中南大學
CENTRAL SOUTH UNIVERSITY

行车环境感知课程
项目报告

题　　目：	行车环境感知之目标检测 ----基于 YOLOv3 和 Faster-RCNN
学生姓名：	孙重豪
学　　号：	8207191520
学　　院：	自动化学院
专业班级：	智能 1902 班

2021 年 11 月

行车环境感知之目标检测

----基于 YOLOv3 和 Faster-RCNN

摘要

环境感知是智能汽车实现自动驾驶的重要一环，也是智能汽车安全性和智能性的保障，感知结果直接影响车辆的决策规划及整车的执行控制感知。一直以来，纯视觉感知方案都被车企以及学术界所否定，但是近些年来，深度学习技术在图像目标检测领域的大规模应用，为面向智能车的行车环境纯视觉感知方案创造了可能。本项目主要研究使用视觉深度学习方法实现车辆环境感知系统，我们在华为云 ModelArts 平台上基于 MindSpore 框架搭建了 YOLOv3 和 Faster-RCNN 网络，实现对行车环境中的行人、车辆、交通标识以及路标进行实时检测，分析并比较了两种模型在测试数据上的效果。

关键词：行车环境感知 目标检测 YOLOv3 Faster-RCNN MindSpore

目录

一、选题意义

行车环境感知是无人驾驶技术中的关键环节，处于自动驾驶汽车于外界环境信息交互的关键一环，是决策与控制的前提与基础，准确、可靠、实时的环境感知是智能车行车安全性和智能性的保障[1]，是无人驾驶汽车最具难度的项目。车辆对行车环境的感知直接决定了自动驾驶的成功与否。

而长久以来，众多传统车企和大多数新兴势力，都以激光雷达作为车辆行车环境感知的主要方式，以计算机视觉作为辅助手段，来实现环境感知。但是，这种方案受限于激光雷达的成本，一直难以得到较为广泛的推广。而通过纯视觉进行行车环境感知方式，受限于技术水平，一直难以做到较好的效果。

近几年，随着深度学习技术在计算机视觉领域的大规模应用及其在图像处理与识别方面取得重大技术突破，并且在部分方面表现出超越人类的表现，使得通过纯视觉方案解决车辆行车环境感知成为可能。而通过纯视觉方案相较激光雷达等其他方案，具有成本低、可感知信息量大、应用面广的优势，同时也和人类驾驶员感知行车环境的方式极为相似，因此通过纯视觉方案可以使得相关行车环境感知技术大规模推广，甚至帮助更高等级的自动驾驶技术落地。

二、方法介绍

2.1 方法概述

在本项目中我们通过 Faster-RCNN 和 YOLOv3 模型对行车环境中经常出现的九类目标：小汽车、行人、公交车、人行横道、地标、卡车、自行车、摩托车、交通标识，进行检测，以验证通过纯视觉方法解决行车感知的可行性。首先，我们将原始的带标注的数据进行了采样，并且通过从 COCO2017 数据集对原有的数据集进行了填充。在数据集创建完毕之后，在华为云 ModelArts 平台上使用 MindSpore 框架搭建 Faster-RCNN 和 YOLOv3 模型，并且进行训练。由于老师提供的数据较少且数据标签极不平衡，难以训练 Faster-RCNN 和 YOLOv3 这种大型目标检测神经网络，虽然进行了上述的数据平衡处理，但仍可能导致结果欠拟合、测试效果不佳，因此对于这两个模型分类网络的训练，我们使用了已有的、基于大量数据训练的预训练模型，并以此为基础训练我们自己的目标检测网络。下图是项目的整体流程：

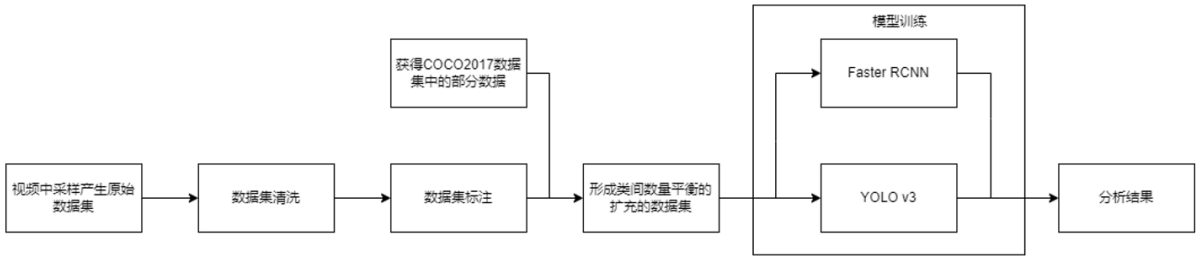

图 2-1　项目整体流程图

2.2　数据预处理

2.2.1　数据分类

从百度网盘上下载老师提供的视觉_前车检测数据集。其中共计有 4591 个 bmp 文件，4413 个已打好的标签的 xml 文件。

首先，用 Python 将 bmp 和 xml 文件分开到两个不同的文件夹，然后将有标签对应的 bmp 图片单独存出放入 BMPImages，其余没有标签的 bmp 文件放入另一个文件夹 BMPImages-带标签，标签放入文件夹 Annotations。

分类时发现有三张错误标签 xml，我们选择将干扰数据剔除，剔除后共计 4410 个标签，4410 张与标签一一对应的 bmp 图片，181 张无标签的 bmp 图片用作测试集。

2.2.2　数据清洗

对数据集进行训练时发现，有很多图片存在重复，可能的原因就是汽车在等红灯时位置一直不变，所以导致了场景重复的图片出现，如下图：

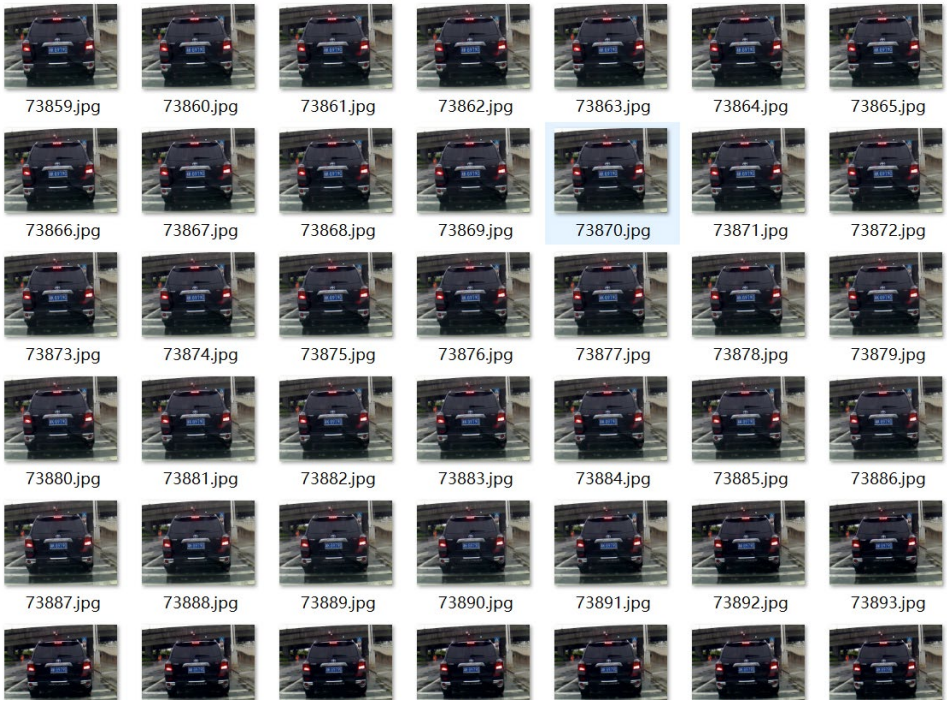

图 2-2　连续重复图片

于是我们进行了冗余图片筛除，同时对所有图片进行每五帧采样，大大降低了数据的重复性，将所有分类的标签使用程序跑出，主要有以下九种：

```
proj_map = {  'pedestrian_crossing': 0,
              'car': 1,
             'traffic_sign': 2,
              'bicycle': 3,
              'truck': 4,
              'pedestrian': 5,
             'road_mark': 6,
              'motorcycle': 7,
              'bus': 8
           }
```

2.2.3 数据标注

在原始数据集中，存在着一部分未标注的数据，我们通过华为八爪鱼平台进行标签标注，如下图所示：

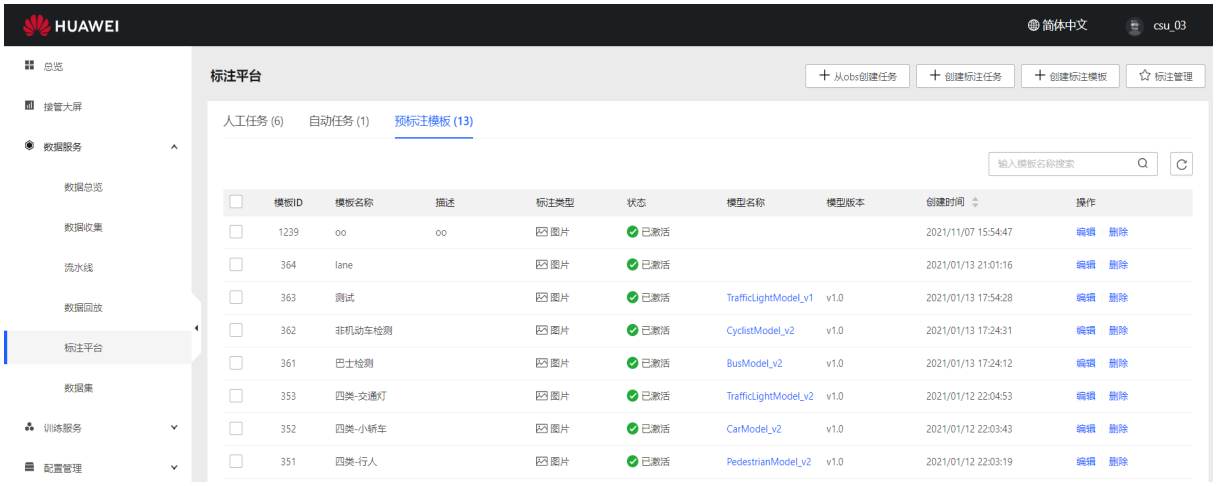

图 2-3　华为八爪鱼平台

2.2.4 数据填充

完成原始数据的处理工作之后，我们发现，原始数据集存在类间样本严重不平衡的现象。如果直接使用该数据集，会对模型的性能产生严重的影响。

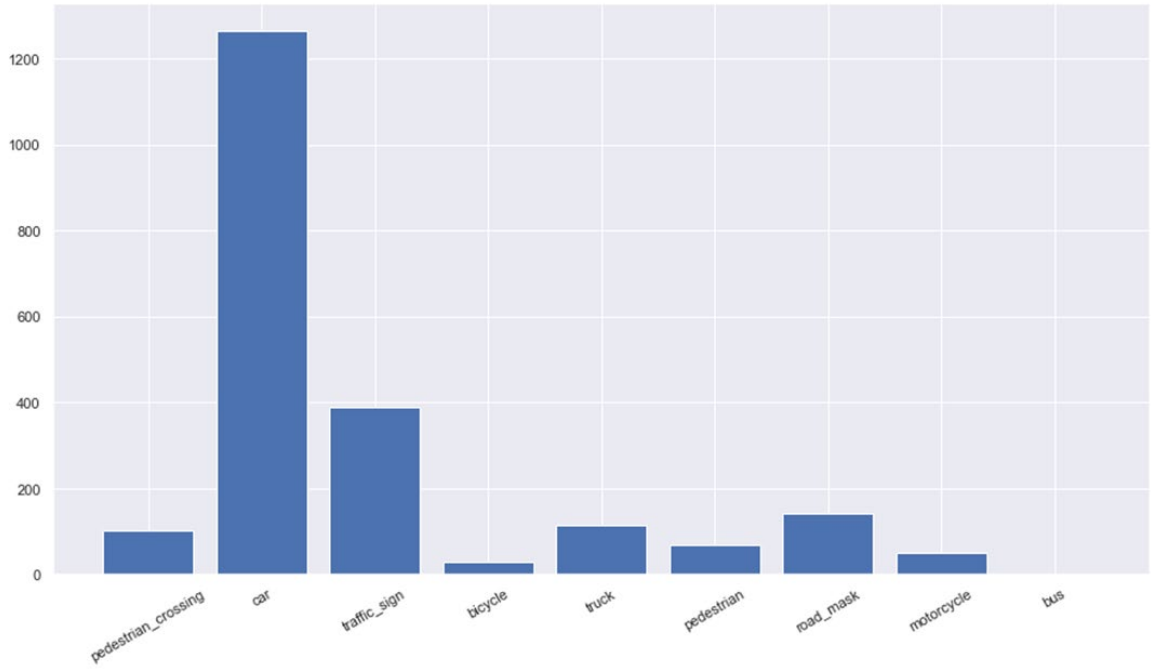

图 2-4　原始数据集类间分布情况

基于此，我们从 COCO2017 数据集中，以表 1 中的优先级进行了数据的选取，来补充我们原始的数据集，并且还对原数据集中最多的几类样本进行了降采样，以使数据集达到类间平衡。

表 2-1　COCO 数据集数据选择优先级

数据筛选优先级
1. 目标类单独出现于图中 2. 目标类出现在图中，但有其他非原始数据的类出现 3. 目标类出现在图中，但有其他原始数据集中非目标类的出现
注：目标类指的是待补充数据的类；非原始数据集的类指的是原始数据集中不包含的类别；原始数据集中非目标类指的是原始数据集中无需补充的类

2.3　Faster-RCNN 模型介绍

2.3.1　模型简介

Faster-RCNN 是以 R-CNN 家族的一种快速目标检测网络。传统的目标检测网络，其速度瓶颈大多是在候选区域的产生。而 Faster-RCNN 通过引入 RPN 网络，从而将候选区域的产生更为精准，减少了候选区域框的数量，从而提高了检测的速度和精度。

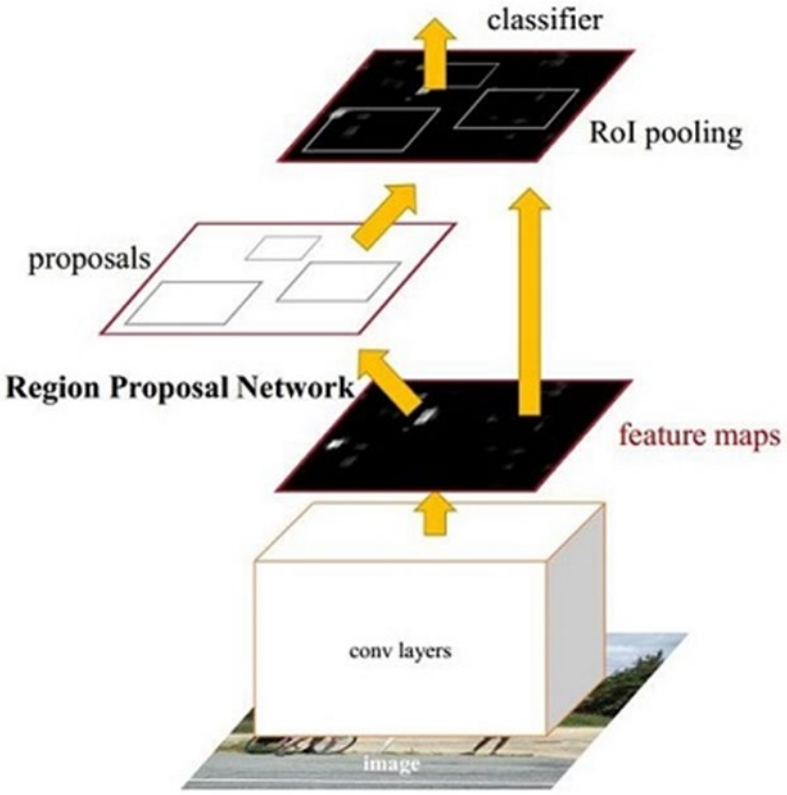

图 2-5　Faster-RCNN 原理图

Faster-RCNN 可以根据功能分为三个部分，分别为特征提取网络、RPN 网络和分类网络。其中，原论文中特征提取网络使用的是 VGG16，通过特征提取网络获得原始图片的特征图，并且送入 RPN 网络进行候选框的产生，进行目标检测的环节，最后再对产生的候选框进行分类和修正，从而提升候选框的精度。

特征提取网络主要起到的作用是提取图片的原始语义特征，得到处理后的特征图，进行信息的压缩表示。而 RPN 网络，则会针对在特征图中的每一个像素产生 K 个不同尺度、不同形状的候选区域，并且对着 K 个候选区域是否包含目标进行评分，再产生所有的候选框侯，将得分高的若干个候选框继续筛选，通过非极大值抑制等方法进一步减少候选区域的数量；与此同时，RPN 网络的另一个分支，会对候选框进行平移缩放，对候选框的位置进行首次修正。将这两条分支合并，便得到了 FPN 的输出，即候选框的位置。随后的分类网络会对候选框中的物体进行分类，并且还会产生一个新的修正量，对候选框进行进一步的修正。通过这三个网络，便可以对一张图片中的目标进行高质量、快速的检测和分类。

下面是其结构图和介绍：

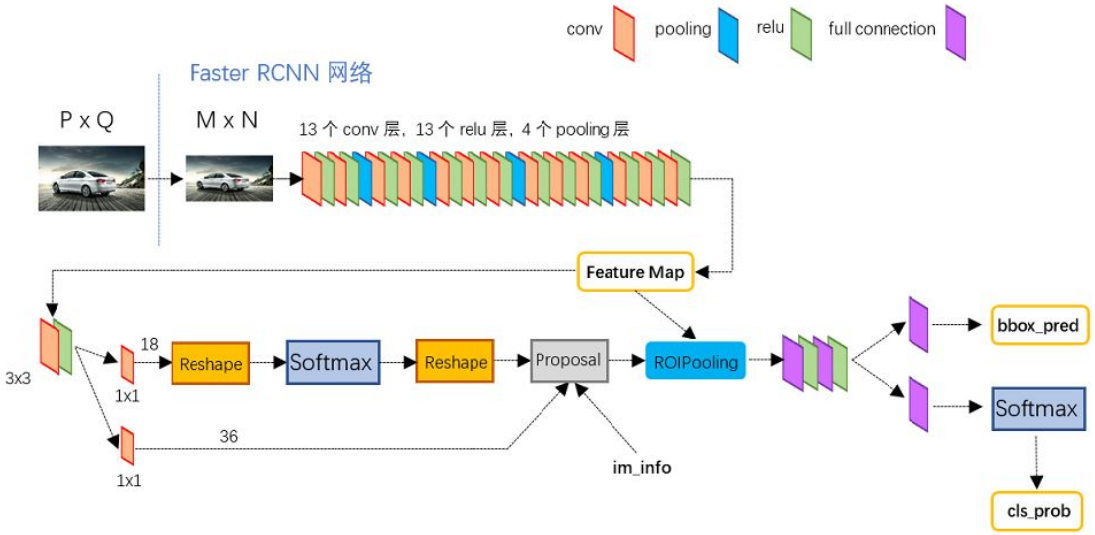

图 2-6　Faster-RCNN 结构图

Faster-R-CNN 由下面几部分组成：

1. 卷积层 CNN 等基础网络：提取特征得到 feature map

2. RPN 层：在经过卷积层提取到的 feature map 上用一个 3x3 的 slide window，去遍历整个 feature map,在遍历过程中每个 window 中心按 rate，scale（1:2,1:1,2:1）生成 9 个 anchors，然后再利用全连接对每个 anchors 做二分类（是前景还是背景）和初步 bbox regression，最后输出比较精确的 300 个 ROIs。把经过卷积层 feature map 用 ROI pooling 固定全连接层的输入维度。

3. 然后把经过 RPN 输出的 rois 映射到 ROIpooling 的 feature map 上进行 bbox 回归和分类。

2.3.2　模型参数

我们使用华为云的 MindSpore 框架进行模型的搭建，其中 Faster-RCNN 参数如下表。

表 2-2　Faster-RCNN 参数

anchor_ratios	[0.5, 1.0, 2.0]
anchor_strides	[4, 8, 16, 32, 64]
anchor_scales	[8, 16, 32]
num_anchors	9
backbone	ResNet 50
learning_rate	0.01
optimizer	SGD+momentum

2.4 YOLOv3 模型介绍

2.4.1 模型简介

YOLO 是“You Only Look Once”的简称，它虽然不是最精确的算法，但在精确度和速度之间选择的折中，效果也是相当不错。YOLOv3 借鉴了 YOLOv1 和 YOLOv2，虽然没有太多的创新点，但在保持 YOLO 家族速度的优势的同时，提升了检测精度，尤其对于小物体的检测能力。YOLOv3 算法使用一个单独神经网络作用在图像上，将图像划分多个区域并且预测边界框和每个区域的概率。YOLOv3 相比 YOLOv2 最大的改进点在于借鉴了 SSD 的多尺度判别，即在不同大小的特征图上进行预测。对于网络前几层的大尺寸特征图，可以有效地检测出小目标，对于网络最后的小尺寸特征图可以有效地检测出大目标。此外，YOLOv3 的 backbone 选择了 DarkNet53 网络，网络结构更深，特征提取能力更强了。下面是 YOLOv3 的结构图，左侧中的红色虚线框部分为 DarkNet53 网络：

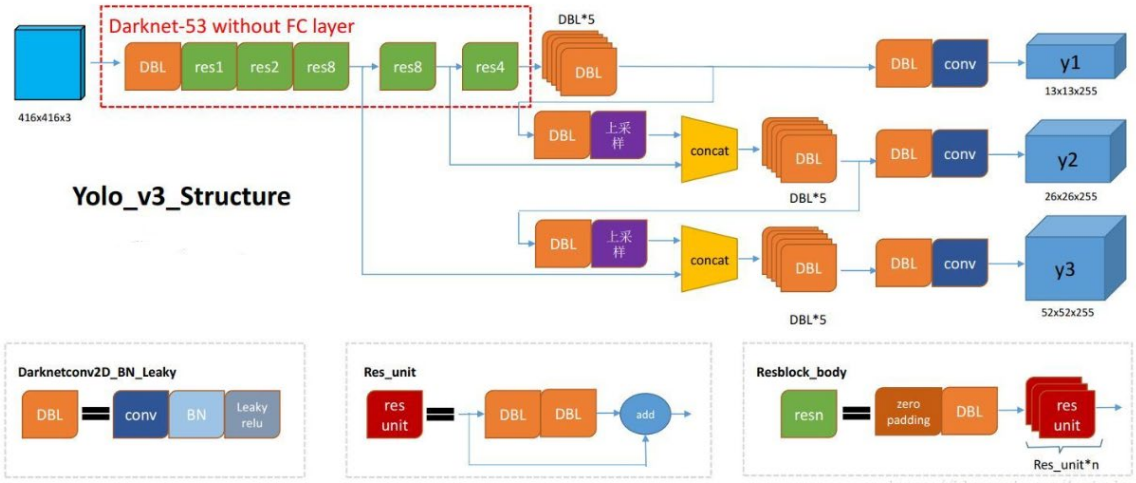

图 2-7 YOLOv3 结构图

对于 YOLOv3 网络结构，有以下几点需要注意：

1、由于网络较深，使用了残差结构。

2、DarkNet53 网络用步长为 2 的卷积代替了池化层。

3、所有的网络层不包含全连接层，因此，输入图像的大小也是可以调整的（有些地方看到的可能是 608x608，其实是一样的），而输入图像的大小同样是最小的输出特征图的 32 倍。

4、YOLOv3 分别在三个尺寸的特征图进行了预测，每个尺寸的特征图使用了 3 个锚点。因此，输出层的维度计算方法为：(4+1+80)x3=255，因此，最后一层 1x1 的卷积层的数量为 255。

5、13 x 13 的特征图会通过上采样层和之前的 26 x 26 的特征图在通道维度拼接在一起，26 x 26 的特征图再经过上采样和 52 x 52 的特征图拼接。

2.4.2 模型参数

（1）DarkNet53

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

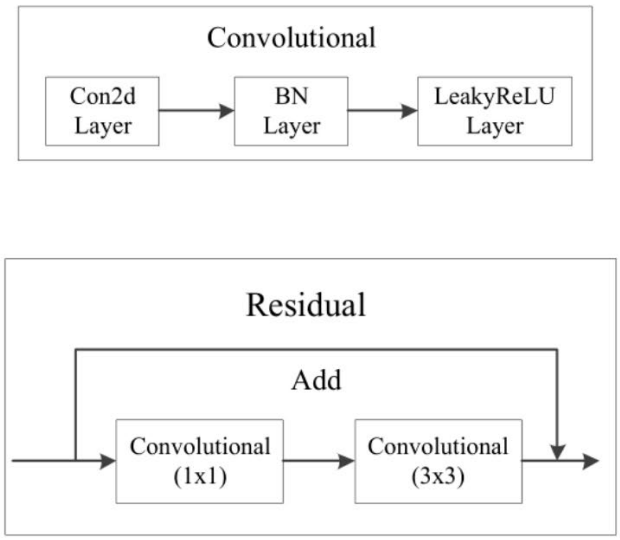

图 2-8　DarkNet53 网络各层参数

（2）目标检测框

随着输出的特征图的数量和尺度的变化，先验框的尺寸也需要相应的调整。YOLO2 已经开始采用 K-means 聚类得到先验框的尺寸，YOLO3 延续了这种方法，为每种下样尺度设定 3 种先验框，总共聚类出 9 种尺寸的先验框。在 COCO 数据集这 9 个先验框是：(10x13)，(16x30)，(33x23)，(30x61)，(62x45)，(59x119)，(116x90)，(156x198)，(373x326)。

特征图	13*13			26*26			52*52		
感受野	大			中			小		
先验框	(116x90)	(156x198)	(373x326)	(30x61)	(62x45)	(59x119)	(10x13)	(16x30)	(33x23)

图 2-9　特征图与先验框

三、实验结果及分析

3.1 Faster-RCNN

3.1.1 训练策略

在训练 Faster-RCNN 的过程中，我们选取了 ResNet50 作为骨干网络，以提取特征。并且选取了在 ImageNet 上进行预训练过的权重作为对应权重。

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43†
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

图 3-1　ResNet 在 ImageNet 上的 top-1 和 top-5 错误率

我们通过图 3-1 所示的训练步骤，对 Faster-RCNN 进行训练，步骤如下图：

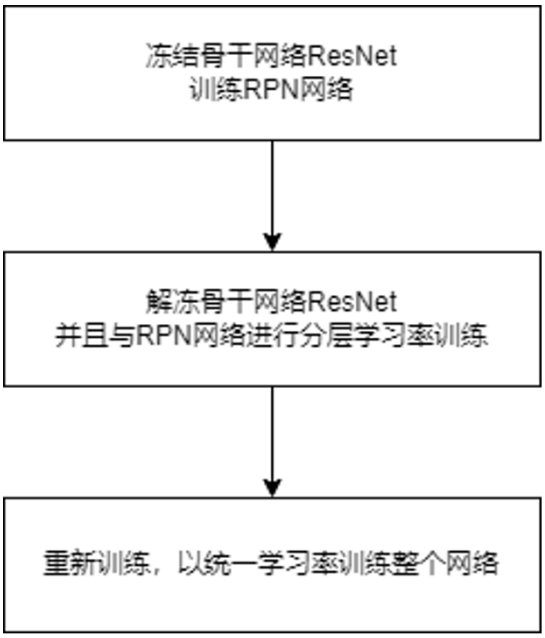

图 3-2　Faster-RCNN 训练步骤

为了保证训练过程的稳定和效果，我们采用了 Warm-up 和余弦退火的学习率策略。

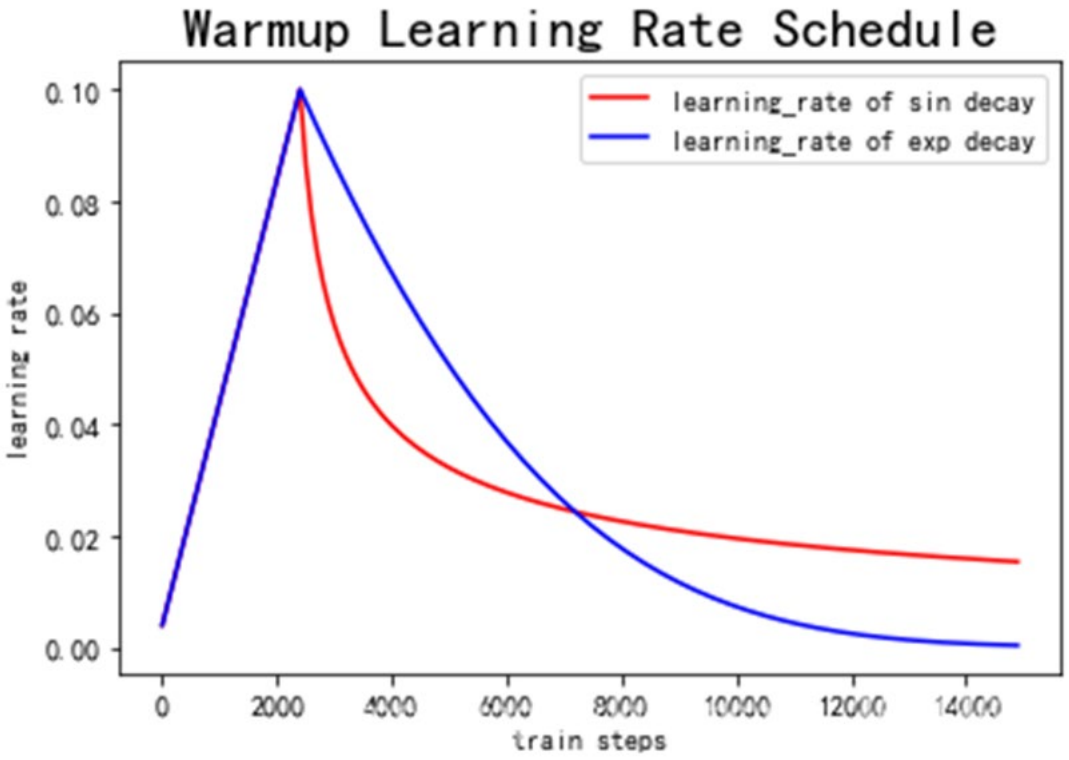

图 3-3　Warmup 和余弦退火学习率策略下的学习率变化图

通过 Warmup 预热学习率的方式，最开始先用小学习率学习，并且随着 step 的增多而增大，在到达设置的阈值时，通过余弦退火的方法，使学习率进行衰减。在实践上，这种技术有助于使模型收敛速度变快，效果更佳。

3.1.2　训练结果

```
Test:  Total time: 0:00:29 (0.0452 s / it)
Averaged stats: model_time: 0.0440 (0.0415)  evaluator_time: 0.0026 (0.0024)
Accumulating evaluation results...
DONE (t=0.27s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.381
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.730
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.355
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.153
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.290
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.459
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.359
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.496
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.499
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.260
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.401
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.552
```

图 3-4　训练输出

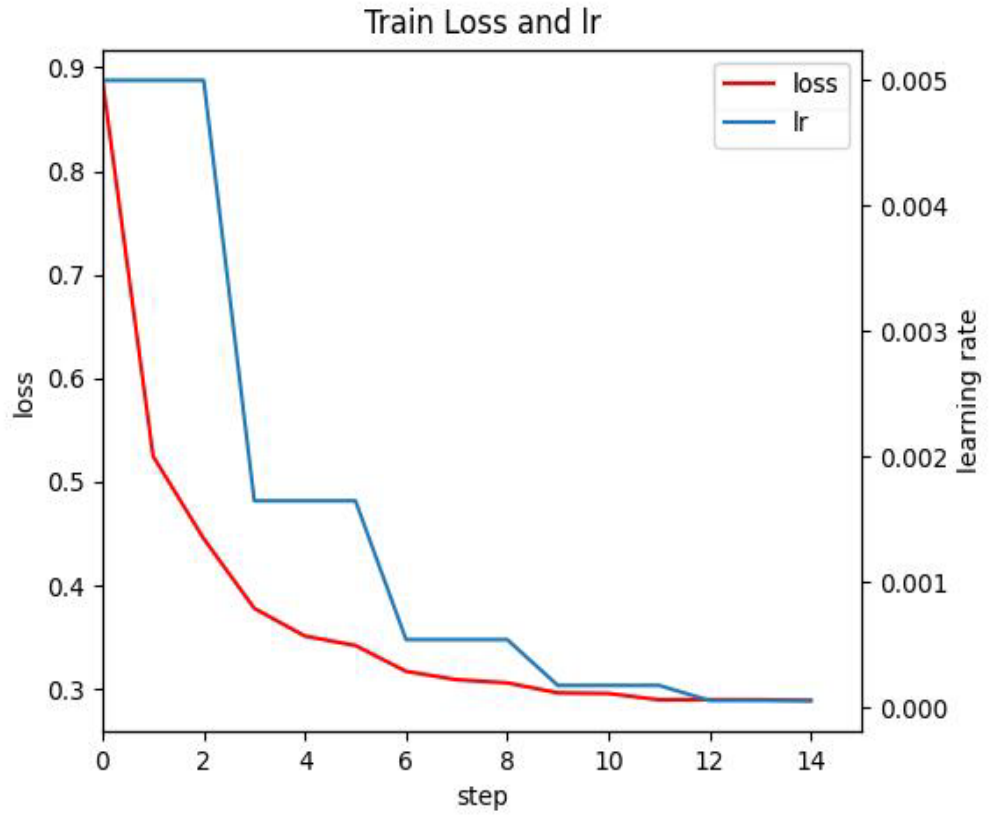

图 3-5　Loss 和学习率变化

图 3-6　测试结果

3.2 YOLOv3

3.2.1 训练策略

```
"distribute": False,
"device_id": 0,
"device_num": 1,
"dataset_sink_mode": True,

"lr": 0.001,
"epoch_size": 500,
"batch_size": 8,
"loss_scale" : 1024,

"pre_trained": None,
"pre_trained_epoch_size":0,

"ckpt_dir": "./ckpt",
"save_checkpoint_epochs" :1,
'keep_checkpoint_max': 1,
```

```
img_shape = [352, 640]
feature_shape = [32, 3, 352, 640]
num_classes = 9
nms_max_num = 50
_NUM_BOXES = 50

backbone_input_shape = [64, 64, 128, 256]
backbone_shape = [64, 128, 256, 512]
backbone_layers = [2, 2, 2, 2]
backbone_stride = [1, 2, 2, 2]

ignore_threshold = 0.5
obj_threshold = 0.3
nms_threshold = 0.4
```

图 3-7 训练参数设置

3.2.2 训练结果

```
Start train YOLOv3, the first epoch will be slower because of the graph compilation.
epoch: 1 step: 163, loss is 151.78075
epoch: 2 step: 163, loss is 74.98442
epoch: 3 step: 163, loss is 55.08742
epoch: 4 step: 163, loss is 41.013165
epoch: 5 step: 163, loss is 54.37825
epoch: 6 step: 163, loss is 38.35581
epoch: 7 step: 163, loss is 39.79799
epoch: 8 step: 163, loss is 21.4313
epoch: 9 step: 163, loss is 42.97113
epoch: 10 step: 163, loss is 32.102135
epoch: 11 step: 163, loss is 18.391888
epoch: 12 step: 163, loss is 36.73929
epoch: 13 step: 163, loss is 23.23796
epoch: 14 step: 163, loss is 18.994617
epoch: 15 step: 163, loss is 27.912682
epoch: 16 step: 163, loss is 57.31811
epoch: 17 step: 163, loss is 30.8367
epoch: 18 step: 163, loss is 28.964546
epoch: 19 step: 163, loss is 45.556713
epoch: 20 step: 163, loss is 22.978683
epoch: 21 step: 163, loss is 25.218836
epoch: 22 step: 163, loss is 22.723663
epoch: 23 step: 163, loss is 19.799068
epoch: 24 step: 163, loss is 41.43291
epoch: 25 step: 163, loss is 36.32434
```

图 3-8 训练输出

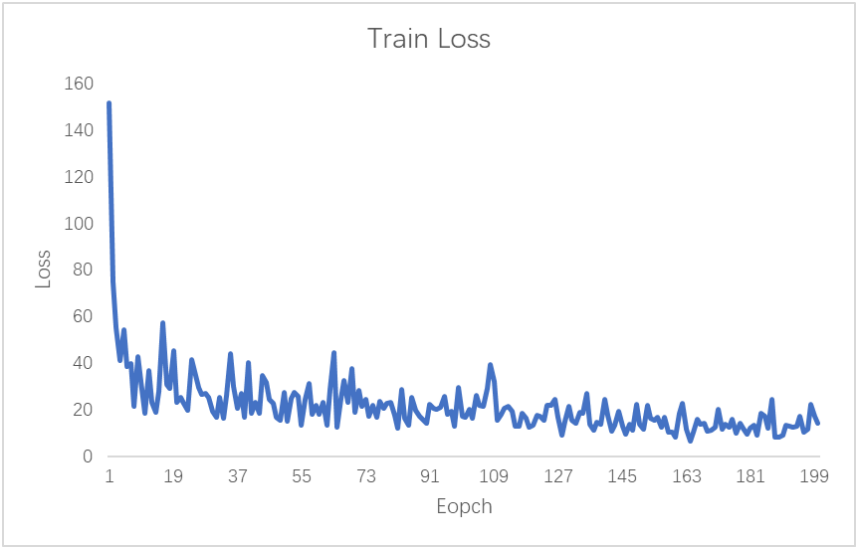

图 3-9 Loss 变化

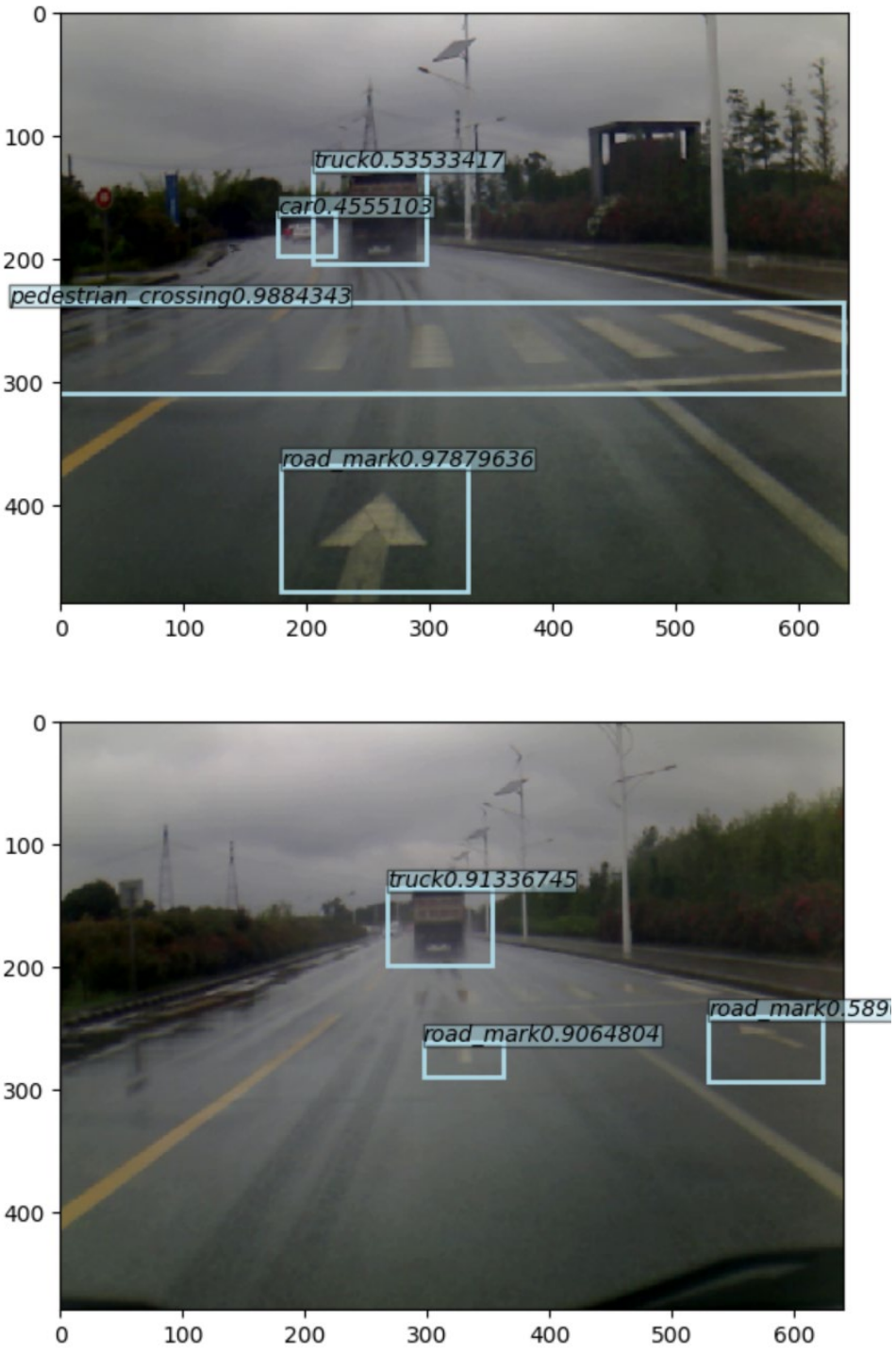

图 3-10　测试结果

3.3　小结

通过上述的实验结果我们可以发现，通过深度学习算法，可以较好的检测出行车环境中出现的几类目标，并且成功定位出其位置。另外，在实验过程中，我们也发现，Faster-RCNN 相较 YOLO v3 更适合解决此种环境下的目标识别，且在小尺度目标的检测上具有更大的优势。但是，通过纯视觉方案实现行车环境感知，仍然存在不足之处。

四、结论与改进设想

4.1 项目结论

在本项目中，先后尝试了 YOLO v3 和 Faster RCNN 两种模型，在同样的训练数据和测试数据上，Faster RCNN 的效果远好于 YOLOv3，其平均精度为 55.72%。除此以外 Faster RCNN 也具有较快的处理速度，且对于一些小尺度的目标检测效果较好，对行车过程产生的积极意义就是能感知到更远的物体，有利于智能汽车提前进行决策规划。这也印证了通过纯视觉感知的方案为车辆提供环境感知能力的可行性。但是这种方案仍然存在不足之处，在过程中发现算法无法对训练数据集中未出现过的物体进行识别或进行误判，从而产生事故的隐患。因此，通过纯视觉方案解决行车感知的问题仍然有很长的一段路需要走，直到足够庞大、丰富的数据集完成，或者新的革命性的未知目标检测算法出现，才能够完全解决现阶段纯视觉感知方案的不足之处。

4.2 改进的设想

训练过程中，我们发现限制 Faster-RCNN 速度进一步的提升主要是在后面的 RPN 网络中。在本项目训练的 Faster-RCNN 模型中，会在骨干网络产生的特征图上针对每个特侦点产生 9 个候选框，而在一个 50 * 38 的特征图上会产生共计 50 * 38 * 9 个候选框，经过多个阶段的判断，最后产生 300 个候选框作为输出，这一部分极大的限制了 Faster-RCNN 的速度。倘若需要进一步提升 Faster-RCNN 的实时性，我们需要引入一些先验知识，对候选框的产生进一步进行筛选，或者减少候选框数量的产生和输出，从而进一步提升 Faster-RCNN 的速度。

若要进一步提升 Faster-RCNN 的检测精度，可以尝试引入一些注意力机制，如 CBAM 或者 Transformer 中的类似机制。

此外，本项目中 Faster-RCNN 的训练过程中，没有改变原始图片的尺度，可以尝试对原始图片的尺度进行修改，产生多种尺度的原始数据，通过多尺度训练的方式，来提高模型对各个尺度目标的检测能力。

五、华为云平台和 MindSpore 框架的使用感受和改进建议

5.1 使用感受

本次实验中，我们全程使用华为的 MindSpore 框架作为模型训练和验证的框架。在整个过程中，我们感受颇深。

首先，华为作为一家民族之光企业，在国内的声誉很好，同时也敢为人先，开源了其自研的 MindSpore 框架，这对于我们而言，又多了一种新的框架的选择，而且还是一个国产的框架，确实让人倍感兴奋。

除此之外，华为云的 OBS 存储功能，能够很方便的帮助我们在不同的 Notebook 中迁移数据，在多人项目中，可以很方便的分享一个大型的文件。

在使用过程中，其实可以感受到华为的 MindSpore 和自家生态的联动的好处。华为的 ModelArts 平台上直接支持 MindSpore 框架，同时结合 MindSporeHub 可以很方便的使用其中的一些开源模型来进行迁移学习或者直接推理使用，免去了我们到处找模型的繁琐工作。另外，在训练 Faster RCNN 的时候，我们发现虽然 MindSpore 在启动的时候会花费很多的时间，但是当启动完成之后，整个模型的训练时十分迅速的。假如还是在 Ascend 处理器上运行，那么其速度比一张 V100 的 GPU 的速度还快，这样可以使得我们无需额外购置 GPU 进行模型的训练，而可以通过一个通用的处理器完成。

5.2 改进建议

5.2.1 遇到的问题

尽管在使用过程中，感受到华为云的很多快捷和方便，但是仍然遇到了一些小问题。下面将一一列举：

1. 产生报错时，网上相关的参考资源较少，且报错信息有些苦涩难懂。
2. MindSporeHub 上的一些链接不可用，模型已过期，却没有及时更新或移除。
3. MindSpore 支持的操作系统、Python 版本、Cuda 版本都较为严格。
4. 使用 MindSpore 时，需要构建完整的项目流程，包括数据流水线在内，对于一些轻型的任务和实验来说过程过于复杂。
5. MindSpore 的官方文档不够详细，而且对于一些算子的解释希望能再多一些可视化的图片，更加图文并茂一些。
6. 对 CPU 和 GPU 的支持不如 Ascend 的好。

7. 没有中文的 API 说明文档。作为中国的开源框架，支持中文可以更好的获得一部分英语能力较弱的中国开发者的青睐，同时也更方便新人的学习。

5.2.1 提出的建议

针对我们在试验过程中遇到的这些问题，我们进行了深入的思考，同时也提出了我们的建议：

1. 完善官方文档，补充一些图片或者案例帮助理解。
2. 继续完善框架并且支持更多版本。
3. 将一些过期、实现的链接和文档及时清除。
4. 提供一些更为轻便的开发模式，可以参考 Keras，制作相关的高阶 API，或者支持 Keras 的方式，成为 Keras 可调用的后端。
5. 开发适合新手入门的接口，不要一味地提供适合工业界的接口，这样不利于框架的入门学习。

参考文献

[1]梁敏健. 智能车行车环境视觉感知关键技术研究[D].长安大学,2017.

[2]董邦宜. 扣件病害检测的不平衡样本学习问题研究[D]. 北京交通大学, 2020.

[3]翟云, 杨炳儒, 曲武. 不平衡类数据挖掘研究综述[J]. 计算机科学, 2010, 37(10): 27-32.

[4]杨晓玲,江伟欣,袁浩然.基于 yolov5 的交通标志识别检测[J].信息技术与信息化,2021(04):28-30.

[5]Jianming Zhang, Manting Huang, Xiaokang Jin, Xudong Li. "A Real-Time Chinese Traffic Sign Detection Algorithm Based on Modified YOLOv2[J]. Inventi Impact - Algorithm,2018:

[6] Gotmare A, Keskar N S, Xiong C, et al. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation[J]. arXiv preprint arXiv:1810.13243, 2018.