

**企业实习中期报告**

学号：

姓名： Koorye

专业方向：

企业名称：­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_

企业指导教师：

院内指导教师：

**信息与软件工程学院**

目 录

[第一章 实习项目进展情况 3](#_Toc104846641)

[1.1 工作完成情况 3](#_Toc104846642)

[1.2 知识技能学习情况 17](#_Toc104846643)

[第二章 存在问题与解决方案 22](#_Toc104846644)

[2.1存在的主要问题 22](#_Toc104846645)

[2.2解决方案与可行性研究 23](#_Toc104846646)

[第三章 前期任务完成度与后续实施计划 27](#_Toc104846647)

[3.1 前期任务完成度 27](#_Toc104846648)

[3.2 后续实施计划 27](#_Toc104846649)

[参考文献 28](#_Toc104846650)

说明:

1.报告要求3000字以上。

2.本模板仅为基本参考，请各位同学根据个人情况进行目录结构扩展。

# 第一章 实习项目进展情况

## 1.1 工作完成情况

（详细介绍已完成的工作情况，包括对复杂软件工程问题的推理分析,针对复杂软件工程问题设计满足特定需求的总体设计和详细设计。对于已经完成工作，需对实施结果进行分析和解释，并通过信息综合得到合理有效的结论。）

* + 1. **工作情况**

进入公司几个月以来，在大家的帮助下，我较好地完成了自己的本职工作和任务。

在阅读相关文献和对现有的目标检测框架的研究中，我发现现有目标检测框架存在的不足：当前的检测框架仅对每个目标所在的感受野本身进行预测，缺乏对目标之间关系的推理。而在现实生活中，利用目标间或目标与背景信息间的关系进行推理是人们观察事物的常用手段。例如，天空中的物体往往是鸟而非鱼，人群中的不明目标有可能也是人等。利用这些关系进行推理，可以提升目标检测框架的性能。

使用ResNeXt101作为Backbone，我使用FCOS模型对FashionPedia数据集进行训练和测试，结果如下：

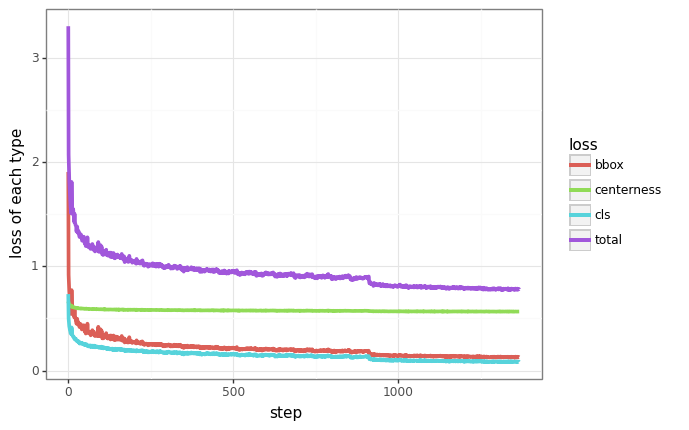


图1-1 损失变化

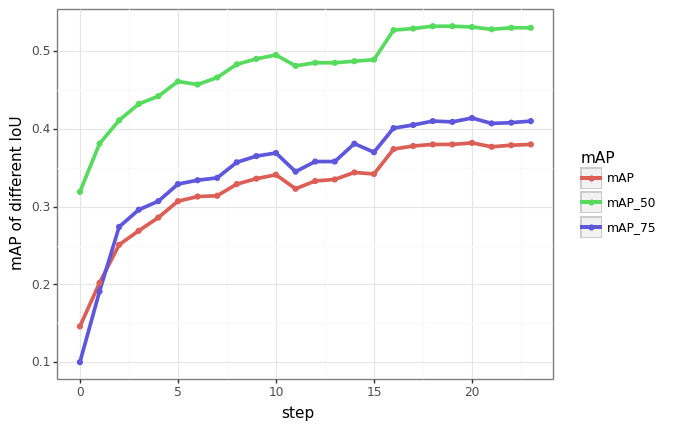


图1-2 不同指标的mAP变化

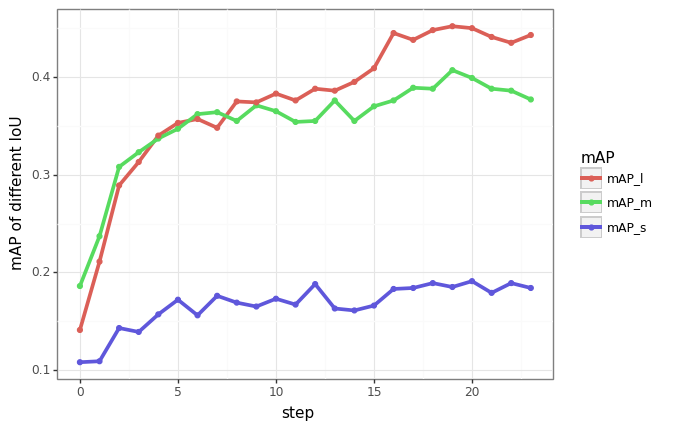


图1-3 不同尺寸目标的mAP变化

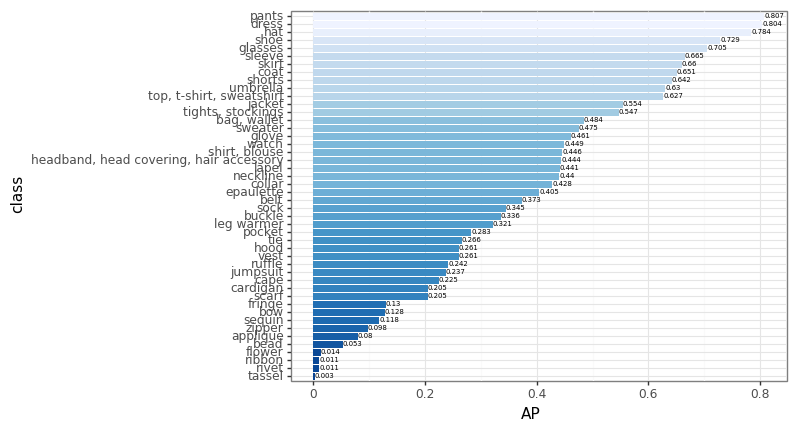


图1-4 不同类别的AP

可以看出，loss已经充分收敛，mAP近似达到该模型所能达到的最佳效果。而对于不同类别，检测效果则呈现很大的差异。效果很好的如pants、dress、hat、shoe、glasses，其AP超过0.7；而效果很差的如tassel、nevt、ribbon、flower、bead、applique、zipper，其AP不及0.1甚至不及0.01。

要提升FCOS的检测性能，就要从这些效果很差的类别入手，分析其原因。

* + 1. **原因推理与分析**

1. **类间样本不均衡**

首先想到的是，Fashion各类别的样本存在不均衡，如衣服、裤子、鞋子等样本，每个人物几乎都有，而对于耳环、配花等样本，则不是这样甚至很少出现。

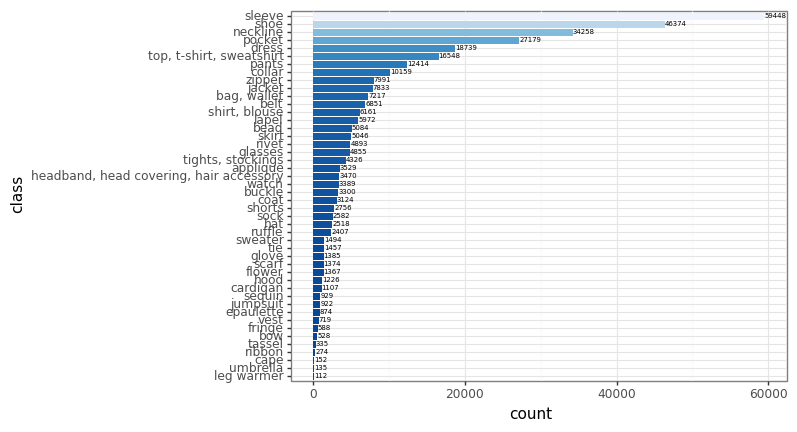


图1-5 各类别的出现次数

如上图，出现最多的如sleeve袖子、shoe鞋子、neckline领口等出现最多，而leg warmer护腿、umbrella雨伞、cape披肩等出现最少，最多和最少出现的样本相差超过500倍。

1. **样本大小差异**

其次，样本的大小对检测效果影响也很大，小目标检测更为困难，而在该数据集中，显然衣服和装饰的大小差异是很大的。通过统计每个目标框的面积，之后对每个类别求平均，计算得到每个类别的平均面积。

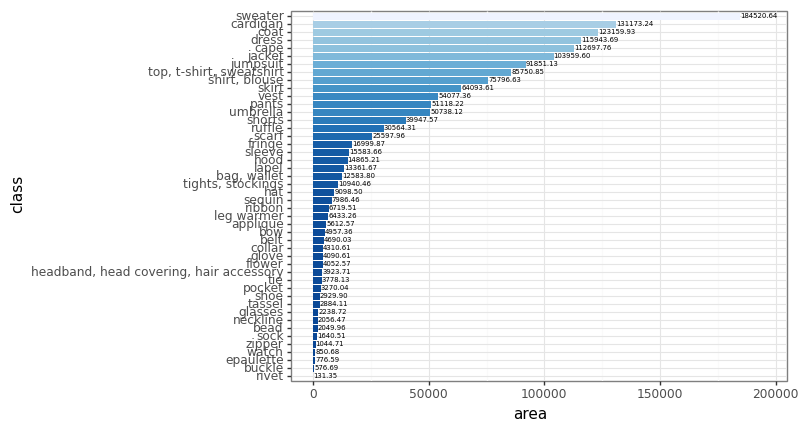


图1-6 各类别的平均面积

如上图，最大如sweater毛衣、cardigan羊毛衫、coat大衣，最小如nvet?、buckle带扣、epaulette肩章，其面积差异超过2000倍。

1. **样本重叠严重**

样本重叠往往会提高检测难度，尤其对于该数据集，类间遮挡现象尤为明显，因为如衣服、裤子等大目标中往往会包含口袋、拉链等小目标，进一步加剧了这些小目标的检测难度。

下面统计了两种重叠严重程度的衡量指标：

前者IoU为两目标框交集的面积/并集的面积，而后者IoG此处将其定义为两目标框交集的面积/被衡量的目标框面积。

对于口袋、拉链等小目标来说，如果使用IoU，其分母几乎就是包含其的衣服、裤子等大目标的面积，而分子几乎就是小目标本身的面积；而如果使用IoG，其值则为自身被其他目标包含的面积比例，比使用IoU更为合理。通过计算每个目标框相对于同一图片中其余所有目标的IoU、IoG，之后对每个类别分别求平均，计算得到每个类别的IoU、IoG均值。

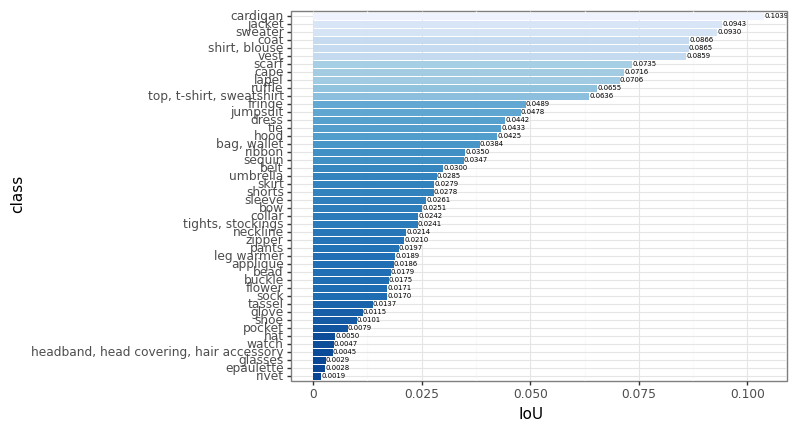


图1-7 各类别的平均IoU

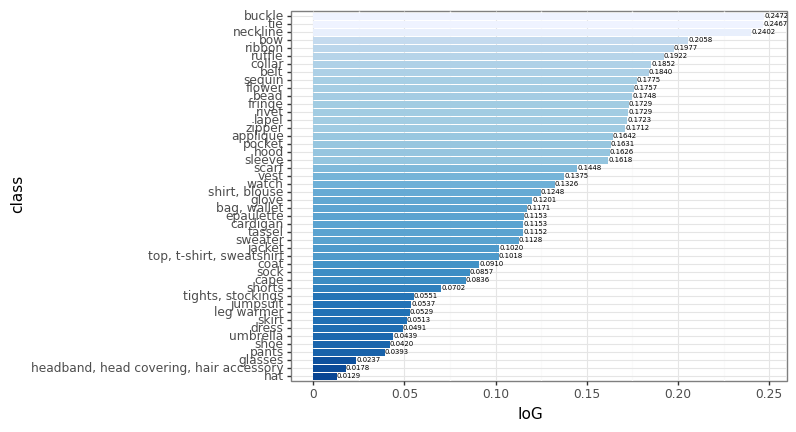


图1-8 各类别的平均IoG

如上图，IoU大的往往就是大目标，如dress裙子、cape披肩，而IoG大的往往是最容易被包含的目标，如neckline领口、bow蝴蝶结。可以看出，不同类别样本重叠遮挡的严重程度存在差异，如buckle带扣、tie领带、neckline领口出现遮挡的比例是hat帽子、glasses眼镜等的20倍。

1. **其它原因**
2. 一些类别本身拥有较为复杂的特征，例如人类就比数字等目标复杂的多，这将影响不同类别目标的检测效果
3. 一些类别的划分也许不合理，例如将“t-shirt、sweatshirt”和“shirt、blouse”分为两类，而它们本身又具有非常相似的特征，这也许会增加检测难度
4. 有些样本本身就很少，划分到验证集中的样本就更少，验证用的样本不够，验证得到的mAP本身就不一定准确

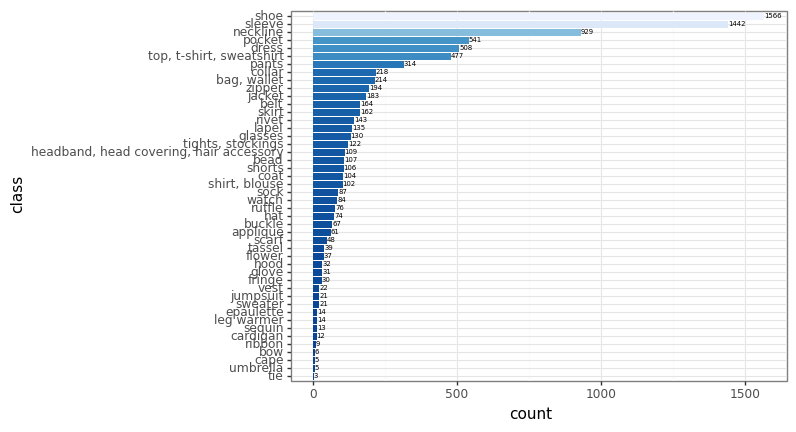


图1-9 验证集中各类别的出现次数

如上图，验证集中的tie领带只有3个、umbrella只有5个，这么少的验证数据得到该类别的mAP具备的参考价值就不大。

经过上述分析，可以发现FashionPedia数据集样本各类别的数量、大小、遮挡程度等方面都存在极大的差异，那么这些因素对检测效果有怎么样的影响？

对于不同类别的样本，分别以出现次数、大小(面积)、IoU、IoG为X轴，以AP为Y轴，绘制散点图：

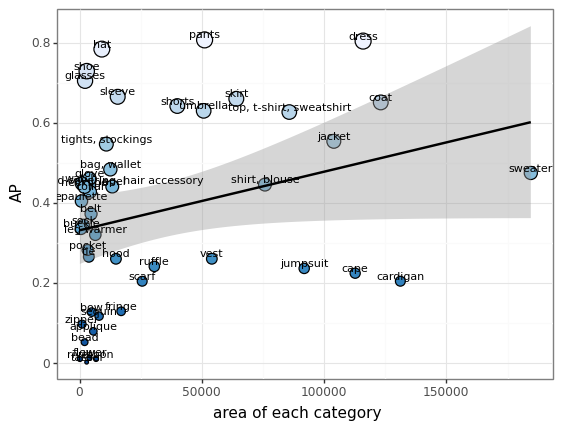
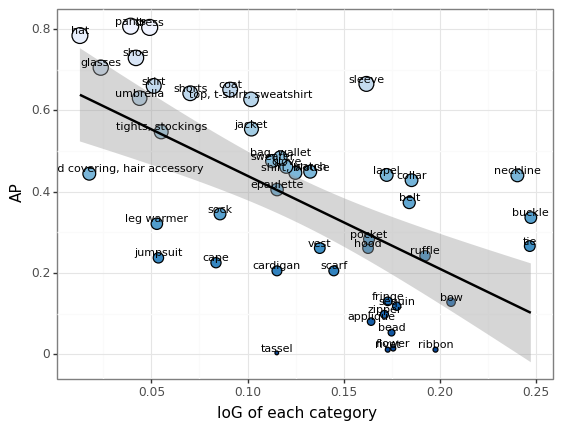
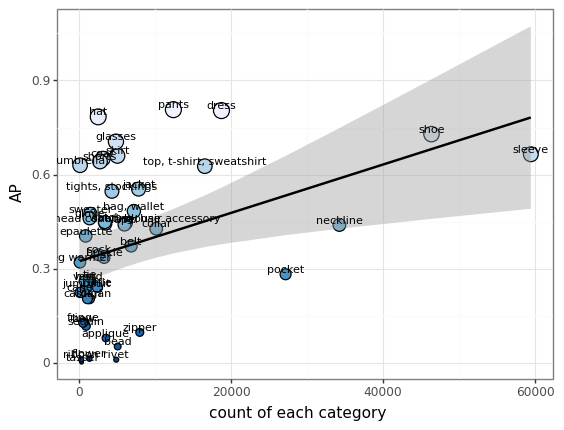
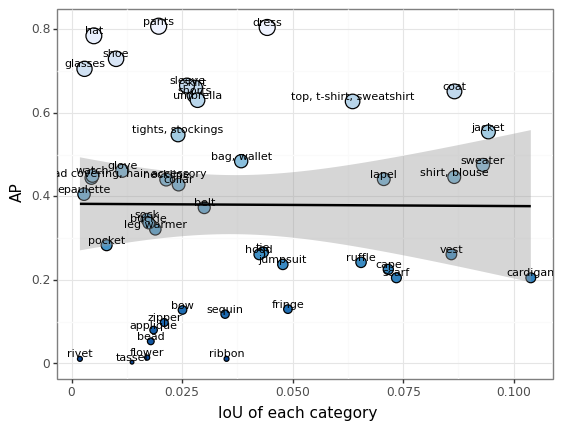


图1-10 各指标和AP分布的散点图

如上图，样本数量、大小和AP呈正相关，样本的数量越多、面积越大，越利于检测；而样本遮挡程度和AP呈负相关，样本越容易被遮挡，越不利于检测。而IoU作为不恰当的衡量样本遮挡程度的指标，则和AP无关。

因此，可以从这4个方面，以缓解FCOS在FashionPedia上的检测弱项，从而提升整体效果，我根据上述分析，提出了一些解决方案。

* + 1. **解决方案设计**

1. **缓解类间样本不均衡**

此处称类别出现次数多的样本为普遍样本，反之为稀少样本，目前已经有很多缓解样本不均衡的方法，如：

1. 数据增强方法：

* 过采样：多次采样包含稀少样本的图片、将稀少样本复制到图片的其他位置、使用GAN生成图片
* Random Sampling：随机抽取普遍样本，即放弃一些普遍样本
* OHEM：挑选难分样本进行训练回传损失，即放弃一些易分样本

1. 更有效的指标：

* Focal Loss：给予正负样本不同的权重，给予易分样本更多的抑制，从而相对的放大难分样本的权重

现有的各种Loss指标，如Focal Loss、GHM，都是从正负样本和难易样本的角度出发，缓解样本不均衡的问题，然而数量多的样本不一定就易分，数量少的样本也不一定难分。

而对于FashionPedia数据集，除了由于目标检测数据集本身特点和检测算法本身原因导致的正负样本不均衡之外，不同类别样本的不均衡也非常突出。为了让稀少样本得到更多训练，可以给稀少样本以更多的权重。

假设第i类的权重为wi，将其代入Focal Loss的计算公式，得

其中

其中pi表示第i个类别的预测值。

然而，如果简单取各类别的占比的倒数作为权重，数量极多和极少的样本相差超过500倍，不同的类别产生的梯度差异极大，这会训练可能是有害的。

论文《Investigating the Challenges of Class Imbalance and Scale Variation in Object Detection in Aerial Images》[[1]](#endnote-1)给出了一些目标检测中类别不平衡的解决策略，对于iSAID航空图像数据集，不同目标的数量分布差异极大，这与FashionPedia的情况是类似的，如下图。

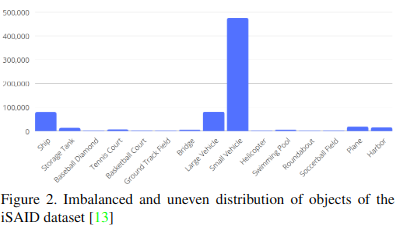


图1-11 iSAID数据集各类别出现次数

论文中给出的方案同样是调整不同类别的损失权重：

其中wi表示第i类的损失权重，ni表示第i类的样本数量。

论文《Learning deep representation for imbalanced classification》[[2]](#endnote-2)中提到，当某个batch的某类别样本数量极少时，直接使用该类别出现频率的倒数会导致训练相当不稳定，损失显著增加，该论文给出了两种权重的取样方法：

其中fj表示单副图像中第j类的平均样本数，k > 0,q > fj为超参数。

论文《Class-Balanced Loss Based on Effective Number of Samples》[[3]](#endnote-3)给出了利用每个类别的有效样本数重新平衡分类损失的策略，作者认为每个样本都在特征空间中占据一定的体积，现有的数据增强方法，如遮挡、旋转、噪声，这些由同一条原型样本所产生的数据会分布在其空间中的邻域。

如下图，所有样本组成的特征空间为虚线包括部分，一个样本能产生的所有样本为灰色部分，而一个新样本只会完全独立或完全属于之前的样本覆盖的空间。

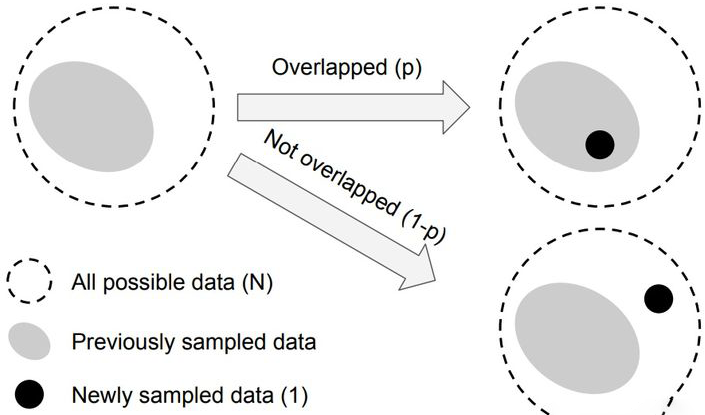


图1-12 特征空间的分布规律

于是有n个样本体积覆盖率的期望：

其中

N表示假设的整个空间的体积，因此是一个超参数，取决于假设的样本空间大小。于是，取En的倒数作为不同类别的损失权重，即

作者在不同数据集上，使用不同的损失函数进行实验，结果如下图，可以看到取得了不错的效果，不过也需要手动调整。

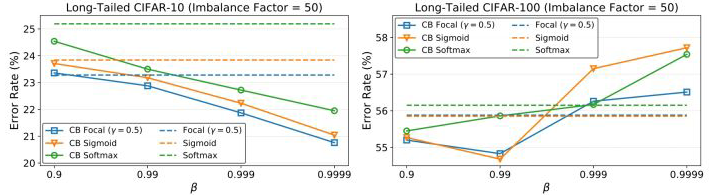


图1-13 CIFAR-10和CIFAR-100数据集上取不同的效果

1. **加强小样本检测**
2. **缓解尺度敏感性：**
   1. **特征提取：**FPN及其各种改进、TridentNet、DCN
   2. **多级学习：**SNIP、FSAF、SAPD
3. **缓解正负样本不平衡：**ATSS(对于小目标ATSS降低阈值以获取更多正样本)
4. **更有效的指标：**GIoU、DIoU、CIoU、NWD等
5. **加强重叠样本检测**

**更有效的指标：**

1. Loss：Rep Loss、Agg Loss
2. NMS：Soft-NMS、Softer-NMS、Adaptive-NMS

而对于FashionPedia数据集，其样本遮挡又有其独特的性质：

1. 类间遮挡很多，类内遮挡很少
2. 有很多大目标完全包含小目标，如衣服包含口袋

由于FashionPedia中多为大目标包含小目标，因此若采用多级学习的策略，将不同大小的目标分到不同特征层，这些遮挡问题可以得到妥善解决。不过既然FashionPedia中的遮挡现象多为完全包含，是否可以对此设计专用的NMS方法？但是这样又没有什么通用性，似乎意义不大。

1. **利用类间信息**

**细粒度学习：**

FashionPedia中包含很多相似的类别，这些类别属于同一个超类：

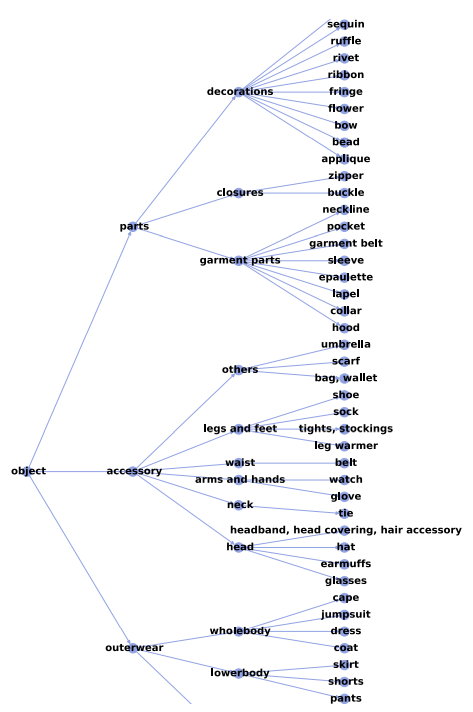


图1-14 Fashionpedia数据集类别分布

对于目标检测来说，尤其是子类数目很多，且子类间特征很相似时，直接对每个子类进行学习效果可能不佳。一个简单的方案是，采用目标检测模型对每个超类进行检测，之后对每个检测框，在其对应的所有子类采用分类器进行分类。

**上下文学习：**

对于人物来说，其帽子、眼镜、上衣、裤子、鞋子等目标的位置信息是相对固定的，如果一个疑似是眼镜的目标出现在帽子和上衣之间，就可以提高该目标为眼镜的置信度，利用这些显式上下文信息，可以提升检测性能。

**AC-CNN：**[[4]](#endnote-4)对于特征图中的每个检测框，取其周围不同范围的区域作为周边环境信息，和检测框一起做ROI Pooling后拼接，得到局部上下文信息特征用于回归；对于整张特征图，将其切片逐次送入LSTM得到全局上下文特征，二者拼接用于最终的分类。

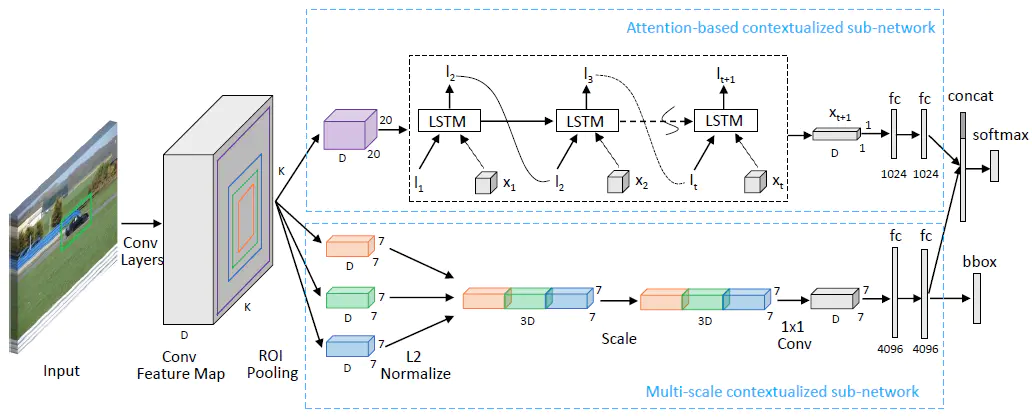


图1-15 AC-CNN

**Object Relation Module[[5]](#endnote-5)：**利用目标间的形状特征信息和相对位置信息提高检测效果。对于要检测的目标框m，对于检出的每个目标框n，计算其外观特征和几何特征，其中几何特征即为物体的坐标和宽高：

之后计算得到所有目标关于第n个目标的关系特征：

其中第m个目标关于第n个目标的权重为：

其中第m个目标关于第n个目标的外观权重为：

几何权重为：

其中表示一种编码方法，它首先计算两目标框的相对位置信息，如

(

将其embed到高维，并添加位置编码信息。之后将多个relation模块得到的与拼接就得到了最终的，如下图：

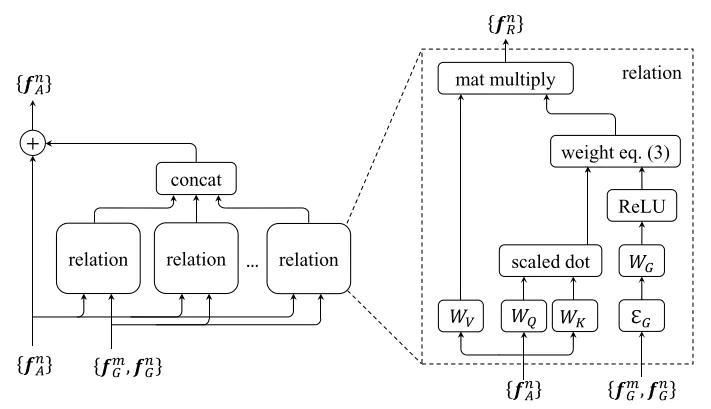


图1-16 Object Relation Module

**IR R-CNN：**[[6]](#endnote-6)根据建议框之间的语义相似度和空间相似度计算两建议框是否有关，构造邻接矩阵，送入GCN更新特征信息，进行最终的训练和预测。

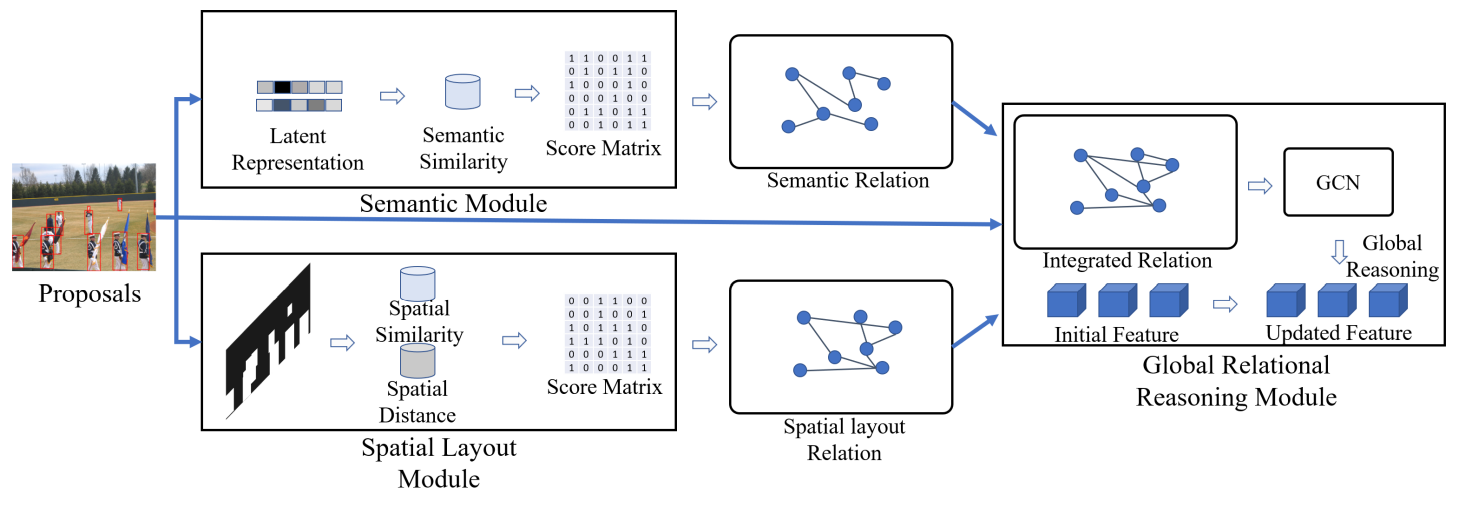


图1-17 IR R-CNN

对于语义相似度的计算，将建议框以某种方式encode为尺寸相同的一维向量后逐元素相乘后相加，作为：

其中是一个示性函数，若i和j高度重叠，则为0，否则为1，是一个投影函数，论文中将其设为一个MLP。

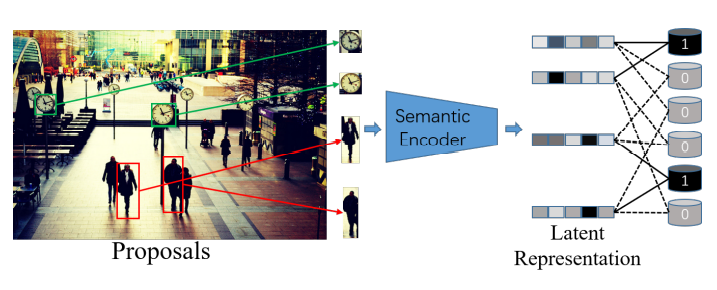


图1-18 语义相似度的计算

对于空间相似度的计算，分别计算空间相似度和空间距离权重后相乘，作为：

其中

其中按经验取5e-4。

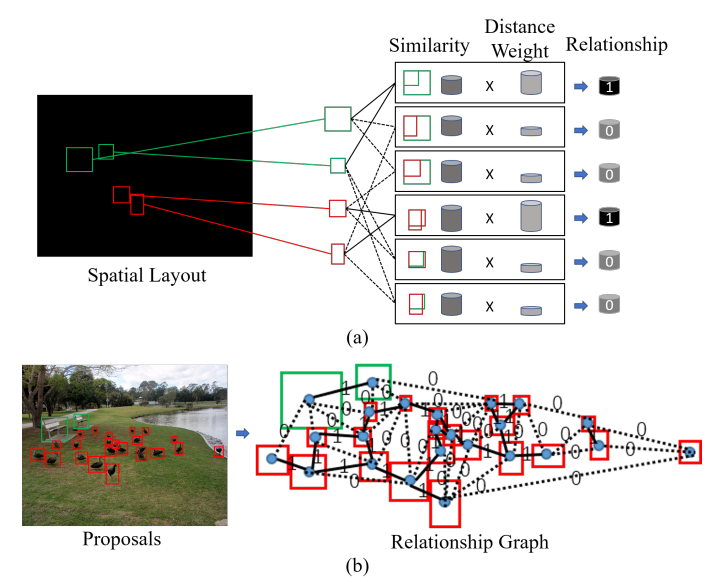


图1-19 空间相似度的计算

将相似度矩阵按行排序后保留每行前K个值作为相关置1，其余为不相关置0，将语义相似度矩阵和空间相似度矩阵求并集，作为最终的邻接矩阵，送入GCN进行特征加强。

1. **提取额外特征**

FashionPedia数据集本身就包含人体，不过可惜的是它并没有给出人体的标注框。也许可以用目标检测或关键点检测的方式提取人体的目标框或各个关键点的坐标，进而提升检测效果。

例如，在得到人体目标框后，可以采用OR-CNN的方式，将人体按比例分割为5个部分，之后对每个部分进行ROI Pooling后相接作为最终特征，不过这个方法只适用于Two-Stage模型。

* + 1. **实验设计与结果**

根据上述论文中的策略，我在FashionPedia数据集上尝试了3种类别加权方式：

* 1. Inverse Class Frequency
  2. Effective Sample
  3. Effective Sample

表1-1 不同加权方式的评估结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 加权方式 |  |  |  |  |  |  |
| Baseline | 0.380 | 0.530 | 0.410 | 0.184 | 0.377 | 0.443 |
| Inverse Class Frequency | 0.383 | 0.534 | 0.406 | 0.173 | **0.381** | **0.450** |
| Effective Sample | **0.391** | **0.542** | **0.420** | **0.186** | 0.380 | 0.442 |
| Effective Sample | 0.370 | 0.514 | 0.398 | **0.186** | 0.366 | 0.417 |

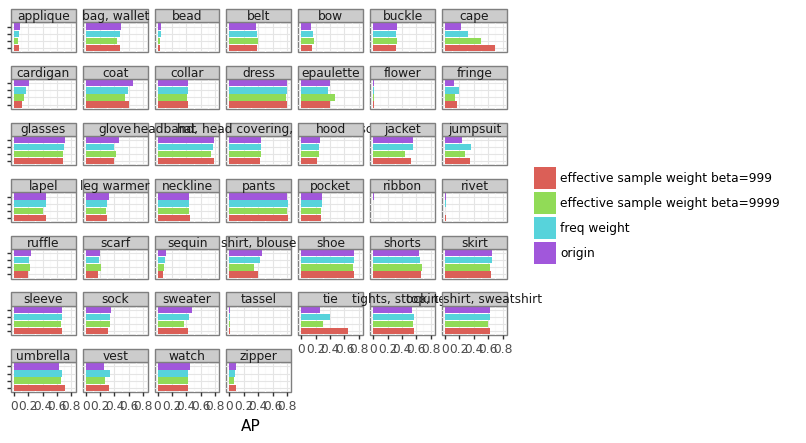


图1-20 各类别在不同加权方式下的AP

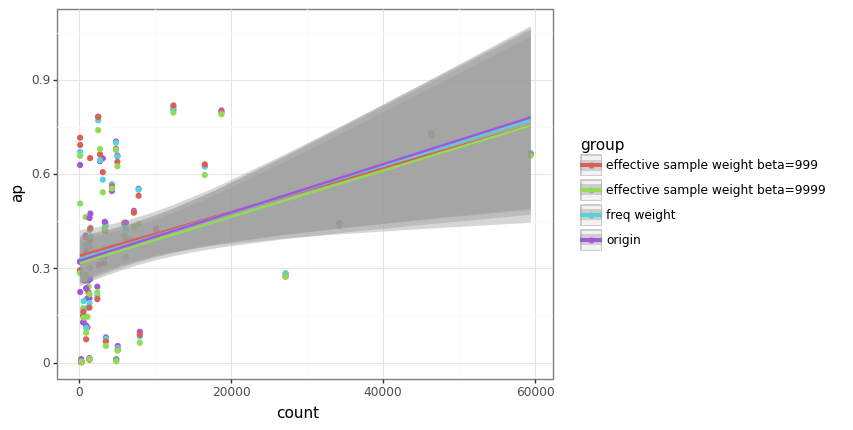


图1-21 各类别的出现次数与AP的分布

* + 1. **结论**

对于数量较少的目标，如cape、umbrella、bow、tie，有效样本加权的策略取得了性能上不小的提升，其中时效果普遍最好，而反词频加权(freq weight)的方式也有一定的提升。

对于数量极少且难分(很小、特征复杂或遮挡严重)的目标，不同加权方式效果并不明显，甚至有所下降。对于数量较多的目标，有效样本加权可以使性能相对不下降，取则导致了比较大的性能下降。

整体上，相对于原样本数量和AP的拟合曲线来说，反词频加权和有效样本加权使得拟合曲线斜率变小，这说明类间样本不均衡带来的预测精度差异被缩小了，其中有效样本加权()效果最好。而对于来说，则造成了总体性能的下降。

## 1.2 知识技能学习情况

（根据课题应用场景，选择开发环境、工具与技术标准；说明工具的学习和使用情况，文献和书籍的查阅、分析、总结和收获；阐述针对个人或职业发展的需求，进行的其它学习等。）

**开发工具：**

1. Visual Studio Code
2. Windows Terminal
3. Electerm

**系统环境：**

1. Tesla V100 x 4
2. Ubuntu 18.04.5 LTS
3. NVIDIA-SMI 470.82.01
4. CUDA 11.4

**软件开发环境：**

1. Miniconda3 4.10.3
2. Python 3.7.0
3. Torch 1.10.0
4. Torchvision 0.11.1
5. Torchaudio 0.10.0
6. Torch-geometric 2.0.4
7. Pycocotools 2.0.4
8. Numpy 1.21.2
9. Pandas 1.3.4
10. Opencv-python 4.5.5.64
11. Mmcv-full 1.4.2
12. Mmdet 2.24.1

在实验过程中，我熟悉了Python的环境搭建、打包与迁移；Python的基本使用，如Python的基本语法、科学运算、数据处理和可视化等；利用Pytorch搭建深度学习模型，进行单机训练和分布式训练，使用Pycocotools进行测试和评估等。在阅读了前人的论文和源代码后，我能根据自己的设计编写对应的模型，并实现并行化的代码设计，使用GPU进行加速。我还掌握了使用MMDetection进行模型的训练与测试，并修改源码加入自己设计的模型、损失函数、数据集等，实现自己需要的功能。

在遇到困难和异常时，我学会了查询相关文档和Github Issues以寻找解决方案，通过对[Welcome to MMDetection’s documentation! — MMDetection 2.24.1 documentation](https://mmdetection.readthedocs.io/en/latest/)，我掌握了MMDetection的各种使用方法，我能够熟练地使用该框架搭建自己需要的实验环境。

例如，我通过阅读官方文档，修改官方示例中提供的COCO数据集设置，适配了自己的FashionPedia数据集：

|  |
| --- |
| 1. \_base\_ = [ 2. '../../\_base\_/datasets/coco\_detection.py', 3. '../../\_base\_/schedules/schedule\_1x.py', '../../\_base\_/default\_runtime.py' 4. ] 5. dataset\_type = 'CocoDataset' 6. data\_root = '/root/Datasets/fashionpedia/' 7. classes = ('shirt, blouse', 'top, t-shirt, sweatshirt', 'sweater', 'cardigan', 8. 'jacket', 'vest', 'pants', 'shorts', 'skirt', 'coat', 'dress', 9. 'jumpsuit', 'cape', 'glasses', 'hat', 10. 'headband, head covering, hair accessory', 'tie', 'glove', 'watch', 11. 'belt', 'leg warmer', 'tights, stockings', 'sock', 'shoe', 12. 'bag, wallet', 'scarf', 'umbrella', 'hood', 'collar', 'lapel', 13. 'epaulette', 'sleeve', 'pocket', 'neckline', 'buckle', 'zipper', 14. 'applique', 'bead', 'bow', 'flower', 'fringe', 'ribbon', 'rivet', 15. 'ruffle', 'sequin', 'tassel') 16. *# model settings* 17. model = dict( 18. type='FCOS', 19. backbone=dict( 20. type='ResNet', 21. depth=50, 22. num\_stages=4, 23. out\_indices=(1, 2, 3), 24. frozen\_stages=1, 25. norm\_cfg=dict(type='BN', requires\_grad=False), 26. norm\_eval=True, 27. style='pytorch', 28. init\_cfg=dict( 29. type='Pretrained', 30. checkpoint='torchvision://resnet50')), 31. neck=dict( 32. type='FPN', 33. in\_channels=[512, 1024, 2048], 34. out\_channels=256, 35. start\_level=0, 36. add\_extra\_convs='on\_output',  *# use P5* 37. num\_outs=5, 38. relu\_before\_extra\_convs=True), 39. bbox\_head=dict( 40. type='FCOSHead', 41. num\_classes=len(classes), 42. in\_channels=256, 43. stacked\_convs=4, 44. feat\_channels=256, 45. strides=[8, 16, 32, 64, 128], 46. center\_sampling=True, 47. center\_sample\_radius=1.5, 48. norm\_on\_bbox=False, 49. centerness\_on\_reg=True, 50. loss\_cls=dict( 51. type='FocalLoss', 52. use\_sigmoid=True, 53. gamma=2.0, 54. alpha=0.25, 55. loss\_weight=1.0), 56. loss\_bbox=dict(type='GIoULoss', loss\_weight=1.0), 57. loss\_centerness=dict( 58. type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0)), 59. *# training and testing settings* 60. train\_cfg=dict( 61. assigner=dict( 62. type='MaxIoUAssigner', 63. pos\_iou\_thr=0.5, 64. neg\_iou\_thr=0.4, 65. min\_pos\_iou=0, 66. ignore\_iof\_thr=-1), 67. allowed\_border=-1, 68. pos\_weight=-1, 69. debug=False), 70. test\_cfg=dict( 71. nms\_pre=1000, 72. min\_bbox\_size=0, 73. score\_thr=0.05, 74. nms=dict(type='nms', iou\_threshold=0.6), 75. max\_per\_img=1000)) 76. *# data settings* 77. img\_norm\_cfg = dict( 78. mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to\_rgb=True) 79. train\_pipeline = [ 80. dict(type='LoadImageFromFile'), 81. dict(type='LoadAnnotations', with\_bbox=True), 82. dict(type='Rotate', level=1.0), 83. dict(type='ColorTransform', level=1.0), 84. dict(type='Resize', img\_scale=(1333, 800), keep\_ratio=True), 85. dict(type='RandomFlip', flip\_ratio=0.5), 86. dict(type='Normalize', \*\*img\_norm\_cfg), 87. dict(type='Pad', size\_divisor=32), 88. dict(type='DefaultFormatBundle'), 89. dict(type='Collect', keys=['img', 'gt\_bboxes', 'gt\_labels']), 90. ] 91. test\_pipeline = [ 92. dict(type='LoadImageFromFile'), 93. dict(type='LoadAnnotations', with\_bbox=True), 94. dict( 95. type='MultiScaleFlipAug', 96. img\_scale=(1333, 800), 97. flip=False, 98. transforms=[ 99. dict(type='Resize', keep\_ratio=True), 100. dict(type='RandomFlip'), 101. dict(type='Normalize', \*\*img\_norm\_cfg), 102. dict(type='Pad', size\_divisor=32), 103. dict(type='ImageToTensor', keys=['img']), 104. dict(type='Collect', keys=['img']), 105. ]) 106. ] 107. data = dict( 108. samples\_per\_gpu=4, 109. workers\_per\_gpu=4, 110. train=dict( 111. type=dataset\_type, 112. classes=classes, 113. ann\_file=data\_root + 'Annotations/instances\_attributes\_train2020.json', 114. img\_prefix=data\_root + 'Images/train/', 115. pipeline=train\_pipeline), 116. val=dict( 117. type=dataset\_type, 118. classes=classes, 119. ann\_file=data\_root + 'Annotations/instances\_attributes\_val2020.json', 120. img\_prefix=data\_root + 'Images/test/', 121. pipeline=test\_pipeline), 122. test=dict( 123. type=dataset\_type, 124. classes=classes, 125. ann\_file=data\_root + 'Annotations/instances\_attributes\_val2020.json', 126. img\_prefix=data\_root + 'Images/test/', 127. pipeline=test\_pipeline)) 128. *# optimizer* 129. optimizer = dict( 130. lr=0.01, paramwise\_cfg=dict(bias\_lr\_mult=2., bias\_decay\_mult=0.)) 131. optimizer\_config = dict( 132. \_delete\_=True, grad\_clip=dict(max\_norm=35, norm\_type=2)) 133. *# learning policy* 134. lr\_config = dict( 135. policy='step', 136. warmup='constant', 137. warmup\_iters=500, 138. warmup\_ratio=1.0 / 3, 139. step=[16, 22]) 140. runner = dict(type='EpochBasedRunner', max\_epochs=24) 141. fp16 = dict(loss\_scale=512.) |

步骤如下：

1. 将Fashionpedia数据集解压到存放目录root/Datasets下：

|  |
| --- |
| 1. tree fashionpedia -d 2. fashionpedia 3. ├── Annotations 4. └── Images 5. ├── test 6. └── train |

1. 设定dataset\_type、data\_root、classes等参数
2. 在data中指定自己数据集的ann\_file位置、img\_prefix、classes
3. 设定自己所需的环境，如使用fp16混合精度训练、全局batch size为4x4等

对于我个人的发展需求，我对经典的机器学习理论，如决策树、SVM、Boosting、GMM、HMM等常见的算法模型进行学习，并作笔记；我对经典的深度学习模型，如LeNet5、VGG、AlexNet、ResNet、Transformer等进行学习；对于个人目前接触的目标检测方向，我对经典框架如SSD、R-CNN系列、YOLO系列、RetinaNet、FCOS、FSAF、ATSS、AutoAssign等进行学习和比较，对其中的思想进行分析，如RetinaNet中的Focal Loss起到了平衡简单和困难样本和正负样本的作用，从而从中提取自己的思想和解决方案。

# 第二章 存在问题与解决方案

## 2.1存在的主要问题

（分析、总结和归纳实习执行过程中存在的主要问题与具体的应对措施，能通过具体事例来说明在心理素质培养以及应对项目风险与挑战的能力锻炼等方面是否有所收获。）

在实习过程中，对于遇到的问题，我能够独立进行分析，给出解决方案，对于个人存在困难的问题，我能够通过查阅文档和其它途径进行解决，我在实习过程中学到了独立思考和解决问题的能力。

例如，我对Fashionpedia数据集分别使用MMDetection提供的FCOS模型和第三方仓库[zhenghao977/FCOS-PyTorch-37.2AP: A pure torch implement of FCOS 37.2AP (github.com)](https://github.com/zhenghao977/FCOS-PyTorch-37.2AP)进行训练和预测，其评估结果却有很大的差异。

表2-1 两种框架的初始性能比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 框架 |  |  |  |  |  |  |
| MMDetection | 0.284 | 0.439 | 0.309 | 0.127 | 0.302 | 0.315 |
| FCOS-Pytorch-37.2AP | 0.335 | 0.478 | 0.373 | 0.169 | 0.344 | 0.387 |
| 结论 | 两种框架存在约5个点的性能差距 | | | | | |

上表反映了两者的性能差距，其中MMDetection落后第三方仓库近5个点，这在使用同一检测框架的情况下是较为奇怪的现象。而要确定模型的baseline，就需要消除同一检测框架下存在的不一致性，否则就无法以模型的AP作为基准值。

我从两者使用的tricks、模型结构、训练环境、训练策略、测试策略等角度进行分析，经过反复实验，找到了造成其差异的原因。在这一过程中，我处理未知的能力有所提升。

## 2.2解决方案与可行性研究

（针对当前尚未解决的复杂工程问题，分析文献寻求解决方案，并在安全、环境、法律等约束条件下，通过技术经济评价对设计方案的可行性进行研究，说明方案的合理性。）

* + 1. **解决方案**

1. 比较与调整两者的tricks
2. 比较与调整两者的模型结构
3. 比较与调整两者的优化器、batch size等超参数设置
4. 比较与调整两者的训练和测试过程
   * 1. **实验过程**
   1. 修改mmdetection提供的fcos\_r50\_caffe\_fpn\_gn-head\_1x\_coco.py，使其匹配Fashion数据集
   2. **修改tricks：**
5. 替换backbone的caffe风格为pytorch风格
6. 修改NMS前的最大预测数为1000，NMS阈值为0.6
7. 替换预处理的图像标准化参数为pytorch的标准化参数
8. 使用Center Sampling
9. 使用Centerness on Reg
10. 将IoU Loss替换为GIoU Loss
    1. **修改模型结构：**将mmdet的backbone输出从C2~C5改为C3~C5，FPN输出从第1层开始改为从第0层开始
    2. **修改训练方式：**使用DistributedDataParallel重写第三方仓库，实现分布式训练，学习率设为0.01，总的batch size为4x4(4卡，每张卡的batch size为4)，共训练24轮，前500次step采用1/3的学习率进行warm up，第16和22次学习率衰减为之前的1/10
    3. **比较测试过程：**真值替换(替换class heatmap、bbox heatmap和centerness)后对两者进行评估
    4. **比较训练过程：**根据MMDetection的损失函数计算方式对第三方仓库进行重写，重新进行训练和评估
       1. **实验结果**

两者的性能起初为：

表2-1 两种框架的初始性能比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 框架 |  |  |  |  |  |  |
| MMDetection | 0.284 | 0.439 | 0.309 | 0.127 | 0.302 | 0.315 |
| FCOS-Pytorch-37.2AP | 0.335 | 0.478 | 0.373 | 0.169 | 0.344 | 0.387 |
| 结论 | 两种框架存在约5个点的性能差距 | | | | | |

比较两种框架，我发现最初始的MMDetection提供的FCOS没有采用诸如centerness on reg、center sampling等tricks，而第三方仓库采用了这些tricks。在步骤2修改tricks后，两者的性能如下：

表2-2 统一tricks后两种框架的性能比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 框架 |  |  |  |  |  |  |
| MMDetection | 0.313 | 0.472 | 0.338 | 0.184 | 0.314 | 0.350 |
| FCOS-Pytorch-37.2AP | 0.335 | 0.478 | 0.373 | 0.169 | 0.344 | 0.387 |
| 结论 | Tricks对于FCOS具有约3个点的提升 | | | | | |

Tricks提供了3个点的性能提升，然而两者仍有2个点的差距。经过比较两者的结构，我发现其backbone和FPN的输入和输出层不同，这可能导致两者关注的目标尺寸有所差异。在步骤3修改模型结构后，两者的性能如下：

表2-3 统一模型结构后两种框架的性能比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 框架 |  |  |  |  |  |  |
| MMDetection | 0.308 | 0.465 | 0.329 | 0.160 | 0.319 | 0.358 |
| FCOS-Pytorch-37.2AP | 0.335 | 0.478 | 0.373 | 0.169 | 0.344 | 0.387 |
| 结论 | 模型结构不是导致两者差异的主要原因 | | | | | |

修改模型结构带来的改变不大。比较两者的训练方式，我发现第三方仓库没有使用分布式多卡训练，且两者的batch size也不一致。通过阅读何恺明的论文[[7]](#endnote-7)，我发现要保持性能尽量不变，batch size与学习率要保持线性缩放的关系，而FCOS的作者采用的基准为4卡、batch size=4，即4x4=16。而我的MMDetection设置为4x16，学习率却没有同时放大。于是我修改第三方仓库为分布式多卡训练，并统一训练环境：

1. 采用DDP分布式训练，设定4张卡，每张卡batch size为4，总共为4x4=16
2. 采用SGD作为优化器，momentum设为0.9，weight decay设为1e-4
3. 采用warm up策略，前500step的采用1/3的学习率
4. 采用的线性衰减的策略，在第16和22epoch，学习率衰减为之前的1/10

在步骤4修改训练环境后，两者的性能如下：

表2-4 统一训练环境后两种框架的性能比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 框架 |  |  |  |  |  |  |
| MMDetection | 0.380 | 0.528 | 0.416 | 0.183 | 0.378 | 0.451 |
| FCOS-Pytorch-37.2AP | 0.327 | 0.459 | 0.360 | 0.193 | 0.337 | 0.367 |
| 结论 | 全局batch size对训练结果的影响很大 | | | | | |

统一训练环境后，MMDetection的性能提升了很多，然而第三方仓库却并没有提升，反而有少许下降。这说明batch size的设置对模型训练结果有很大的影响，但这仍然不是导致两者性能差距的最终原因。

于是我对模型的训练和测试阶段进行比对。在步骤5将两者的预测替换为真值后，两者的性能如下：

表2-5 真值替换后两种框架的性能比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 框架 |  |  |  |  |  |  |
| MMDetection | 0.965 | 0.965 | 0.965 | 0.947 | 0.983 | 0.986 |
| FCOS-Pytorch-37.2AP | 0.965· | 0.965 | 0.965 | 0.947 | 0.983 | 0.986 |
| 结论 | 两者的真值构造和预测过程是一致的 | | | | | |

在真值替换后，两者得到了完全相同的结果，这说明两者真值构造和预测过程完全一致，于是我对训练阶段进行比较，根据MMDetection的Loss函数重写第三方仓库，在步骤6重写Loss后，两者的性能如下：

表2-6 统一训练环境后两种框架的性能比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 框架 |  |  |  |  |  |  |
| MMDetection | 0.358 | 0.510 | 0.392 | 0.168 | 0.361 | 0.423 |
| FCOS-Pytorch-37.2AP | 0.352 | 0.498 | 0.390 | 0.185 | 0.375 | 0.402 |
| 结论 | 全局batch size对训练结果的影响很大 | | | | | |

在统一训练环境后，两者取得了相近的结果。

* + 1. **结论**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 模型 | Tricks | 分布式训练 | GPU数量 | Batch size | 使用mmdet的loss | AP |
| 1 | MMDetection | N | Y | 4 | 16 | Y | 0.284 |
| 2 | FCOS-Pytorch-37.2AP | Y | N | 4 | 8 | N | 0.335 |
| 3 | MMDetection | Y | Y | 4 | 16 | Y | 0.313 |
| 4 | FCOS-Pytorch-37.2AP | Y | Y | 4 | 4 | N | 0.321 |
| 5 | MMDetection | Y | Y | 4 | 4 | Y | 0.358 |
| 6 | MMDetection | Y | Y | 4 | 2 | Y | 0.380 |
| 7 | FCOS-Pytorch-37.2AP | Y | Y | 4 | 4 | Y | 0.352 |

1. 从序号1、3可以看出，从Tricks造成了两者3个点的性能差异
2. 从序号1、5、6可以看出，GPU x batch size的值对结果影响很大
3. 从序号4、7可以看出，在统一tricks和batch size后，两者的性能差异在于Loss的设计不同

**最终结论：导致两种框架性能差异的原因，在于tricks、总的batch size和loss的计算方式的差异。**

# 第三章 前期任务完成度与后续实施计划

## 3.1 前期任务完成度

请详细列出目前已完成了哪些工作，还有哪些工作没有完成，已完成工作占总任务的百分比。

**已完成的工作：**

1. **基本任务：**
2. 学习Python的编写，以及Numpy、Pandas等基础科学计算库的使用
3. 学习李宏毅机器学习和深度学习课程
4. 学习CS231N公开课并完成相关作业
5. 学习Pytorch Tutorial，编写和运行相关代码
6. 学习并复现MMDetection中的任意模型
7. 学习并复现MMPose或MMAction中的任意模型
8. **正式任务：**
9. 对小目标和行人检测领域进行探索，有自己的理解，并作报告
10. 对one-stage模型进行研究，在Fashionpedia数据集进行某个one-stage模型的训练和测试
11. 学习MMDetection，使用MMDetection训练和测试Fashionpedia数据集
12. 对Fashionpedia数据集进行分析，找到现有目标检测框架的不足，提出解决方案

**未完成的工作：**

1. 对自己的解决方案进行实验，确认其有效性
2. 后续的消融实验、可视化、写作等

## 3.2 后续实施计划

请详细写出未完成工作内容后期的实施计划，最好以表格形式来写，参见表3-1所示：

表 3-1 后期任务计划表

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 工作内容 | 工作开始时间 | 工作结束时间 |
| 1 | 在现有的目标检测模型上提出自己的改进方案 | 2022-6-1 | 2022-6-30 |
| 2 | 对方案进行实验，确认其有效性 | 2022-7-1 | 2022-7-30 |
| 3 | 消融实验、可视化和写作 | 2022-8-1 | 2022-8-30 |

## 参考文献

1. [1] Elhagry A , Saeed M . Investigating the Challenges of Class Imbalance and Scale Variation in Object Detection in Aerial Images[J]. 2022. [↑](#endnote-ref-1)
2. [2] Huang C , Li Y , Loy C C , et al. Learning Deep Representation for Imbalanced Classification[C]// Computer Vision & Pattern Recognition. IEEE, 2016:5375-5384. [↑](#endnote-ref-2)
3. [3] Cui Y , Jia M , Lin T Y , et al. Class-Balanced Loss Based on Effective Number of Samples[J]. arXiv, 2019. [↑](#endnote-ref-3)
4. [4] Li J , Wei Y , Liang X , et al. Attentive Contexts for Object Detection[J]. IEEE Transactions on Multimedia, 2016, 19(5):944-954. [↑](#endnote-ref-4)
5. [5] Hu H , Gu J , Zhang Z , et al. Relation Networks for Object Detection[J]. [↑](#endnote-ref-5)
6. [6] Fu K , Li J , Ma L , et al. Intrinsic Relationship Reasoning for Small Object Detection[J]. 2020. [↑](#endnote-ref-6)
7. [7] Goyal P , P Dollár, Girshick R , et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour[J]. 2017. [↑](#endnote-ref-7)