人工智能项目报告

**软件学院 周城程 51194501213**

虽然本科阶段没有接触过人工智能相关的课程或者是项目，研究生阶段的研究方向也不是和人工智能相关的，但是”All in AI”让我一直都对人工智能领域充满了好奇和兴趣，因此这学期选了“人工智能”课程，希望能了解一点人工智能相关的知识。

课程作业是做人工智能相关的项目，由于我是单人组队，所以选择的是“读论文复现”任务。我选择的是CVPR2018年的一篇论文“Cascade R-CNN: Delving into High Quality Objection”。

这篇论文是针对目标检测算法做的一些改进，从目标检测的两方面——分类和定位做了分析观察和实验，发现当前流行的目标检测算法存在的问题，并通过实验效果和原理分析提出了一个有效而又简单的解决方法。

下面我将从论文理解、代码分析、结果展示与分析、总结四方面进行展示。

1. 论文理解

该篇论文是针对detection的classification和location做了细致的观察和实验分析发现当前流行的detection网络存在的问题，并提出了简单易行但十分有效的办法，能够有效的提高目标检测的准确率，特别是在coco数据集上效果很明显。

由于文章信息量比较大，下面我将从发现问题、解决方法、实验结果详细介绍我从该篇文章获得的信息。

1. 发现问题

在二级的目标检测网络中，例如：RCNN、Faster-RCNN等，有一个卷积层用来提取图片的特征图（feature map），然后根据获得的特征图来提取候选区域（proposal，一般选取～2000个），接着会有1个header结构（在RCNN中就是SVM分类器；在Faster-RCNN中就是RPN网络）来对每个proposal做两件事情：分类（classification）和边界框回归（bounding box regression）。

在训练阶段，由于候选区域可能只包含物体的一部分，所以需要界定正负样本。每个proposal都会和标注好的真值（ground truth）进行交并比（IoU值）计算。在神经网络搭建时，会提前定义好交并比的阈值（IoU thresholds）参数。所以，不同的proposal会被分为正样本（positives）和负样本（negatives）。

在bbox reg中，对每个标记为正样本的bbox会向预先标定好的框进行回归。

所以，可以观察到，通过IoU来界定候选框是正样本或是负样本，IOU阈值的选取对训练和测试都有较大的影响，所以调IoU阈值的参数也成为目标检测工作的一个难题。

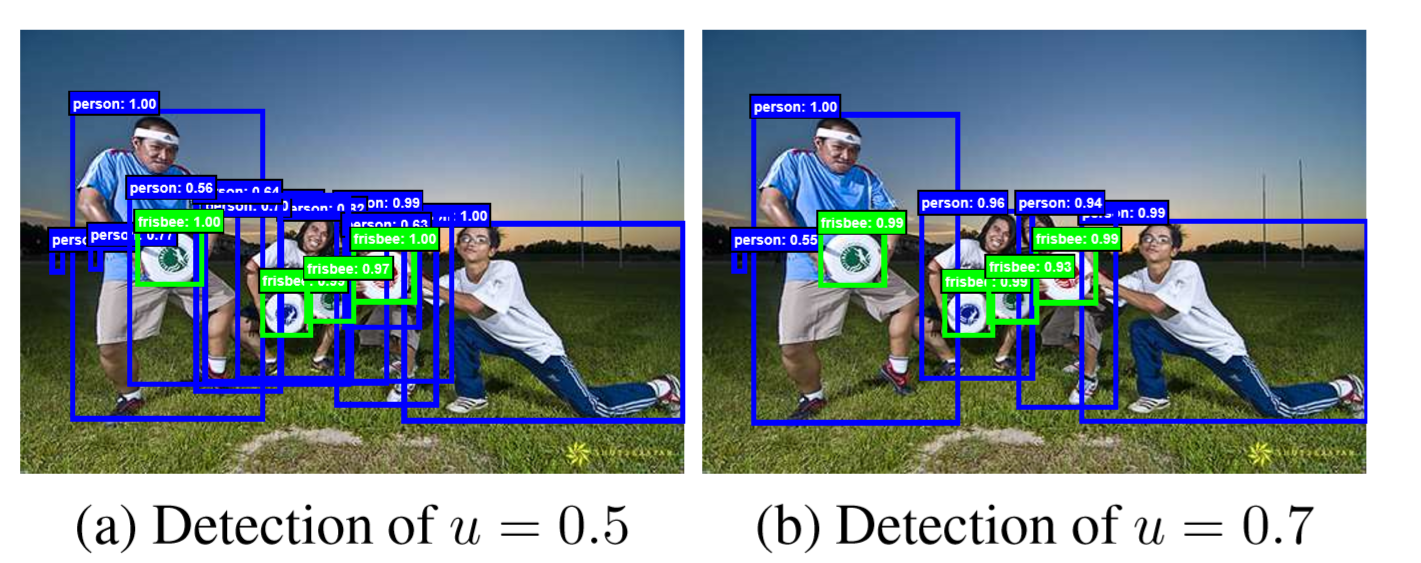


图 1 不同阈值下目标检测的效果

文中给了一个例子，如上图展示。常见的IoU阈值设定为u = 0.5，由于阈值设定的较为宽松，容易引入较多的噪声，如上左图，它框定了许多实际上并不是目标的结果，这反映u = 0.5，无法排除一些假阳性样本（close false positive）。当将阈值提高u = 0.7,框定的目标准确，但是这也会引起另一个问题——由于阈值设定较高，造成正样本数量大量减少，如下图展示的是候选区域数量随阈值的提高减少的情况，这容易引起过拟合的情况。综合，当阈值设定较低时，样本数量较多，但是容易引入噪声；当阈值设定较高时，有效过滤噪声，但是样本数量较少，容易过拟合。

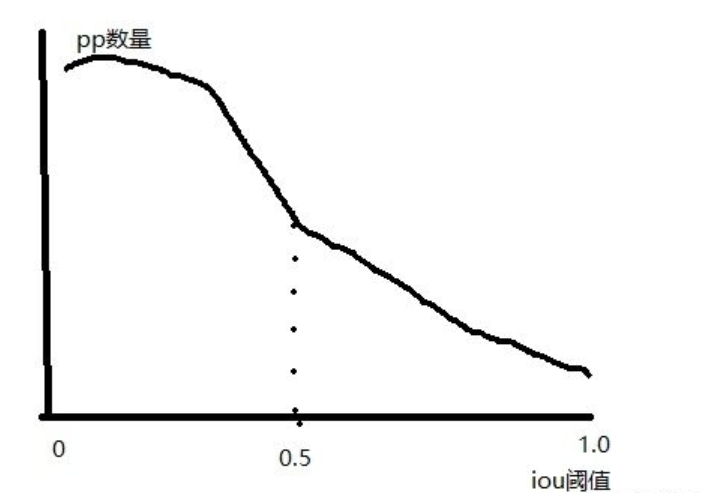


图 2 候选区域（pp）的数量随着IOU阈值的变化情况

作者做了一些实验验证了问题情况，如下图。

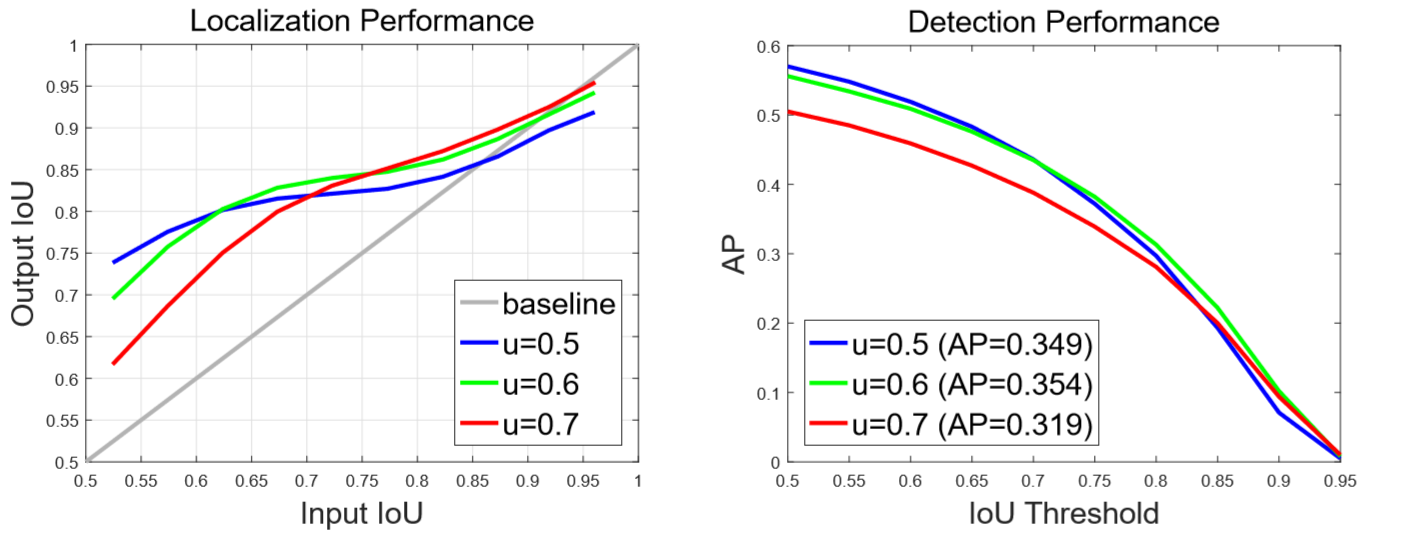


图 3 单一阈值检测器定位和检测效果

左图，横轴Input IoU代表的是候选区域自身的IoU值，纵轴是经过“边界框回归”得到的修正后的候选区域的IoU值，不同线条代表的是利用不同的IoU阈值u（u = 0.5，0.6，0.7）训练出来的单一阈值检测器。可以看到

* 回归得到的候选区域的IoU普遍比输入的IoU要高（根据灰色的线界定，在灰线上的，就是输出 > 输入）；
* IoU值在0.55-0.6范围的候选区域，用阈值u= 0.5训练出来的检测器的回归效果是最好；IoU在0.6-0.75范围内的候选区域，用阈值u = 0.6训练出来的检测器回归效果最好；IoU在0.75-的候选区域，用阈值u = 0.7训练出来的检测器回归效果最好。

所以根据左图展示的和我们观察得到如下结论：

* 检测器有效果，能提高定位的准确度；
* 候选区域的IoU值和训练器训练时设定的阈值u相近的时候，效果最好，阈值相距太远，就会存在mismatch问题。

所以自然联想到，由于候选区域IoU的分布范围广，用单一阈值的检测器不管如何修改参数，均不能达到很好的训练和检测效果。我们以IoU = 0.5（常见的界定正负样本的阈值）作为阈值，那么所有IoU > 0.5的候选区域都会被当作正样本进行边界框回归，那些在0.5-0.6左右的候选区域经过定位后得到的边界框效果很好；但是那些候选框的IoU > 0.6，经过定位后得到的效果就不是很好。

但是单纯的提高阈值训练出来的单一检测器的检测也不会很好，上右图给出了数据展示。右图的横坐标表示inference阶段，判定box为TP的IoU阈值，纵轴为Map。可以看到，在相同的测试阈值的设定下，经过不同阈值训练出来的检测器效果是不一样的。当候选区域划定正负样本的阈值设定为u = 0.5时，u = 0.7的检测器的效果是最差的——u = 0.7的检测器的AP值最小。原因是用高阈值训练时，由于阈值较高，正样本数量就会大大减少，因为它的过拟合（overfitting）情况严重。

所以总结得到单个IoU级别优化的检测器在其他级别不一定是最佳的，高IoU值的候选区域需要匹配高IoU值训练的检测器。

1. 解决方法

既然单一阈值的检测器没法在各个IoU值下的候选区域有良好效果，作者就想到了采用多级（muti-stage）检测器，每个每个级别有不同的IoU阈值。

它的基本思想是:用一个级别输出的经过bbox reg优化后的候选区域去训练下一个级别。根据图3左，某个候选区域经过检测器后输出的IoU都是比原来好的，它的所以它的输出再经过下一个更高阈值的detector后效果一定更好。

如图为论文提出的multi-stage的结构图。

假设有3个训练好的detector串联，它们分别是用IoU阈值0.5/0.6/0.7训练得到的，现在输入一个IOU=0.55的候选区域，经过IoU阈值为0.5的detector后，候选区域的IOU变成0.7；再经过0.6的detector，候选区域的IoU变成0.8；再经过0.7的detector，候选区域的IOU变成0.89，可以看到经过一系列的detector，候选区域的输出的定位效果比单一的detector要好。

经过一个卷积层提取图片整体的特征图，除了第一个bbox ( B0 )是原始的神经网络（eg，fast-rcnn的RPN）提供的proposal。之后的都是根据前一低阈值回归得到的bbox作为输入——B1->pool、B2->pool，每个检测头的阈值不一样——H1、H2、H3（逐渐提高），并且由于训练的样本池（pool）是前一回归的结果，所以使已经经过优化的，所以pool内的样本质量优于之前的，在训练中，能保证每个检测头有足够多的正样本进行训练；在测试时，经过多个检测头的bbox reg的优化，最后输出的bbox效果也会变好。

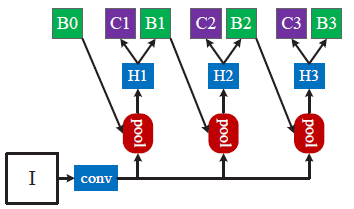


图 4 Cascade-RCNN网络结构图

(I:代表输入的图片；conv：代表神经网络骨架，主要就是提取图片的特征图；B：是边界框；C：分类；pool：池化B0：是通过图片特征图提取的候选区域； H：检测头)

在训练过程中，也用相同的策略，那么训练低阈值的检测器用了低IoU选出的正负样本，通过bbox reg得到的样本的IoU会普遍提高，样本质量不断的提高，正好适配了高阈值的检测器，并且虽然阈值提高，但是样本的IoU值也普遍提高，使得正样本数量不会减少太多，从而也避免了样本数目太少而引起的过拟合问题。

论文也给出了图证明在训练阶段，3级级联的检测器输入的样本的分布情况。如下图。

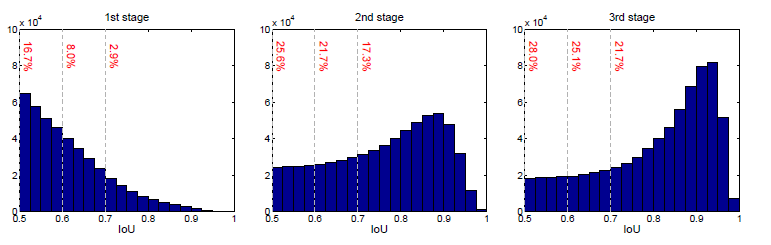


图 5 训练样本的IoU直方图

（红色字代表的是候选区域IoU值大于IoU阈值的样本占总样本的比例；第一阶段输入的样本是Faster-RCNN的RPN网络的输出的候选区域）

可以看出虽然阈值不断提高，但是对应的阈值内的样本没有变少。初始从RPN网络出来的proposal的IoU都较小——质量较差；经过第一次bbox reg之后，大部分的proposal的IoU均明显提高，所以高IoU值的候选区域变多，所以高阈值下的样本数目没有变少，甚至比例还有所提高。这可以证明Cascade-RCNN的训练结果的确是有效的。

论文提出在训练和验证的过程中，均是应用相同的网络（而不是单独训练3个阈值不同的header）。

可以看到这个网络的结构是很明确的。前人也有做过类似的多级目标检测网络，但是它们多少都存在些问题，所以Cascade-RCNN相较有较好的性能优势。论文中对一些类似的神经网络结构和Cascade-RCNN网络相比较，在理论上分析了自身的优势性。

如下图（a）为基础的Faster-RCNN基础网络结构，Cascade-RCNN也是基于该网络基础上的；（b）Iterative BBox的网络结构；（c）为Integral Loss的网络结构。

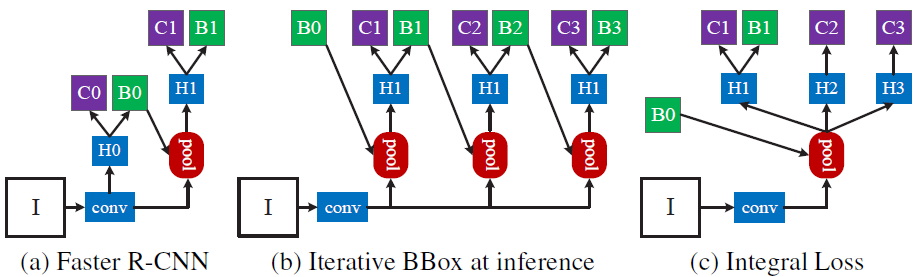


图 6 不同的几个网络结构图

分别和图4的网络相比较，可以明显的看到不同之处。

Iterative BBox网络结构和Cascade-RCNN很像，但是它的每个检测头都是相同的——根据图3了解到的问题：单一阈值的检测器无法对所有IoU范围的proposal进行良好的bbox reg；即B1回归得到的样本的IoU一般都从0.5->0.75以上了，再用相同阈值的检测器就没有很好的效果。

并且detector会改变样本的分布，如果依旧使用原来的阈值的检测器效果不好，如下图。第一行：横纵轴分别表示回归目标中的box的x方向和y方向的偏移量。可以发现，从1st->2nd，proposal的分布发生了很大的变化。样本逐渐靠近Ground Truth——质量变高了；噪声经过bbox reg之后也提高了IoU，离群点变多，这时如果不提高阈值就会引入大量的噪声干扰。所以，Iterative BBox网络结构还存在较大的缺陷，而Cascade-RCNN的多级结构中不同阈值的检测器，就相当于重新采样的过程。

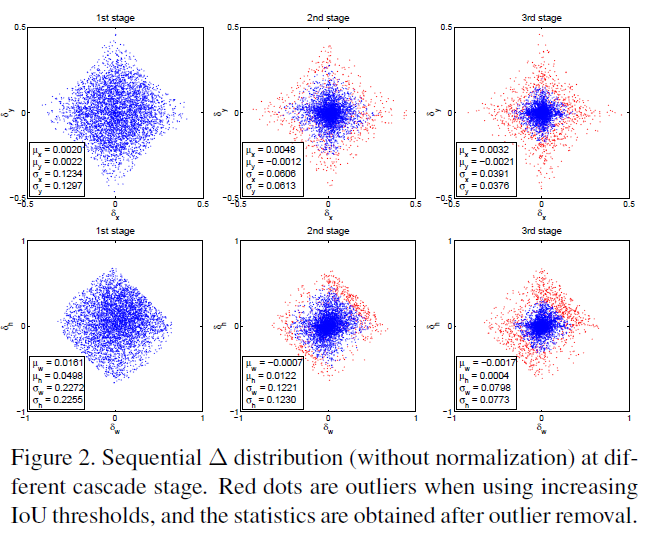


图 7 不同级联阶段的∆分布（未进行归一化）

（红点是异常值，需要剔除）

Integral Loss网络结构是用相同样本根据不同阈值训练出不同的检测器，然后根据输入的候选区域分别丢入不同阈值的检测器中，然后综合看三个结果进行输出。在训练过程中，由于样本都是一样的，对于低IoU阈值的检测器存在噪声过多的问题；对于高IoU阈值检测器的还是存在过拟合的问题。所以实际上训练出来的检测器的效果并不好。

从论文前4部分的理论层面的分析，Cascade-RCNN的创新点还是比较明显的。

* 在训练阶段，由于前一阶段的进行优化的候选区域作为下一阶段的输入，所以样本的IoU值普遍变高，所以下一阶段高阈值下的样本数量不会变少；
* 在验证阶段，即使一开始RPN提出的候选区域的质量不高，但是不断经过不同阶段不同阈值的检测器的回归优化后，候选区的质量提高，与该阶段的检测器的阈值相匹配，不会存在mismatch 的问题。

1. 实验结果

作者构建的Cascade-RCNN结构是4级结构：1个RPN+3个检测器（阈值分别为0.5/0/6/0.7），3级检测器类似于多个Faster-RCNN第二阶段检测器的第二阶段。

作者尝试了多个骨干结构（backbone）网络上搭建Cascade-RCNN的网络结构跑COCO数据集。评估之后，发现在计算量没有大量增加的基础上，均能够提高3～4个点的AP，体现了该网络的普适性。

如下图显示，每一类的基线目标检测器，第一行是未应用Cascade-RCNN的训练速度、测试速度以及它的准确率，第二行是应用Cascade-RCNN的效果。可以看到增加了Cascade-RCNN之后训练速度和测试速度有所减慢，但是相差不大，准确率均有较大的提高。

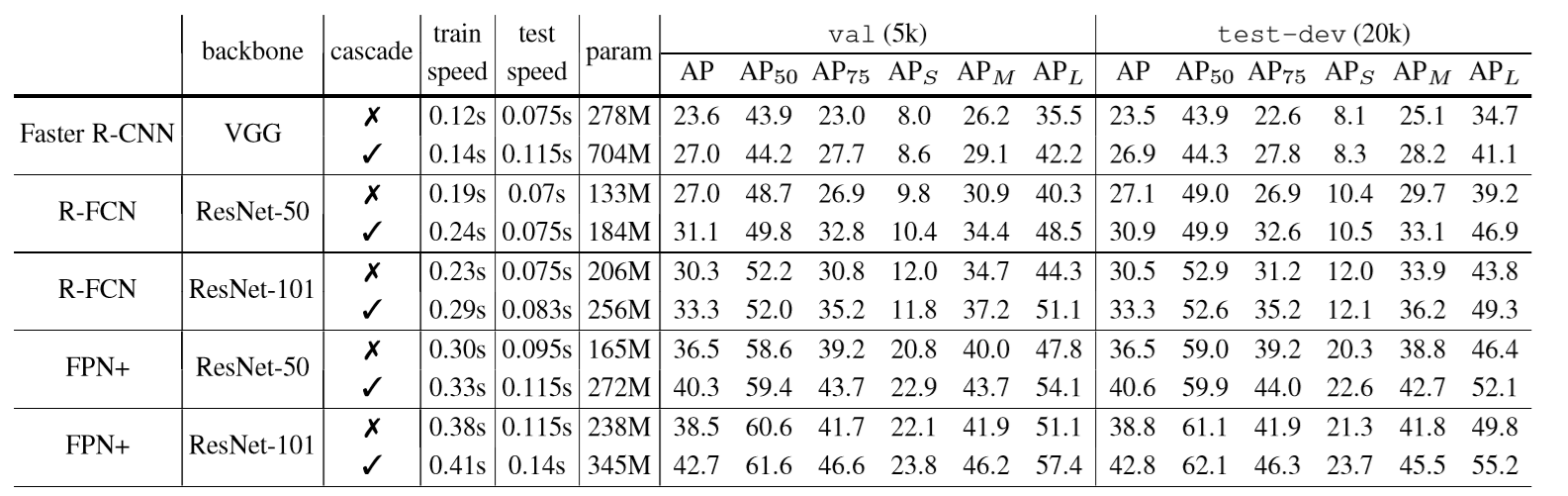


图 8 在多个流行的基线目标检测器上进行的比较

此外作者还对Cascade-RCNN的级数对性能的影响做了实验。如下图。增加一级：1-stage到2-stage，性能有3.3个点的提升；再增加一级，也有0.7个点的提升，但是AP70下准确率开始下降了；再增加一级，性能开始下降，但是下降不多，并且AP90下性能是最好的。

所以综合分析，从训练、测试的效率和准确率多方分析，3级的Cascade-RCNN性价比最高。

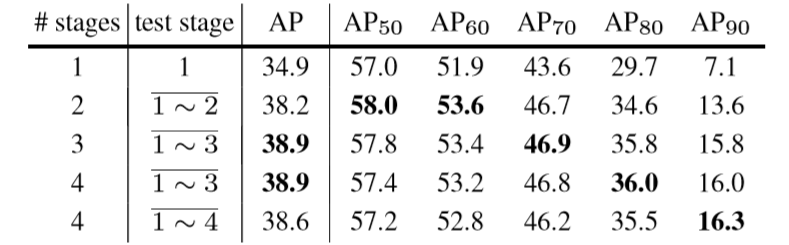


图 9 Cascade-RCNN级数对性能的影响

1. 论文总结

作者通过分析目标检测设定参数时的一些小细节和问题，提出了一个改进思路Cascade-RCNN，它能够有较大的性能提升空间，并且有普适性，实现也较为简单，计算量却没有大量增加。

它给目标检测的性能优化提供了一个新的思路。

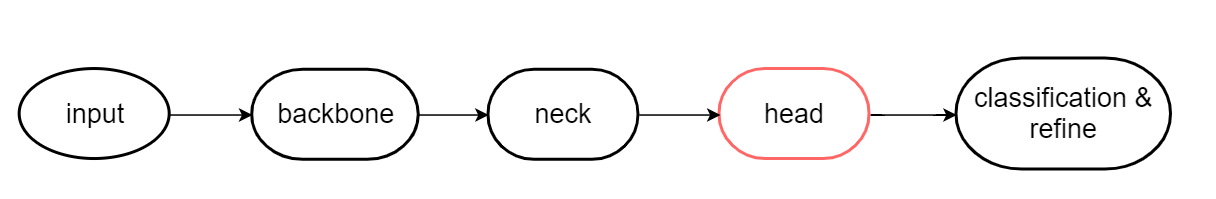
1. 结构实现和部分源码

论文作者在GitHub 上面公布了源码，它是使用了Caffe的matlab接口实现，对没有matlab基础的人来说，学习周期较长，并且caffe的安装过程也较为麻烦，上手难度较大。所以安装完成caffe之后就果断放弃使用作者提供的源码。

后面看到有推荐开源mmdetection检测库，它是基于PyTorch的一个深度学习目标检测工具箱，安装过程较为简单，比较友好，所以就选择它来实现Cascade-RCNN。

首先是环境配置，由于mmdetection必须在Ubuntu系统下，基于CUDA的运行的，所以我首先根据电脑配置安装了NVIDIA驱动，并安装了适配的CUDA版本和对应的Pytorch版本。

由于Cascade-RCNN有普适性，可以基于一些成熟的深度学习的目标检测网络的基础上修改。大致思路是，修改检测头（head）的结构，将1-stag head修改为3-stage head，并且将不同级之间的输入输出串联起来就能实现Cascade-RCNN的核心思想。流程上来说就是：



1. 首先是对输入图片进行特征提取，主要是在backbone阶段和neck阶段。在backbone阶段利用了ResNet，利用neck是FPN（特征金字塔）作为通用的特征提取器的一部分，用来处理多尺度变化的问题；
2. 候选区域的提取，发生在head阶段。根据前面获得的图片特征图去提取proposal（2000个左右）：是head阶段的初始部分，称为rpn\_head，利用RPN网络提取候选区域；
3. 分类正负样本，发生在head阶段。根据输入的proposal，先区分正负样本，由于候选区域较多，所以要根据比例进行采样，得到sampling\_result。这边就是Cascade-RCNN开始不同的地方，采样要进行3次循环，每次循环的IoU阈值是逐渐提高的
4. 池化阶段，将已经获得每个图片的采样之后的正负样本，进行一次RoI Pooling，利用的是SingleRoIExtractor，将不同大小的框映射成固定大小。
5. 池化之后的结果送到bbox head——classification+detection，针对每个框进行classification和bbox的修正。之前rpn为单纯的二分类——前景、背景，这里分为N+1类(类别+背景)。调用的是bbox\_head——并且将优化后的bbox应用到proposal中，并且更新proposal中，则第二次循环就是用的优化过的proposal，这是Cascade的另一个优势；
6. 重复步骤3，4，5三次。

以上是网络搭建的整体流程。由于mmdetection有实现一些常见的基础神经网络的结构，所以只需要关注网络的搭建步骤、网络的选择和参数的选择即可。当你将整个网络的结构配置完成后，可以直接调用mmcv库下面的Runner类来更好的进行Pytorch模型训练，Runner类来操控安排训练的各个环节，我们只需要把定义好的模型结构、数据集等都丢给runner就能实现模型的训练：具体调用类似于：runner.run(data\_loaders, cfg.workflow, cfg.total\_epochs)。

首先定义一个配置文件，存放网络结构各个阶段的网络选取，并且配置好输入、输出、步数、阈值等相关参数。如下图就是搭建一个Cascade-RCNN的过程：

model = dict(  
 type=**'CascadeRCNN'**,  
 *#三个detector* num\_stages=3,  
 *#主体网络是resnet，深度为50* backbone=dict(  
 type=**'ResNet'**,  
 depth=50,  
 num\_stages=4,  
 out\_indices=(0, 1, 2, 3),  
 frozen\_stages=1,  
 style=**'pytorch'**),  
 *#FPN进行特征提取，获得proposal* neck=dict(  
 type=**'FPN'**,  
 in\_channels=[256, 512, 1024, 2048],  
 out\_channels=256,  
 num\_outs=5),  
 *#RPN网络* rpn\_head=dict(  
 type=**'RPNHead'**,  
 in\_channels=256,  
 feat\_channels=256,  
 anchor\_scales=[8],  
 anchor\_ratios=[0.5, 1.0, 2.0],  
 anchor\_strides=[4, 8, 16, 32, 64],  
 target\_means=[.0, .0, .0, .0],  
 target\_stds=[1.0, 1.0, 1.0, 1.0],  
 use\_sigmoid\_cls=**True**),  
 bbox\_roi\_extractor=dict(  
 type=**'SingleRoIExtractor'**,  
 roi\_layer=dict(type=**'RoIAlign'**, out\_size=7, sample\_num=2),  
 out\_channels=256,  
 featmap\_strides=[4, 8, 16, 32]),  
 bbox\_head=[  
 dict(  
 type=**'SharedFCBBoxHead'**,  
 num\_fcs=2,  
 in\_channels=256,  
 fc\_out\_channels=1024,  
 roi\_feat\_size=7,  
 num\_classes=81,  
 target\_means=[0., 0., 0., 0.],  
 target\_stds=[0.1, 0.1, 0.2, 0.2],  
 reg\_class\_agnostic=**True**),  
 dict(  
 type=**'SharedFCBBoxHead'**,  
 num\_fcs=2,  
 in\_channels=256,  
 fc\_out\_channels=1024,  
 roi\_feat\_size=7,  
 num\_classes=81,  
 target\_means=[0., 0., 0., 0.],  
 target\_stds=[0.05, 0.05, 0.1, 0.1],  
 reg\_class\_agnostic=**True**),  
 dict(  
 type=**'SharedFCBBoxHead'**,  
 num\_fcs=2,  
 in\_channels=256,  
 fc\_out\_channels=1024,  
 roi\_feat\_size=7,  
 num\_classes=81,  
 target\_means=[0., 0., 0., 0.],  
 target\_stds=[0.033, 0.033, 0.067, 0.067],  
 reg\_class\_agnostic=**True**)  
 ]  
)

1. 结果展示

mmdetection默认支持的是COCO数据集，由于COCO数据集数据量较大，所以我首先尝试训练了Pascal VOC2007的数据集（训练集有5011张图片，加上背景一共有21类物体），如图是在训练过程中不同head的准确率和loss值的统计——可以看到随着IoU阈值上升，准确率有较明显的提高，损失值也较小，这说明通过级联head能够优化候选区域的分布情况和样本质量，从而提高网络的整体性能。

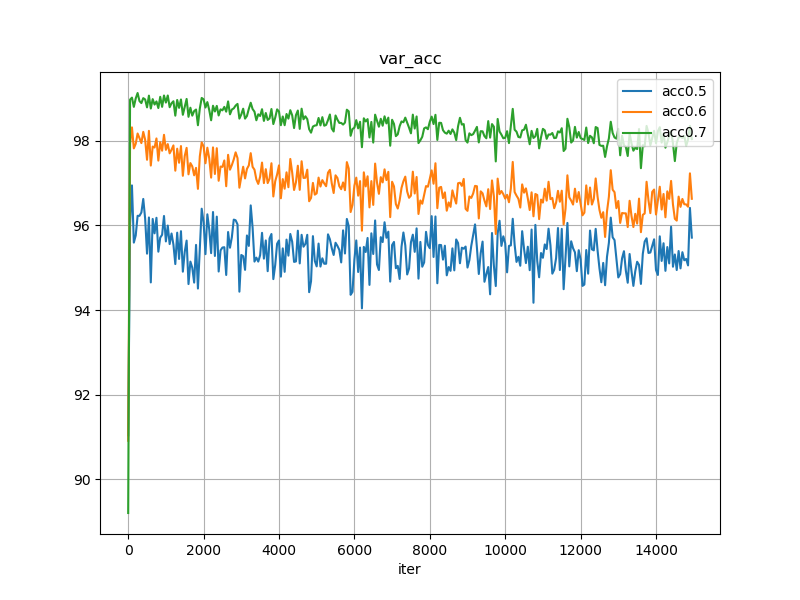


图 10 Pascal VOC2007训练过程的head的准确率

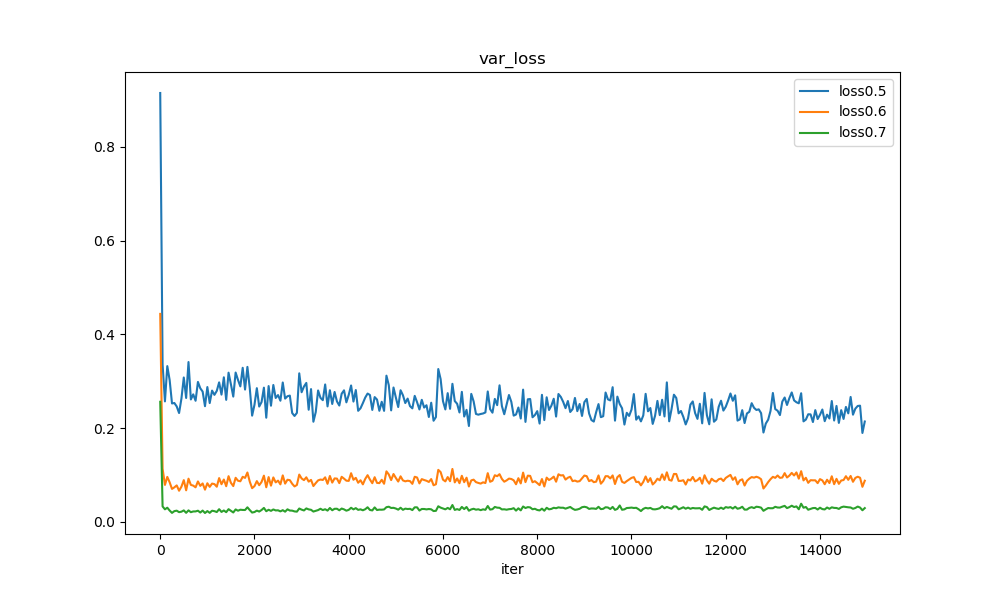


图 11 Pascal VOC2007 训练过程的loss值

根据论文的介绍，Cascade-RCNN在COCO数据集上的表现尤为的突出，所以就尝试使用COCO数据集（COCO加上背景有81类物体）来测评Cascade-RCNN的性能。

在mmdetection中默认支持的是COCO的测评标准。它的底层实现已经出现在工具箱中，具体的位置为：mmdet.core.evaluation.coco\_eval，它也是直接调用Microsoft的coco API中的pycocotools包来实现的。

如下图就是对训练好的网络进行性能评估的结果。

它是一个检测评价矩阵（detection evaluation metrics），分别计算了box和segm在小目标、中目标以及大目标上的AP和AR值，其中AP就是mAP，是所有类别的平均值。AP仅对IoU值大于0.5的物体进行统计。可以看到指标大致和论文实现的性能指标相近。

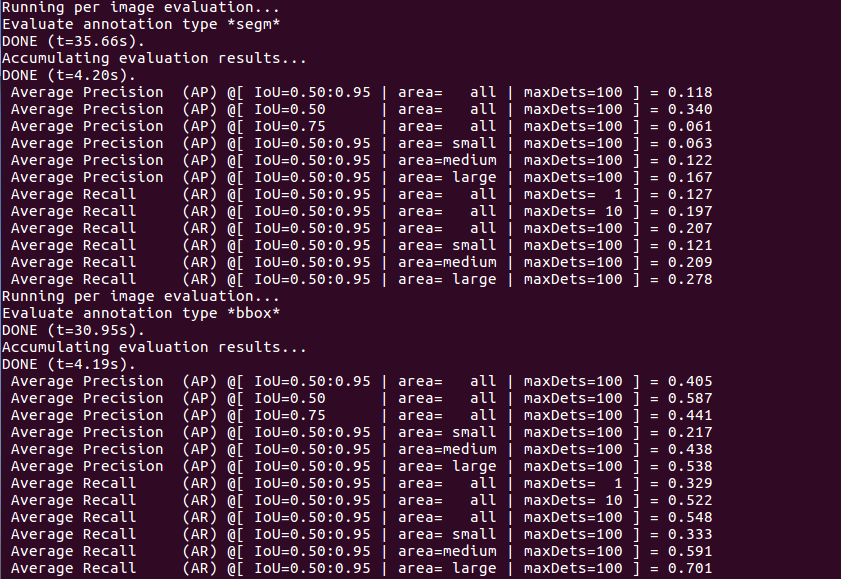


图 12 在COCO数据集上的评价矩阵

当然整个网络训练的目的就是在实际中进行目标检测的。通过输入一张或者多张图片，能够可视化的看到目标类别和定位结果。



图 13 随机找的图片的运行效果（物体是存在于80类目标中）

1. 总结

Cascade-RCNN这篇论文从一个很小的参数点入手，对IoU阈值这个参数变化中的网络性能进行深入分析，最后总结出了单一阈值检测器存在的问题，以及针对该问题提出的改进措施。整篇论文思路很清晰，论证也完整可信，并且进行了多方实验论证了Cascade-RCNN在改进目标检测准确率上的确有较大的进步，其普适性、运行效率也得到了证明。读懂这篇论文后，由衷感叹作者的仔细和分析能力。

这篇论文是深入了整体的神经网络结构内部，对结构进行的一个修改。对一个没有一点人工智能基础的人来说难度较大，论文中的术语也比较多，所以我的整个学习流程是：先学习了CNN卷积神经网络，RCNN（区域卷积神经网络）的整个网络结构的原理，由于本论文首先是基于Faster-RCNN的网络结构改进得到的，所以又学习了Faster-RCNN整体网络结构。这时候读这篇论文就比较能理解其中的原理和优化方向了。（记录了整个学习过程在，并且存放在GitHub：https://github.com/amyZhoucc/readAIpaper）在读懂这篇论文的基础上，我了解了一些主流的目标检测的数据集以及它们的评价指标，例如COCO,Pascal等。

整体流程还是以理解主流的目标检测的网络结构框架构造为主，此外还搭建了基于Cascade-RCNN思想的神经网络，并且训练了数据集，通过运行测试集分析了该网络的性能。

通过整个学期对人工智能的学习以及做项目的经历，我对人工智能有了初步的了解，尤其是目标检测领域的相关知识，收获了很多知识，也有了一点实践经验。

参考文献

[1] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into High Quality Objection. Computer Vision and Pattern Recognition (cs.CV).

[2] R. B. Girshick. Fast R-CNN. In ICCV, pages 1440–1448, 2015. 1, 2, 3, 5.

[3] Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. Computer Vision and Pattern Recognition (cs.CV).

[4] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Computer Vision and Pattern Recognition (cs.CV).