合肥工業大學

计算机与信息学院

方向领域综合设计报告

设　计　题　目　基于飞桨平台的车辆检测系统设计与实现

学生姓名与学号　孙淼 2018211958

专 业 与 班 级　计算机科学与技术 18-2 班

指　导　教　师　余 烨

完　成　日　期　2022/01/10

课程目标(Course Objectives, CO)	支撑的毕业要求 (Graduation Requirement，GR)及二级指标点	
CO1 运用计算机科学与技术及其应用领域工程的思想与方法，从需求分析开始，独立完成领域方向系统的设计与开发，兼顾领域方向系统开发的社会评价。	GR6 工程与社会：能够基于计算机工程相关背景知识进行合理分析，评价计算机软硬件开发、系统设计等计算机科学与技术工程实践过程和复杂计算机工程问题解决方案对社会、健康、安全、法律以及文化的影响，并理解应承担的责任。	GR6.2 了解社会发展形势，能分析和评价计算机科学与技术工程实践对社会、健康、安全、法律、文化的影响，以及这些制约因素对项目实施的影响，并理解应承担的责任。
CO2 理解领域方向系统运行应遵守的社会规范，能够在领域方向系统开发实践中自觉遵守工程职业道德和规范，履行责任。	GR8 职业规范：具有人文社会科学素养、社会责任感，能够在计算机科学与技术工程实践中理解并遵守工程职业道德和规范，履行责任。	GR8.2 理解计算机科学与技术工程实践活动有可能涉及公众的安全、健康和福祉，以及环境保护的社会责任，能够在计算机科学与技术工程实践中自觉履行责任。
CO3 理解领域方向系统开发的人员组织方式，理解团队成员及负责人角色。	GR9 个人和团队：能够在多学科背景下的团队中承担个体、团队成员以及负责人的角色。	GR9.2 能够在团队中独立或合作开展工作，能够组织、协调和指挥团队开展工作。
CO4 理解领域方向系统开发相关的工程配置管理方法，掌握领域方向系统项目管理能力。	GR11 项目管理：理解并掌握计算机工程管理原理与经济决策方法，并能在多学科环境中应用。	GR11.3 能在多学科环境下(包括模拟环境)，在设计开发解决方案的过程中，运用工程管理与经济决策方法。

验收成绩

CO1 运用计算机科学与技术及其应用领域工程的思想与方法，从需求分析开始，独立完成领域方向系统的设计与开发，兼顾领域方向系统开发的社会评价。（15 分）	CO2 理解领域方向系统运行应遵守的社会规范，能够在领域方向系统开发实践中自觉遵守工程职业道德和规范，履行责任。（5 分）	CO3 理解领域方向系统开发的人员组织方式，理解团队成员及负责人角色。（5 分）	CO4 理解领域方向系统开发相关的工程配置管理方法，掌握领域方向系统项目管理能力。（5 分）

报告成绩

CO1 运用计算机科学与技术及其应用领域工程的思想与方法，从需求分析开始，独立完成领域方向系统的设计与开发，兼顾领域方向系统开发的社会评价。（55 分）	CO2 理解领域方向系统运行应遵守的社会规范，能够在领域方向系统开发实践中自觉遵守工程职业道德和规范，履行责任。（5 分）	CO3 理解领域方向系统开发的人员组织方式，理解团队成员及负责人角色。（5 分）	CO4 理解领域方向系统开发相关的工程配置管理方法，掌握领域方向系统项目管理能力。（5 分）

总成绩：

指导教师评语（验收/报告）：

指导教师签名：

日期：2022 年 1 月 21 日

1. 需求与任务功能描述

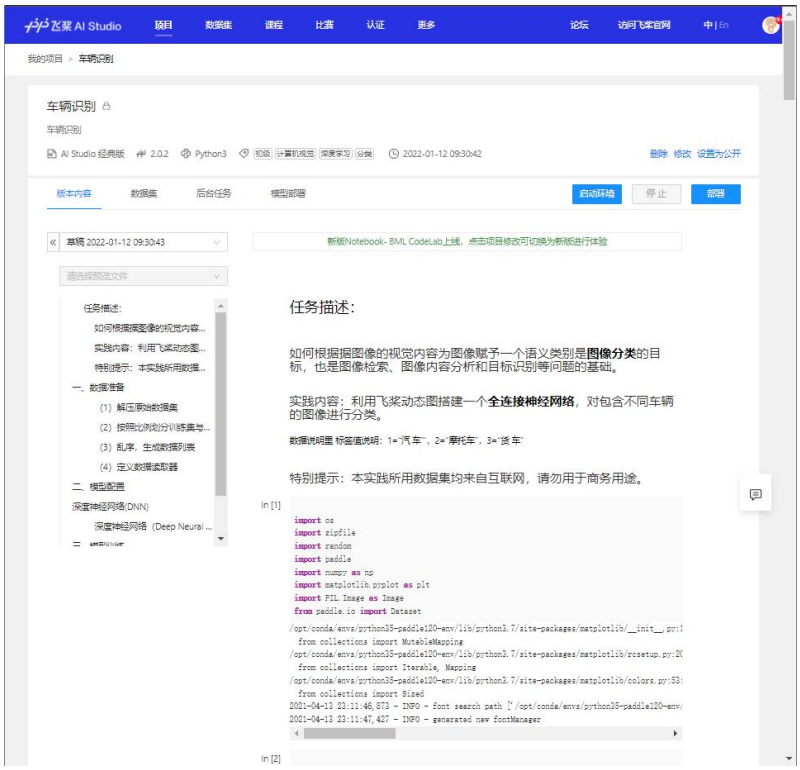
背景描述：车辆防碰撞预警系统中车辆检测与跟踪是最基本且核心的技术，许多的辅助驾驶功能都是在识别到有效车辆目标的基础上实现。车辆检测技术随着计算机视觉领域理论的发展，分成了很明显的几个阶段：从最早的简单图像处理到基本的模式识别方法，再从传统的手动设计特征的机器学习方法到基于深度卷积神经网络的方法。车辆跟踪技术也从最开始的模板匹配、卡尔曼滤波延伸出许多改进的方法。

问题描述：车辆检测在智能交通、无人驾驶、智慧城市等领域均具有十分重要的应用。飞桨平台由百度团队开发，集深度学习核心训练和推理框架、基础模型库、端到端开发套件、丰富的工具组件于一体。本课题拟研究飞桨平台的应用、车辆检测的原理，并基于飞桨平台，进行车辆检测系统的设计和实现。

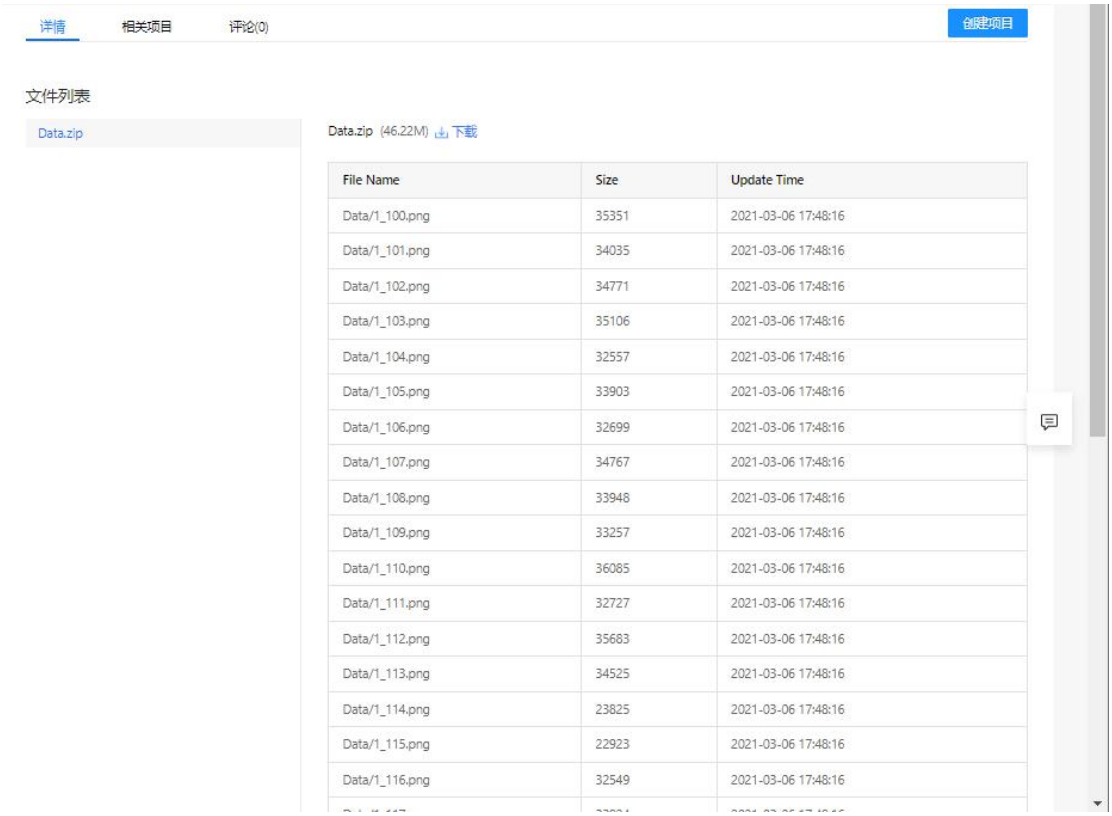
编程任务：利用飞桨动态图搭建一个全连接神经网络，对包含不同车辆的图像进行分类。如何根据据图像的视觉内容为图像赋予一个语义类别是图像分类的目标，也是图像检索、图像内容分析和目标识别等问题的基础。

预备知识：首先需要对飞桨平台的界面进行熟悉，对于以下界面，其中：

版本内容：展示当前 Notebook 最新内容.



数据集：支持部分数据类型预览.此处我采用的是摩托车、家用汽车、货车数据来自 2005 PASCAL 视觉类挑战赛（VOC2005）所使用的数据的筛选处理结果，货车图片来自网络收集，后期通过筛选处理得到。**数据说明里 标签值说明：1=“汽车”’，2=“摩托车”，3=“货车”。**



后台任务：基于 ipynb 文件在后台完成训练任务.

模型部署：用于生成在线 API 服务和趣味体验馆.

此外，飞桨平台还提供了 VisualDL 工具，VisualDL 是一个面向深度学习任务设计的可视化工具。VisualDL 利用了丰富的图表来展示数据，用户可以更直观、清晰地查看数据的特征与变化趋势，有助于分析数据、及时发现错误，进而改进神经网络模型的设计。

AI Studio Notebook 项目（Paddle1.8.0 及以上版本）已经集成 VisualDL 工具以便于您的使用,可在可视化 tab 中启动 VisualDL 服务。目前,VisualDL 支持 scalar, image, audio, graph, histogram, pr curve, high dimensional 七个组件，项目正处于高速迭代中。

其中主要的组件有：

组件名称	展示图表	作用
Scalar	折线图	动态展示损失函数值、准确率等标量数据
Image	图片可视化	显示图片，可显示输入图片和处理后的结果，便于查看中间过程的变化
Audio	音频可视化	播放训练过程中的音频数据，监控语音识别与合成等任务的训练过程
Graph	网络结构	展示网络结构、节点属性及数据流向，辅助学习、优化网络结构
Histogram	直方图	展示训练过程中权重、梯度等张量的分布
PR Curve	折线图	权衡精度与召回率之间的平衡关系
High Dimensional	数据降维	将高维数据映射到 2D/3D 空间来可视化嵌入，便于观察不同数据的相关性

2. 设计

(1) 总体设计

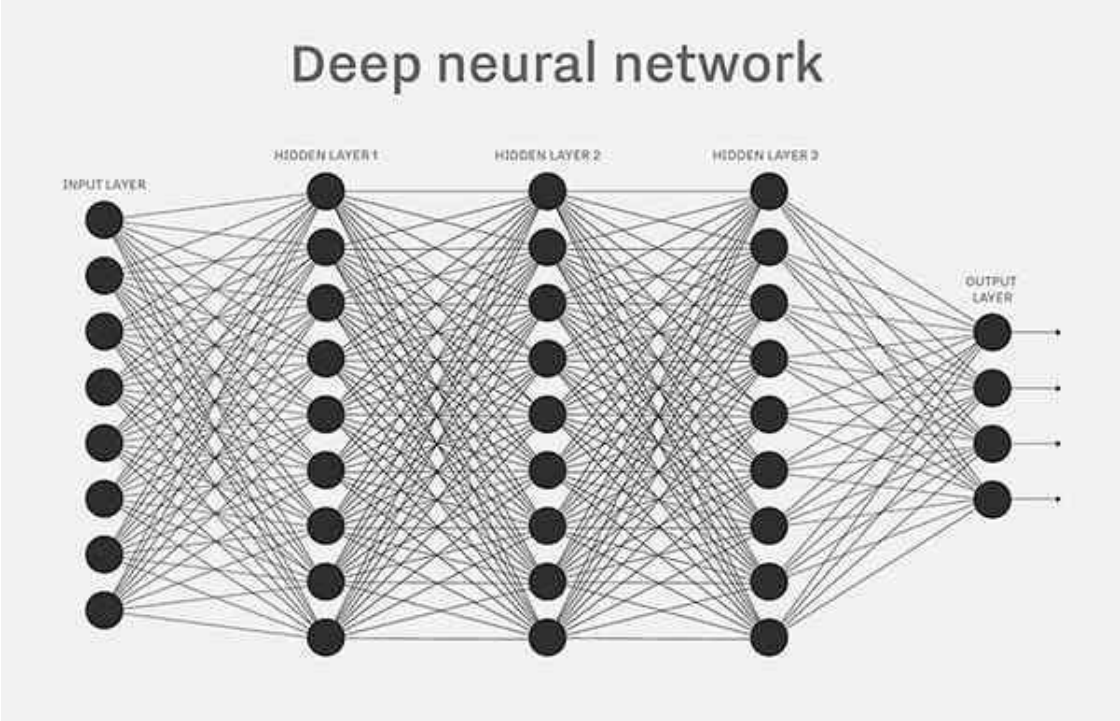
深度神经网络，（Deep Neural Networks，简称 DNN）是深度学习的基础，其结构为 input、hidden（可有多层）、output，每层均为全连接。是一种判别模型，可以使用反向传播算法进行训练。权重更新可以使用随机梯度下降法求解。

但是深度网络同时也存在问题，与其他神经网络模型类似，如果仅仅是简单地训练，深度神经网络可能会存在很多问题。常见的两类问题是过拟合和过长的运算时间。

深度神经网络很容易产生过拟合现象，因为增加的抽象层使得模型能够对训练数据中较为罕见的依赖关系进行建模。对此，权重递减化）或者稀疏）等方法可以利用在训练过程中以减小过拟合现象。另一种较晚用于深度神经网络训练的正规化方法是丢弃法（"dropout" regularization），即在训练中随机丢弃一部分隐层单元来避免对较为罕见的依赖进行建模。

反向传播算法和梯度下降法由于其实现简单，与其他方法相比能够收敛到更好的局部最优值而成为神经网络训练的通行方法。但是，这些方法的计算代价很高，尤其是在训练深度神经网络时，因为深度神经网络的规模（即层数和每层的节点数）、学习率、初始权重等众多参数都需要考虑。扫描所有参数由于时间代价的原因并不可行，因而小批量训练（mini-batching），即将多个训练样本组合进行训练而不是每次只使用一个样本进行训练，被用于加速模型训练。而最显著地速度提升来自 GPU，因为矩阵和向量计算非常适合使用 GPU 实现。但使用大规模集群进行深度神经网络训练仍然存在困难，因而深度神经网络在训练并行化方面仍有提升的空间。

其在图像分类方面的著名结果是：图像分类领域中一个公认的评判数据集是 MNIST 数据集。MNIST 由手写阿拉伯数字组成，包含 60,000 个训练样本和 10,000 个测试样本。与 TIMIT 类似，它的数据规模较小，因而能够很容易地在不同的模型配置下测试。Yann LeCun 的网站给出了多种方法得到的实验结果。截至 2012 年，最好的判别结果由 Ciresan 等人在当年给出，这一结果的错误率达到了 0.23%。



(2) 详细设计

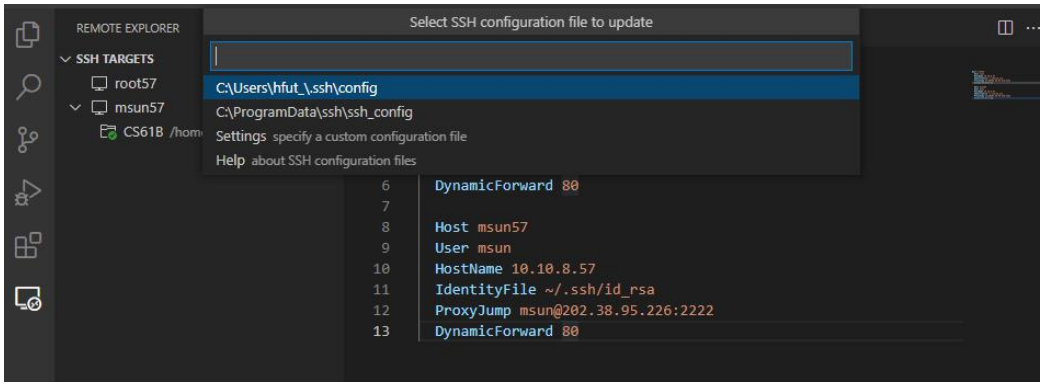
2.1 ssh & VSCode

由于 AI Studio 是不给 GPU 算力的，所以我先在带 GPU 的本机上训练好模型，通过 ssh 连接导入模型到 AI Studio，最后在飞桨上完成测试。

平常用惯了 Windows 环境的人可能会很喜欢这个办法。利用 VS Code 的远程开发插件 Remote SSH，像编辑本地文件一样地编辑远程文件，并且能充分利用 VS Code 的所有其他插件的功能。VS Code 中安装 Remote SSH 插件，在插件中使用上述命令添加服务器，即可利用 VS Code 进行远程开发。

生成 RSA 密钥对文件，并将公钥内容复制到堡垒机和服务器上。之后即可用命令 ssh -J hpzhong@202.38.95.226:2222 root@10.10.8.45 快速登录服务器。（或者用 C:\Users\hfurt_.ssh\config，VSCode 能检测到你 Windows 上的 config 文件。

```
Host msun57
User msun
HostName 10.10.8.57
IdentityFile ~/.ssh/id_rsa
ProxyJump msun@202.38.95.226:2222
DynamicForward 80
```



Connect to Host in Current Window and Open Folder.

p.s. Remember SSH TARGETS --> Configure --> Setting --> clean search bar --> change the Auto Save to "onFocusChange"

同理可以连接到 AI Studio。然后把我本机训练好的模型添加到 AI Studio 环境内，然后在飞桨平台的 Notebook 上完成验证演示。

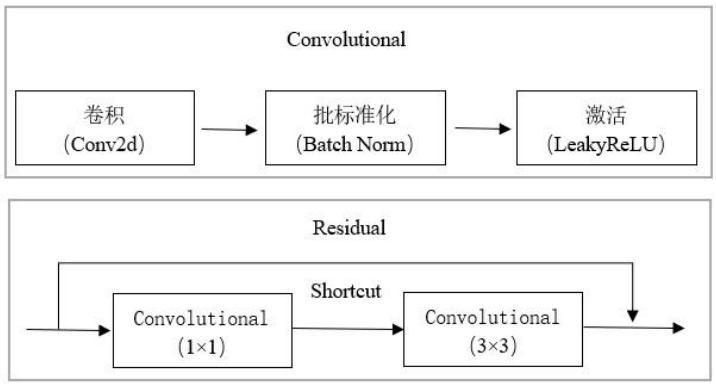
此处，我选择了 PaddleDetection 内置的 Yolov3DarkNet53 作为网络，训练

得到权重文件 yolov3.weights 进行验证。

网络层	卷积核数量	卷积核大小/步幅	特征图大小	
Convolutional	32	3×3	416×416	
Convolutional	64	3×3 / 2	208×208	
Convolutional	32	1×1		
Convolutional	64	3×3		×1
Residual			208×208	
Convolutional	128	3×3 / 2	104×104	
Convolutional	64	1×1		
Convolutional	128	3×3		×2
Residual			104×104	
Convolutional	256	3×3 / 2	52×52	
Convolutional	128	1×1		
Convolutional	256	3×3		×8
Residual			52×52	
Convolutional	512	3×3 / 2	26×26	
Convolutional	256	1×1		
Convolutional	512	3×3		×8
Residual			26×26	
Convolutional	1024	3×3 / 2	13×13	
Convolutional	512	1×1		
Convolutional	1024	3×3		×4
Residual			13×13	

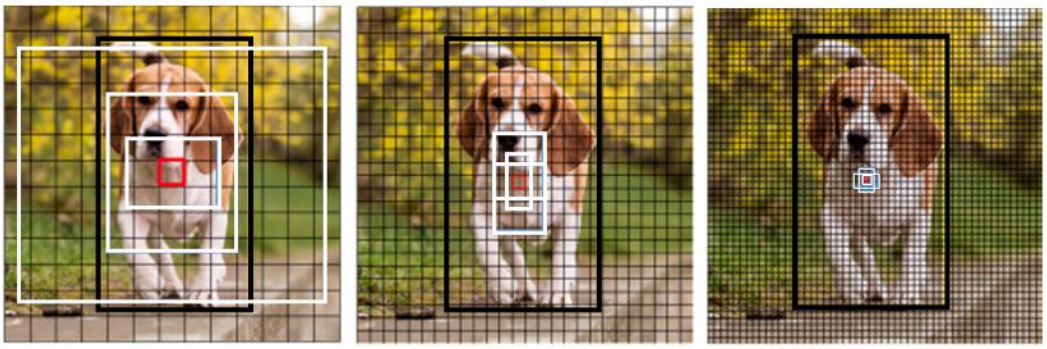
论文链接：https://arxiv.org/abs/1804.02767

Darknet-53 中主要包含卷积层（Convolutional）和残差块（Residual）这两种网络层。在每个卷积层中，使用各种卷积核完成卷积运算后，都进行了一次批标准化，之后再使用 Leaky ReLU 函数进行激活；表格中的各个方框即代表一类残差块，方框右边标识的是相应残差块的数量，每个残差块中均包含两个卷积层。它们的具体结构如下图所示：



卷积过程中，除了一直使用 3×33×3 大小的卷积核来提取特征外，还延续了 Darknet-19 中使用 1×11×1 大小的卷积核来增加特征图维度的思想，两类卷积核的数量都随着网络层数的增加而呈 22 的幂次方增长。网络中没有引入全局平均池化层，而采用简单的增大卷积步幅策略来压缩特征。每次将卷积步幅从 11 增加 22，特征图的大小就会成原来的一半，经过 55 次这样的下采样（Subsampled）操作后，最后输出的特征图大小被压缩到 13×1313×13，即输入的 1/321/32。

YOLOv3 和 YOLOv2 一样，仍采用了 Faster R-CNN 中提出的 Anchor Box 机制从特征图中检测各类目标。Anchor Box 机制相当于给模型提供了更多了先验知识，拥有了这些先验知识，模型只要学会预测真实包围盒各参量相对于 Anchor Box 的偏移大小，而非 Bounding Box 的各个具体参数，该机制使模型的学习过程容易了很多。此外，YOLOv2 中还使用了 K-Means 聚类算法对训练集中真实 Bounding Box 的大小进行聚类运算，这样得到几种 Anchor Box 尺寸要比原 Faster R-CNN 手工选取的尺寸具有更优的先验性。

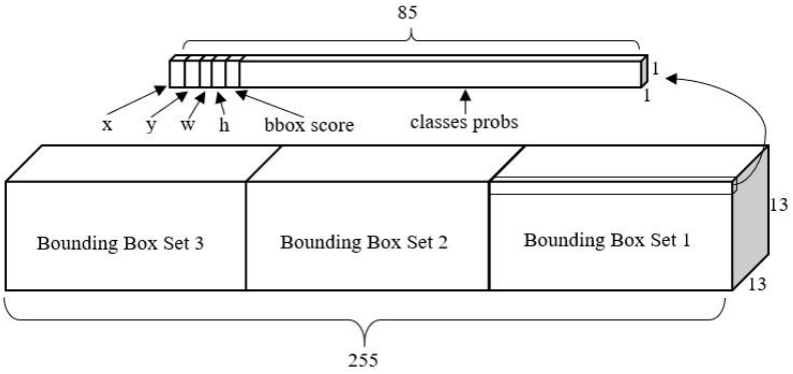


(a) 13×13　　(b) 26×26　　(c) 52×52

YOLOv3 检测模型输出的是三个尺度不一的张量，需要把它们进行拆分，才能得到想要的检测结果。从前面的流程图可知，在输入大小为 416×416416×416 的情况下，输出张量分别包含 13×1313×13、26×2626×26、52×5252×52 个网格。除网格数量外，这些张量的结构都是相同的，这里仅以其中大小为 13×1313×13 张量为例进行分析。

如下图，由于各尺度的特征图都使用了 33 种不同的 Anchor Box 来实现包围盒回归，因此输出张量的各网格中均包含了 33 个 Bounding Box 预测结果，各预测结果均由 Bounding Box 的 44 个位置（xx、yy、ww、hh）、11 个置信度评分

（bbox score）以及 CC 类目标的 CC 个存在概率值（classes probs）组成。作者使用了包含 8080 类常见物体的 COCO 数据集来训练 YOLOv3，因此一个预测结果是一个 8585 维的向量，整个张量的深度则为 255255。



具体训练和验证

数据准备

（1）解压原始数据集

```
#解压原始数据集
def unzip_data(src_path,target_path):
    '''''
    解压原始数据集，将 src_path 路径下的 zip 包解压至 target_path 目录下
    '''
    if(not os.path.isdir(os.path.join(target_path,'Data'))):
        z = zipfile.ZipFile(src_path, 'r')
        z.extractall(path=target_path)
        z.close()
        print('数据集解压完成')
    else:
        print('文件已存在')

#数据集解压完成
unzip_data('data/data72920/Data.zip','/home/aistudio/work/')
```

（2）按照比例划分训练集与验证集

```

    # 对训练列表进行乱序
    random.shuffle(all_data_list)

    with open(train_list_path, 'a') as f1:
```

```
        with open(eval_list_path, 'a') as f2:
            for ind, img_path_label in enumerate(all_data_list):
                #划分测试集和训练集
                if ind % 10 == 0:
                    f2.write(img_path_label)
                else:
                    f1.write(img_path_label)
    print ('生成数据列表完成！')
```

（3）乱序，生成数据列表

```
def get_data_list(target_path, train_list_path, eval_list_path):
    '''''
    生成数据列表
    '''
    data_dir = 'work/Data'
    all_data_list  = []

    for im in os.listdir(data_dir):
        img_path = os.path.join(data_dir, im)
        img_label = str(int(im.split('_')[0])-1)
        all_data_list.append(img_path + '\t' + img_label + '\n')
```

```
#参数初始化
src_path=train_parameters['src_path']
target_path=train_parameters['target_path']
train_list_path=train_parameters['train_list_path']
eval_list_path=train_parameters['eval_list_path']

#解压原始数据到指定路径
unzip_data(src_path,target_path)

#每次生成数据列表前，首先清空 train.txt 和 eval.txt
with open(train_list_path, 'w') as f:
    f.seek(0)
    f.truncate()
with open(eval_list_path, 'w') as f:
    f.seek(0)
    f.truncate()

#生成数据列表
get_data_list(target_path,train_list_path,eval_list_path)

#文件已存在
```

```
#生成数据列表完成!
```

（4）定义数据读取器

```
class dataset(Dataset):
    def __init__(self, data_path, mode='train'):
        """
        数据读取器
        :param data_path: 数据集所在路径
        :param mode: train or eval
        """
        super().__init__()
        self.data_path = data_path
        self.img_paths = []
        self.labels = []

        if mode == 'train':
            with open(os.path.join(self.data_path, "train.txt"), "r", encoding="utf-8") as f:
                self.info = f.readlines()
            for img_info in self.info:
                img_path, label = img_info.strip().split('\t')
                self.img_paths.append(img_path)
                self.labels.append(int(label))

        else:
            with open(os.path.join(self.data_path, "eval.txt"), "r", encoding="utf-8") as f:
                self.info = f.readlines()
            for img_info in self.info:
                img_path, label = img_info.strip().split('\t')
                self.img_paths.append(img_path)
                self.labels.append(int(label))


    def __getitem__(self, index):
        """
        获取一组数据
        :param index: 文件索引号
        :return:
        """
        # 第一步打开图像文件并获取 label 值
        img_path = self.img_paths[index]
        img = Image.open(img_path)
```

```
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = np.array(img).astype('float32')
        img = img.transpose((2, 0, 1)) / 255
        label = self.labels[index]
        label = np.array([label], dtype="int64")
        return img, label

    def print_sample(self, index: int = 0):
                print(" 文 件 名 ",  self.img_paths[index],  "\t 标 签 值 ", self.labels[index])

    def __len__(self):
        return len(self.img_paths)
```

```
#训练数据加载
train_dataset = dataset('/home/aistudio/data',mode='train')
train_loader = paddle.io.DataLoader(train_dataset,
                                    batch_size=train_parameters['train_batch_size'],
                                   shuffle=True
                                   )
#测试数据加载
eval_dataset = dataset('/home/aistudio/data',mode='eval')
eval_loader = paddle.io.DataLoader(eval_dataset,
                                   batch_size=train_parameters['train_batch_size'],
                                  shuffle=False
                                  )
In [9]

train_dataset.print_sample(200)
print(train_dataset.__len__())
eval_dataset.print_sample(0)
print(eval_dataset.__len__())
print(eval_dataset.__getitem__(10)[0].shape)
print(eval_dataset.__getitem__(10)[1].shape)
文件名 work/Data/1_178.png     标签值 0
1414
文件名 work/Data/3_1349.png     标签值 2
158
(3, 120, 120)
(1,)
```

模型训练：

```
def draw_process(title,color,iters,data,label):
    plt.title(title, fontsize=24)
    plt.xlabel("iter", fontsize=20)
    plt.ylabel(label, fontsize=20)
    plt.plot(iters, data,color=color,label=label)
    plt.legend()
    plt.grid()
    plt.show()

model = MyDNN()
model.train()
cross_entropy = paddle.nn.CrossEntropyLoss()
optimizer = paddle.optimizer.Adam(learning_rate=train_parameters['learning_strategy']['lr'],
                                  parameters=model.parameters())

steps = 0
Iters, total_loss, total_acc = [], [], []

for epo in range(train_parameters['num_epochs']):
    for _, data in enumerate(train_loader()):
        steps += 1
        x_data = data[0]
        y_data = data[1]
        predicts = model(x_data)
        loss = cross_entropy(predicts, y_data)
        acc = paddle.metric.accuracy(predicts, y_data)
        loss.backward()
        optimizer.step()
        optimizer.clear_grad()
        if steps % train_parameters["skip_steps"] == 0:
            Iters.append(steps)
            total_loss.append(loss.numpy()[0])
            total_acc.append(acc.numpy()[0])
            #打印中间过程
            print('epo: {}, step: {}, loss is: {}, acc is: {}'\
                  .format(epo, steps, loss.numpy(), acc.numpy()))
        #保存模型参数
        if steps % train_parameters["save_steps"] == 0:
            save_path = train_parameters["checkpoints"]+"/"+"save_dir_" + str(steps) + '.pdparams'
            print('save model to: ' + save_path)
```

```
            paddle.save(model.state_dict(),save_path)
paddle.save(model.state_dict(),train_parameters["checkpoints"]+"/"+"save_dir_final.pdparams")
draw_process("trainning loss","red",Iters,total_loss,"trainning loss")
draw_process("trainning acc","green",Iters,total_acc,"trainning acc")
epo: 0, step: 50, loss is: [1.5851377], acc is: [0.375]
epo: 0, step: 100, loss is: [1.012798], acc is: [0.5]
epo: 0, step: 150, loss is: [0.93412054], acc is: [0.5]
epo: 1, step: 200, loss is: [1.1832764], acc is: [0.625]
epo: 1, step: 250, loss is: [0.47708118], acc is: [0.875]
epo: 1, step: 300, loss is: [0.49956387], acc is: [0.875]
epo: 1, step: 350, loss is: [0.94739217], acc is: [0.375]
epo: 2, step: 400, loss is: [0.9151466], acc is: [0.875]
epo: 2, step: 450, loss is: [0.66895646], acc is: [0.75]
epo: 2, step: 500, loss is: [1.4960159], acc is: [0.375]
save model to: /home/aistudio/work/checkpoints/save_dir_500.pdparams
epo: 3, step: 550, loss is: [0.6696862], acc is: [0.75]
epo: 3, step: 600, loss is: [0.89107466], acc is: [0.625]
epo: 3, step: 650, loss is: [0.63641155], acc is: [0.75]
epo: 3, step: 700, loss is: [0.7211526], acc is: [0.75]
epo: 4, step: 750, loss is: [1.0665362], acc is: [0.5]
epo: 4, step: 800, loss is: [0.6904267], acc is: [0.5]
epo: 4, step: 850, loss is: [0.716236], acc is: [0.625]
/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:2349: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working
  if isinstance(obj, collections.Iterator):
/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:2366: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working
  return list(data) if isinstance(data, collections.MappingView) else data
```

验证：

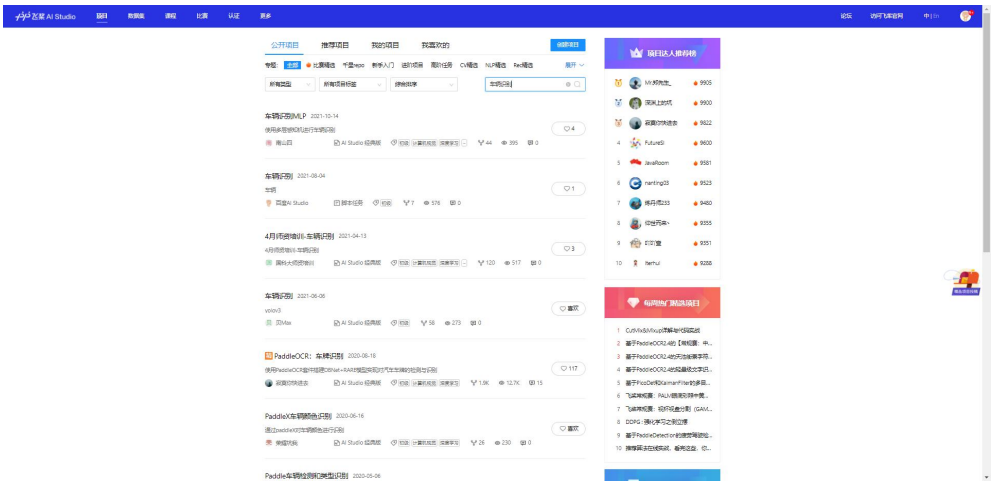
```
import numpy as np
import cv2 as cv
import os
import time
import matplotlib.pyplot as plt
'''''
yolo_dir = '/home/aistudio/data/data93876/yolov3.weights'  # YOLO 文件路径
yolo_dir1 = 'work'
weightsPath = os.path.join(yolo_dir, 'yolov3.weights')  # 权重文件
```

```
configPath = os.path.join(yolo_dir1, 'yolov3.cfg')  # 配置文件
labelsPath = os.path.join(yolo_dir1, 'coco.names')  # label 名称
imgPath = os.path.join(yolo_dir1, '1.png')  # 测试图像
'''
weightsPath = '/home/aistudio/data/data93876/yolov3.weights'
configPath = 'work/yolov3.cfg'
labelsPath = 'work/coco.names'
imgPath = 'work/1.png'
CONFIDENCE = 0.5  # 过滤弱检测的最小概率
THRESHOLD = 0.4  # 非最大值抑制阈值

net = cv.dnn.readNetFromDarknet(configPath, weightsPath)  # #  利用下载的文件
print("[INFO] loading YOLO from disk...")  # # 可以打印下信息

img = cv.imread(imgPath)
blobImg = cv.dnn.blobFromImage(img, 1.0/255.0, (416, 416), None, True,
                               False)  # # 用 blobFromImage 这个函数来转格式，将输入的图层转为 biob 格式
net.setInput(blobImg)  # # 调用 setInput 函数将图片送入输入层

# 获取网络输出层信息（所有输出层的名字），设定并前向传播
outInfo = net.getUnconnectedOutLayersNames()  # #yolo 在每个 scale 都有输出，outInfo 是每个 scale 的名字信息，供 net.forward 使用
start = time.time()
layerOutputs = net.forward(outInfo)  # 得到各个输出层的、各个检测框等信息，是二维结构。
end = time.time()
print("[INFO] YOLO took {:.6f} seconds".format(end - start))  # # 可以打印下信息

# 拿到图片尺寸
(H, W) = img.shape[:2]
# 过滤 layerOutputs
#      layerOutputs      的      第      1      维      的      元      素      内容: [center_x, center_y, width, height, objectness, N-class score data]
# 过滤后的结果放入：
boxes = [] # 所有边界框（各层结果放一起）
confidences = [] # 所有置信度
classIDs = [] # 所有分类 ID

# # 1）过滤掉置信度低的框框
for out in layerOutputs:  # 各个输出层
    for detection in out:  # 各个框框
```

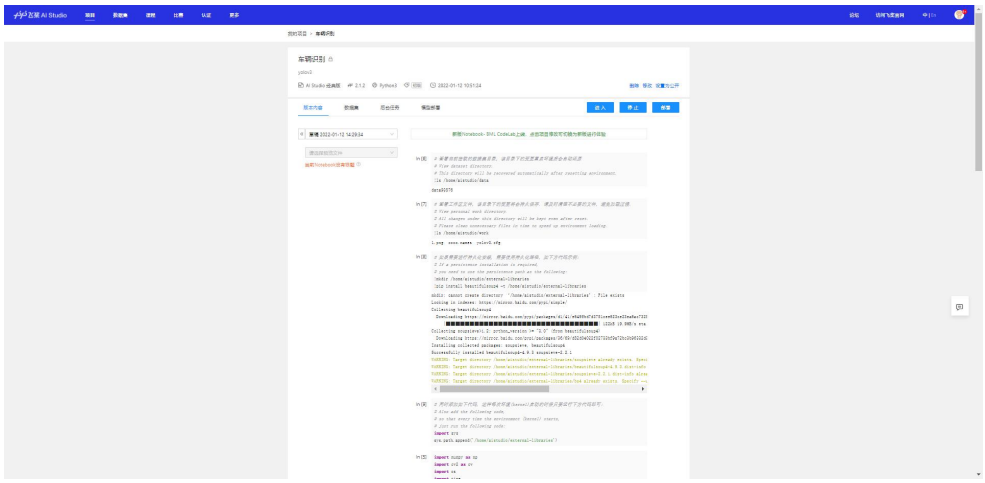
```
        # 拿到置信度
        scores = detection[5:]  # 各个类别的置信度
        classID = np.argmax(scores)  # 最高置信度的 id 即为分类 id
        confidence = scores[classID]  # 拿到置信度

        # 根据置信度筛查
        if confidence > CONFIDENCE:
            box = detection[0:4] * np.array([W, H, W, H])  # 将边界框放会图片尺寸
            (centerX, centerY, width, height) = box.astype("int")
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))
            boxes.append([x, y, int(width), int(height)])
            confidences.append(float(confidence))
            classIDs.append(classID)

# # 2）应用非最大值抑制(non-maxima suppression，nms)进一步筛掉
idxs = cv.dnn.NMSBoxes(boxes, confidences, CONFIDENCE, THRESHOLD) # boxes 中，保留的 box 的索引 index 存入 idxs
# 得到 labels 列表
with open(labelsPath, 'rt') as f:
    labels = f.read().rstrip('\n').split('\n')
# 应用检测结果
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(labels), 3), dtype="uint8")  # 框框显示颜色，每一类有不同的颜色，每种颜色都是由 RGB 三个值组成的，所以 size 为 (len(labels), 3)
if len(idxs) > 0:
    for i in idxs.flatten():  # indxs 是二维的，第 0 维是输出层，所以这里把它展平成 1 维
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        color = [int(c) for c in COLORS[classIDs[i]]]
        cv.rectangle(img, (x, y), (x+w, y+h), color, 2)  # 线条粗细为 2px
        text = "{}: {:.4f}".format(labels[classIDs[i]], confidences[i])
        cv.putText(img, text, (x, y-5), cv.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)  # cv.FONT_HERSHEY_SIMPLEX 字体风格、0.5 字体大小、粗细 2px
plt.figure()
title="car_detect"
plt.imshow(img)
plt.title(title)
plt.show()
```

3. 用户手册

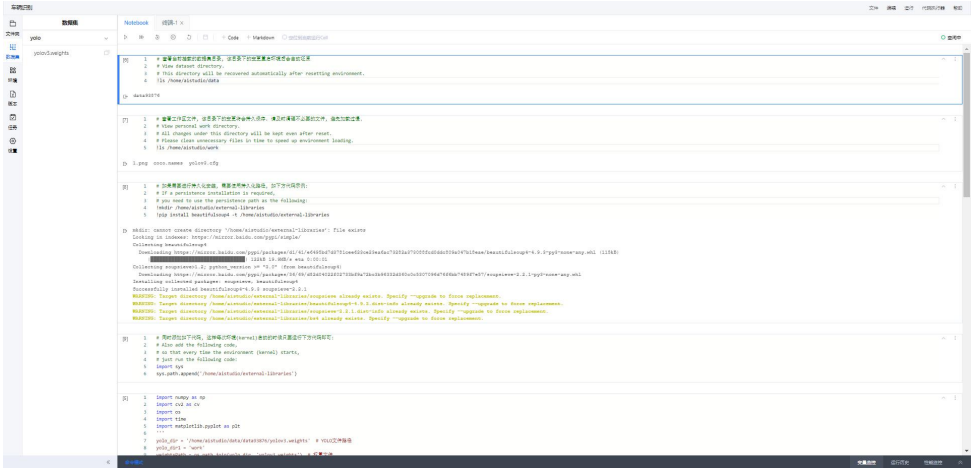
1、打开 AI Studio，Fork 本 yolov3 车辆识别项目（或是自己创建）；



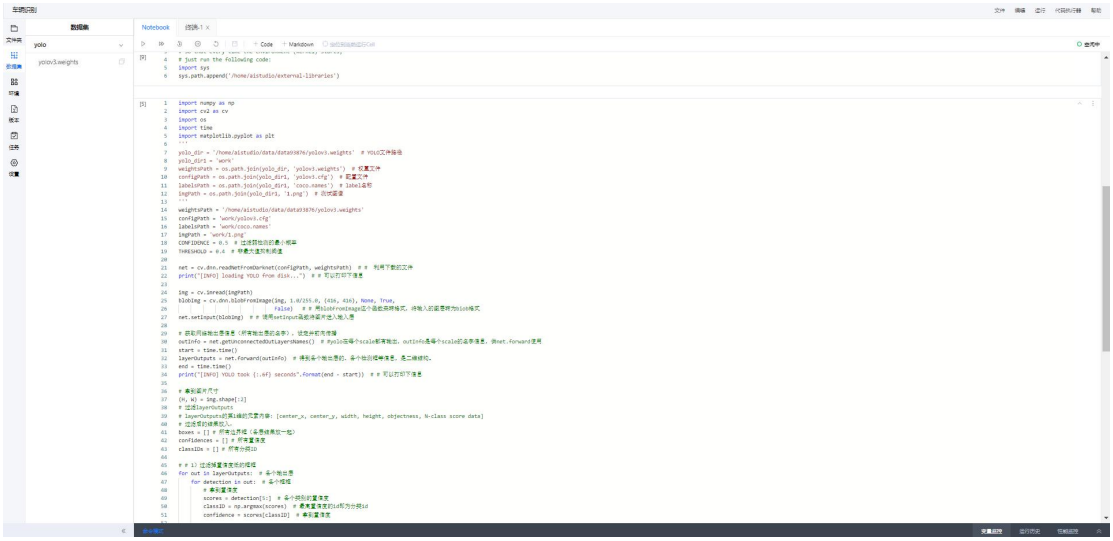
2. 点击创建环境，然后进入



3、进入之后，找到测试主代码（前三个 cell 是系统创建新项目自动生产的，可以忽略）



4、点击主测试代码的运行按钮



5、即可在该页面得到运行结果

4. 系统功能测试

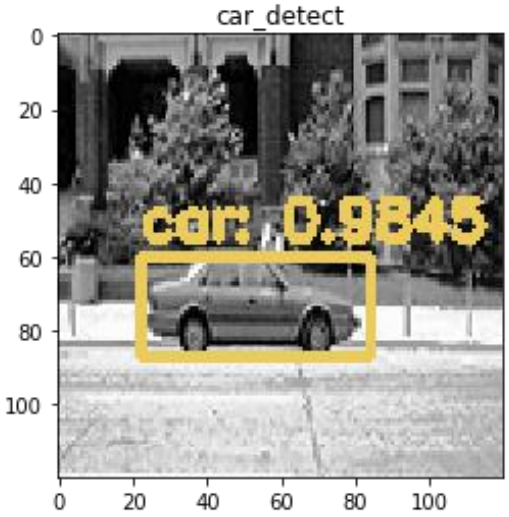
测试样例：

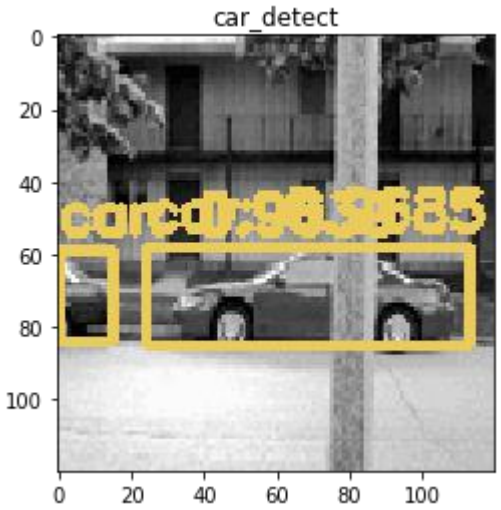
1_42.png

1_43.png

1_44.png

测试结果：





5．存在的问题

1、标签重写问题

yolov3 存在的问题，标签重写是指由于 yolo 特有的网格负责预测 bbox 的特点，可能会出现两个物体分配给了同一个 anchor，导致仅仅有一个物体被保留负责预测，另一个物体被当做背景忽略了。当输入分辨率越小，物体越密集，物体的 wh 大小非常接近时候，标签重写现象比较严重。如上图所示，红色表示被重写的 bbox，可以看出 27 个物体有 10 个被重写了。

具体来说，以 416＊416 大小的图像为例，在图像分辨率随着卷积下降到 13＊13 的特征图大小时，这时候特征图一个像素点的对应是 3232 大小的图像 patch。而 YOLOV3 在训练时候，如果出现相同两个目标的中心位于同一个 cell，且分配给同一个 anchor，那么前面一个目标就会被后面目标重写，也就是说两个目标由于中心距离太近以至于在特征图上将采样成为同一个像素点的时候，这时候其中有个目标会被重写而无法进行到训练当中。

这种现象在 coco 数据上不明显的原因是 bbox 分布比较均匀，不同大小物体会分配到不同预测层，标签重写概率比较低。但是在很多实际应用中，比如工业界的特定元件检测时候，物体排布非常紧密，且大小几乎一致，此时就可能会出现标签重写问题了，作者论文指出在 Cityscapes 数据上该现象也比较明显。

2、kmean 计算 anchor 存在的问题

yolo 系列采用 kmean 算法聚类得到特定要求的 9 个 anchor，并且以每三个为一组，用于大输出图(检测小物体)，中等输出图和小输出图层(检测大物体)的默认 anchor。可以看出不同大小的物体会被这三组 anchor 分配到不同预测层进行预测。

但是这种 kmean 算法得出的结果是有问题的，在实际项目中也发现了。前面说过大部分特定场景的目标检测数据集，并不是和 coco 自然场景一样，啥尺度都有，实际项目中大部分物体都是差不多大的，或者说仅仅有特定的几种尺度，此时采用 kmean 这一套流程就会出现：几乎一样大的物体被强制分到不同层去预测，这个训练方式对网络来说非常奇怪，因为物体和物体之间 wh 可能就差了一

点点，居然强制分层预测，这明显不合理。本文作者生成的仿真数据其实也是这个特点。

作者论文指出,kmean 这种设置，仅仅在：［公式］情况下采用合理的。其中 r 是输入图片分辨率，例如 416。该式子的意思是物体的大小分布是满足边界为 0 到 r 的均匀分布，也就是说在 416x416 图片上，各种大小尺度的 bbox 都会存在的情况下，kmean 做法是合理的。但是可能大部分场景都是：［公式］即均值为 0.5r，标准差为 r 的物体分布，如果按照默认的 kmean 算法对 anchor 的计算策略，那么由于大部分物体都是中等尺寸物体，会出现其余两个分支没有得到很好训练，或者说根本就没有训练，浪费网络。

6．进一步改进及其思路

对于标签重新问题，没有啥特别好的办法，只能通过要么增加输入图片分辨率大小；要么增加输出特征图大小实现。本文的做法是增加输出特征图大小。原始的 yolov3，输入大小是输出特征图的 8/16 和 32 倍，通过上述数据可以发现标签重写比例蛮高的。而通过增加输出特征图大小后可以显著降低重写比例。

而对于 kmean 聚类带来的问题，有两种解决办法：

(1) kmean 聚类流程不变，但是要避免出现小物体被分配到小输出特征图上面训练和大物体被分配到大输出特征图上面训练问题，具体就是首先基于网络输出层感受野，定义三个大概范围尺度，然后设置两道阈值，强行将三个尺度离散化分开；然后对 bbox 进行单独三次聚类，每次聚类都是在前面指定的范围内选择特定的 bbox 进行，而不是作用于整个数据集。主要是保证 kmean 仅仅作用于特定 bbox 大小访问内即可，就可以避免上面问题了。但是缺点也非常明显，如果物体大小都差不多，那么几乎仅仅有一个输出层有物体分配预测，其余两个尺度在那里空跑，浪费资源。

(2) 就只有一个输出层，所有物体都是在这个层预测即可。可以避免 kmean 聚类问题，但是为了防止标签重写，故把输出分辨率调高，此时就完美了。作者实际上采用的是 1/4 尺度输出，属于高分辨率输出，重写概率很低。

附录——源程序

见压缩文件 PaddleDetection.zip。