTXGetForce上的检测效率将提高两倍；

提高了高分辨率视频中的检测效率；

使用数据增强进行训练，将性能提高3.5倍，并且解决了多GPU环境下的瓶颈；

增加了训练锚的计算；

增加了Windows上的支持。

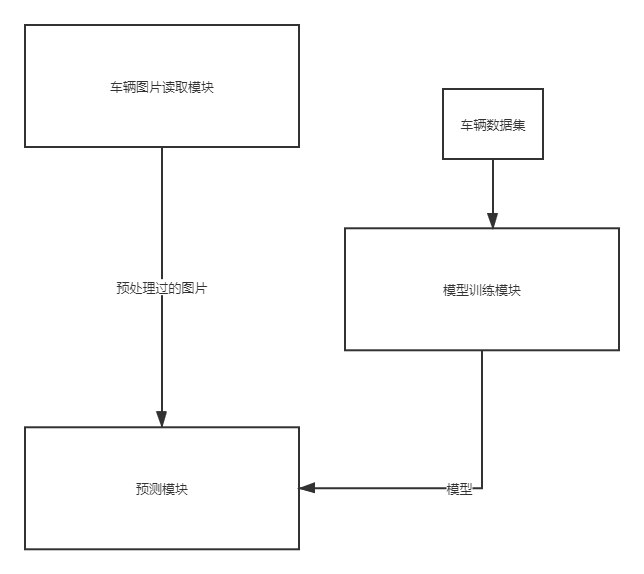
## 2.4 本章小结

本章内容分三节，2.1节主要内容为机器学习的介绍，其中2.1.1节简单介绍了机器学习的概念，机器学习的分类；2.1.2节先是较为详细介绍了神经网络与卷积神经网络，接着引出深度学习的概念；2.1.3节讲述了机器学习的未来发展与展望。2.2节主要内容为TensorFlow的简介与发展。2.3节介绍了当前效果最好的图像检测模型YOLO的发展与原理以及Darknet小众深度学习框架。

# 第三章 系统的总体设计和平台搭建

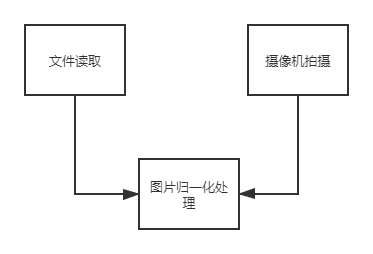
## 3.1 系统的总体设计框架

如图3-1所示，基于机器学习的车型识别系统（以下简称VDS）由三大模块组成：车辆图片读取模块，模型训练模块，预测模块。



**图 3-1 VDS的整体框架**

车辆图片读取模块由图片读取，图片预处理两个子模块构成，图片读取子模块又根据读取方式分为文件读取和摄像头捕捉两种，其结构如图3-2所示。



**图3-2 车辆图片读取模块**

在1.0版本中，模型采用Tensorflow框架，运行于jupyter环境。在2.0版本中，模型使用darknet框架，使用VS2015进行编译，运行于命令行。

最后的预测模块接收图片读取模块的输出调用训练出的模型，得到车辆的196个型号分类。

## 3.2 车型识别系统的需求分析

### 3.2.1 系统定义

基于机器学习的车型识别系统（以下简称车型识别系统）是用于道路上静止或行驶中车辆分类的智能系统，它旨在帮助用户实施捕捉、分类、统计不同车型的车辆。从用户的角度来说，通过该系统可以获得各时段道路上某车型的流量、流量最大的车型等统计数据用以数据分析。

### 3.2.2 功能需求

|  |  |
| --- | --- |
| **需求名称** | 摄像头调用 |
| **需求编号** | VDS-2.0-001 |
| **优先级** | 3 |
| **功能描述** | 1. 适配不同机型摄像头 2. 适配外置摄像头 |
| **约束条件** |  |
| **备注** |  |

|  |  |
| --- | --- |
| **需求名称** | 车辆捕捉 |
| **需求编号** | VDS-2.0-002 |
| **优先级** | 3 |
| **功能描述** | 可捕捉静止的和运动中的车辆  可捕捉遮挡成都不超过一半的车辆  可于阴天、夜晚、雨天、晴天等不同环境下工作 |
| **约束条件** |  |
| **备注** |  |

|  |  |
| --- | --- |
| **需求名称** | 图片数据规范加工处理 |
| **需求编号** | VDS-1.0-001 |
| **优先级** | 2 |
| **功能描述** | 对图片进行大小、格式统一处理，，方便后续使用 |
| **约束条件** |  |
| **备注** |  |

|  |  |
| --- | --- |
| **需求名称** | 训练模型 |
| **需求编号** | VDS-1.0-002 |
| **优先级** | 1 |
| **功能描述** | 使用图片进行训练，得到精度达标的型 |
| **约束条件** |  |
| **备注** | 多分类模型；  精度不小于0.80。 |

|  |  |
| --- | --- |
| **需求名称** | 预测模块 |
| **需求编号** | VDS-1.0-003 |
| **优先级** | 1 |
| **功能描述** | 将处理好的图片传入该模块、得到汽车的车型信息 |
| **约束条件** |  |
| **备注** | 信息实时显示在摄像头影像中 |

### 3.2.1 性能需求

|  |  |
| --- | --- |
| **需求名称** | 性能要求 |
| **需求编号** | VDS-1.0-PR-001 |
| **优先级** | 1 |
| **需求描述** | 为保证实时性要求，预测结果应在0.5秒钟之内得出 |
| **约束条件** |  |
| **备注** |  |

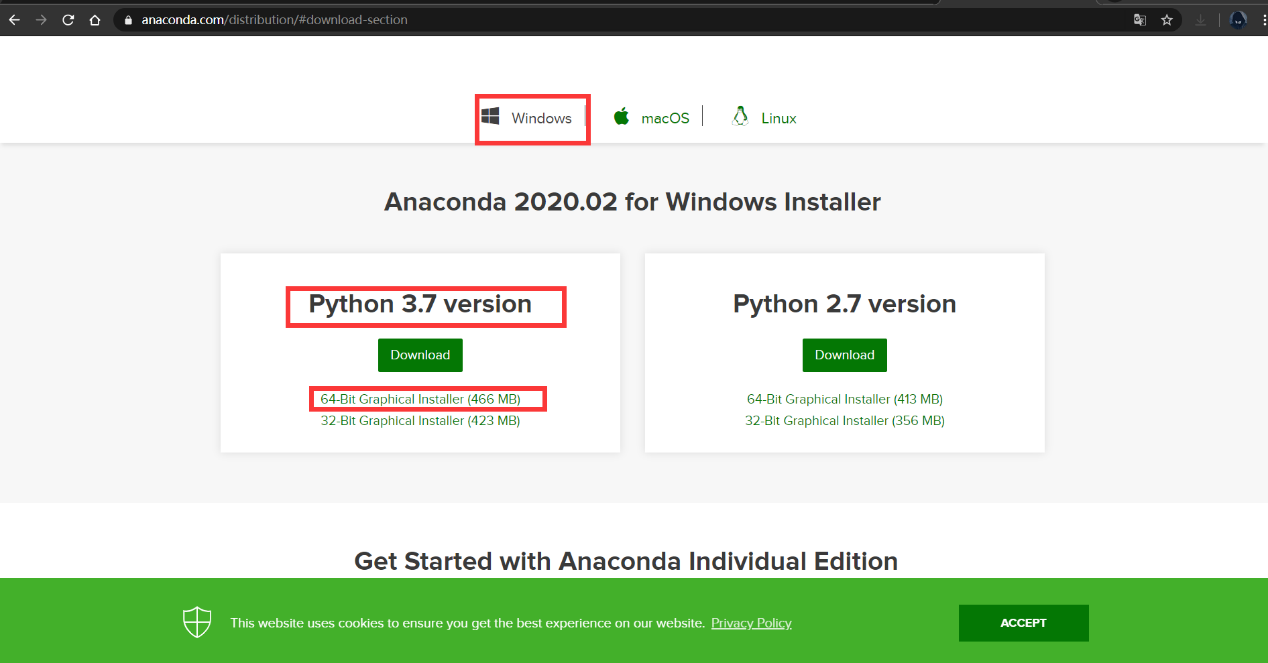
## 3.3 开发环境平台搭建

本系统使用Python开发语言。各类开发工具版本如下：

|  |  |
| --- | --- |
| 开发工具 | 版本号 |
| Python | 3.7.4 |
| TensorFlow | 2.0.0 |
| CUDA | 10.0.130 |
| CUDNN | 7.4 |
| OPENCV | 3.4 |
| VS | 2015 |

### 3.3.1 Anaconda环境搭建

1）从官方下载安装包。根据操作系统、Python版本和机器位数选怎合适的安装包。如果选用了错误的安装包，可能会埋下隐患。这里我们选择Windows、x64位系统、Python3.7版本，不建议选择更高的Python版本，因为高版本的Python很可能没有对应的显卡驱动与之匹配。



**图3-4 下载Anaconda安装包**

2） 根据向导进行安装。如果不熟悉安装过程，所有设置均可采用默认。因为在软件行业，软件系统测试有一关就是保证默认设置可用。

### 3.3.2 安装TensorFlow-GPU

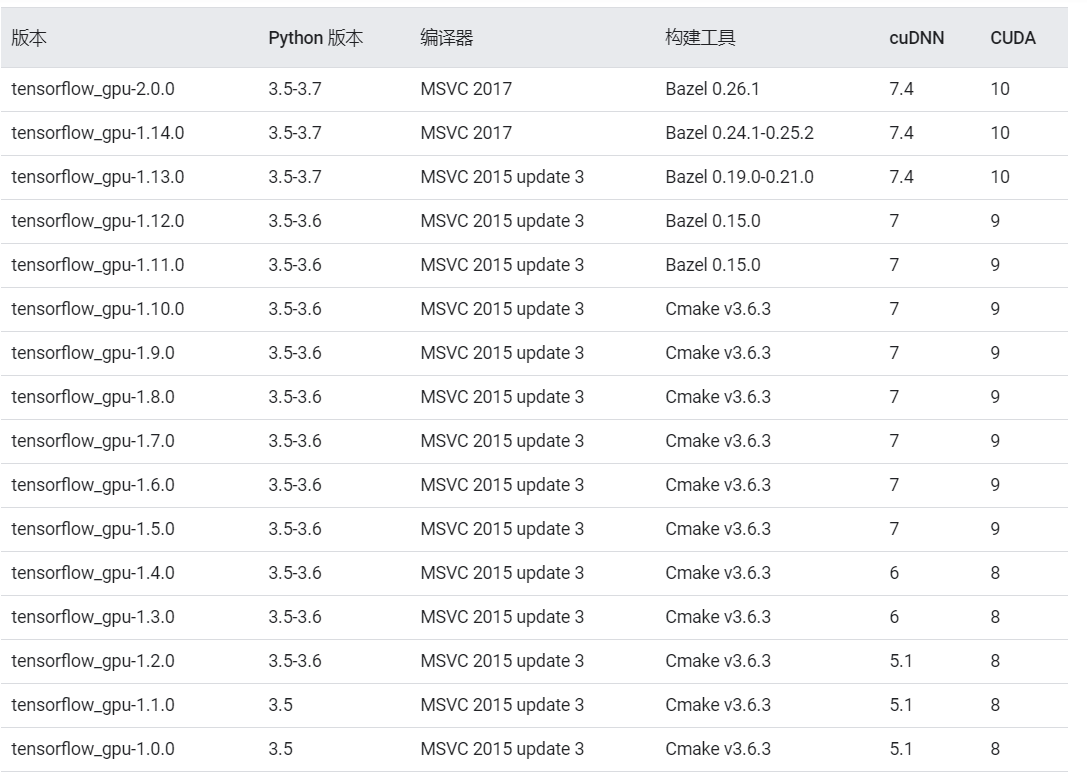
打开Anaconda Prompt，执行命令

conda install tensorflow-gpu=2.0.0

2.0.0是TensorFlow的版本，如果使用其他版本（如2.1.0），只需将其对应替换即可。

### 3.3.3 安装CUDA和CUDNN

1） 先到官网下载CUDA和CUDNN，一定要注意CUDA版本与Tensorflow版本的对应，图3-5是从TensorFlow官网截取的版本对应关系。根据图3-5可知，Python3.7对应cuda和cudnn版本分别是10.0与7.4 。而python3.7也是cuda和cudnn支持的最高python版本，这就是当初选择python3.7而放弃3.8或者3.9的原因。补充一点，安装CUDA时要勾选其中的VisualStudio插件，不然无法在VisualStudio中使用CUDA编译darknet。如果忘记勾选也没有关系，因为你可以随时运行此安装器补充安装其中的插件。



**图3-5 TensorFlow与CUDA、CUDNN版本对应关系**

2）下载完成后，按照向导安装CUDA，并配置系统环境变量即可。

3）执行命令nvcc -V验证安装。如果可以看到cuda版本号，则证明cuda安装成功。

4）对于cudnn直接将其解开压缩包，然后需要将以下bin,include,lib文件复制粘贴到cuda的文件夹下C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0

5）如果以上方式无法安装，可以使用命令行安装。（使用命令行安装可能存在找不到源的情况）依次执行命令：

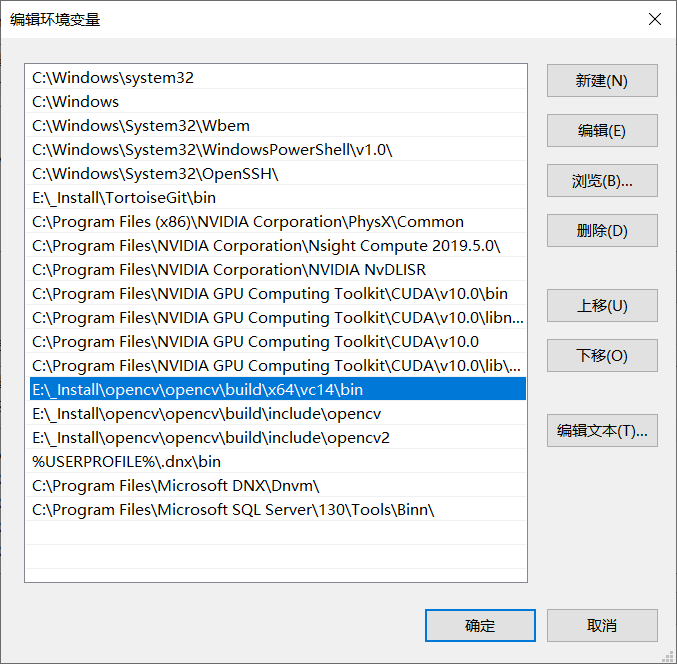
Pip install tensorflow-gpu=2.0.0

Pip install cudatoolkit=10.0

Pip install cudnn =7.4

### 3.3.4 安装OpenCV

到opencv官网下载opencv3.4的压缩包，解压后按照图3-6配置电脑环境变量即可 。



**图3-6 opencv环境变量配置**

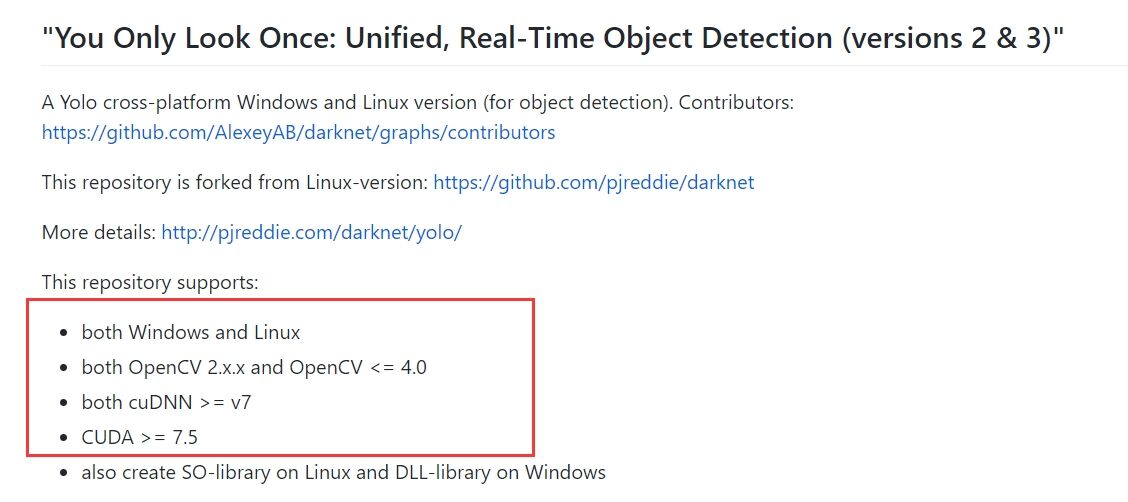
### 3.3.4 编译darknet

本系统使用的是2.3.5节介绍的AB版darknet，它的配置过程比较繁琐，准备工作的每一个细节都要保证精准无误，否则ERROR将如决堤之水。图2-7是作者专门说明的环境要求。

首先，opencv版本要在2.0到4.0之间，本系统采用的是opencv3.4 。

其次，CUDA版本到大于7.5，cudnn版本大于7.0。

最后，要保证NVIDIA驱动版本、CUDA版本、CUDNN版本之间的对应关系。CUDA与CUDNN的关系对照关系可以在图3-5中查到。而高版本NVIDIA 驱动兼容低版本，所以只要将其更新到最新版即可。我的显示适配器是NVIDIA GTX GeForce 1660Ti，驱动版本为430，CUDA版本为10.0，CUDNN版本为7.4 。



**图3-7 编译环境要求**

以下是darknet的编译步骤：

1）按照以上要求做好环境准备；

2）配置好以上环境之后，使用git工具下载AB版darknet源码（也可以访问其github网站下载压缩包 ）

git clone <https://github.com/AlexeyAB/darknet>

3）进入到build/darknet/x64目录，将opencv3.4安装目录中的两个dll文件：opencv\_ffmpeg340\_64.dll和opencv\_world340.dll复制到该文件夹下，不然后面运行会报错找不到dll。如果你找不到这两个dll文件，可以参考图3-8 。opencv\_world340后面带有小d的是debug版本；不带d的是release版本，这里需要选择release版本。

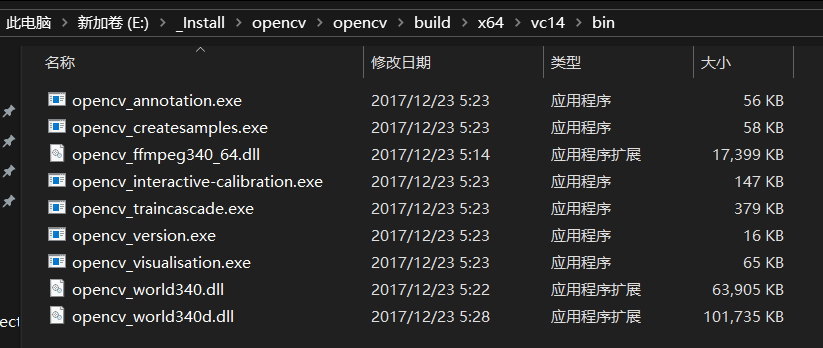
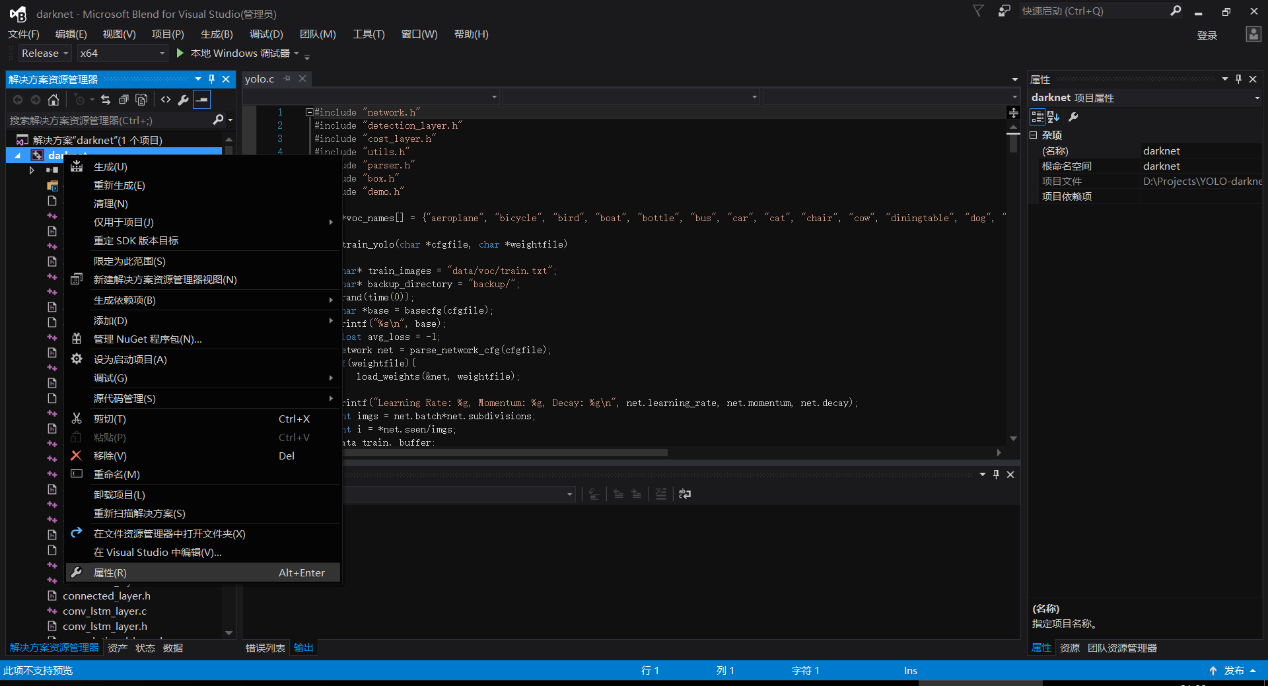


图3-8 opencv库文件的位置

4）使用VisualStudio2015（也可以使用2013或者2017版）打开darknet.sln。

5）如图3-9所示，右键对darknet.sln进行配置，注意上面应选为release和x64。



**图3-9 进行项目配置**

6）如图3-9所示，配置SDK版本为8.1，平台工具集为v140（如果使用VS2017或者2013，SDK和平台工具集要选择与之对应的版本）

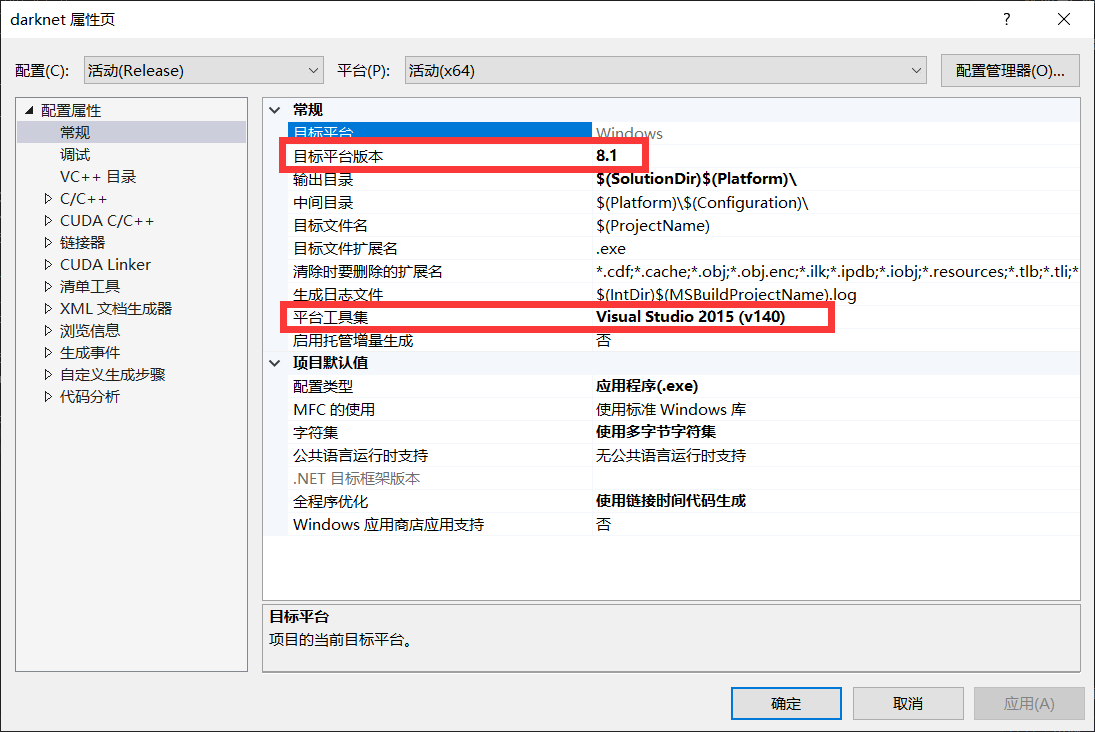
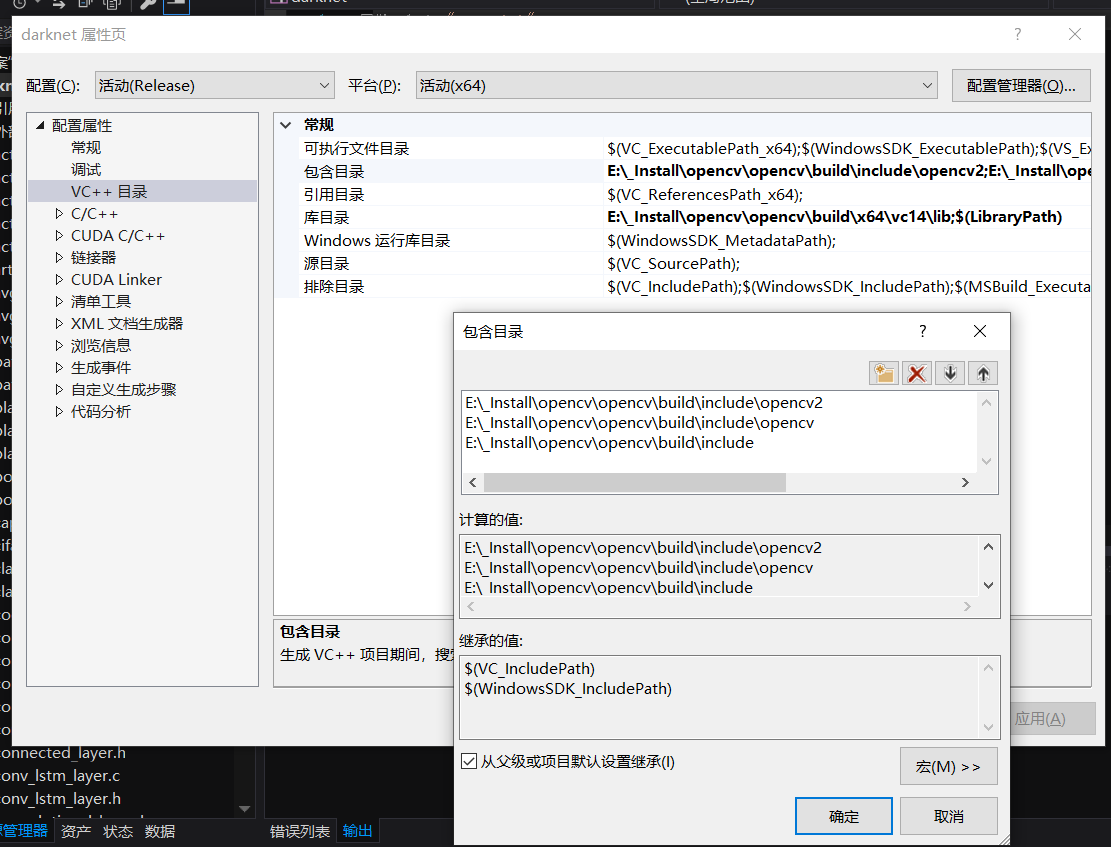
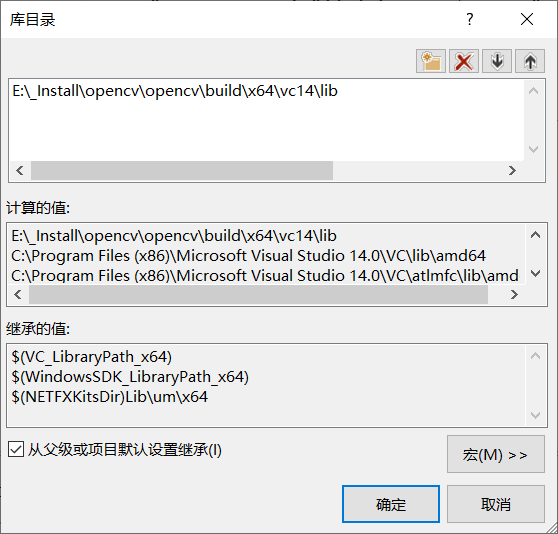


图3-9 SDK与平台工具集的配置

7）如图3-10和3-11所示，修改包含目录和库目录，即添加opencv3.4的包含目录和库目录（按照自己的opencv3.4的路径来进行添加）

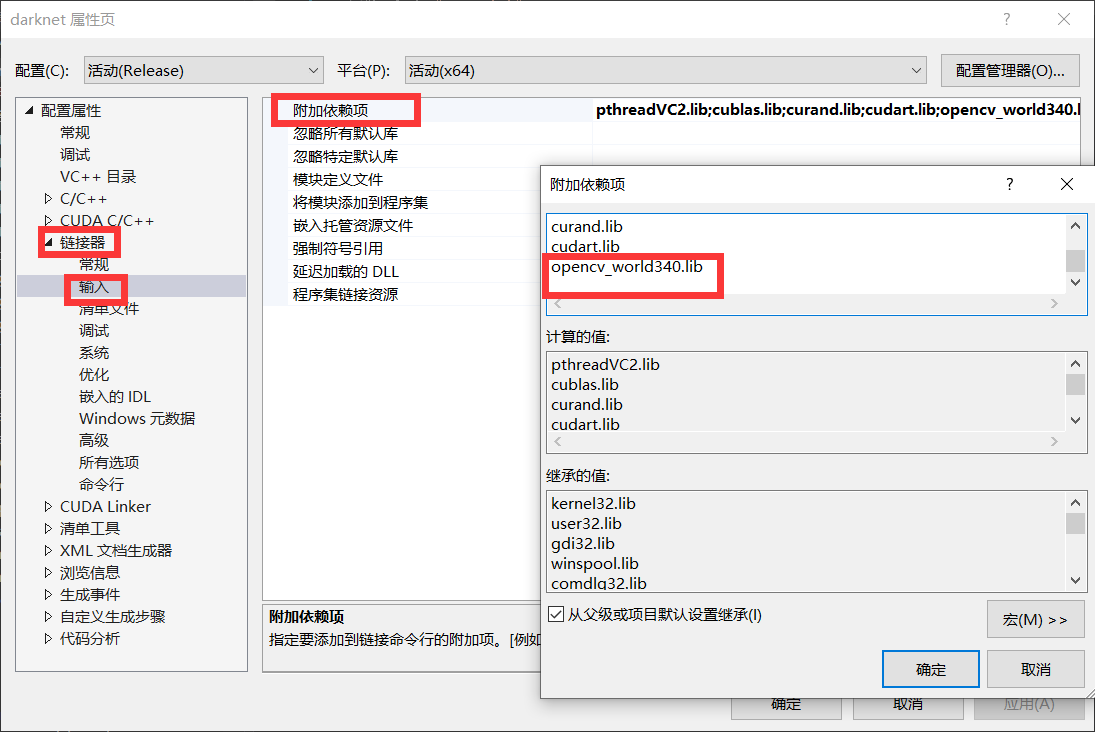


**图3-10 VC++目录的配置（包含目录）**



**图3-11 VC++目录的配置（库目录）**

8）如图3-12所示，在链接器的配置项中添加opencv的附加依赖。



**图3-12 附加依赖项的配置**

9）生成解决方案，生成darknet。至此，darknet的编译就完成了。

## 3.4 本章小结

本章介绍了系统总体设计框架、系统的具体需求分析、开发环境的搭建三部分内容，其中重点介绍了darnnet的编译。

# 第四章 车型识别系统软件设计

按照车型识别系统（VDS）需求优先级，将系统为1.0-alpha, 1.0, 2.0, 三个版本实现。其中1.0-alpha版实现了图片和标签的读取、数据清洗、训练集划分、模型训练、预测分类等一系列功能。它是一个“探索性”的版本，忽略一些细节，只是走通了一条最为主干的路；1.0版本在1.0-alpha版的基础上截弯取直，去掉对程序结果本身无影响的“探索性代码”，实现BodyStyle分类85%的精度，支持文件读取的方式输入车辆图片；3.0版改用YOLO模型，支持图片、摄像头、视频三种输入，对车辆的Make&Module进行分类预测。

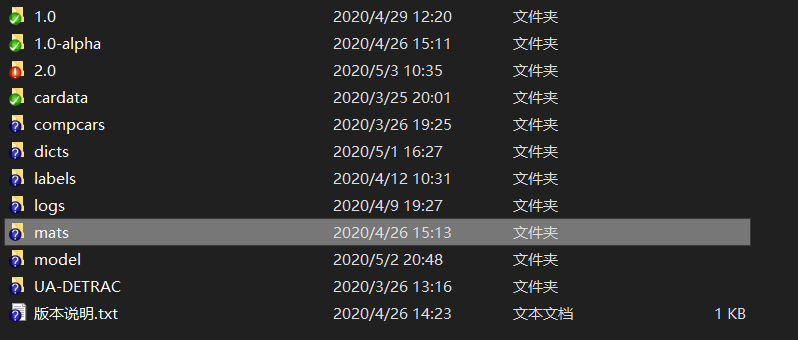
## 4.1 VDS1.0-alpha

### 4.1.1 简介

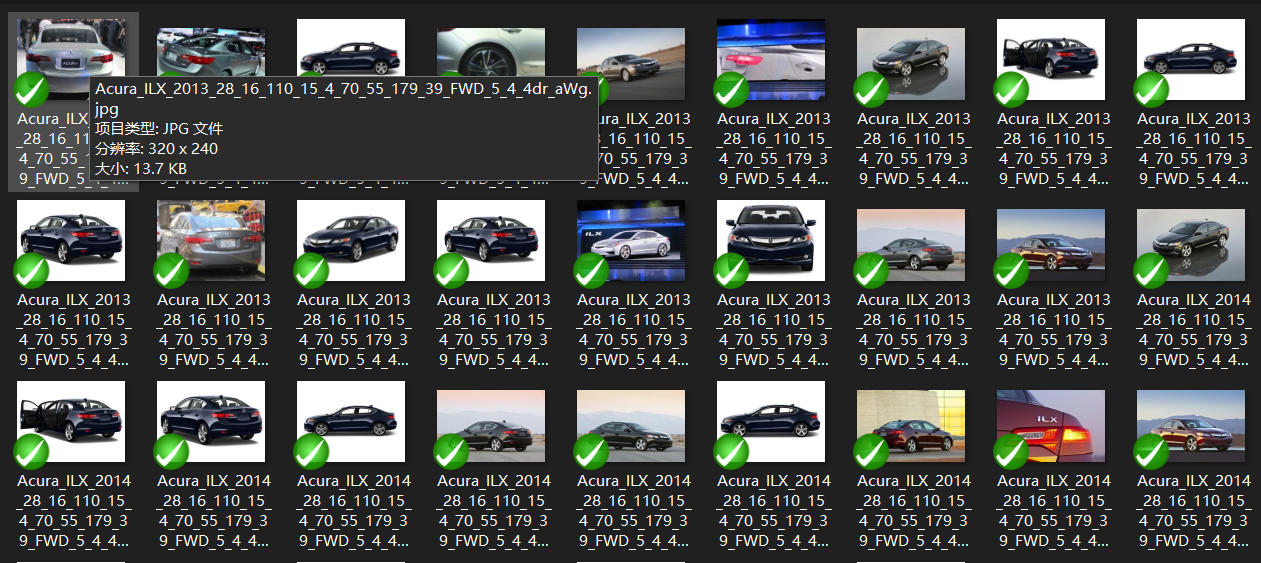
VDS1.0—alpha为本系统第一个探索性版本，其目的是找到一条适合的开发路径。它包括图片和标签的预处理、BodyStyle分类器的网络模型设计、生成对照字典三个主要功能。具体如何实现以及设计思路和什么是字典将在下文一一阐述。

### 4.1.2 数据预处理

介绍该模块之前，先对程序的文件存放位置进行简单介绍。如图4-1所示，当前版本的程序源代码存放在1.0-alpha文件夹中；图片文件放在cardata文件夹当中，其命名格式如图4-2所示；compcars存放的是另一个图片数据集，当前版本暂时用不到，在这里不予介绍；dicts文件夹存放的是字典文件，字典即分类号与分类名称的对照，如分类号3对应的分类名称为SUV；labels文件夹存放数据标签；logs文件夹存放日志文件；mats文件夹存放的是图片的数字化格式；model文件夹存放保存的分类器和回归模型。



**图4-1 系统文件的存放目录**



**图4-2 图片数据集的命名格式**

接下来正式介绍数据预处理模块。

1）首先加载图片，并生成标签：、

# 加载所有图片，共64467张

def load\_data():

data = np.empty((64467, 60, 80, 3), dtype='float32')

# data\_arr=[[[[]]]]

# label = np.empty((64467, 16), dtype='string')

label\_arr = [["" for i in range(16)]for j in range(64467)]

# os.listdir(filename)返回filename中所有文件的文件名列表

imgs = os.listdir('cardata')

num = len(imgs)

print(num)

for i in range(num):

# PIL 的 open() 函数用于创建 PIL 图像对象

img\_0 = Image.open('cardata/'+imgs[i])

img = img\_0.resize((80, 60), Image.ANTIALIAS)

# 转化为矩阵

arr = np.asarray(img, dtype='float32')

data[i, :, :, :] = arr

label\_arr[i] = imgs[i].split('\_')[0:16] # 这个是字符串类型需要编码

if i % 100 == 0:

print(i/646, "%")

del arr

del img

del img\_0

gc.collect()

return data, label\_arr

为了方便处理，一次性将所有图片压缩，读入到内存。在读取图片的同时，将文件名按照下划线分割保存到label\_arr变量中 。为了避免重复加载耗费大量时间，我们需要将变量保存到文件当中, 如此，下次使用即可直接从文件读取图片数据和标签。

2）加载完数据之后，要对数据进行清洗：

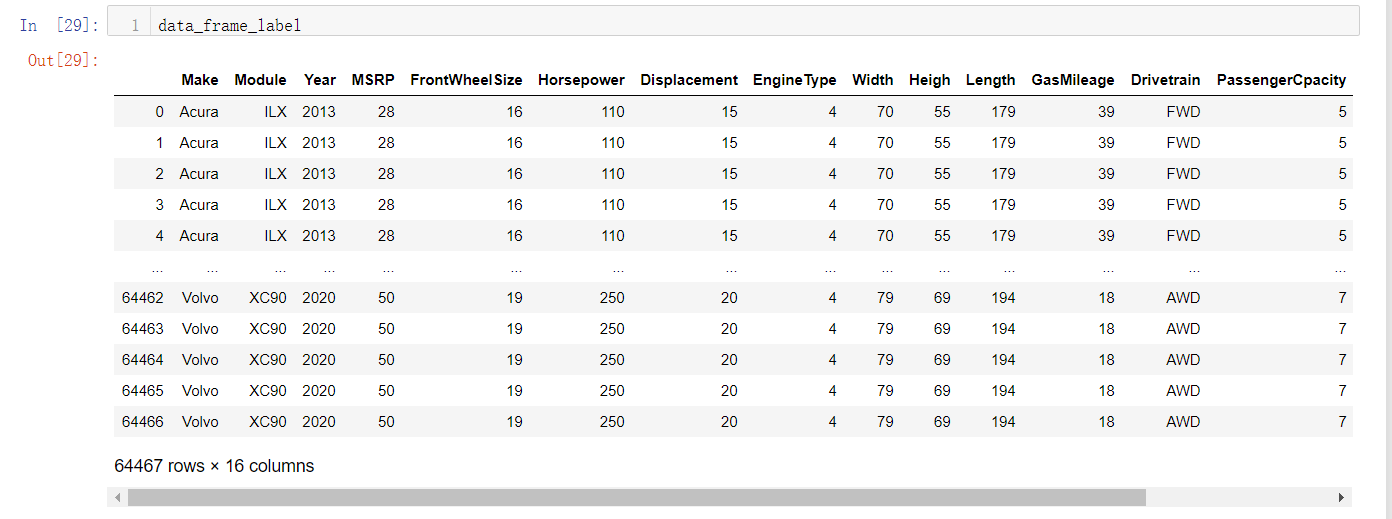
因为读取mat文件会因对齐增添一些空格，所以需要先去除空格

for i in range(len(label)): # 去除因对齐添加的空格

for j in range(len(label[0])):

label[i][j] = label[i][j].strip()

清洗完之后，标签数据格式如图4-3：



**图4-3 标签格式**

接下来对其进行均值填充，核心代码如下：

num\_cols\_list = ['Year', 'MSRP', 'FrontWheelSize', 'Horsepower',

'Displacement', 'EngineType', 'Width', 'Heigh', 'Length', 'GasMileage', 'PassengerCpacity', 'PassengerDoors']

# avg\_year = avg\_msrp = avg\_fws = avg\_horsr = avg\_disp = avg\_engine = avg\_w = avg\_h = avg\_l = avg\_gas = avg\_passcap = avg\_passdoor = 0

avg = [0 for i in range(len(num\_cols\_list))]

sums = [0 for i in range(len(num\_cols\_list))]

num = [0 for i in range(len(num\_cols\_list))]

for i in range(row):

for j in range(len(num\_cols\_list)):

x = data\_frame\_label[num\_cols\_list[j]][i]

# if match(x,[0-9]\d\*):

if match(x,"[0-9]\*"):

num[j] += 1

sums[j] += int(x)

上述代码思路为求得非空行数与数值之和，两者相除得平均值

# 有了均值之后，对非法值进行修改

for i in range(row):

for j in range(len(num\_cols\_list)):

x = data\_frame\_label[num\_cols\_list[j]][i]

if match(x, "[0-9]\*"):

data\_frame\_label[num\_cols\_list[j]][i]=int(x)

else:

data\_frame\_label[num\_cols\_list[j]][i]=avg[j]

if i%1000!=0:

pass

else:

print(100\*i/row,"%")

4）空值填充完之后，对非数值标签进行独热编码。

encoder = LabelBinarizer()

y\_onehot = encoder.fit\_transform(y\_label)

5）接下来是打乱顺序，划分训练集、测试集，然后为了快速收敛，对data进行归一化。为了卷面整洁，这些代码将不在文中呈现，最终在附录当中给出。

### 4.1.3 神经网络结构探索

1）神经网络的构造过程也是颇为坎坷，第一个网络模型是这样的：

# 输入层

# 卷积层 64，7\*7,2S

# 最大池化层64，3\*3，2S

# 局部响应归一化

# 卷积层 64，1\*1，1S

# 卷积层192,3\*3,1S

# 局部响应归一化

# 最大池化层192，(3,3)，2S

# 初始化层

# 初始化层

# 最大池化层,3\*3,2S

# 初始化层

# 初始化层

# 最大池化层,3\*3,2S

# 初始化层

# 初始化层

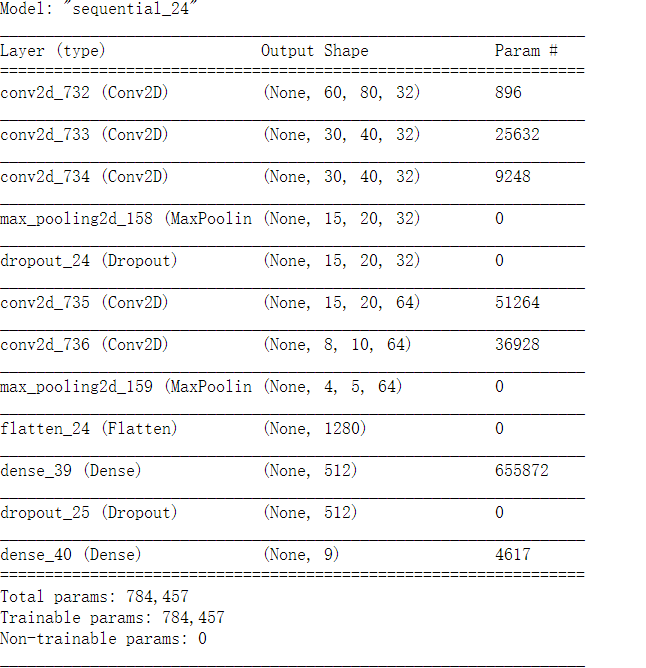
# 平均池化7\*7,1V

# dropout 40%

# 全链接

它参考了GoogleNet，共有19层，采用了GoogleNet中的初始化层，当然，如此复杂的模型处理简单的分类有种杀鸡用宰牛刀的感觉，结果可想而知。虽然模型不可用，为什么在文中介绍它呢？这是因为之后的网络结构都是由此衍生出的。

2）最后，经过一系列的修正，得到一个精度为87%的BodyStyle分类器，其结构如4-4所示：



**图4-4 BodyStyle分类器神经网络模型结构**

因为初始化层训练速度过于缓慢，当前分类器没有采用它，不过在后面的分类器中，还是采用了初始化层，这个将在后文详细介绍。

3）此外，1.0-alpha版还经过一系列尝试，得到合适的批量（40张图片）、学习率（0.01）等等参数。

## 4.2 VDS1.0

### 4.2.1 简介

有了1.0-alpha版本的探索经验后，针对第一优先级的需求，正式推出1.0版。它实现了文件读取汽车图片的方式，使用图4-4展示的神经网络实现对bodyStyle的预测，并将预测结果展示在原图片之上。

### 4.2.2 设计思路

按照需求分析优先级，1.0版本解决了BodyStyle的分类问题。它首先将模型训练独立出系统，单独对BodyStyle构造一个网络模型，并进行合适的训练，将模型保存到models文件夹。在系统中只需要加载模型进行预测即可。为了减小耦合，1.0版本对图片读取和预处理，分类预测，展示结果分别进行了模块化封装。各模块的具体实现如下：

1）调用保存的分类器

bodystyle\_model=Sequential()

MODEL\_PATH='model/bodystyle\_model.h5'

bodystyle\_model=load\_model(MODEL\_PATH)

2）定义图片读取和预处理模块

def read\_image(file\_name):

path=file\_name

img = Image.open(path)

img = img.resize((80, 60), Image.ANTIALIAS)

return img

3）定义预测函数

def predict\_class(img):

arr = np.asarray(img, dtype='float32')

arr=arr/255

arr=arr.reshape(1,60,80,3)

result = bodystyle\_model.predict\_classes(arr)

return bodyStyleDict[result[0]]

4）调用预测函数得到预测结果，将结果展示。效果如图4-5.

from PIL import Image, ImageFont, ImageDraw # 导入模块

print(img.format, img.size, img.mode)

draw = ImageDraw.Draw(img) #修改图片

#font = ImageFont.truetype(None, size = 40)

draw.text((20,50), result, fill = (255, 0 ,0)) #利用ImageDraw的内置函数，在图片上写入文字

img.show()



**图4-5 VDS-1.0 BodyStyle分类效果图**

### 4.2.3 做出的改进

假设A1A2A3为三个相互独立的模型，这三个模型旨在解决同一个问题PA，且精度均为0.8。对于一个相同的输入IA，其预测均正确的概率为0.83，即0.512；有两个模型预测正确，一个模型预测错误的概率为C23 \*0.82\*0.2，即0.384。也就是说，如果采用简单投票的方式，三个模型整体预测精度为0.512+0.384，即0.896。如果使用五个模型，同理可推算出其精度可达0.85+C45\*0.84\*0.2+C35\*0.83\*0.22+C52\*0.82\*0.23，即0.99328 。这里之所以将两个模型计算正确三个模型计算错误的情况也算进去，是因为在多分类问题中其类别比例大概率是2:1:1:1,而不是2:3或者2:2:1，为了简化计算，将后两种情况忽略（在二分类问题中 ，不可忽略）。

当然上述讨论的前提是各模型之间相互独立。实际上，很难对同一问题设计出5个完全独立的分类器。我们能做的只是设计结构尽可能差异的神经网络，使得各网络尽量不输出相同的错误结果。

## 4.3 VDS2.0

### 4.3.1 简介

VDS2..0将与1.0完全不同。首先，在分类结果上，VDS2.0有196类分类结果，远远高于1.0版的8类。其次，VDS2.0改用compcars数据集，这意味着又要重新设计数据预处理模块。最后，VDS2.0基于YOLOv3和darknet，采用C++编译器。 2.0版将支持视频实时检测。

### 4.3.2 设计思路

2.0版本的基本结构与图3-1所示仍是大致相同。首先是要有数据预处理模块，处理数据集的图片以及标签，使之与darknet框架所需标签格式一致。然后，最重要的是模型训练模块，使用3.3.4节编译后的darknet框架训练compcars数据集，得到一个196类分类器。最后的预测模块，使用python调用命令行，执行预测命令，这样一来，即可将预测模块嵌入python程序。下面结合关键代码详细介绍每个模块的具体实现。

#### 4.3.2.1 数据预处理

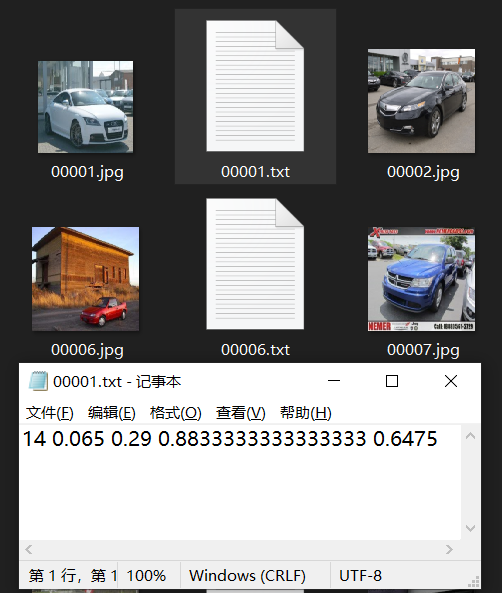
1）在具体实现之前，先介绍一下YOLOv3训练数据集需要的文件。

首先，要有训练集，也就是训练所需图片。本系统使用的是compcars数据集。

其次，对应每一张图片，要有一个与之同名的txt文件，该文件中每行包含5个数字，第一个数字代表类别，后面的四个数据是标注框的信息，分别是标注框左上角的x，y坐标，标注框的宽w，高h。 每一行是一个标注对象， 如果一张图片中有多个目标，则txt就对应有多少行。需要特别注意的是，坐标和标注框的宽高都要进行归一化，即使数据落在0-1之间。示例请参照图4-6 。

再次，在train.txt中记录要参与训练的图片的文件名。如图4-7所示，每行记录一张图片。并以同样的格式生成valid,txt。

最后，编辑cfg文件。这个文件与训练与测试有关而与数据预处理无关，所以具体细节将在后文介绍。



**图4-6 训练集图片与标签格式**



图4-7 train.txt

2）下面，再介绍一下compcars数据集里的标签样式以及将这些标签转换为YOLO需要的格式的方式。

在数据集的devkit文件夹内有一个README.txt。在这个文件中有标签数据格式的介绍。

训练集的标签存放在一个名为cars\_train\_annos.mat的文件中。调用如下代码读取该文件，解析后的内容如图4-8所示。

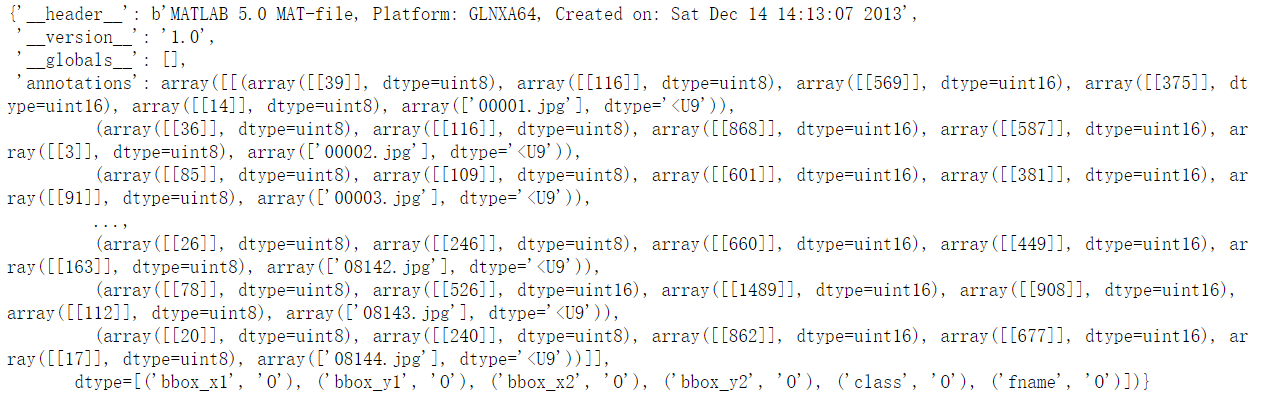
import scipy.io as io #使用scipy读取mat文件

train\_label\_path = '../compcars/devkit/cars\_train\_annos.mat'

# test\_label\_path = '../compcars/devkit/cars\_test\_annos.mat'

train\_label = io.loadmat(train\_label\_path)

print(train\_label)



**图4-8 解析cars\_train\_annos.mat文件**

从README.txt中可知，每个array分别记录了类别class，图片左上角x坐标，左上角y坐标，右下角x坐标，右下角y坐标，图片文件名。

这些数据糅合在一起很难处理，接下来我们要将它们剥离开。

import os

import gc

from PIL import Image

import numpy as np

train\_label\_anno=train\_label['annotations']

xmins = []

ymins = []

xmaxs = []

ymaxs = []

classes=[]

filenames=[]

nums=len(train\_label\_anno[0])

for i in range(nums):

annotations = train\_label\_anno[0][i]

xmin = annotations[0][0][0]

ymin = annotations[1][0][0]

xmax = annotations[2][0][0]

ymax = annotations[3][0][0]

# xmin = xmin/width

# ymin = ymin/height

# xmax = xmax/width

# ymax = ymax/height

xmins.append(xmin)

ymins.append(ymin)

xmaxs.append(xmax)

ymaxs.append(ymax)

classes.append(annotations[4][0][0])

filenames.append(annotations[5][0])

上述代码将五个维度剥离开，分别存放在xmins,ymins,xmaxs,ymaxs,classes,filenames五个list中。这一步工作完成之后，还要对其归一化，除此之外，因为YOLO需要图片宽高，所以还要想办法获得图片的宽高。

# 生成边框width和height

# box\_width = xmaxs-xmins

# box\_height = ymaxs-ymins

box\_width = []

box\_height = []

for i in range(nums):

box\_height.append(ymaxs[i]-ymins[i])

box\_width.append(xmaxs[i]-xmins[i])

#归一化

#由于每张图片的分辨率不一定一样，所以要遍历每张图片，以获得图片分辨率。

path='../compcars/cars\_train/' #图片存储路径

for i in range(nums):

img\_path = path+filenames[i] #拼接图片路径与文件名

img = Image.open(img\_path) #打开图片

img\_width, img\_height = img.size #获得分辨率

# 进行归一化

c = str(classes[i])

x = str(xmins[i]/img\_width)

y = str(ymins[i]/img\_height)

w = str(box\_width[i]/img\_width)

h = str(box\_height[i]/img\_height)

# 拼接字符串

row = ''+c+' '+x+' '+y+' '+w+' '+h

# txt\_filename=img\_path+'.txt'

# 替换文件名，将jpg替换为txt

txt\_filename = img\_path[:-3]+'txt'

f=open(txt\_filename, "w+")

f.write(row) #写入文件

f.close()

经过上述步骤后，YOLOv3所需每张图片的标签即可获得。最后再生成train.txt即可，valid.txt内容可以与train.txt一致。

f = open('train.txt', "a+")

for i in range(nums):

row = filenames[i]

f.write('data/compcars/'+row+'\n')

f.close()

训练之前，还有一个准备工作要做。虽然我们已经知道了每个类的编号，但是不知道这些编号具体对应哪一类。所以，要建立类编号和类名之间的映射。映射关系在cars\_meta.mat中已经给出，现在只要将其解析出来即可。

meta\_label\_path = '../compcars/devkit/cars\_meta.mat'

train\_meta = io.loadmat(meta\_label\_path)

class\_dict=train\_meta ['class\_names'][0]

#将class\_dict的内容复制，替换voc\_label.py中的classes。

#### 4.3.2.2 训练

训练之前要做一些准备工作，这些工作在这里不予详细列出，有兴趣的请到附录三中进行查看。

训练模块使用python调用Windows原生命令窗口，执行已经用VS生成的exe文件

具体代码如下：

def start\_training(path, classes, names, weights,data,cfg, train='data/train.txt', valid='data/valid.txt', backup='backup/', nocfg=False):

if nocfg:

create\_datafile(path, classes, names, train, valid)

command = 'cd D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/ &\

darknet.exe detector train '+data+' '+cfg+' '+weights

# print(command)

with os.popen(command, "r")as f:

text = f.read()

print(text) # cmd输出结果

return text

经封装后，就可以在python环境训练了。也可以直接使用dos命令行执行命令进行训练，二者效果一致，只不过后者可以实时查看训练过程。

# nocfg=true的话，系统会生成data文件

# 如果data文件已存在，请设置nocfg=False

weights = 'backup/yolov3-compcars\_last.weights'

data = 'data/compcars.data'

cfg = 'yolov3-compcars.cfg'

start\_training(path, classes, names, weights, data,

cfg, train, valid, nocfg=False)

#### 4.3.2.3 预测

和训练模块类似，本系统对预测模块同样进行了封装，代码如下：

def predict(data, cfg, weights, filepath, mode='test'):

command = 'cd D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/ &\

darknet.exe detector '+mode+' '+data+' '+cfg+' '+weights + ' '+filepath

print('执行命令:\n', command)

with os.popen(command, "r")as f:

text = f.read()

print(text) # cmd输出结果

return text

mode有两种状态，当它为test时，predict函数对图片进行预测。当mode为demo时，predict函数使用视频作为输入进行预测，当视频路径为空时，系统默认使用摄像头。具体实例如下：

weights = 'backup/yolov3-compcars\_last.weights'

data = 'data/compcars.data'

cfg = 'yolov3-compcars.cfg'

mode='test'  
 img\_path='D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/data/compcars/cars\_test/00001.jpg'

# 展示原图

from PIL import Image

import os

img = Image.open(img\_path)

img

# 测试结果

text = predict(data, cfg, weights, img\_path, mode='test')

mode='demo'

vidio\_path = 'D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/data/test.mp4'

text = predict(data, cfg, weights, vidio\_path, mode='demo')

## 4.4 本章小结

本章先后介绍了VDS三个版本的设计思路与部分具体实现。

# 第五章 车型识别系统的验证测试

## 5.1 VDS1.0

### 5.1.1功能点VDS-1.0-001

功能描述：对图片进行大小、格式统一处理，，方便后续使用

测试用例：输入分辨率为320\*240\*3的图片5-1，预期结果：得到60\*80\*3的矩阵

验证结果：见图5-2

****

**图5-1 VDS-1.0-001测试图片**



**图5-2 VDS-1.0-001测试结果**

### 5.1.2功能点VDS-1.0-002

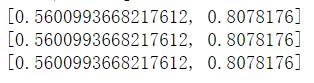
功能描述：使用图片进行训练，得到精度达0.8的多分类模型

验证结果：四个不同的bodyStyle分类模型在测试集上的loss和acc如图5-3所示，在训练集上的loss和acc如图5-4所示。

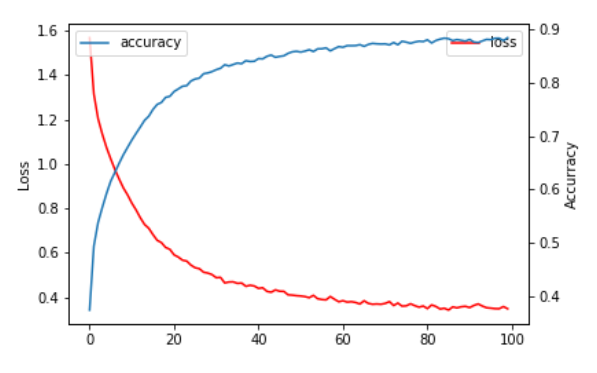
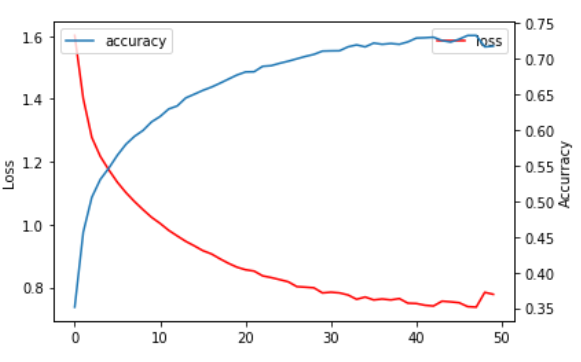


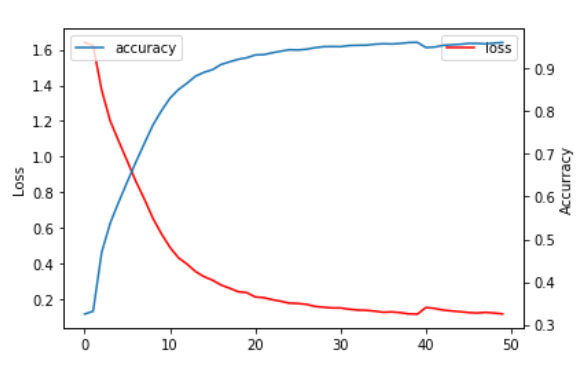
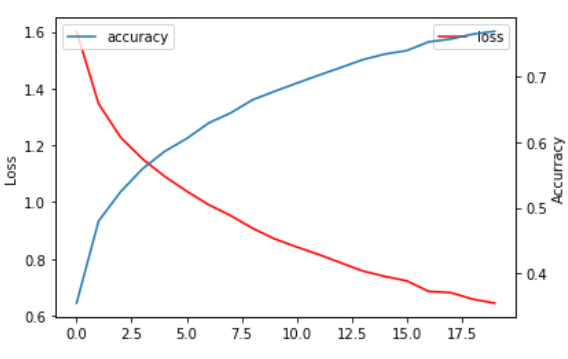






**图5-3 bodyStyle分类器在测试集上的损失与精度**

**图5-4 bodyStyle分类器在训练集上的损失与精度**

### 5.1.3功能点VDS-1.0-003

功能描述：将处理好的图片传入该模块、得到汽车的车型信息

使用图5-5（1）所示的验证图片作为分类器的输入得到图5-5（2）。



1. （2）

**图5-5分类器的输入（1）和输出（2）**

## 5.2 VDS2.0

### 5.2.1功能点VDS-1.0-001

功能描述：对图片进行大小、格式统一处理，方便后续使用

验证：2.0版本增加了适配不同图片分辨率的功能，所以取消了统一格式的预处理模块。

使用不同分辨率的图片作为输入，如果系统能够识别，则该功能点通过验证。

使用图5-6中四张不同分辨率的图片作为系统的输入，其对应的输出见图5-7



**图5-6 VDS-1.0-001测试图片**

**图5-7 VDS-1.0-001测试结果**

### 5.2.2功能点VDS-1.0-002

### 5.2.3功能点VDS-1.0-003

### 5.2.4功能点VDS-2.0-001

### 5.2.5功能点VDS-2.0-002

## 5.3 本章小结

本章针对3.2.3节的需求功能点进行验证性测试，在实验环境下各功能点均完成。

# 结论与展望

**1. 总结**

本文从机器学习在汽车领域的应用角度出发，先是介绍了机器学习的概念与发展历程 。 接着又介绍了常用深度学习框架Tensorflow与小众框架darknet，其中详细介绍了darknet与YOLO的原理。然后介绍了本系统的设计框架与具体实现，最后根据需求分析中的功能点进行系统测试，测试结果符合要求。在硬件条件允许的情况下，本系统可以应用于道路车流量分析、车型鉴定等方面。

**2. 展望**

虽然如今的机器学习框架可以做到几十毫秒的时间内计算出预测结果，但是这种实时性实际上还没有实用性。之所以这么说，是因为它们的实时性是建立在强大的显示适配器的基础上，而因为高性能显示适配器成本过高，诸如本系统之类的识别系统很难做到大规模应用。要实现大规模应用，依本人愚见，有两条路可以走：一是研发出可在嵌入式系统或者微处理器系统中做到实时性的机器学习框架；二是利用新的网络传输技术的低延迟性，将终端采集到的数据传输给高性能的计算机集群进行运算，再将结果返回终端予以显示。

# 致 谢

本论文的顺利完成，离不开我的导师悉心的指导。是他帮助我消除了选题的迷茫、修正了草案的缺陷、克服了设计的困难、提高了论文格式的规范性。在这些天的交流与指导当中，导师认真的工作作风、一丝不苟的教学习惯以及学校优良的学习风气使我受益良多。在此，我对我的导师以及培育我的母校表示衷心的感谢与敬佩。此外，我还要感谢在毕业设计中帮我打开思路与耐心指导的学姐，是她让我少走了很多弯路，帮助我顺利完成毕业设计与论文撰写。

# 参考文献

[1] [Kaicheng Yu](https://arxiv.org/search/cs?searchtype=author&query=Yu%2C+K). Evaluating the Search Phase of Neural Architecture Search[D].纽约:康奈尔大学.2019

[2] 维基百科.Machine Learning.[EB/OL].‘https://www.thefreedictionary.com/Machine learning’. 2014/2020.

[3] 泰坦妮克号. [为什么CNN能自动提取图像特征](https://www.cnblogs.com/ymd12103410/p/9874878.html)[EB/OL]. ‘https://www.cnblogs.com/ymd12103410/p/9874878.html’.2019/2020

[4] Redmon, Joseph and Farhadi.YOLOv3:An Incremental Improvement[D].arXiv.2018

[5] Ross Girshick. Fast R-CNN[D].arXiv.2015

*[6]* Kaiming He. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[D].arXiv.2015

[7] Wei Liu. SSD: Single Shot MultiBox Detector[D].ECCV.2016

# 附录一 VDS1.0-alpha源码

**# VDS 1.0-alpha 数据加载模块 源代码**

import os

import gc

# import cv2

from PIL import Image

import numpy as np

# 加载所有图片，共64467张

def load\_data():

# Return a new array of given shape and type, without initializing entries.

data = np.empty((64467, 60, 80, 3), dtype='float32')

# data\_arr=[[[[]]]]

# label = np.empty((64467, 16), dtype='string')

label\_arr = [["" for i in range(16)]for j in range(64467)]

# os.listdir(filename)返回filename中所有文件的文件名列表

imgs = os.listdir('cardata')

num = len(imgs)

print(num)

for i in range(num):

# PIL 的 open() 函数用于创建 PIL 图像对象

img\_0 = Image.open('cardata/'+imgs[i])

img = img\_0.resize((80, 60), Image.ANTIALIAS)

# Convert the input to an array

arr = np.asarray(img, dtype='float32')

data[i, :, :, :] = arr

label\_arr[i] = imgs[i].split('\_')[0:16] # 这个是字符串类型需要编码

if i % 100 == 0:

print(i/646, "%")

del arr

del img

del img\_0

gc.collect()

return data, label\_arr

data, label\_arr = load\_data()

# 保存到文件 避免下次重复计算

import scipy.io as io

data\_mat\_path = 'mats/data\_old'

label\_mat\_path = 'labels/label\_old'

io.savemat(data\_mat\_path, {'name': data})

io.savemat(label\_mat\_path, {'name': label})

# 加载data、label

import scipy.io as io

data\_mat\_path = 'mats/'

label\_mat\_path = 'labels/'

data\_filename = 'data.mat'

label\_filename = 'label.mat'

data = io.loadmat(data\_mat\_path+data\_filename)["name"]

label = io.loadmat(label\_mat\_path+label\_filename)["name"]

for i in range(64467): # 去除因对齐添加的空格

for j in range(16):

label[i][j] = label[i][j].strip()

**# VDS 1.0-alpha 数据清洗模块 源代码**

# 选取所有label作为使用，因为这样的估价会更加准确

# 首先对label进行编码，数字类型的和字符串类型的要使用不同的编码方式

# 参考 <https://blog.csdn.net/luguanyou/article/details/80598122>

# 先将label转换为DataFrame的格式,方便可视化观察

import pandas as pd

from sklearn.preprocessing import LabelBinarizer

from pandas import DataFrame

data\_frame\_label = DataFrame(label, index=None, columns=['Make', 'Module', 'Year', 'MSRP', 'FrontWheelSize', 'Horsepower', 'Displacement', 'EngineType', 'Width', 'Heigh', 'Length', 'GasMileage', 'Drivetrain', 'PassengerCpacity', 'PassengerDoors', 'BodyStyle'])

row,col=data\_frame\_label.shape

# 这时候需要进行数据清洗，因为有空值的存在

# 先查看一下有多少记录存在空值

cols\_list = ['Make', 'Module', 'Year', 'MSRP', 'FrontWheelSize', 'Horsepower',

'Displacement', 'EngineType', 'Width', 'Heigh', 'Length', 'GasMileage',

'Drivetrain', 'PassengerCpacity', 'PassengerDoors', 'BodyStyle']

del\_index = []

for cols in cols\_list:

for i in range(row):

if data\_frame\_label[cols][i] == 'nan':

del\_index.append(i)

len(del\_index)

# 可以看到有24947个记录存在nan

# 处理空值主要有两种方式：剔除、替换

# 在这里存在空值的记录占到了三分之一，用剔除法不太合适，所以决定进行均值填充

# 但是有四个字段是字符串类型，无法进行均值计算，我们选择保留它

import re

from fnmatch import fnmatchcase as match

num\_cols\_list = ['Year', 'MSRP', 'FrontWheelSize', 'Horsepower',

'Displacement', 'EngineType', 'Width', 'Heigh', 'Length', 'GasMileage',

'PassengerCpacity', 'PassengerDoors']

# avg\_year = avg\_msrp = avg\_fws = avg\_horsr = avg\_disp = avg\_engine = avg\_w = avg\_h = avg\_l = avg\_gas = avg\_passcap = avg\_passdoor = 0

avg = [0 for i in range(len(num\_cols\_list))]

sums = [0 for i in range(len(num\_cols\_list))]

num = [0 for i in range(len(num\_cols\_list))]

for i in range(row):

for j in range(len(num\_cols\_list)):

x = data\_frame\_label[num\_cols\_list[j]][i]

# if match(x,[0-9]\d\*):

if match(x,"[0-9]\*"):

num[j] += 1

sums[j] += int(x)

for i in range(len(num\_cols\_list)):

avg[i]=int(sums[i]/num[i])

# 有了均值之后，对非法值进行修改

for i in range(row):

for j in range(len(num\_cols\_list)):

x = data\_frame\_label[num\_cols\_list[j]][i]

if match(x, "[0-9]\*"):

data\_frame\_label[num\_cols\_list[j]][i]=int(x)

else:

data\_frame\_label[num\_cols\_list[j]][i]=avg[j]

if i%1000!=0:

pass

else:

print(100\*i/row,"%")

# 然后对label中的不同标签分别使用sklearn的LabelBinarizer进行编码，替换原先label的标签

encoder = LabelBinarizer()

make\_onehot = encoder.fit\_transform(data\_frame\_label['Make'])

module\_onehot = encoder.fit\_transform(data\_frame\_label['Module'])

drivetrain\_onehot = encoder.fit\_transform(data\_frame\_label['Drivetrain'])

bodystyle\_one\_hot = encoder.fit\_transform(data\_frame\_label['BodyStyle'])

# 现在已经将字符串类型的标签进行独热编码，放到四个变量当中。接下来，移除原dataframelabel中这四列

# 在移除之前，要先建立独热编码后的标签和之前的联系，用字典表示

dict\_make = {}

dict\_module = {}

dict\_drivetrain = {}

dict\_bodystyle = {}

# 要作为字典的key，应该将独热编码转换为string

n = len(make\_onehot) # 64467

str\_make\_one\_hot = [''for i in range(n)]

str\_module\_one\_hot = [''for i in range(n)]

str\_drivetrain\_onehot = [''for i in range(n)]

str\_bodystyle\_one\_hot = [''for i in range(n)]

# tuple\_make\_one\_hot = tuple(make\_onehot)

# tuple\_module\_one\_hot = tuple(module\_onehot)

# tuple\_drivetrain\_onehot = tuple(drivetrain\_onehot)

# tuple\_bodystyle\_one\_hot = tuple(bodystyle\_one\_hot)

for i in range(n):

for j in range(len(make\_onehot[0])):

str\_make\_one\_hot[i] += str(make\_onehot[i][j])

for j in range(len(module\_onehot[0])):

str\_module\_one\_hot[i] += str(module\_onehot[i][j])

for j in range(len(drivetrain\_onehot[0])):

str\_drivetrain\_onehot[i] += str(drivetrain\_onehot[i][j])

for j in range(len(bodystyle\_one\_hot[0])):

str\_bodystyle\_one\_hot[i] += str(bodystyle\_one\_hot[i][j])

dict\_make[str\_make\_one\_hot[i]] = data\_frame\_label['Make'][i].strip()

dict\_module[str\_module\_one\_hot[i]] = data\_frame\_label['Module'][i].strip()

dict\_drivetrain[str\_drivetrain\_onehot[i]

] = data\_frame\_label['Drivetrain'][i].strip()

dict\_bodystyle[str\_bodystyle\_one\_hot[i]

] = data\_frame\_label['BodyStyle'][i].strip()

# 移除四列,保存在新变量data\_frame\_label\_nums 中

data\_frame\_label\_nums = data\_frame\_label.drop(

['Make', 'Module', 'Drivetrain', 'BodyStyle'], axis=1)

# 现在想办法将移除的四列重新以独热编码的格式插入到data\_frame\_label中去

# 查看data\_frame\_label\_num的格式

data\_frame\_label\_nums.head()

print(type(data\_frame\_label\_nums),type(make\_onehot))

# 可以看到一个是DataFrame类型，另外四个是ndarray类型，所以我们首先将类型统一

data\_ndarray\_label\_nums=data\_frame\_label\_nums.values

# 格式统一之后，为了进行拼接，需要对shape统一

data\_ndarray\_label\_nums.shape

# 一种实现思路是将data\_ndarray\_label\_nums进行升维，如将3变为[3]，这样就可以和其他四个独热编码同构(需要把其它四个也升三维），可以进行拼接

# 另一种实现思路是直接将5个array进行拼接，表头的含义通过字典进行查找

# 由于第一种方案更接近人的思想，我们优先考虑这种

data\_ndarray\_label\_nums=data\_ndarray\_label\_nums.reshape(64467,12,1)

make\_onehot=make\_onehot.reshape(64467,1,-1)

module\_onehot=module\_onehot.reshape(64467,1,-1)

drivetrain\_onehot=drivetrain\_onehot.reshape(64467,1,-1)

bodystyle\_one\_hot=bodystyle\_one\_hot.reshape(64467,1,-1)

data\_ndarray\_label\_one\_hot = np.c\_[data\_ndarray\_label\_nums, make\_onehot]

row, width\_make\_one\_hot = make\_onehot.shape

row, width\_module\_one\_hot = module\_onehot.shape

row, width\_drivetrain\_one\_hot = drivetrain\_onehot.shape

row, width\_bodystyle\_one\_hot = bodystyle\_one\_hot.shape

print(width\_make\_one\_hot, width\_module\_one\_hot,

width\_drivetrain\_one\_hot, width\_bodystyle\_one\_hot)

# 这里要记住data\_ndarray\_label\_one\_hot每一列的含义

# 0: 12 ['Year', 'MSRP', 'FrontWheelSize', 'Horsepower', 'Displacement', 'EngineType', 'Width', 'Heigh', 'Length', 'GasMileage',

# 'PassengerCpacity', 'PassengerDoors']

# 12:12+42 Make

# 12+42:12+42+323 Module

# 12+42+323: 12+42+323+5 DriveTrain

# 12+42+323+5: BodyStyle

**VDS 1.0-alpha 训练模块**

# 到此为止，所有的图片保存在了data中，所有的标签保存到了data\_ndarray\_label\_one\_hot

# 标签名字过长，我们进行简化

import copy

label=copy.deepcopy(data\_ndarray\_label\_one\_hot)

#打乱顺序

#读取data矩阵的第一维数（图片的个数）

num\_example=data.shape[0]

#产生一个num\_example范围，步长为1的序列

arr=np.arange(num\_example)

#调用函数，打乱顺序

np.random.shuffle(arr)

#按照打乱的顺序，重新排序

data=data[arr]

label=label[arr]

r\_train = 0.8

s\_train = np.int(num\_example \* r\_train)

X\_train = data[:s\_train]

y\_train = label[:s\_train]

X\_test = data[s\_train:]

y\_test = label[s\_train:]

# TensorFlow and tf.keras

import tensorflow as tf

from tensorflow import keras

#导入必备包

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import AveragePooling2D

from tensorflow.keras.layers import BatchNormalization

from tensorflow.keras.layers import Activation

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Input

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import concatenate

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_)

print(tf.test.is\_gpu\_available)

print(tf.test.is\_built\_with\_cuda)

print(tf.test.gpu\_device\_name)

print(tf.test.is\_built\_with\_gpu\_support)

class ConvBNRelu(keras.Model):

def \_\_init\_\_(self, ch, kernelsz=3, strides=1, padding='same'):

super(ConvBNRelu, self).\_\_init\_\_()

self.model = keras.models.Sequential([

keras.layers.Conv2D(ch, kernelsz, strides=strides, padding=padding), # 卷积

keras.layers.BatchNormalization(), #批量归一化

keras.layers.ReLU() #激活函数

])

def call(self, x, training=None):

x = self.model(x, training=training)

return x

class InceptionBlk(keras.Model):

def \_\_init\_\_(self, ch, strides=1):

super(InceptionBlk, self).\_\_init\_\_() # 父类方法

self.ch = ch # 赋值

self.strides = strides # 赋值

self.conv1 = ConvBNRelu(ch, strides=strides) # 构造第1个CNN基本单元

self.conv2 = ConvBNRelu(ch, kernelsz=3, strides=strides) # 构造第2个CNN基本单元，卷积核大小初始为3\*3

self.conv3\_1 = ConvBNRelu(ch, kernelsz=3, strides=strides)# 构造第3\_1个CNN基本单元，卷积核大小初始为3\*3

self.conv3\_2 = ConvBNRelu(ch, kernelsz=3, strides=1) # 构造第3\_2个CNN基本单元，卷积核大小初始为3\*3

self.pool = keras.layers.MaxPooling2D(3, strides=1, padding='same') # 最大池化层，same

self.pool\_conv = ConvBNRelu(ch, strides=strides) #构造CNN基本单元，卷积核大小初始为3\*3

def call(self, x, training=None):

x1 = self.conv1(x, training=training)

x2 = self.conv2(x, training=training)

x3\_1 = self.conv3\_1(x, training=training)

x3\_2 = self.conv3\_2(x3\_1, training=training)

x4 = self.pool(x)

x4 = self.pool\_conv(x4, training=training)

# concat along axis=channel

x = tf.concat([x1, x2, x3\_2, x4], axis=3) # 通道数扩充，上图理解。

return x

# Res Block 模块。继承keras.Model或者keras.Layer都可以

class Inception(keras.Model):

def \_\_init\_\_(self, num\_layers, num\_classes, init\_ch=16, \*\*kwargs):

super(Inception, self).\_\_init\_\_(\*\*kwargs)

self.in\_channels = init\_ch # 初始输入通道

self.out\_channels = init\_ch # 初始输出通道数

self.num\_layers = num\_layers # 初始层数，就是多少个res block

self.init\_ch = init\_ch # 初始通道

self.conv1 = ConvBNRelu(init\_ch) # 构造1个CNN基本单元

self.blocks = keras.models.Sequential(

name='dynamic-blocks') # 创建一个Sequential容器对象

# for循环，循环次数为num\_layers传输的参数；

for block\_id in range(num\_layers):

# 2个Inception Block模块构建一个参考块，循环2次；

for layer\_id in range(2):

if layer\_id == 0: # 如果为第1次循环的话

# 创建1个Inception Block模块，并且步长为2；

block = InceptionBlk(self.out\_channels, strides=2)

else: # 如果为第2次循环的话

# 创建1个Inception Block模块，并且步长为1；

block = InceptionBlk(self.out\_channels, strides=1)

self.blocks.add(block) # 把block放进容器对象blocks中

# enlarger out\_channels per block # 通道数扩大为2倍。

self.out\_channels \*= 2

self.avg\_pool = keras.layers.GlobalAveragePooling2D()

self.fc = keras.layers.Dense(num\_classes)

def call(self, x, training=None):

out = self.conv1(x, training=training)

out = self.blocks(out, training=training)

out = self.avg\_pool(out)

out = self.fc(out)

return out

# 构造第二个网络

# def conv\_module(x,k,kx,ky,stride,chandim,padding="same"):

# #conv-bn-relu

# x = Conv2D(k,(kx,ky),strides=stride,padding=padding)(x)

# x = BatchNormalization(axis=chandim)(x)

# x = Activation('relu')(x)

# return x

# def inception\_module(x,num1x1,num3x3,chandim):

# conv\_1x1 = conv\_module(x,num1x1,1,1,(1,1),chandim)

# conv\_3\_3 = conv\_module(x,num3x3,3,3,(1,1),chandim)

# x = concatenate([conv\_1x1,conv\_3\_3],axis=chandim)

# return x

# def downsample\_module(x,k,chandim):

# #conv downsample and pool downsample

# conv\_3x3 = conv\_module(x,k,3,3,(2,2),chandim,padding='valid')

# pool = MaxPooling2D((3,3),strides=(2,2))(x)

# x = concatenate([conv\_3x3,pool],axis=chandim)

# return x

# #然后定义整个MiniGoogLeNet

# def minigooglenet\_functional(width,height,depth,classes):

# inputshape=(height,width,depth)

# chandim=-1

# #define inputs and firse conv

# inputs = Input(shape=inputshape)

# x = conv\_module(inputs,96,3,3,(1,1),chandim)

# #def two inception and followed by a downsample

# x = inception\_module(x,32,32,chandim)

# x = inception\_module(x,32,48,chandim)

# x = downsample\_module(x,80,chandim)

# #def four inception and one downsample

# x = inception\_module(x,112,48,chandim)

# x = inception\_module(x,96,64,chandim)

# x = inception\_module(x,80,80,chandim)

# x = inception\_module(x,48,96,chandim)

# x = downsample\_module(x,96,chandim)

# #def two inception followed by global pool and dropout

# x = inception\_module(x,176,160,chandim)

# x = inception\_module(x,176,160,chandim)

# x = AveragePooling2D((7,7))(x)

# x = Dropout(0.5)(x)

# #softmax

# x = Flatten()(x)

# x = Dense(classes)(x)

# x = Activation('softmax')(x)

# #create the model

# model = Model(inputs,x,name='MiniGoogLeNet')

# return model

# Inference 这个函数创建对输入表示创建一个预测。它对每个输入返回一个 1×10的张量。张量中的值会传递给损失函数，计算这个预测与基准相差多少。

# 根据参数batch\_size，每次需要处理128个图像。这个技术称为mini-batch。

# 在更小的数据集上batch上处理输入数据，而不是整个数据集，这样就可以有效使用内存。

# 因为是在每个小型数据集上更新权重，而不是处理所有样本，模型也能够更快收敛。

# Loss 可以用softmax cross entropy 函数执行N路分类。Softmax可以归一化（将tensor加成1个数值）从inference函数产生的输入数据。

# 张量归一化后，Cross entropy 计算编码标注。cross entropy计算预测值与基准值之间的差异。每次迭代，优化器都最小化cross entropy的值。

# 调用Inception

model = Inception(2, num\_labels) # 第一参数为残差块数，第二个参数为类别数；

# derive input shape for every layers.

# print(width,height,channel)

model.build(input\_shape=(None, width, height, channel))

model.summary()

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=200, callbacks=[save\_model\_cb])

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print("准确率: %.4f，共测试了%d张图片 " % (test\_acc, len(y\_test)))

# 附录二 VDS1.0 源码

**VDS1.0 bodyStyle分类器的训练**

# 加载data、label

import scipy.io as io

data\_mat\_path = '../mats/'

label\_mat\_path = '../labels/'

data\_filename = 'data\_old.mat'

label\_filename = 'label\_old.mat'

data = io.loadmat(data\_mat\_path+data\_filename)["name"]

label = io.loadmat(label\_mat\_path+label\_filename)["name"]

for i in range(64467): # 去除因对齐添加的空格

for j in range(16):

label[i][j] = label[i][j].strip()

# 取shape一列

row=len(label)

y=['' for i in range(row)]

for i in range(row):

y[i]=label[i][15]

from sklearn.preprocessing import LabelBinarizer

encoder = LabelBinarizer()

# y = encoder.fit\_transform(y)

# 划分训练集测试集

from sklearn.model\_selection import train\_test\_split

X\_bs\_train, X\_bs\_test, y\_bs\_train, y\_bs\_test = train\_test\_split(

data, y, test\_size=0.2)

# 归一化

X\_bs\_train=X\_bs\_train/255

X\_bs\_test=X\_bs\_test/255

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import AveragePooling2D

from tensorflow.keras.layers import BatchNormalization

from tensorflow.keras.layers import Activation

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Input

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import concatenate

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

row,height, width, channel=data.shape

input\_shape = (height, width, channel)

# 减小复杂度解决过拟合

# 增加dropout

# 构造网络

bodyStyleModel = Sequential()

# 第一层

bodyStyleModel.add(Conv2D(filters=32,

input\_shape=input\_shape,

kernel\_size=(3, 3),

strides=(1, 1),

activation='relu',

padding='same'

)

)

# 中间各层

bodyStyleModel.add(Conv2D(filters=32,

kernel\_size=(5, 5),

strides=(2, 2),

activation='relu',

padding='same'

)

)

bodyStyleModel.add(MaxPooling2D(pool\_size=(2, 2)))

bodyStyleModel.add(Dropout(0.3))

bodyStyleModel.add(Conv2D(filters=64,

kernel\_size=(5, 5),

strides=(1, 1),

activation='relu',

padding='same'

)

)

bodyStyleModel.add(Conv2D(filters=64,

kernel\_size=(3, 3),

strides=(2, 2),

activation='relu',

padding='same'

)

)

bodyStyleModel.add(MaxPooling2D(pool\_size=(2, 2)))

bodyStyleModel.add(Dropout(0.3))

bodyStyleModel.add(Flatten())

bodyStyleModel.add(Dense(512, activation='relu'))

bodyStyleModel.add(Dropout(0.4))

bodyStyleModel.add(Dense(len(y[0]), activation='softmax'))

from tensorflow.keras.optimizers import SGD

learning\_rate = 0.01

decay = 1e-6

momentum = 0.8

nesterov = True

sgd\_optimizer = SGD(lr=learning\_rate, decay=decay,

momentum=momentum, nesterov=nesterov)

bodyStyleModel.compile(optimizer=sgd\_optimizer,

loss="categorical\_crossentropy",

metrics=['accuracy']

)

batch\_size = 50 # 每批训练数据量的大小

epochs = 200

bodyStyleModel.fit(X\_bs\_train,

y\_bs\_train,

epochs=epochs,

batch\_size=batch\_size,

shuffle=True,

# validation\_data=(valid\_input, valid\_output)

)

print(bodyStyleModel.evaluate(X\_bs\_test,y\_bs\_test,verbose=0))

# 显示loss和acc

import matplotlib.pyplot as plt

fig = plt.figure()

ax1 = fig.add\_subplot(111)

ax1.plot(history.history['loss'], label='loss',color='r')

ax1.legend(loc=1)

ax1.set\_ylabel('Loss')

ax2 = ax1.twinx()

ax2.plot(history.history['accuracy'], label='accuracy')

ax2.legend(loc=2)

ax2.set\_ylabel('Accurracy')

#保存模型

MODEL\_PATH = 'model/bodystyle\_model\_85.h5'

bodyStyleModel.save(MODEL\_PATH)

**# VDS1.0-预测模块**

import cv2

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras import Sequential

from PIL import Image

bodystyle\_model=Sequential()

MODEL\_PATH='../model/bodystyle\_model.h5'

bodystyle\_model=load\_model(MODEL\_PATH)

# 预测分类号

def predict\_num(file\_name):

path=file\_name

img = Image.open(path)

img = img.resize((80, 60), Image.ANTIALIAS)

arr = np.asarray(img, dtype='float32')

arr=arr/255

arr=arr.reshape(1,60,80,3)

result = bodystyle\_model.predict\_classes(arr)

return result[0]

#预测分类名称

def read\_image(file\_name):

path=file\_name

img = Image.open(path)

img = img.resize((80, 60), Image.ANTIALIAS)

return img

def predict\_class(img):

# path=file\_name

# img = Image.open(path)

# img = img.resize((80, 60), Image.ANTIALIAS)

arr = np.asarray(img, dtype='float32')

arr=arr/255

arr=arr.reshape(1,60,80,3)

result = bodystyle\_model.predict\_classes(arr)

# print(result[0])

return bodyStyleDict[result[0]]

from PIL import Image, ImageFont, ImageDraw # 导入模块

print(img.format, img.size, img.mode)

draw = ImageDraw.Draw(img) #修改图片

#font = ImageFont.truetype(None, size = 40)

draw.text((20,50), result, fill = (255, 0 ,0)) #利用ImageDraw的内置函数，在图片上写入文字

img.show()

# im.save('mysmile.jpg', jpg)

# 附录三 VDS 2.0源码

**# 数据预处理**

import scipy.io as io

import os

import gc

from PIL import Image

import numpy as np

train\_label\_path = '../compcars/devkit/cars\_train\_annos.mat'

test\_label\_path = '../compcars/devkit/cars\_test\_annos.mat'

train\_label = io.loadmat(train\_label\_path)

train\_label\_anno=train\_label['annotations']

for i in range(nums):

annotations = train\_label\_anno[0][i]

xmin = annotations[0][0][0]

ymin = annotations[1][0][0]

xmax = annotations[2][0][0]

ymax = annotations[3][0][0]

# xmin = xmin/width

# ymin = ymin/height

# xmax = xmax/width

# ymax = ymax/height

xmins.append(xmin)

ymins.append(ymin)

xmaxs.append(xmax)

ymaxs.append(ymax)

classes.append(annotations[4][0][0])

filenames.append(annotations[5][0])

# 生成边框width和height

# box\_width = xmaxs-xmins

# box\_height = ymaxs-ymins

box\_width = []

box\_height = []

for i in range(nums):

box\_height.append(ymaxs[i]-ymins[i])

box\_width.append(xmaxs[i]-xmins[i])

# # 每个图片对应一个txt文件，第一个数字代表类别，后面的四个数据是标注框的信息，分别是标注框左上角的x，y坐标，标注框的宽w，高h。

# 每一行是一个标注对象，# 如果一张图片中有多个目标，则txt就对应有多少行。

# 12 0.3 0.56 0.440 0.70

# 2 0.56 0.45 0.67 0.70

path='../compcars/cars\_train/'

for i in range(nums):

img\_path = path+filenames[i]

img = Image.open(img\_path)

img\_width, img\_height = img.size

c = str(classes[i])

x = str(xmins[i]/img\_width)

y = str(ymins[i]/img\_height)

w = str(box\_width[i]/img\_width)

h = str(box\_height[i]/img\_height)

row = ''+c+' '+x+' '+y+' '+w+' '+h

# txt\_filename=img\_path+'.txt'

txt\_filename = img\_path[:-3]+'txt'

f=open(txt\_filename, "w+")

f.write(row)

f.close()

f = open('train.txt', "a+")

for i in range(nums):

row = filenames[i]

f.write('data/compcars/'+row+'\n')

f.close()

meta\_label\_path = '../compcars/devkit/cars\_meta.mat'

train\_meta = io.loadmat(meta\_label\_path)

class\_dict=train\_meta ['class\_names'][0]

**# 训练模块**

# def read\_yolocfg(cfg\_path, classes):

# 暂时先用手动配置的方式

# Ctrl+F 搜索yolo，总共可以搜索到3个

# 在每个搜索结果修改classes为当前使用数据集中分类数

# 在每个搜索结果的上一个卷积层，修改filiter为（classes+5）\*3

# 保存

# def change\_mode(mode,batch=32,subdivisions=16):

# 训练时 batch=32,subdivisions=16，内存较大时，batch可设置为64.

# 测试时，batch=1，subdivisions=1

def create\_datafile(path, classes, names, train='data/train.txt', valid='data/valid.txt', backup='backup/'):

f = open(path, "w+")

f.write('classes= '+classes+'\n')

f.write('train= '+train+'\n')

f.write('valid= '+valid+'\n')

f.write('names= '+names+'\n')

f.write('backup= '+backup)

f.close()

return

# 手动修改darknet-master\build\darknet\x64\data\voc\voc\_label.py

# 将第10行classes改为names文件中的分类名

# 检查cfg文件batch是否为32或者64，如果是在测试模式，请手动修改为训练模式

#

def start\_training(path, classes, names, weights,data,cfg, train='data/train.txt', valid='data/valid.txt', backup='backup/', nocfg=False):

if nocfg:

create\_datafile(path, classes, names, train, valid)

# command = 'cd D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/ &\

# darknet.exe detector train data/compcars.data yolov3-compcars.cfg backup/yolov3-compcars\_last.weights'

command = 'cd D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/ &\

darknet.exe detector train '+data+' '+cfg+' '+weights

# print(command)

with os.popen(command, "r")as f:

text = f.read()

print(text) # cmd输出结果

return

# nocfg=true的话，系统会生成data文件

# 如果data文件已存在，请设置nocfg=False

weights = 'backup/yolov3-compcars\_last.weights'

data = 'data/compcars.data'

cfg = 'yolov3-compcars.cfg'

start\_training(path, classes, names, weights, data,

cfg, train, valid, nocfg=False)

**# 测试模块**

def predict(data, cfg, weights, filepath, mode='test'):

command = 'cd D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/ &\

darknet.exe detector '+mode+' '+data+' '+cfg+' '+weights + ' '+filepath

print('执行命令:\n', command)

with os.popen(command, "r")as f:

text = f.read()

print(text) # cmd输出结果

return text

weights = 'backup/yolov3-compcars\_last.weights'

data = 'data/compcars.data'

cfg = 'yolov3-compcars.cfg'

img\_path='D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/data/compcars/cars\_test/00001.jpg'

# 展示原图

from PIL import Image

import os

img = Image.open(img\_path)

img

text = predict(data, cfg, weights, img\_path, mode='test')

vidio\_path = 'D:/Projects/YOLO-darknet/darknet-master/build/darknet/x64/data/test.mp4'

text = predict(data, cfg, weights, vidio\_path, mode='demo')