

浙江大學

本科生毕业论文



题目: 面向实时行人检测的快速卷积神经网络

姓 名: 叶东强子 3130104011

学 号: 叶东强子

指导教师: 朱建科

专 业: 2013 级 计算机科学与技术

学 院: 计算机科学与技术学院

A Dissertation Submitted to Zhejiang University for the Degree of Bachelor of Engineering



TITLE: Fast Pedestrian Detection
Based on Convolution Neural Network

Author: Dongqiangzi Ye 3130104011

Supervisor: Jianke Zhu

Major: Computer Science and Technology

College: College of Computer Science and Technology

Submitted Date: _____

浙江大学本科毕业生毕业论文(设计)诚信承诺书

1. 本人郑重地承诺所呈交的毕业论文(设计),是在指导教师的指导下严格按照学校和学院有关规定完成的。
2. 本人在毕业论文(设计)中引用他人的观点和参考资料均加以注释和说明。
3. 本人承诺在毕业论文(设计)选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。
4. 在毕业论文(设计)中对侵犯任何方面知识产权的行为,由本人承担相应的法律责任。

毕业论文(设计)作者签名:

_____年____月____日

摘 要

此处填入论文摘要 (字数在 300 字以内)

关键词： 此处填入论文关键词

Abstract

此处填入英文摘要

Keywords: 此处填入英文关键词

目 录

摘要	I
Abstract	II
目录	IV
第 1 章 绪论	1
1.1 课题背景	1
1.2 本文研究目标和内容	3
1.3 本文结构安排	3
第 2 章 技术路线	5
2.1 相关工作	5
2.2 关键技术	7
2.3 项目框架	9
2.4 KITTI 数据集	9
2.5 本章小结	10
第 3 章 研究方案	11
3.1 数据预处理	11
3.2 网络结构	14
3.3 验证方法	17
3.4 本章小结	18
第 4 章 实验结果与分析	19
4.1 实验平台	19
4.2 结果对比	20
4.3 预测效果	24
4.4 泛化性	26
4.5 本章小结	29

第 5 章 本文总结	30
5.1 工作成果总结	30
5.2 未来工作	30
参考文献	32
致谢	33

第1章 绪论

1.1 课题背景



图 1.1 Google 无人车

当下机器学习发展迅猛，机器学习渗透进入了许多领域，为很多问题的处理带来了新思路、新方案，是未来计算机领域发展的一个很有前景的趋势。而在机器学习普及的同时，智能驾驶成为一个非常热门的方向。智能驾驶是一个非常复杂的系统，包括对实时数据的收集（如传感器），对当前状况的预测（如行人检测），对之后状态的决策（如轨迹生成），控制车辆操作。近来全球都刮起了一股无人车驾驶的热潮，从最开始的实验室领头，到后来国外的 google、特斯拉、Uber，国内的百度、图森、滴滴，智能驾驶成为一个非常火热的方向。不久前刚在美国加州领到无人车驾驶证的百度，又或者曾经在 KITTI 数据集第一的图森，又或者是已经将无人车商业化的特斯拉都在致力于成为无人驾驶领域的领军企业。而行人检测就是自动驾驶（或辅助驾驶）中

相当重要的一环,而保证行人检测的准确性和实时性自然是相当重要的事情,本项目便是基于智能驾驶的背景,探究行人检测的新算法、新模型,实现实时的行人检测。

行人检测是一种典型的物体检测。行人检测具有极其广泛的应用:智能辅助驾驶,智能监控,行人分析以及智能机器人等领域,在过去几年中引起了广泛的关注。目前传统的行人检测方法大多是使用的滑动窗口,特征提取,然后分类建模,如积分通道特征的方法。对于相关论文的研究 [9, 1],传统行人检测算法能分为两类:

1. 基于背景建模:利用背景建模方法,提取出前景运动的目标,在目标区域内进行特征提取,然后利用分类器进行分类,判断是否包含行人
2. 基于统计学习:这也是目前行人检测最常用的方法,根据大量的样本构建行人检测分类器。提取的特征主要有目标的灰度、边缘、纹理、颜色、梯度直方图等信息。

而实时行人检测也是一个比较复杂的问题。在实验过程中,使用的图片大多都是光照充足,物体特征明确的情况。而在实际的问题中,我们不得不面对光照改变的情况,或者天气变化的情况,甚至行人之间也会有重合的情况。实际的复杂情况大大加大了行人检测预测准确的难度,而车辆驾驶的实时性也对行人检测预测的速度要求更高。

得益于硬件设备的提升,深度学习在近几年发展迅猛,深度学习通过构建一个深层的网络结构,通过监督学习的方法来构建一个物体检测的模型。而卷积神经网络就是其中很重要的一个分支,卷积神经网络与其他神经网络不同之处在于它通过每一层的卷积核去处理图像信息。由于对局部特征处理效果好,参数少训练快,网络结构复杂可塑性强,卷积神经网络成为处理图像信息的一个主流的方向。在物体检测上,基于卷积神经网络的模型 [7, 8, 5, 2, 3] 也是层出不穷,效果越来越好,速度越来越快。

卷积神经网络在目标识别检测上取得了巨大的成功,而目前行人检测领域的主流结果仍采用积分通道特征。本项目将从之前的研究成果出发,参考近来比较火热的物体检测的卷积神经网络,研究面向主动驾驶安全的行人检测算法。本项目将参考 YOLO 的网络模型,在主流的行人数据集上,对比各模型的准确率和速度,探讨实时行人检测的方案与技术。

针对上述问题以及技术背景,在导师的指导下,提出了本项目的设计课题:面向实时行人检测的快速卷积神经网络研究。

1.2 本文研究目标和内容

本项目的全称为面向实时行人检测的快速卷积神经网络研究。该项目是从 KITTI 的数据集中下载训练和测试的图片和标签,将数据集预处理成 darknet 可读入的形式。再配置 darknet 的环境,修改 darknet 的参数,修改网络结构,训练网络,最后验证数据集,检验 AP、FPS 值,得到实验结果。总的来说,该项目是基于 darknet 的深度学习框架下,参考 YOLO 的网络结构基础,使用 KITTI 的数据集来研究实时行人检测的实验。

具体而言,该模型目标如下:

1. 实现带有物体检测的卷积神经网络
2. 测试不同网络结构的性能与速度
3. 检验网络的泛化性,针对特殊场景、特殊光照、特殊天气条件的检测效率,优化训练数据采样策略

研究的内容:

1. 配置 darknet 环境
2. 预处理 KITTI 数据集
3. 修改网络结构
4. 设置评价函数,评估检验结果
5. 对比实验性能和速度

1.3 本文结构安排

本论文的组织结构如下:

第一章介绍面向实时行人检测的快速卷积神经网络研究的背景与内容,主要包括本项目所要解决的实际需求,项目意义和难点创新点以及本项目需要完成的工作内容等。

第二章介绍项目的相关文献,概况之前研究的成果。

第三章重点描述研究方案和项目的理论依据,概况项目的流程和算法。

第四章介绍实验参数及环境,对比和分析测试结果,分析实验效果,探究实验的泛化性特征。

第五章为项目总结,包括对目前成果总结,对项目不足之处的分析和相关改进思路以及系统可能发展的讨论。

其中第三章到第五章为本文作者的重点工作。

第 2 章 技术路线

2.1 相关工作

2.1.1 HOF 特征和 CSS 特征 +HIKSVM 分类器 [9]

该研究以多种方式推动了行人检测的最新技术: 该研究已经为行人检测引入了强大的自相似特征, 当应用于色彩通道时, 在单帧设置和附加运动信息方面提供了显著的改进。精心实现的 HOG 特征, HOF 特征图像运动的变体和新的 CSS 特征以及 HIKSVM 作为分类器的组合, 超过了之前技术 5%-20% 的精度。

2.1.2 积分通道特征 [1]

积分通道特征的大概思路是: 通过对输入图像做各种线性和非线性的变换, 诸如局部求和、直方图、haar-like 及它们的变种之类的特征便可以通过积分图来快速计算出来。给定一个输入图像 I , 其所对应的通道是原始输入图像的某种输出响应。对于灰度图而言, 其对应的通道 $C=I$ 。而对于彩图而言, 其每个颜色通道则对应一个通道。其他类似的通道可以通过各种线性和非线性的方法计算而来。令 Ω 代表图像的某种通道计算函数, 则, $C = \Omega(I)$ 。为了能利用滑动窗口快速计算, 通道应该具有变换不变性, 即, 对于原图 I 及其对应的某种变换 I' 而言, $C = \Omega(I)$ 和 $C' = \Omega(I')$ 应该成立, 如此一来, 便允许 Ω 在整个图像上计算一次, 从而避免了变换之后的重复计算。

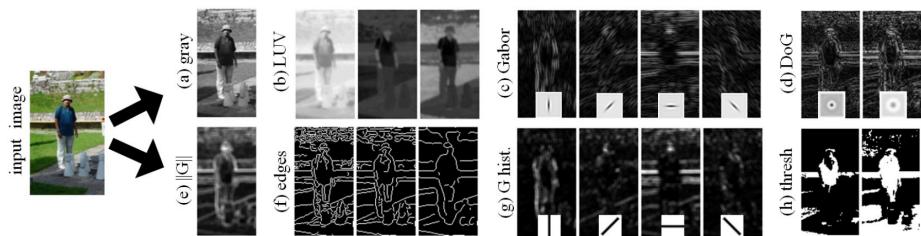


图 2.1 积分通道特征

2.1.3 Faster R-CNN[8]

Faster R-CNN(其中 R 对应于“Region(区域)”)是基于深度学习 R-CNN 系列目标检测最好的方法。使用 VOC2007+2012 训练集训练,VOC2007 测试集测试 mAP 达到 73.2%, 目标检测的速度可以达到每秒 5 帧。技术上将 RPN 网络和 Fast R-CNN 网络结合到了一起, 将 RPN 获取到的 proposal 直接连到 ROI pooling 层, 是一个 CNN 网络实现端到端目标检测的框架。

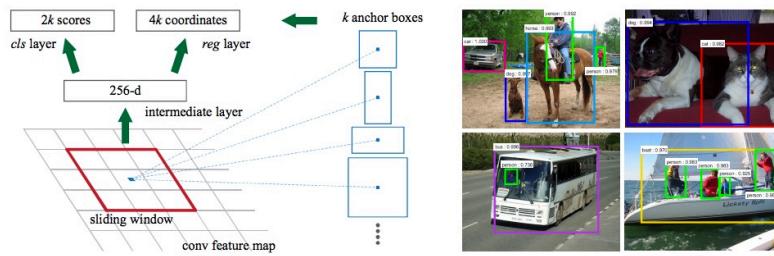


图 2.2 Faster R-CNN

2.1.4 SSD[5]

SSD 也是一种深度学习的神经网络方法。它将原网络后再加入几层特征卷积层, 原全连接层提取的特征进入特征提取层, 能提取原图各个尺度的特征, 通过这些特征计算出检测的物体。

2.1.5 YOLO[7]

YOLO 将对象检测重新映射为单个回归问题, 直接从图像像素到边界框坐标和类概率。YOLO 将预测该物体是什么对象以及它在什么位置。

YOLO 非常简单: 单个卷积神经网络同时预测了这些框的各个边界框和类概率。YOLO 通过完整的图片训练, 直接优化检测性能。这种统一的模型比传统的对象检测方法有好几个好处。

首先, YOLO 非常快。由于 YOLO 将框检测作为回归问题, YOLO 不需要复杂的流水线。

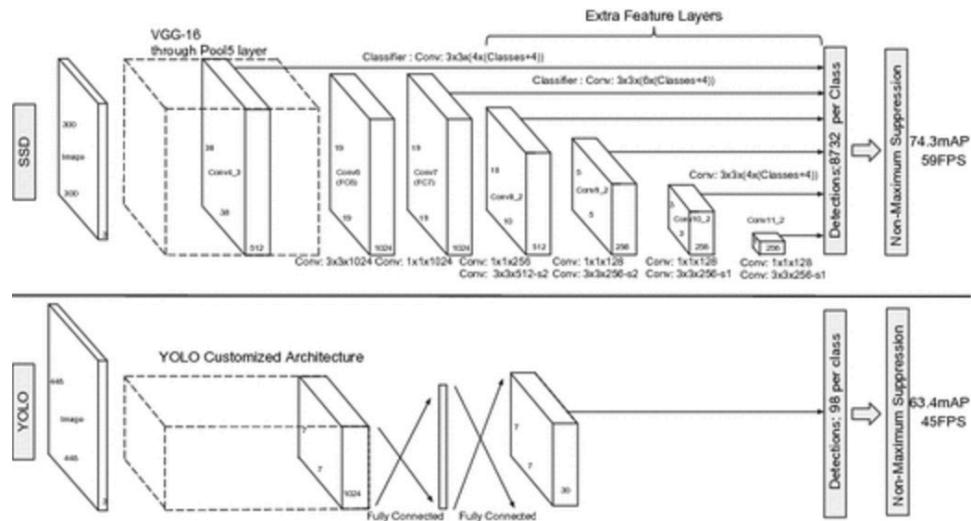


图 2.3 SSD 网络模型

第二, YOLO 在做出预测时,会全局地推理图片信息。与滑动窗口和基于区域提案的技术不同, YOLO 在训练和测试时间期间处理整个图像,因此它可以编码关于类及其外观的语义信息。

第三, YOLO 的泛化性更强。

2.2 关键技术

YOLO 将对象检测的单独组件统一为单个神经网络。YOLO 网络使用整个图像的特征来预测每个边界框。它还同时预测图像的所有边界框。这意味着 YOLO 的网络全局地推断整张图像和所有对象。

如图 2.5, YOLO 将整个输入图像分为 $S \times S$ 格。如果物体的中心落入网络单元格中,则该网络单元负责检测该对象。

每个网格单元预测 B 边界框和置信度分数。这些置信度分数反映了模型对于框包含对象的概率,以及它认为边界框预测的准确程度。我们将置信度定义为 $Pr(\text{Object}) * IOU_{pred}^{truth}$ 。如果该单元格中没有对象,则置信度分数应为零。否则,我们希望置信度分数等于预测框和真值之间的交集(IoU)。

每个边界框包括 5 个预测: x, y, w, h 和置信度。 (x, y) 坐标表示相对于网格单元

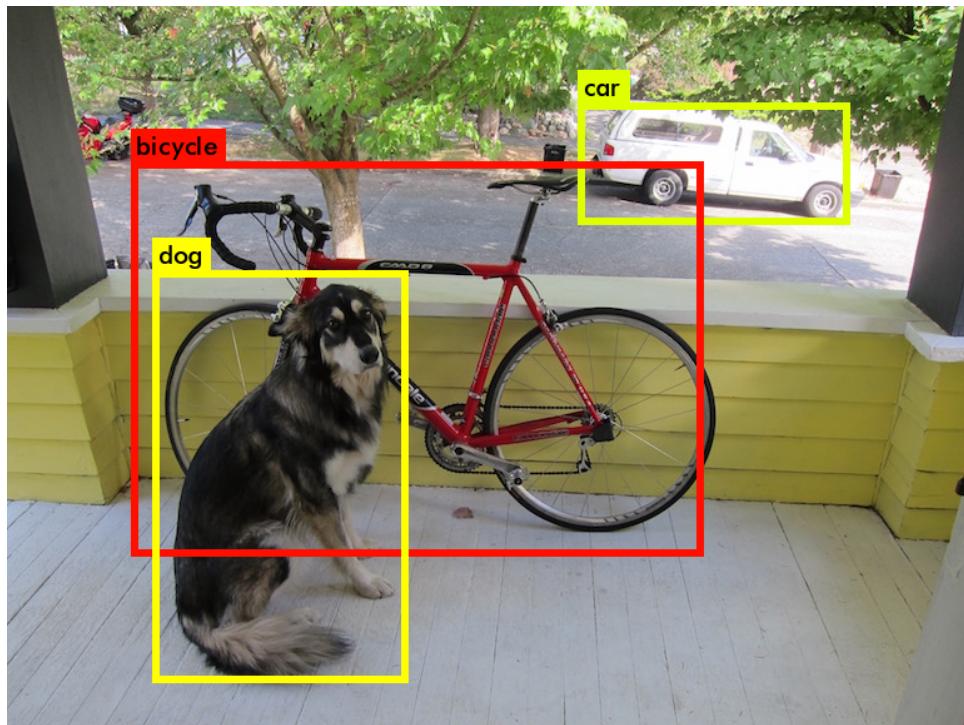


图 2.4 YOLO 检测

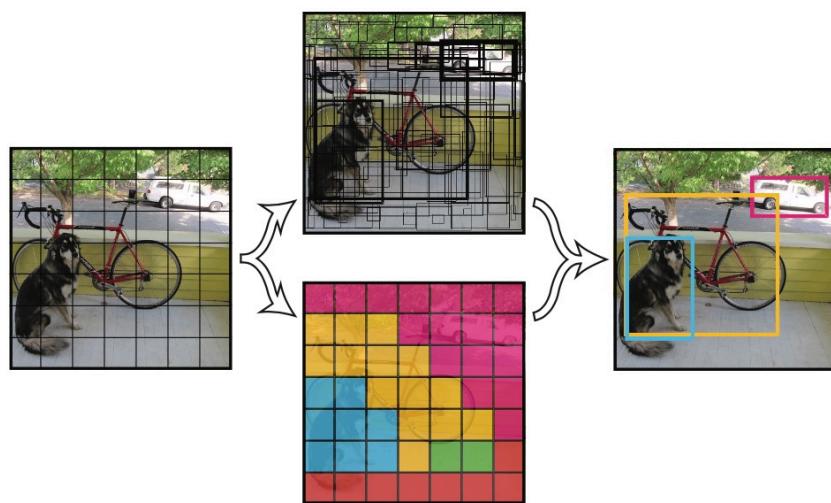


图 2.5 YOLO 模型

边界的框的中心。从整个图像预测宽度和高度。最后置信度预测表示预测框和任何真值框之间的 IOU。

每个网格单元还预测了 C 类的条件概率, $Pr(Class_i|Object)$ 。这些概率在包含物体的网格单元格上被约束。我们只预测每个网格单元的一组类概率,而不考虑框的数量 B。

在测试的时候,我们乘以类的条件概率和单独的框置信度, $Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$ (1) 这给了我们每个边界框的特定类的置信分数。这些分数对该类出现在框中的概率进行了编码,并且预测框有多符合该物体。

2.3 项目框架

Darknet[6] 是一个基于 C 语言和 CUDA 的开源神经网络框架。它支持 CNN、RNN、YOLO 等各种神经网络。YOLO 作者使用的框架,同时也是 YOLO 效果最好的框架。Darknet 支持 GPU 加速,支持 OpenCV,同时由于它是个轻量级的框架配置较为方便,各个操作系统也都能配置,可移植性强。同时本项目是基于 YOLO 的网络结构的实验,所以选择了 Darknet 作为本项目的实验框架。

2.4 KITTI 数据集

KITTI[4] 利用自主驾驶平台 Annieway 来开发新的具有挑战性的现实世界的计算机视觉基准。KITTI 感兴趣的任务是:立体声,光流,视觉测距,3D 物体检测和 3D 跟踪。为此,KITTI 配备了两台高分辨率彩色和灰度摄像机的标准车厢。Velodyne 激光扫描仪和 GPS 定位系统提供了准确的地面上实况。卡尔斯鲁厄市中心的农村地区和高速公路上都是通过驾驶卡车来捕捉的。每张图片最多可显示 15 辆汽车和 30 名行人。除了以原始格式提供所有数据,KITTI 提取每个任务的基准。对于我们的每个基准,KITTI 还提供评估指标和评估网站。初步实验表明,在实验室迁移到现实世界之前,诸如米德尔伯里等已建立基准的方法排名高于平均水平。我们的目标是通过向社会提供新的困难的现实基准,来减少这种偏差并补充现有的基准。

KITTI 包含市区、乡村和高速公路等场景采集的真实图像数据,每张图像中最

多达 15 辆车和 30 个行人,还有各种程度的遮挡与截断。整个数据集由 389 对立体图像和光流图,39.2 km 视觉测距序列以及超过 200k 3D 标注物体的图像组成,以 10Hz 的频率采样及同步。总体上看,原始数据集被分类为'Road', 'City', 'Residential', 'Campus' 和 'Person'。对于 3D 物体检测, label 细分为 car, van, truck, pedestrian, pedestrian(sitting), cyclist, tram 以及 misc 组成。

2.5 本章小结

本章主要介绍了之前的一些研究工作。接着描述了本项目的关键技术,然后介绍了本项目的框架 Darknet,最后简单介绍了本项目的数据集 KITTI。

基于本项目的需求:性能、速度。本项目将参考 YOLO 的网络结构,在 Darknet 上进行网络的训练和测试,使用 KITTI 数据集作为我们训练和测试的数据集。

第 3 章 研究方案

本项目分为四块内容: KITTI 数据集的转换以及数据预处理, YOLO 关键技术的实现, 网络结构的调整与优化, 测试模型的评估方法。

3.1 数据预处理

3.1.1 KITTI 数据集详解

#Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Here, 'DontCare' labels denote regions in which objects have not been labeled, for example because they have been too far away from the laser scanner. To prevent such objects from being counted as false positives our evaluation script will ignore objects detected in don't care regions of the test set. You can use the don't care labels in the training set to avoid that your object detector is harvesting hard negatives from those areas, in case you consider non-object regions from the training images as negative examples.

图 3.1 KITTI 格式

KITTI 数据集中共有 7480 张图片可以用来训练和测试。图 3.1 展示了 KITTI 数据集的典型样本, 分为 'Road', 'City', 'Residential', 'Campus' 和 'Person' 五类。原始数据采集于 2011 年的 5 天, 共有 180GB 数据。

数据图像如图 4.6, 每张图片大小在 1224*370 左右。每张图片最多可显示 15 辆汽车和 30 名行人。



图 3.2 KITTI 图像

```
007347.txt ▾
Car 0.00 1 2.72 833.99 146.26 930.90 178.56 1.52 1.76 4.10 13.72 -0.12 35.10 3.09
Car 0.00 0 -1.51 538.26 168.34 567.42 191.04 1.50 1.78 3.69 -3.58 0.69 49.29 -1.58
Cyclist 0.00 0 -0.02 600.56 165.32 624.34 188.37 1.74 0.60 1.79 0.59 0.59 54.33 -0.01
Pedestrian 0.00 1 3.13 600.06 165.36 610.52 188.69 1.80 0.68 1.44 0.25 0.63 55.44 3.14
Pedestrian 0.00 2 0.25 463.56 170.65 472.99 196.09 1.92 0.57 1.32 -10.02 1.18 54.01 0.07
DontCare -1 -1 -10 407.49 171.65 440.41 206.62 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 3.57 195.29 87.88 228.21 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 782.62 156.24 829.93 175.80 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 514.38 166.51 539.07 185.04 -1 -1 -1 -1000 -1000 -1000 -10
```

图 3.3 KITTI 标签

数据标签如图 3.3, 每行是一个单独的物体。第一列是类名, 而 KITTI 总共把所有的物体分了八个类(还有一个类名是 DontCare, 表示该区域没有被标注, 防止假阳性); 第二列代表物体超出边界多少; 第三列表示物体被覆盖的情况; 第四列是观察物体的角度; 第五列到第八列是物体在图片中的位置(Xmin, Ymin, Xmax, Ymax); 第九列到第十一列是三维物体的长宽高; 第十二列到第十四列是三维物体在摄像机中的坐标(x, y, z), 第十五列是物体在摄像机的坐标中离 y 轴的弧度。

3.1.2 数据转换和预处理

在实验过程中, 我们将前 7000 张图片作为训练图片, 后 480 张图片作为测试图片。

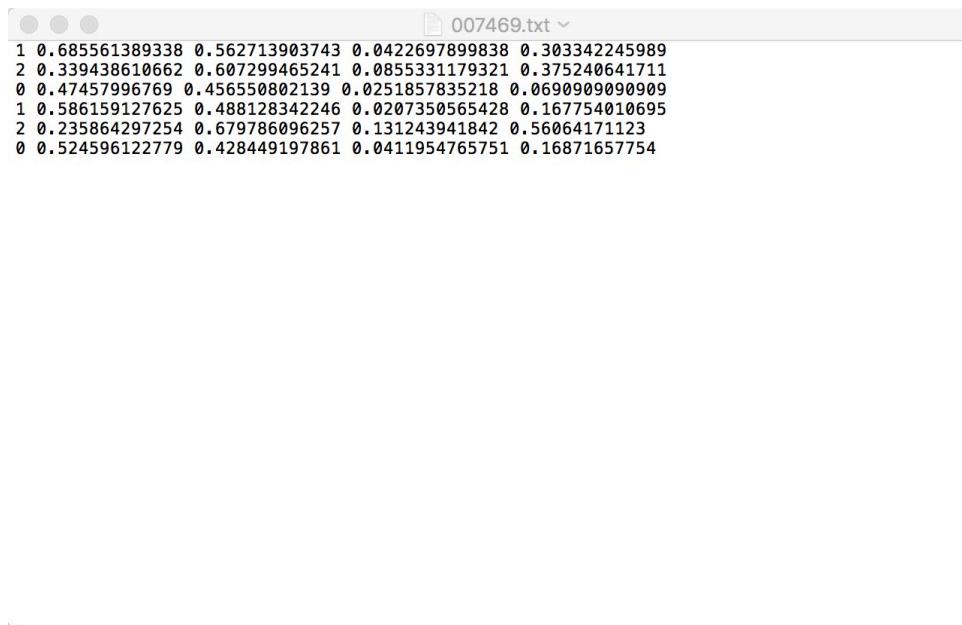


图 3.4 Darknet 标签

首先, 我们需要把数据集转换为 Darknet 所需要的形式, 如图 3.4。Darknet 中: 每行第一列是类的序号, 第二列是框中心的 x 坐标, 第三列是框中心的 y 坐标, 第四列是框的宽度, 第五列是框的高度。另外这四个数据都需要归一化到 0-1 之间。所以我们需要将 KITTI 标签中的(Xmin, Ymin, Xmax, Ymax)四个坐标进行下面的变换:

$$X = (X_{min} + X_{max}) / (2 * Weight)$$

$$Y = (Y_{min} + Y_{max}) / (2 * Height)$$

$$W = (X_{max} - X_{min}) / Weight$$

$$H = (Y_{max} - Y_{min}) / Height$$

来变换到归一化后的(X, Y, W, H)。另外由于 KITTI 数据集中的类过多, 我们适当调整为仅有三类: Car、Pedestrian、Cyclist, 对应到 0、1、2 的标签。将原数据集中的 Car、Van、Truck、Tram 转换成 Car; Pedestrian、Person_sitting 转换成 Pedestrian; Cyclist 转换成 Cyclist; 同时忽略掉 Misc 和 DontCare 的类。

```
train.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000000.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000001.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000002.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000003.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000004.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000005.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000006.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000007.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000008.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000009.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000010.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000011.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000012.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000013.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000014.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000015.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000016.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000017.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000018.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000019.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000020.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000021.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000022.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000023.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000024.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000025.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000026.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000027.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000028.txt
/Users/EowinYe/Downloads/data_object_image_2/training/image_2/000029.txt
```

图 3.5 Darknet 标签

然后, 我们需要生存可以供 Darknet 读入的 train_list 和 test_list, 如图 3.5。这里直接将图片的前 7000 张作为训练图片, 后 480 张作为测试图片然后生成对应的 list 文件即可。

3.2 网络结构

网络结构参考 YOLO 的网络结构, 在前面的章节我们已经讨论过 YOLO 的关键技术, 这一节将主要讨论具体实现过程中的网络结构选择。

3.2.1 损失函数

损失函数是使用的平方误差。

我们优化了模型输出中的平方误差。我们使用平方误差，因为它很容易优化，但是它不能完全符合我们的最大化平均精度的目标。我们增加了边界框坐标预测的损失，并减少了对不包含对象的框的置信预测的损失。我们使用两个参数， λ_{coord} 和 λ_{noobj} 来完成这个。我们设置 $\lambda_{coord} = 5$ 和 $\lambda_{noobj} = 0.5$ 。平方误差也平等地对待大框和小框中的误差。我们的误差度量应该反映出，大框中的小偏差比小框小。为了部分解决这个问题，我们直接预测边界框宽度和高度的平方根，而不是宽度和高度。

在训练期间，我们优化以下多部分损失函数：

$$\begin{aligned}
 & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \\
 & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
 & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
 & + \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned}$$

其中 1_i^{obj} 表示如果对象出现在单元 i 中，并且 1_{ij}^{obj} 表示单元 i 中的第 j 个边界框预测器对于该预测是“负责的”。如果对象存在于该网格单元中，损失函数只会惩罚分类错误。如果该预测因子对于真值框是“负责的”，则它也只对边界框坐标误差进行估计（即具有该网格单元中的任何预测变量的最高 IOU）。

3.2.2 完整网络结构 YOLO

如图 3.6，参考 YOLO 和 GoogLeNet 模型 []，我们的网络有 24 个卷积层，其次是 2 个全连接层。部分卷积层后还连接了一些 1*1 的还原层。最后的卷积层使用了 40 个滤波器 (num=5 * (class=3 + coords=4 + bias_match=1))。除开最终层使用了线性激活函数，所有层使用以下 leaky 激活函数 3.7：

该网络比较大，实际训练过程中所需显存较多，训练出来的模型参数更多，理论上效果也会更好，不过速度会偏慢。

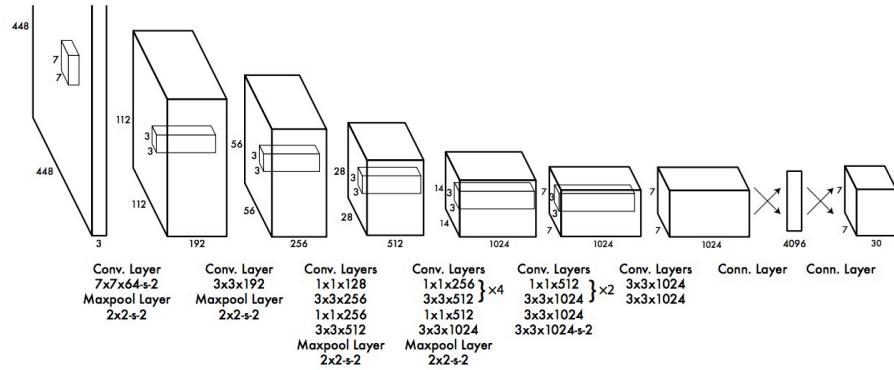


图 3.6 完整网络结构 YOLO

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

图 3.7 leaky 激活函数

3.2.3 缩小网络结构 tiny-YOLO

针对速度的要求,我们又采用了一种 tiny-YOLO 的网络来进行试验。

tiny-YOLO 仅有 9 个卷积层,并且每个卷积层使用了更少的滤波器(输出的特征),其它网络结构和之前的网络结构一样。理论上来说, tiny-YOLO 将实现非常快速度的检测,同时模型大小也将降低不少,不过将造成部分精度的损失。

3.3 验证方法

本项目将采用计算各个类别的 AP(Average Precision)的方法实现对检测结果的评估 []。

comp4_det_test_Pedestrian.txt						
007000	0.007039	281.713806	30.470688	758.445374	369.998291	
007001	0.006015	394.650696	1.000000	835.286316	375.000000	
007001	0.007974	184.124969	1.000000	1058.138428	375.000000	
007002	0.008022	20.230259	143.595108	81.493851	263.438049	
007002	0.009988	672.592041	145.850342	738.182007	195.371033	
007002	0.005635	747.313232	161.470245	844.262451	226.355957	
007002	0.005605	21.967134	169.955078	55.513687	354.650940	
007002	0.012218	762.952576	181.146454	873.789734	316.450714	
007002	0.006676	1201.522339	171.545410	1242.000000	375.000000	
007002	0.091169	742.487854	149.178375	933.687927	359.951660	
007003	0.015406	87.790077	182.501694	223.942719	250.834213	
007003	0.005315	145.894028	183.429016	306.845459	240.619751	
007003	0.010455	111.199463	193.145523	199.496582	304.123871	
007003	0.012998	297.782593	27.517700	746.401489	355.929199	
007003	0.005527	1.000000	1.000000	974.705933	375.000000	
007004	0.006379	111.119507	139.416962	172.093475	196.234863	
007004	0.035021	801.491394	143.631210	824.405334	207.263016	
007004	0.498468	875.536194	144.876770	908.349060	219.262268	
007004	0.144587	957.829285	150.985931	1007.594910	224.329285	
007004	0.047739	1062.591919	152.858185	1123.302124	219.098419	
007004	0.071278	1132.814087	153.814285	1224.000000	229.512405	
007004	0.023677	1102.858276	119.645523	1224.000000	262.942993	
007004	0.006874	1092.251099	160.011932	1217.853882	346.469116	
007004	0.021735	65.229370	1.000000	794.908997	370.000000	
007004	0.006349	3.195862	1.000000	866.547180	370.000000	
007005	0.027958	1174.179565	149.185562	1213.567749	248.314926	
007005	0.008835	1126.706421	80.934723	1235.491333	321.131775	
007008	0.016437	691.410278	167.001175	732.870850	220.203445	
007008	0.006946	1190.899048	158.421967	1226.818726	282.891205	
007008	0.005408	128.899963	1.000000	928.772034	376.000000	

图 3.8 行人验证结果

训练好的模型对于每张图片会返回各个框的检测结果(x,y,w,h,class,precision)。而在实际验证中,每个类别会返回所有大于阈值的检测结果,如图 3.8。每种类都有单独的文件存放检测结果,文件形式为(img_id, Xmin, Ymin, Xmax, Ymax, score)。我们就需要用这个检测结果和原标签对比,得到最终的实验结果。

对于每个类,AP 的具体算法如下:

1. 按照预测结果的 score 将预测结果按降序排序。

2. 对于每个预测结果 $(\hat{X}_{min}, \hat{Y}_{min}, \hat{X}_{max}, \hat{Y}_{max})$, 找到所对应的文件中该类的各个真值 $(X_{imin}, Y_{imin}, X_{imax}, Y_{imax})$, 计算覆盖量 IOU

$$intersection = (\max(\hat{X}_{min}, X_{imin}) - \min(\hat{X}_{max}, X_{imax})) * (\max(\hat{Y}_{min}, Y_{imin}) - \min(\hat{Y}_{max}, Y_{imax}))$$

$$union = (\hat{X}_{max} - \hat{X}_{min}) * (\hat{Y}_{max} - \hat{Y}_{min}) + (X_{imax} - X_{imin}) * (Y_{imax} - Y_{imin}) - intersection$$

$$IOU = union / intersection$$

3. 更新 TP (true positive), FP (false positive): 选取匹配的真值中 IOU 最高的 IOU。如果 IOU 大于阈值 (0.5) 并且该真值没有被检测到过 TP 就加一, 否则 FP 加一。

4. 计算 AP 值: 做出 PR 曲线, 找到 score 降序的数组中, 所有 TP 加一的位置, 然后将这些位置 i 乘以前 i 个预测的准确率, 再求和

$$AP = \sum_{IOU > thresh} 1 * pre(i) / N$$

其中 N 是真值的数量。

3.4 本章小结

本章主要讲述了本项目的整体研究流程。首先从数据集的处理上分析了 KITTI 数据集的特征, 以及转换成 YOLO 所需的格式的数据预处理等; 接着, 我们讨论了该项目网络的损失函数, 分析了所使用的各种网络结构; 最后, 我们讨论了测试结构的格式以及相对应的验证方法 AP。

第 4 章 实验结果与分析

本章将对实验进行检验并对比分析;将统计各个类的 AP 值并做出 PR 曲线;预测效果展示,并展现其中效果不佳的结果;探究实验泛化性能力。

4.1 实验平台

基本环境:

1. 操作系统: CentOS Linux release 7.1
2. 显卡: GTX1080
3. CUDA 版本: 8.0
4. 实验框架: Darknet
5. 数据集: KITTI

YOLO 训练参数:

1. batch size: 64
2. width: 416
3. height: 416
4. channels: 3
5. momentum: 0.9
6. decay: 0.0005
7. learning rate: 0.001
8. max batches: 50000

tiny-YOLO 训练参数:

1. batch size: 64
2. width: 416
3. height: 416
4. channels: 3
5. momentum: 0.9
6. decay: 0.0005
7. learning rate: 0.001
8. max batches: 50000

如图 4.1、4.2, 展现了两个模型的训练过程。

```
26: 524.731812, 530.401367 avg, 0.000000 rate, 6.280000 seconds, 1664 images
Loaded: 0.000000 seconds
Region Avg IOU: 0.077426, Class: 0.259889, Obj: 0.191617, No Obj: 0.495858, Avg Recall: 0.000000, count: 37
Region Avg IOU: 0.041767, Class: 0.283997, Obj: 0.165505, No Obj: 0.496245, Avg Recall: 0.000000, count: 30
Region Avg IOU: 0.114092, Class: 0.359812, Obj: 0.234454, No Obj: 0.496528, Avg Recall: 0.000000, count: 19
Region Avg IOU: 0.084310, Class: 0.263465, Obj: 0.271619, No Obj: 0.496076, Avg Recall: 0.025000, count: 40
Region Avg IOU: 0.104786, Class: 0.385687, Obj: 0.252009, No Obj: 0.497408, Avg Recall: 0.000000, count: 42
Region Avg IOU: 0.087030, Class: 0.317330, Obj: 0.154607, No Obj: 0.494996, Avg Recall: 0.000000, count: 41
Region Avg IOU: 0.058146, Class: 0.380191, Obj: 0.185132, No Obj: 0.496261, Avg Recall: 0.000000, count: 32
Region Avg IOU: 0.062913, Class: 0.398259, Obj: 0.248613, No Obj: 0.494852, Avg Recall: 0.000000, count: 28
27: 509.014679, 528.262695 avg, 0.000000 rate, 6.490000 seconds, 1728 images
Loaded: 0.000000 seconds
Region Avg IOU: 0.098187, Class: 0.333843, Obj: 0.291236, No Obj: 0.497792, Avg Recall: 0.000000, count: 53
Region Avg IOU: 0.095328, Class: 0.336044, Obj: 0.232018, No Obj: 0.494961, Avg Recall: 0.023256, count: 43
Region Avg IOU: 0.099860, Class: 0.243875, Obj: 0.184972, No Obj: 0.494606, Avg Recall: 0.048780, count: 41
Region Avg IOU: 0.153518, Class: 0.246061, Obj: 0.298524, No Obj: 0.494275, Avg Recall: 0.100000, count: 30
Region Avg IOU: 0.119655, Class: 0.312289, Obj: 0.177828, No Obj: 0.494970, Avg Recall: 0.100000, count: 30
Region Avg IOU: 0.084038, Class: 0.358863, Obj: 0.208506, No Obj: 0.497025, Avg Recall: 0.000000, count: 44
Region Avg IOU: 0.103840, Class: 0.337353, Obj: 0.233433, No Obj: 0.496394, Avg Recall: 0.000000, count: 35
Region Avg IOU: 0.025110, Class: 0.209897, Obj: 0.168186, No Obj: 0.498033, Avg Recall: 0.000000, count: 34
```

图 4.1 YOLO 训练过程

4.2 结果对比

4.2.1 AP 比较

我们首先对比本项目各个模型对于各个类的 AP 值以及 FPS 比较, 如表 4.1。其中 Faster R-CNN 的数据参考自 []。

```

156: 21.874781, 21.986486 avg, 0.001000 rate, 3.210000 seconds, 9984 images
Loaded: 0.000000 seconds
Region Avg IOU: 0.225866, Class: 0.622047, Obj: 0.079425, No Obj: 0.005303, Avg Recall: 0.076923, count: 52
Region Avg IOU: 0.236667, Class: 0.840507, Obj: 0.084323, No Obj: 0.005293, Avg Recall: 0.217391, count: 23
Region Avg IOU: 0.299134, Class: 0.808581, Obj: 0.059308, No Obj: 0.005542, Avg Recall: 0.260870, count: 23
Region Avg IOU: 0.321630, Class: 0.894890, Obj: 0.094913, No Obj: 0.005162, Avg Recall: 0.138889, count: 36
Region Avg IOU: 0.278542, Class: 0.785117, Obj: 0.075642, No Obj: 0.004782, Avg Recall: 0.107143, count: 28
Region Avg IOU: 0.261610, Class: 0.870945, Obj: 0.186651, No Obj: 0.005373, Avg Recall: 0.217391, count: 23
Region Avg IOU: 0.314563, Class: 0.870868, Obj: 0.183406, No Obj: 0.005152, Avg Recall: 0.218750, count: 32
Region Avg IOU: 0.326335, Class: 0.839568, Obj: 0.081353, No Obj: 0.004614, Avg Recall: 0.130435, count: 23
157: 17.267338, 21.514572 avg, 0.001000 rate, 3.320000 seconds, 10048 images
Loaded: 0.000000 seconds
Region Avg IOU: 0.158779, Class: 0.854986, Obj: 0.084412, No Obj: 0.004538, Avg Recall: 0.071429, count: 28
Region Avg IOU: 0.254858, Class: 0.816093, Obj: 0.078008, No Obj: 0.004755, Avg Recall: 0.096774, count: 31
Region Avg IOU: 0.234688, Class: 0.854571, Obj: 0.094775, No Obj: 0.005023, Avg Recall: 0.075000, count: 40
Region Avg IOU: 0.254167, Class: 0.636910, Obj: 0.185094, No Obj: 0.005739, Avg Recall: 0.170732, count: 41
Region Avg IOU: 0.227086, Class: 0.839824, Obj: 0.077659, No Obj: 0.005277, Avg Recall: 0.090909, count: 22
Region Avg IOU: 0.251827, Class: 0.432411, Obj: 0.132082, No Obj: 0.005509, Avg Recall: 0.136364, count: 22
Region Avg IOU: 0.223834, Class: 0.827659, Obj: 0.110631, No Obj: 0.004805, Avg Recall: 0.096774, count: 31
Region Avg IOU: 0.205552, Class: 0.707390, Obj: 0.073666, No Obj: 0.005361, Avg Recall: 0.083333, count: 48

```

图 4.2 tiny-YOLO 训练过程

表 4.1 各模型各类 AP 和 FPS 结果对比

	Pedestrian	Car	Cyclist	FPS
Faster R-CNN	65.91 %	79.11 %	62.81 %	5
YOLO	51.44 %	68.05 %	50.97 %	45
tiny-YOLO	26.67 %	50.75 %	25.04 %	120

通过图表，我们可以发现虽然 tiny-YOLO 有 120fps，而 yolo 只有 45fps，比 YOLO 的速度快上 2-3 倍，但是性能远不如 YOLO。从速度上来看，YOLO 能达到每秒 45 帧的速度，已经基本可以满足实时行人检测的需求，tiny-YOLO 甚至能够达到每秒 120 帧的速度，两个模型均能满足实时行人检测的要求。从性能上来看，不管是哪个类的 AP 值，YOLO 都全面领先于 tiny-YOLO，尤其是在行人和自行车的检测上，而 YOLO 不仅在行人和自行车上能分别达 51.44AP 和 50.97AP，在车辆检测上更是能达到 68.05AP，效果较好。而反观 tiny-YOLO，它在车辆检测上只有 50.75AP，而在行人检测和自行车检测上更是仅有 26.67AP 和 25.04AP。

从整体上看，模型对车辆的检测明显高于对行人和自行车的检测，这是由于 KITTI 数据集中车辆数据较多造成的。而模型的整体表现也部分受限于训练数据过少。

对比 Faster R-CNN，Faster R-CNN 仅有 5fps，无法满足实时行人检测的要求，不管是 YOLO 还是 tiny-YOLO，在速度上都明显比 Faster R-CNN 要快上很多。并且 Faster R-CNN 在行人、车辆、自行车上的检测分别有 65.91AP、79.11AP、62.81AP，YOLO 在这三项上的损失并没有想象上的那么大。可见 YOLO 仅牺牲了部分的精度确在检测速度上提高了很多。

总的来说, YOLO 在 AP 上的表现达到预期效果, 而 tiny-YOLO 虽然速度很快, 但是效率上损失过多。

4.2.2 PR 曲线比较

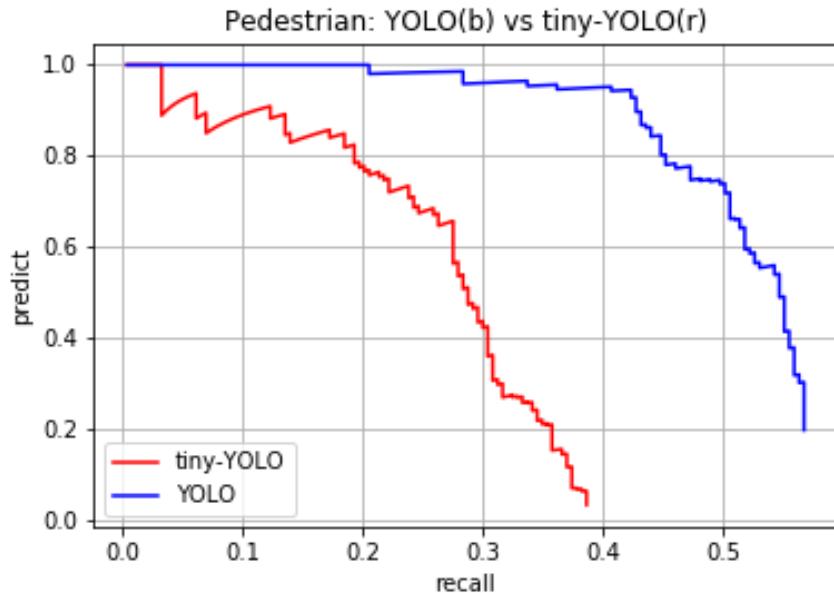


图 4.3 行人检测比较

我们分别对各个模型各个类做出 PR 曲线并进行比较, 结果如图 4.3、4.4、4.5。

从图 4.3 中可以看出, 对于行人的检测上, YOLO 模型明显好于 tiny-YOLO 的模型, 在召回率 0.4 的时候, tiny-YOLO 的准确率已经接近 0, 而 YOLO 的准确率还是接近 1。

从图 4.4 中可以看出, 对于车辆的检测, 两个模型都有很好的表现, PR 曲线都很飘。tiny-YOLO 大概在召回率大于 0.45 后准确率开始快速下降, 而 YOLO 则在召回率大于 0.6 后准确率快速下降。

从图 4.5 中可以看出, 对于自行车的检测, YOLO 模型也明显好于 tiny-YOLO 模型。tiny-YOLO 下降速度比较稳定, 而 YOLO 在召回率大于 0.5 后准确率开始快速下降。

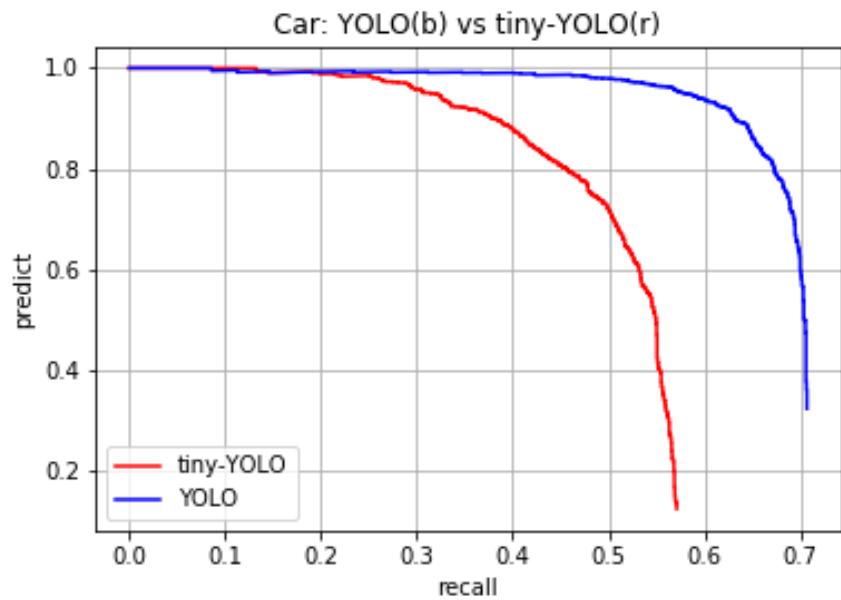


图 4.4 车辆检测比较

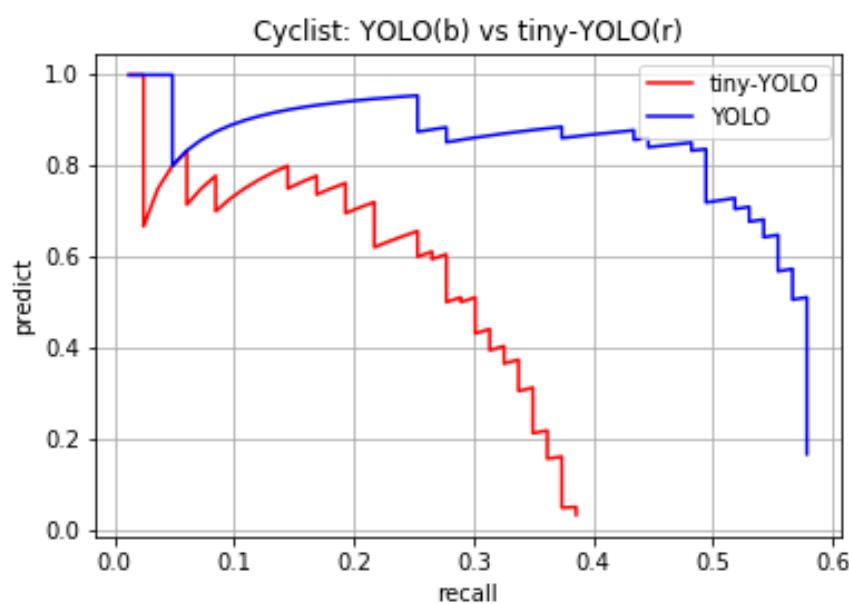


图 4.5 自行车检测比较

4.3 预测效果

4.3.1 效果展示



图 4.6 输入图像

我们选择了一张情形较为复杂的图作为输入,如图 4.6。

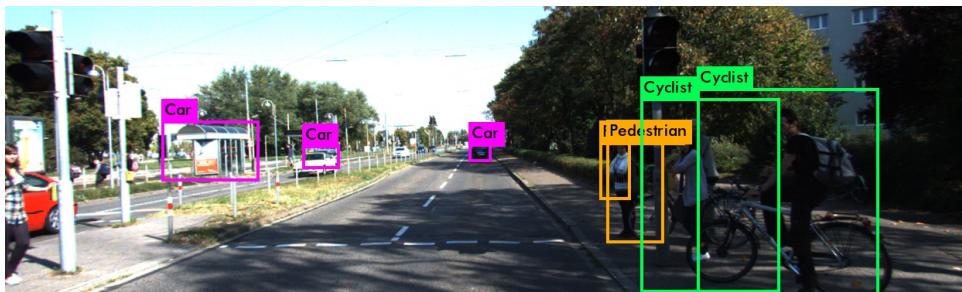


图 4.7 YOLO 预测结果



图 4.8 tiny-YOLO 预测结果

测试结果如图 4.7、4.8所示。YOLO 的测试结果中,预测出来的结果基本正确,而且预测框很好的覆盖了物体,不过其中出现了两个错误,一个是重复预测了一个行

人，一个是将公交车站预测成为了车辆；另外一些不完整的车辆和行人没有预测出来。而 tiny-YOLO 的情况就较为差了，很大一部分容易预测出来的物体并没有成功检测到，将自行车预测成行人，预测框与物体的真值框差别较大。

从上述分析中我们可以看出 YOLO 基本能满足行人检测的要求，而 tiny-YOLO 的检测性能过低就不太适合了。另外使用 YOLO 网络对不完整的物体检测性能不佳，有重复检测情况，对过于细小的物体检测不佳（大部分模型对过于细粒度物体的检测都不太支持）。

4.3.2 错误结果分析

我们分别检测了 YOLO 模型在细小物体靠的近时候的表现以及不完整物体的表现。

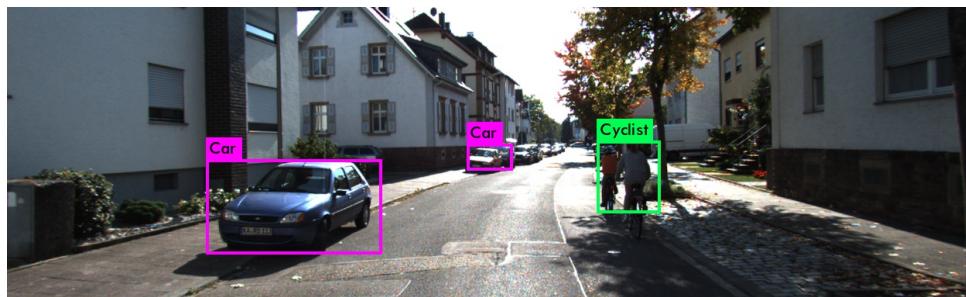


图 4.9 细小集群的测试

如图 4.9，YOLO 队友两个靠的很近骑自行车的人检测成了一个骑自行车的人，这主要是由于 YOLO 将物体检测分为了 $S \times S$ 的方格，而多个细粒度的物体如果在一个方格内的话，YOLO 会很容易将多个物体检测成一个物体。所以 YOLO 对细粒度的集群检测性能不佳。

如图 4.10，YOLO 对不完整物体的检测没有想象中的糟糕，对于大部分不完整的物体还是能够准确检测出来的，除开该物体残缺部分过多。另外由于 YOLO 对细小集群表现不佳，物体间的覆盖也会导致检测的性能损失。

如图 4.11，YOLO 对不同物体的覆盖检测性能不佳。该问题与 YOLO 对细小集群检测不佳的问题类似，这是由于 YOLO 把检测问题分为 $S \times S$ 的方格，而每个方格

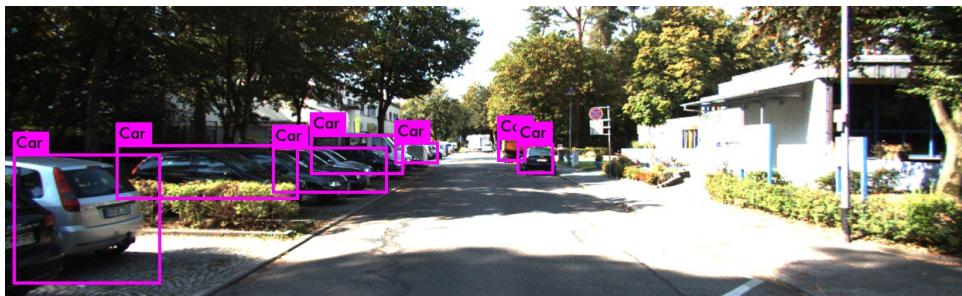


图 4.10 不完整物体的测试

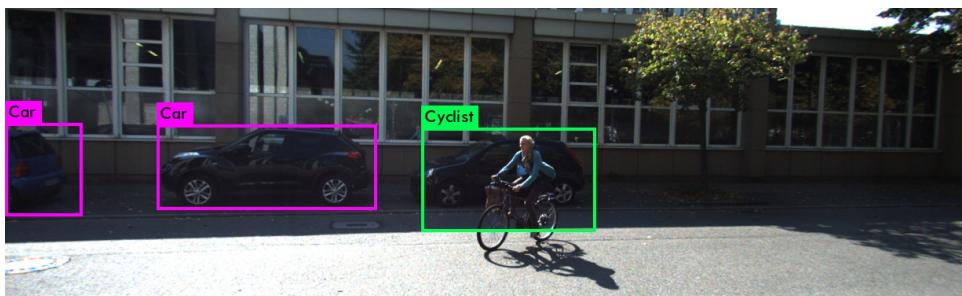


图 4.11 不同物体覆盖的测试

只预测其中一个具体的类,如果不同物体产生覆盖现象,往往不能同时预测成功出这些不同的物体来。

4.4 泛化性

本项目讨论实时行人检测的可行性,针对日常生活驾驶、极端天气、夜晚等各种情形进行实验。

我们针对日常生活驾驶过程中的情况做了测试,测试输入如图 4.12, 预测结果如图 4.13。模型对正常驾驶情况下的预测性能很好,基本能够满足实时行人检测的要求。

我们针对极端天气情况进行了泛化性测试,结果如图 4.14、4.15。从图中可以看出,模型对极端天气情况下表现较差,对于雪中的预测,可能是主要受到噪声(雪)的干扰,所以行人和车辆预测效果都不佳;对于雨中的预测,则可能是受到行人拿伞的影响,所以对行人检测性能不佳但是对车辆检测性能还行。

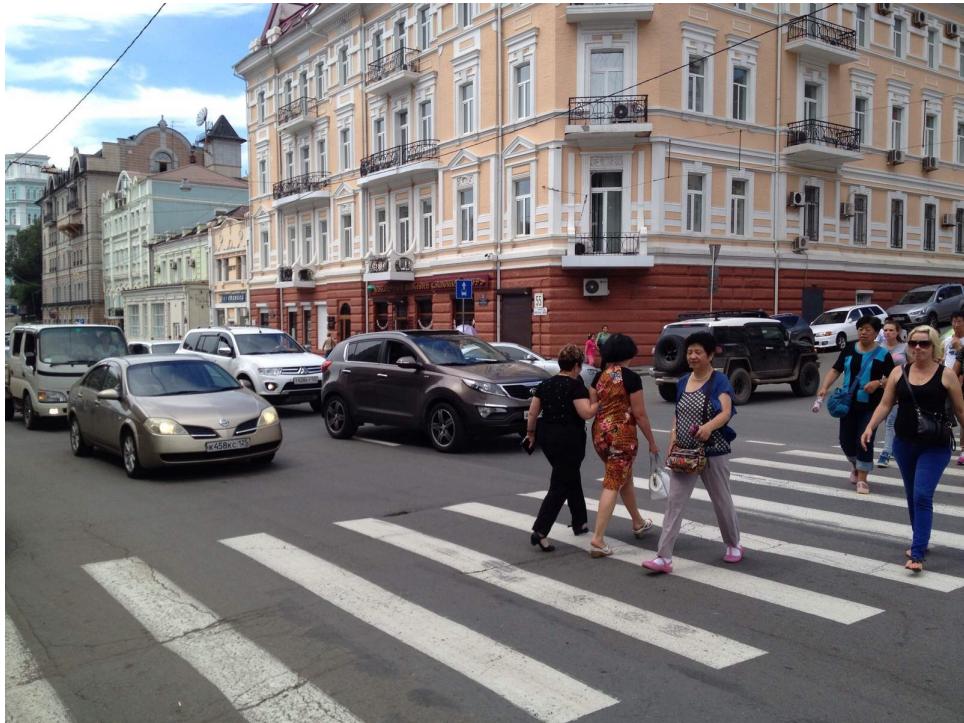


图 4.12 日常生活驾驶

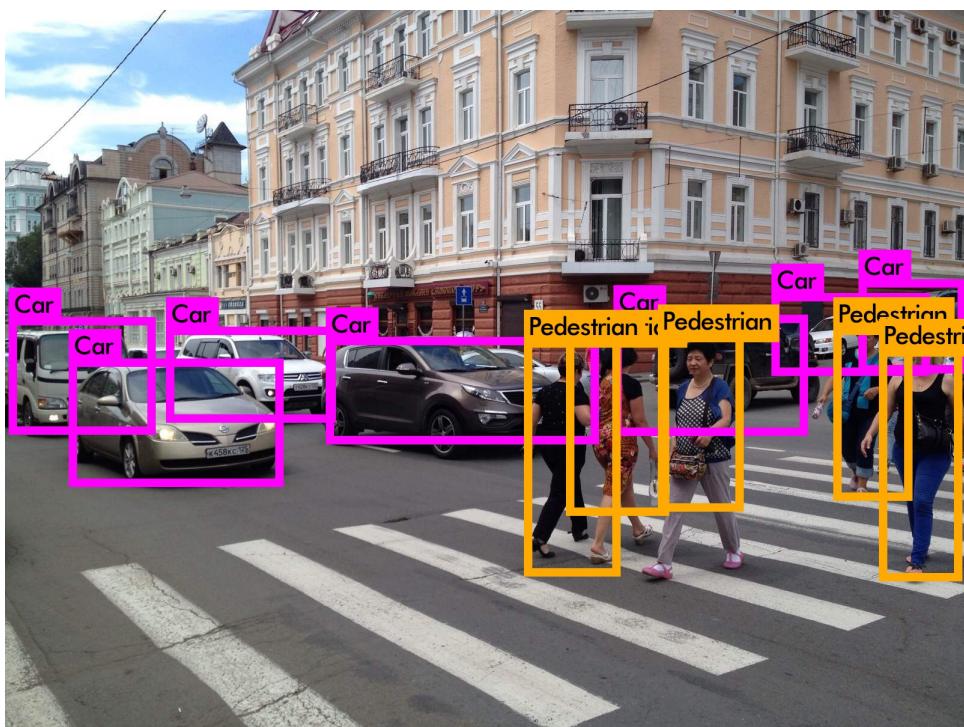


图 4.13 日常生活驾驶预测结果



图 4.14 雨中行人检测

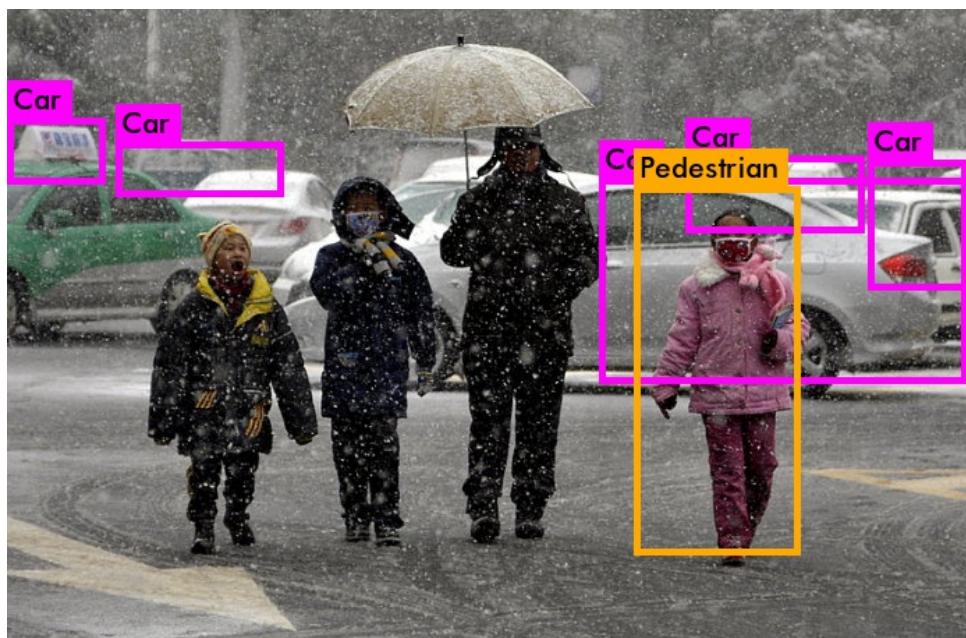


图 4.15 雪中行人检测

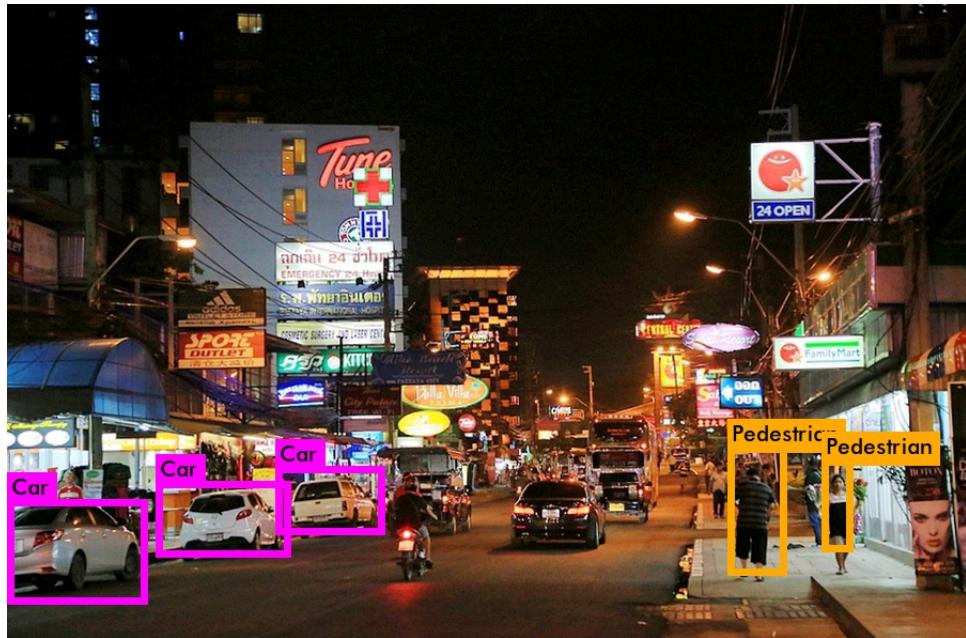


图 4.16 夜晚行人检测

我们针对夜晚的情况进行了泛化性测试，结果如图 4.16。从图中可以看出，模型对夜晚情况下行人和车辆的检查性能还行，虽然还是远不如正常情况下的行人检测，但是已经能够说明模型能在夜晚或光照不好的情况下进行行人检测。

4.5 本章小结

本章对本项目做了详细的实验和分析。我们首先介绍了我们的实验环境和训练参数等。接着，我们研究了各个模型的 AP、FPS 值和 PR 曲线，对比分析得到 YOLO 网络基本满足实时行人检测的要求。之后我们具体分析了预测的效果以及错误结果的分析，分析出了 YOLO 进行人检测的一些弊端。最后我们对模型进行了泛化性的检验，检验结果表明虽然对于极端天气情况下模型表现不佳，但是总体而言模型的泛化性程度较高。通过实验，我们探究了模型的实时行人检测的能力和不足。

第 5 章 本文总结

5.1 工作成果总结

整个项目完成了面向实时行人检测的卷积神经网络的研究，并分析对比了各个模型的实验结果。描述了使用 YOLO 网络作为实时行人检测的研究方案，分析了 YOLO 最后的表现与性能。

搭建了 Darknet 的深度学习框架，学习了 Darknet 的使用方法，参考了 Darknet 的源码来满足实验的需求。

使用了 KITTI 数据集作为本项目的训练和测试数据集，介绍了 KITTI 数据集的形式，通过预处理数据将 KITTI 数据集转换为 YOLO 所需的形式，并划分为训练、测试数据，供之后 YOLO 网络训练和测试使用。

调整了网络结构，分别参考了 YOLO 和 tiny-YOLO 的网络结构来进行实验。基于 Darknet 框架完成了整个模型的训练和测试过程。

验证了实验结果，通过 AP、PR 曲线等方法对比分析了各模型实验结果，最后的实验结果也基本符合预期效果，而 YOLO 的网络结构更是不仅在速度上达到了令人满意的程度，模型性能也相对较好。

综上所述，本项目基本完成了预期的研究成果，同时也对未来的研究有所启迪。

5.2 未来工作

当前研究的不足之处在于：

1. 受限于算法问题，模型对于细小集群的检测性能不佳。
2. 受限于算法问题，模型对于不同小物体覆盖时检测性能不佳。
3. 受限于训练数据等影响，模型对于极端天气条件和夜晚泛化性不佳。

对于面向实时行人检测的卷积神经网络的研究，还可以继续完善的工作如下：

1. 改进模型。测试深度学习模型,如 SSD、Faster R-CNN 在行人检测上的性能和速度。
2. 尝试其他数据集。针对其他不同的行人数据集分别进行训练和测试,研究不同数据集下各个模型的表现。
3. 增强模型的泛化性。尝试提取图像特征,或优化采样策略针对特殊场景进行训练以提高模型对不同场景、天气、夜晚的泛化性。
4. 采用二值化网络来进行网络的训练和测试 []。近来深度学习网络二值化是一个火热的话题,使用二值化网络理论上可以大大减少网络的大小,大大加快网络的传递速度,未来可以尝试从这个方向研究实时行人检测。

参考文献

- [1] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. 2009.
- [2] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.
- [3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [4] <http://www.cvlibs.net/datasets/kitti/>. Kitti vision benchmark.
- [5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.
- [6] Joseph Redmon. Darknet: Open source neural networks in c. <http://pjreddie.com/darknet/>, 2013–2016.
- [7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.
- [8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [9] Stefan Walk, Nikodem Majer, Konrad Schindler, and Bernt Schiele. New features and insights for pedestrian detection. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 1030–1037. IEEE, 2010.

致 谢

本科生毕业论文(设计)任务书

一、题目: _____

二、指导教师对毕业论文(设计)的进度安排及任务要求:

起迄日期 200 年 月 日至 200 年 月 日

指导教师(签名) _____ 职称 _____

三、系或研究所审核意见:

负责人(签名) _____

年 月 日

毕业论文(设计)考核

一、指导教师对毕业论文(设计)的评语:

指导教师(签名) _____

年 月 日

二、答辩小组对毕业论文(设计)的答辩评语及总评成绩:

成绩比例	文献综述 占(10%)	开题报告 占(20%)	外文翻译 占(10%)	毕业论文(设计) 质量及答辩 占(60%)	总评成绩
分值					

答辩小组负责人(签名) _____

年 月 日