**信息与软件工程学院**

**企业实习初期报告**

学 号：

姓 名： Koorye

专业方向：

企业名称：

实习岗位名称：

企业指导教师：

院内指导教师：

目 录

[第一章 实习岗位介绍 1](#_Toc71278155)

[1.1企业情况 1](#_Toc71278156)

[1.2岗位目标、任务与复杂工程问题 2](#_Toc71278157)

[1.2.1 岗位目标 2](#_Toc71278158)

[1.2.2 任务 2](#_Toc71278159)

[1.2.3 复杂工程问题 3](#_Toc71278160)

[第二章 知识技能学习情况 19](#_Toc71278161)

[2.1开发环境和工具 19](#_Toc71278162)

[2.2预备知识 19](#_Toc71278163)

[2.3新知识点学习和掌握情况 20](#_Toc71278164)

[第三章 前期任务完成度与后续实施计划 23](#_Toc71278165)

[3.1 前期任务完成度 24](#_Toc71278166)

[3.2 后续实施计划 25](#_Toc71278167)

[参考文献 26](#_Toc71278168)

说明：

1、报告要求2000字以上。

2、如果参与了多个项目，请在相应栏目内依次写出。

# 第一章 实习岗位介绍

## 1.1企业情况

...

## 1.2岗位目标、任务与复杂工程问题

### 1.2.1 岗位目标

研究目标检测领域现状，学习现有的目标检测模型和方法，寻找目标检测领域的发展方向，通过研究现有模型和算法在小目标检测、人群检测等领域找到性能上的提升策略。

### 1.2.2 任务

**1. 学习基础机器学习和深度学习模型和方法**

1.1. 学习机器学习的基础知识

1.1.1. 吴恩达公开课<https://www.coursera.org/learn/machine-learning>

1.1.2. 李宏毅机器学习中文课程<https://www.bilibili.com/video/av10590361?from=search&seid=2567563418955556168>

1.1.3. 李航统计机器学习

1.2. 学习深度学习的基础知识

1.2.1. 斯坦福公开课CS231n <http://cs231n.stanford.edu/>

1.2.2. 基于numpy实现常见的神经网络结构，如全连接、卷积、BN等

**2. 学习和使用深度学习框架**

2.1. 学习和运行Pytorch官方教程代码 <https://pytorch.org/tutorials/>

2.2. 使用Pytorch复现常见的深度学习模型，如ResNet、GAN、Style Transfer等

**3. 学习目标检测领域的基础模型和方法**

3.1. 学习MMDetection框架的使用

3.2. 选择MMDetection中的一个或几个目标检测算法进行复现

3.3. 选择MMPose或MMAction2中的一个或几个行为识别和姿态估计算法进行复现

**4. 探索小目标检测和行人检测领域**

4.1. 阅读小目标检测和行人检测的相关论文

4.2. 学习目标检测模型，如RetinaNet、FCOS、FSAF等，探索其中的各种提升检测性能的策略，寻找对小目标和密集目标检测的启发

4.3. 针对One-Stage模型，在FashionPedia数据集上研究提升性能的策略

### 1.2.3 复杂工程问题

**1. 复现Faster-RCNN**

实习过程中复现了很多模型，我这里打算以Faster-RCNN为例详细讲述模型复现的完整过程。

Faster-RCNN的源码比较复杂，因此我基于非官方仓库<https://github.com/pangkun248/Faster-RCNN-PyTorch>尝试复现代码，该仓库相比官方仓库少了一些功能，比如多批次训练等，不过足够用来理解Faster-RCNN的完整结构。

Faster-RCNN由三大部分组成：

* backbone：从图片提取特征信息
* rpn：从特征图中提取候选框
* classfier：对每个候选框进行分类和位置信息的微调

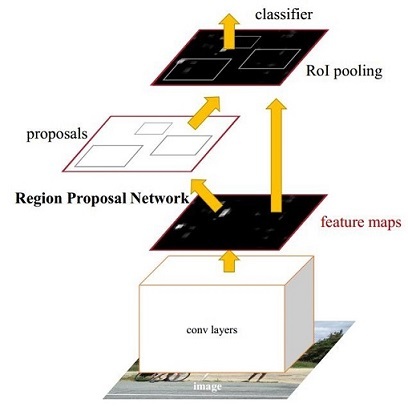


图1-1 Faster-RCNN

这里我选用ResNet50作为backbone。

RPN(Region Proposal Network)是Faster-RCNN中最为复杂的结构，对于每个特征点，RPN经过两条路径分别产生分类(正负样本分数)和回归信息(x,y,w,h与anchor的差值)，之后挑选正样本分数最大的若干候选框进行NMS，截取前若干个候选框供后续的分类和回归。

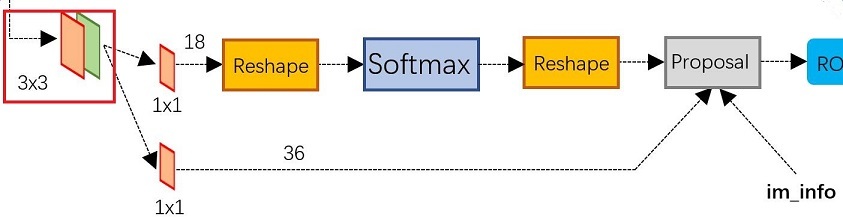


图1-2 RPN

RPN要实现的内容包括：

* **Anchor的设计：**对于小中大三种尺寸的目标，设计长宽比1:1，1:2，2:1三种比例的Anchor，共9种Anchor。

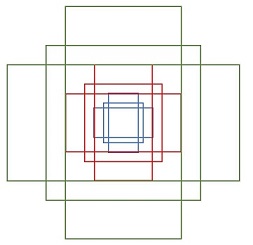
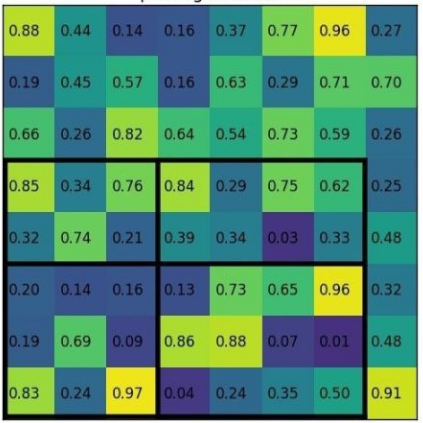
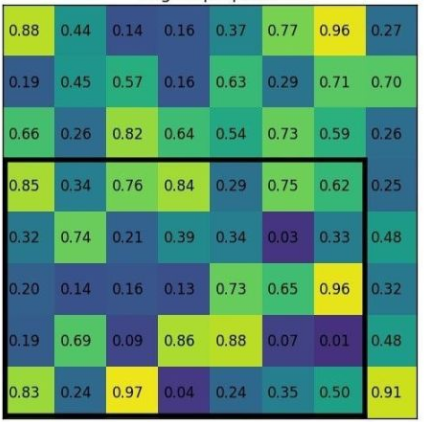


图1-3 Anchor的设计

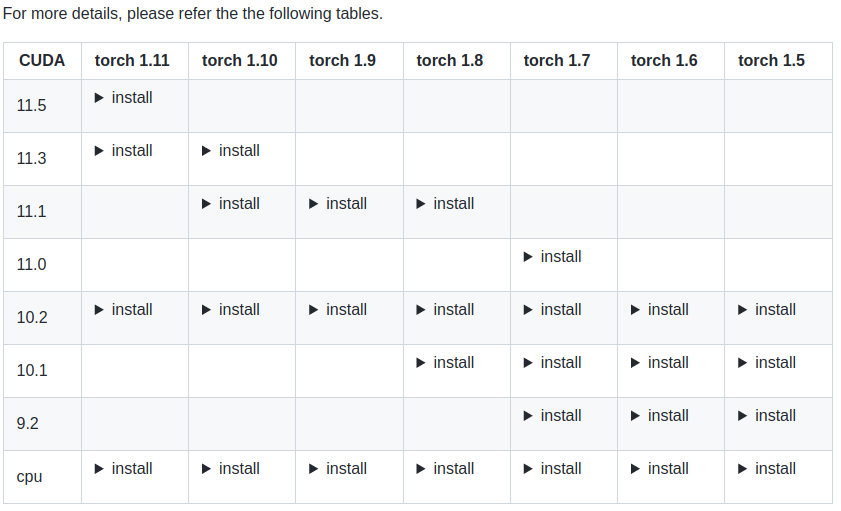
* **Anchor的产生策略：**原图大小是特征图大小的16倍，为了覆盖到每个特征点，以步长为16遍历原图，就可以得到所有Anchor的平移坐标，与Anchor相加就可以得到所有平移后的Anchor
* **特征点的分类和回归：**对于回归，对每个特征点产生9x4=36个值，分别表示9个Anchor的(dx,dy,dw,dh)；对于分类，对每个特征点产生9x2=18个值，分别表示9个Anchor的正负样本分数(s+,s-)，之后作softmax并提取正样本分数
* **候选框的后处理：**根据分数挑选前12000个预测框(预测阶段则为6000个)，将偏移量应用到Anchor上得到候选框，限制坐标范围在原图宽高内，作NMS后挑选分数最大的前2000个预测框(预测阶段则为300个)

1. Classifier的核心内容在于RoI-Pooling，它可以将不同尺寸的候选框统一尺寸，对于每个候选框，RoI-Pooing会根据设定的宽高分割候选框的区域(如Pooling后宽为2,候选框宽为7，就取7/2=3...1，于是取每个区域宽为3，最后区域不被整除就直接取满)，之后取每个区域的最大值。
2. 
3. 图1-4 RoI-Pooling
4. 对于Pooling后尺寸统一的特征图，就可以直接进行分类和回归，得到每个候选框的类别分数和坐标偏移，修正之前的候选框就可以得到最终预测结果。
5. Faster-RCNN的一大难点在于训练，对于classfier，需要设计一种策略筛选候选框的正负样本供训练。这里的做法是，将RPN生成的候选框与真实目标框比较，对于每个候选框，取与所有目标框最大的IoU，若IoU>0.5，认为是正样本，否则是负样本，最后分别随机抽取128个正负样本供其训练。
6. 而对于RPN，这里的做法是，通过预先生成的Anchor与真实目标框比较，对于每个Anchor,取与所有目标框最大的IoU，若IoU>0.7，认为是正样本；若IoU<0.3，认为是负样本，最后随机抽取128个正负样本供其训练。
7. 在数据集上，我使用PASCAL VOC2012进行训练。
8. 复现代码见<https://github.com/Koorye/faster-rcnn-demo>。
9. 部分其他模型的复现代码如下：

* FCOS: <https://github.com/Koorye/fcos>
* 基于StackHourglass、Simple Baselines、HRNet等模型的姿态估计：<https://github.com/Koorye/pose-estimation>
* Swin Transformer: <https://github.com/Koorye/swin-transformer>
* Vision Transformer:<https://github.com/Koorye/vision-transformer>

1. **离线配置GPU集群服务器的深度学习环境**由于服务器不能连接公网，我在环境的配置上遇到了比较大的困难，经过几天的探索，我找到了离线配置深度学习环境的方法。我需要安装的主要依赖有Pytorch和MMDetection。
2. 服务器配置：

* 系统：Ubuntu18.04
* GPU：V100 \* 4
* CPU：24核
* RAM：160GB

1. 首先确保本机拥有Linux环境，在Windows下可采用Windows Linux Subsystem2，如果是Windows，打包后的环境Linux将无法使用！
2. 其次确保本机已配置CUDA和CUDNN，对于Windows可以到NVIDIA官网下载，并确保Windows版本高于2b12，这样WSL2会自动适配宿主机的CUDA环境。我这里是Manjaro，CUDA环境是系统自带的，无需多余配置。
3. 其次确保本机和服务器都已安装Conda环境，我这里已经安装了Miniconda3。
4. 接下来可以正式开始配置环境，需要的环境有：
5. addict 2.4.0
6. autopep8 1.6.0
7. Bottleneck 1.3.4
8. certifi 2021.10.8
9. cycler 0.11.0
10. **Cython 0.29.28**
11. fonttools 4.31.0
12. kiwisolver 1.4.0
13. matplotlib 3.5.1
14. mkl-fft 1.3.1
15. mkl-random 1.2.2
16. mkl-service 2.4.0
17. **mmcv-full 1.4.2**
18. numexpr 2.8.1
19. **numpy 1.21.2**
20. **opencv-python-headless 4.5.5.64**
21. packaging 21.3
22. pandas 1.3.4
23. **Pillow 9.0.1**
24. pip 22.0.4
25. pycodestyle 2.8.0
26. pyparsing 3.0.7
27. python-dateutil 2.8.2
28. pytz 2021.3
29. PyYAML 6.0
30. setuptools 60.10.0
31. six 1.16.0
32. terminaltables 3.1.10
33. toml 0.10.2
34. **torch 1.10.0**
35. **torchaudio 0.10.0**
36. **torchvision 0.11.1**
37. typing\_extensions 4.1.1
38. wheel 0.37.1
39. yapf 0.32.0
40. 我这里使用Conda创建了名为mmdet的环境用于安装这些依赖，其中重要的依赖用加粗形式标注，opencv选择headless版本，否则服务器由于缺乏GUI无法使用；pytorch的安装需要注意匹配CUDA版本，我这里是11.3，并安装了Conda，因此pytorch的安装命令如下：
41. conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
42. 其次是mmcv-full，需要根据Pytorch和CUDA版本选择版本，官网给出了不同版本的安装命令<https://github.com/open-mmlab/mmcv>：
43. 
44. 图1-5 MMCV版本选择
45. pip install mmcv-full=={mmcv\_version} -f https://download.openmmlab.com/mmcv/dist/cu113/torch1.10.0/index.html
46. 此处我指定mmcv\_version为1.4.2。
47. 接下来切换到base环境，安装打包工具：
48. conda install -c conda-forge conda-pack
49. 之后就可以指定环境打包
50. conda pack -n mmdet
51. 打包完成后会生成名为mmdet.tar.gz的文件，使用scp将其发到服务器上，在Conda的envs目录中创建名为mmdet的文件夹，将其解压到文件夹中，至此，就可以在服务器上激活mmdet环境。
52. 以上是MMDetection的环境准备，接下来安装MMDetection，从Github下载源码：
53. git clone https:*//github.com/open-mmlab/mmdetection.git*
54. 将其发送到服务器上，之后切换到目录中进行编译：
55. python setup.py install
56. 至此MMDetection就安装成功了。
57. **3. 在MMDetection上使用自定义数据集**
58. 由于要使用MMDetection训练FashionPedia数据集，需要进行一些环境的配置。下载FashionPedia数据集并发送到服务器上，路径为：
59. /home/koorye/datasets/fashionpedia
60. ├── Annotations
61. │   ├── attributes\_train2020.json
62. │   ├── attributes\_val2020.json
63. │   ├── info\_test2020.json
64. │   ├── instances\_attributes\_train2020.json
65. │   ├── instances\_attributes\_val2020.json
66. │   ├── test.json
67. │   └── train\_test.json
68. └── Images
69. ├── test
70. └── train
71. 数据集包含两个文件，Annotations和Images，其中Images存放训练和测试图片，Annotations存放不同类型的注释文件，我这里使用的是instances开头的文件，用于实例级别的任务，如目标检测。
72. FashionPedia数据集的注释文件遵循COCO格式，包含images、annotations、categories三部分，其中categories包含所有类别信息，annotations包含所有注释信息，指向每个标注框：
73. 'images': [
74. {
75. 'file\_name': 'COCO\_val2014\_000000001268.jpg',
76. 'height': 427,
77. 'width': 640,
78. 'id': 1268
79. },
80. ...
81. ],
82. 'annotations': [
83. {
84. 'segmentation': [[192.81,
85. 247.09,
86. ...
87. 219.03,
88. 249.06]],  *# 如果有 mask 标签*
89. 'area': 1035.749,
90. 'iscrowd': 0,
91. 'image\_id': 1268,
92. 'bbox': [192.81, 224.8, 74.73, 33.43],
93. 'category\_id': 16,
94. 'id': 42986
95. },
96. ...
97. ],
98. 'categories': [
99. {'id': 0, 'name': 'car'},
100. ]

因此我不需要做过多修改就可以直接用它进行训练。为了适配FCOS，需要修改config文件，我这里选择configs/fcos/fcos\_x101\_64x4d\_fpn\_gn-head\_mstrain\_640-800\_2x\_coco.py进行修改，将其命名为fcos\_x101\_64x4d\_fpn\_gn-head\_mstrain\_640-800\_2x\_fashion.py：

1. \_base\_ = './fcos\_r50\_caffe\_fpn\_gn-head\_1x\_coco.py'
2. dataset\_type = 'CocoDataset'
3. clses = ['shirt, blouse', 'top, t-shirt, sweatshirt', 'sweater', 'cardigan', 'jacket', 'vest', 'pants', 'shorts', 'skirt', 'coat', 'dress', 'jumpsuit', 'cape', 'glasses', 'hat', 'headband, head covering, hair accessory', 'tie', 'glove', 'watch', 'belt', 'leg warmer', 'tights, stockings', 'sock', 'shoe', 'bag, wallet', 'scarf', 'umbrella', 'hood', 'collar', 'lapel', 'epaulette', 'sleeve', 'pocket', 'neckline', 'buckle', 'zipper', 'applique', 'bead', 'bow', 'flower', 'fringe', 'ribbon', 'rivet', 'ruffle', 'sequin', 'tassel']
4. model = dict(
5. backbone=dict(
6. type='ResNeXt',
7. depth=101,
8. groups=64,
9. base\_width=4,
10. num\_stages=4,
11. out\_indices=(0, 1, 2, 3),
12. frozen\_stages=1,
13. norm\_cfg=dict(type='BN', requires\_grad=True),
14. norm\_eval=True,
15. style='pytorch',
16. init\_cfg=dict(
17. type='Pretrained', checkpoint='open-mmlab://resnext101\_64x4d')),
18. bbox\_head=dict(
19. num\_classes=len(clses),
20. loss\_cls=dict(
21. type='FocalLoss',
22. use\_sigmoid=True,
23. gamma=2.0,
24. alpha=0.25,
25. loss\_weight=1.0))
26. )
27. img\_norm\_cfg = dict(
28. mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to\_rgb=True)
29. train\_pipeline = [
30. dict(type='LoadImageFromFile'),
31. dict(type='LoadAnnotations', with\_bbox=True),
32. dict(
33. type='Resize',
34. img\_scale=[(1333//2, 640//2), (1333//2, 800//2)],
35. multiscale\_mode='value',
36. keep\_ratio=True),
37. dict(type='RandomFlip', flip\_ratio=0.5),
38. dict(type='Normalize', \*\*img\_norm\_cfg),
39. dict(type='Pad', size\_divisor=32),
40. dict(type='DefaultFormatBundle'),
41. dict(type='Collect', keys=['img', 'gt\_bboxes', 'gt\_labels']),
42. ]
43. test\_pipeline = [
44. dict(type='LoadImageFromFile'),
45. dict(
46. type='MultiScaleFlipAug',
47. img\_scale=(1333, 800),
48. flip=False,
49. transforms=[
50. dict(type='Resize', keep\_ratio=True),
51. dict(type='RandomFlip'),
52. dict(type='Normalize', \*\*img\_norm\_cfg),
53. dict(type='Pad', size\_divisor=32),
54. dict(type='ImageToTensor', keys=['img']),
55. dict(type='Collect', keys=['img']),
56. ])
57. ]
58. data = dict(
59. samples\_per\_gpu=4,
60. workers\_per\_gpu=4,
61. train=dict(pipeline=train\_pipeline,
62. type=dataset\_type,
63. classes=clses,
64. ann\_file='/home/koorye/datasets/fashionpedia/Annotations/train\_test.json',
65. img\_prefix='/home/koorye/datasets/fashionpedia/Images/train'
66. ),
67. val=dict(pipeline=test\_pipeline,
68. type=dataset\_type,
69. classes=clses,
70. ann\_file='/home/koorye/datasets/fashionpedia/Annotations/train\_test.json',
71. img\_prefix='/home/koorye/datasets/fashionpedia/Images/train'
72. ),
73. test=dict(pipeline=test\_pipeline,
74. type=dataset\_type,
75. classes=clses,
76. ann\_file='/home/koorye/datasets/fashionpedia/Annotations/train\_test.json',
77. img\_prefix='/home/koorye/datasets/fashionpedia/Images/train')
78. )
79. *# optimizer*
80. optimizer = dict(
81. lr=0.01, paramwise\_cfg=dict(bias\_lr\_mult=2., bias\_decay\_mult=0.))
82. optimizer\_config = dict(
83. \_delete\_=True, grad\_clip=dict(max\_norm=35, norm\_type=2))
84. *# learning policy*
85. lr\_config = dict(step=[16, 22])
86. runner = dict(type='EpochBasedRunner', max\_epochs=24)
87. 修改的部分由红色标出，包括指定类别、类别数、数据集格式、训练和测试读取的注释文件和图片目录，以及使用的每个GPU上的batch数，我这里测试使用4时显存达到饱和。
88. 接下来就可以开始训练，不过如果使用预训练模型，需要提前将模型文件下载并发送到Pytorch缓存目录中，否则由于服务器无法连接公网将报错。
89. 至此，可以使用命令进行多卡分布式训练：
90. tool/dist\_train.sh config/fcos/fcos\_x101\_64x4d\_fpn\_gn-head\_mstrain\_640-800\_2x\_fashion.py 4
91. 最后一个参数指定使用4个GPU。训练完成后，可以在work\_dirs目标中查看模型文件和日志：
92. (base) root@ai:/home/koorye/mmdetection/work\_dirs/fcos\_x101\_64x4d\_fpn\_gn-head\_mstrain\_640-800\_2x\_fashion*# ls*
93. 20220404\_025330.log       epoch\_19.pth
94. 20220404\_025330.log.json  epoch\_2.pth
95. 20220405\_022050.log       epoch\_20.pth
96. 20220405\_022050.log.json  epoch\_21.pth
97. 20220405\_022245.log       epoch\_22.pth
98. 20220405\_022245.log.json  epoch\_23.pth
99. epoch\_1.pth               epoch\_24.pth
100. epoch\_10.pth              epoch\_3.pth
101. epoch\_11.pth              epoch\_4.pth
102. epoch\_12.pth              epoch\_5.pth
103. epoch\_13.pth              epoch\_6.pth
104. epoch\_14.pth              epoch\_7.pth
105. epoch\_15.pth              epoch\_8.pth
106. epoch\_16.pth              epoch\_9.pth
107. epoch\_17.pth              fcos\_x101\_64x4d\_fpn\_gn-head\_mstrain\_640-800\_2x\_fashion.py
108. epoch\_18.pth              latest.pth
109. **4. 修改MMDetection源码并使用自定义模型**

由于MMDetection自带的Focal Loss不支持类别加权的训练策略，我这里修改了源码，由于是工程问题展示，此处不展示太多原理，只讲述工程实现策略。

在mmdetection/mmdet/models/losses中创建名为cls\_balanced\_focal\_loss.py的文件，该文件在focal\_loss.py的基础上做出修改：

1. *# Copyright (c) OpenMMLab. All rights reserved.*
2. import numpy as np
3. import torch
4. import torch.nn as nn
5. import torch.nn.functional as F
6. from mmcv.ops import sigmoid\_focal\_loss as \_sigmoid\_focal\_loss
7. from ..builder import LOSSES
8. from .utils import weight\_reduce\_loss
9. def py\_sigmoid\_focal\_loss(pred,
10. target,
11. weight=None,
12. gamma=2.0,
13. alpha=0.25,
14. reduction='mean',
15. avg\_factor=None):
16. pred\_sigmoid = pred.sigmoid()
17. target = target.type\_as(pred)
18. pt = (1 - pred\_sigmoid) \* target + pred\_sigmoid \* (1 - target)
19. focal\_weight = (alpha \* target + (1 - alpha) \*
20. (1 - target)) \* pt.pow(gamma)
21. loss = F.binary\_cross\_entropy\_with\_logits(
22. pred, target, reduction='none') \* focal\_weight
23. if weight is not None:
24. if weight.shape != loss.shape:
25. if weight.size(0) == loss.size(0):
26. *# For most cases, weight is of shape (num\_priors, ),*
27. *#  which means it does not have the second axis num\_class*
28. weight = weight.view(-1, 1)
29. elif weight.size(0) == loss.size(1):
30. weight = weight.unsqueeze(0)
31. else:
32. *# Sometimes, weight per anchor per class is also needed. e.g.*
33. *#  in FSAF. But it may be flattened of shape*
34. *#  (num\_priors x num\_class, ), while loss is still of shape*
35. *#  (num\_priors, num\_class).*
36. assert weight.numel() == loss.numel()
37. weight = weight.view(loss.size(0), -1)
38. assert weight.ndim == loss.ndim
39. weight = weight.to(loss.device)
40. loss = weight\_reduce\_loss(loss, weight, reduction, avg\_factor)
41. return loss
43. ...
44. @LOSSES.register\_module()
45. class ClassBalancedFocalLoss(nn.Module):
46. def \_\_init\_\_(self,
47. use\_sigmoid=True,
48. gamma=2.0,
49. alpha=0.25,
50. reduction='mean',
51. loss\_weight=1.0,
52. cls\_weight=None,
53. activated=False):
54. super(ClassBalancedFocalLoss, self).\_\_init\_\_()
55. assert use\_sigmoid is True, 'Only sigmoid focal loss supported now.'
56. self.use\_sigmoid = use\_sigmoid
57. self.gamma = gamma
58. self.alpha = alpha
59. self.reduction = reduction
60. self.loss\_weight = loss\_weight
61. self.activated = activated
62. self.cls\_weight = torch.tensor(cls\_weight).float()
63. def forward(self,
64. pred,
65. target,
66. weight=None,
67. avg\_factor=None,
68. reduction\_override=None):
69. assert reduction\_override in (None, 'none', 'mean', 'sum')
70. reduction = (
71. reduction\_override if reduction\_override else self.reduction)
72. if self.use\_sigmoid:
73. if self.activated:
74. calculate\_loss\_func = py\_focal\_loss\_with\_prob
75. else:
76. if torch.cuda.is\_available() and pred.is\_cuda:
77. calculate\_loss\_func = sigmoid\_focal\_loss
78. else:
79. num\_classes = pred.size(1)
80. target = F.one\_hot(target, num\_classes=num\_classes + 1)
81. target = target[:, :num\_classes]
82. calculate\_loss\_func = py\_sigmoid\_focal\_loss
83. loss\_cls = self.loss\_weight \* calculate\_loss\_func(
84. pred,
85. target,
86. self.cls\_weight,
87. gamma=self.gamma,
88. alpha=self.alpha,
89. reduction=reduction,
90. avg\_factor=avg\_factor)
91. else:
92. raise NotImplementedError
93. return loss\_cls

主要做法是，在FocalLoss类上添加名为cls\_weight的参数，将其转换为tensor后作为weight传入focal\_loss的计算函数，在focal\_loss的计算过程中，利用boardcast机制将每个类别的权重分别乘到对应的每个loss上。

之后在config文件中指定权重，这里根据论文《Learning deep representation for imbalanced classification》和《Class-Balanced Loss Based on Effective Number of Samples》给出的类别权重计算策略事先计算了权重：

1. \_base\_ = './fcos\_r50\_caffe\_fpn\_gn-head\_1x\_coco.py'
2. dataset\_type = 'CocoDataset'
3. clses = ['shirt, blouse', 'top, t-shirt, sweatshirt', 'sweater', 'cardigan', 'jacket', 'vest', 'pants', 'shorts', 'skirt', 'coat', 'dress', 'jumpsuit', 'cape', 'glasses', 'hat', 'headband, head covering, hair accessory', 'tie', 'glove', 'watch', 'belt', 'leg warmer', 'tights, stockings', 'sock', 'shoe', 'bag, wallet', 'scarf', 'umbrella', 'hood', 'collar', 'lapel', 'epaulette', 'sleeve', 'pocket', 'neckline', 'buckle', 'zipper', 'applique', 'bead', 'bow', 'flower', 'fringe', 'ribbon', 'rivet', 'ruffle', 'sequin', 'tassel']
4. *# effective sample loss weight beta=.999*
5. cls\_weights = [0.5631061188393242, 0.561921578338332, 0.7244106412078546, 0.8391473841042447, 0.5621435234313629, 1.0955003085592585, 0.56192381011002, 0.5999960165758521, 0.5655517032185713, 0.5877300634948965, 0.5619215461318604, 0.932710808698791, 3.983082777928567, 0.5663221112906932, 0.6111291621835592, 0.5799361290655214, 0.7323975301807979, 0.7493789932444386, 0.5815098383815367, 0.5625148687150053, 5.300854527481565, 0.5694333585943118, 0.6078277161308269, 0.5619215420827506, 0.5623328072695912, 0.7521557429290706, 4.44758388433754, 0.7951166329329684, 0.5619431935485343, 0.5633533229210334, 0.964004666095468, 0.5619215420827506, 0.5619215420836217, 0.5619215420827512, 0.5834038242845697, 0.5621110548793051, 0.5788723641845773, 0.5654154367035752, 1.3692965293589698, 0.7539496180167501, 1.2635220769033613, 2.343564655133554, 0.566156709160444, 0.6174804038061301, 0.9284351951783096, 1.9731651981171856]
6. model = dict(
7. backbone=dict(
8. type='ResNeXt',
9. depth=101,
10. groups=64,
11. base\_width=4,
12. num\_stages=4,
13. out\_indices=(0, 1, 2, 3),
14. frozen\_stages=1,
15. norm\_cfg=dict(type='BN', requires\_grad=True),
16. norm\_eval=True,
17. style='pytorch',
18. init\_cfg=dict(
19. type='Pretrained', checkpoint='open-mmlab://resnext101\_64x4d')),
20. bbox\_head=dict(
21. num\_classes=len(clses),
22. loss\_cls=dict(
23. type='ClassBalancedFocalLoss',
24. use\_sigmoid=True,
25. gamma=2.0,
26. alpha=0.25,
27. loss\_weight=1.0,
28. cls\_weight=cls\_weights))
29. ...

通过在loss\_cls中指定Loss的类型和cls\_weight的参数，就可以实现类别加权损失策略。

# 第二章 知识技能学习情况

## 2.1开发环境和工具

**系统和硬件：**

* 个人PC：Linux Manjaro KDE、4核CPU、GTX 1050Ti、16G内存
* 服务器：Linux Ubuntu 18.04、24核CPU、4 \* V100、160G内存

1. **开发环境：**

* Miniconda3 4.12.0
* Python 3.7.0
* Pytorch 1.10.0
* Torchvision 0.11.1
* CUDA 11.3
* MMCV-Full 1.4.2
* MMDetection2 2.23.0

1. **开发工具：**

* Visual Studio Code 1.66.0
* Electerm 1.20.10
* SSH, SCP, SFTP
* Konsole + ZSH

## 2.2预备知识

**基本的数学知识：**微积分、线性代数等

**机器学习基础知识：**决策树、AdaBoost、朴素贝叶斯、高斯混合模型、主成分分析、隐马尔可夫模型等

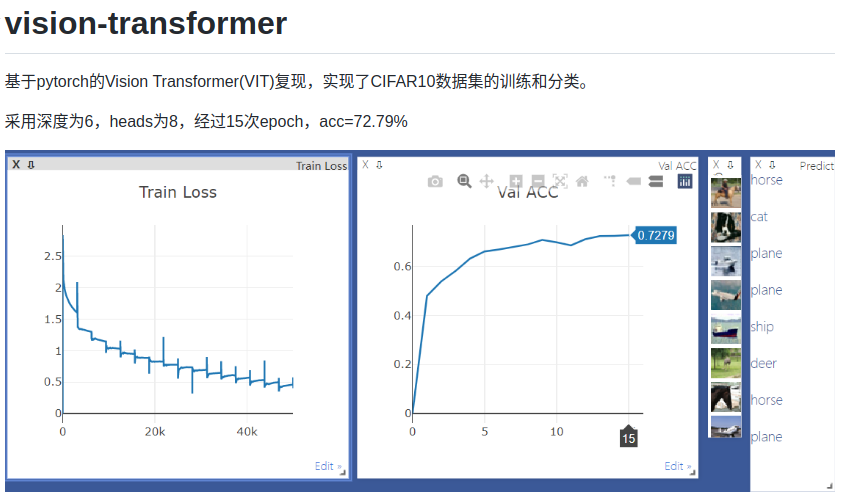
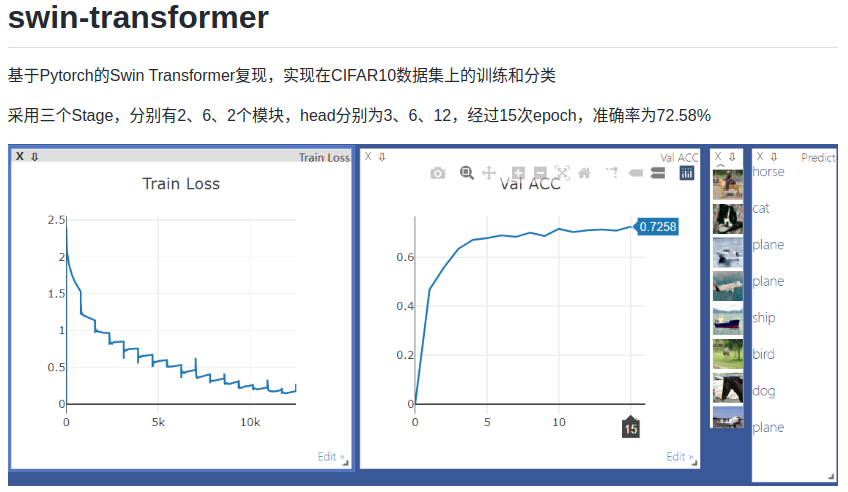
**数学建模基础知识：**插值、拟合、图和网络、规划问题、遗传算法、时间序列模型、多元分析等

**深度学习基础知识：**梯度下降法，全连接神经网络、卷积、BN、池化、Dropout等操作，LeNet5、VGG、AlexNet、ResNet等经典网络结构，SGD、RMSProp、Alex等经典优化器，常见的训练策略、学习率策略、衡量指标、迁移学习策略等

**Python的基本使用：**python语法、numpy科学计算、pandas数据处理、使用python进行数据可视化等

**深度学习框架的使用：**Pytorch、Torchvision等

## 2.3新知识点学习和掌握情况

1. **深度学习进阶知识**
2. 通过CS231n公开课等课程内容，我学习了基于numpy的神经网络实现，加深了对深度学习领域的理解。
3. 通过Pytorch Tutorial，我学习了GAN、Style Transfer、NLP、强化学习等领域的模型和方法，实现了基于DCGan的图像生成，基于CycleGAN、Gram矩阵的风格迁移，基于RNN、Transformer的聊天机器人、文字翻译、看图解释，以及一个马里奥游戏的AI训练DEMO，部分代码可见：<https://github.com/Koorye/pytorch-tutorial>, <https://github.com/Koorye/freshman/tree/master/section1>。
4. 在导师的指导下，我学习了Vision Transformer、Swin Transformer，以及基于Transformer的目标识别模型DETR，复现并实现了图片分类和目标检测功能，部分代码可见：<https://github.com/Koorye/vision-transformer>, <https://github.com/Koorye/swin-transformer>。
5. 
6. 图2-1 VIT复现
7. 
8. 图2-2 Swin Transformer复现

之后在组内学习的过程中，我接触了姿态估计领域的模型和方法，学习了Hourglass、Simple Baselines、HRNet等模型，并了解到Heatmap Regression作关键点回归的方法，自我复现了相关模型实现姿态估计，相关代码可见：<https://github.com/Koorye/pose-estimation>

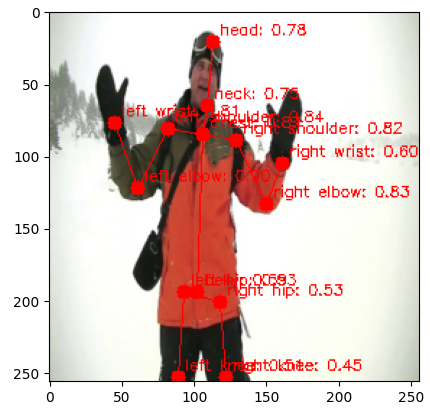
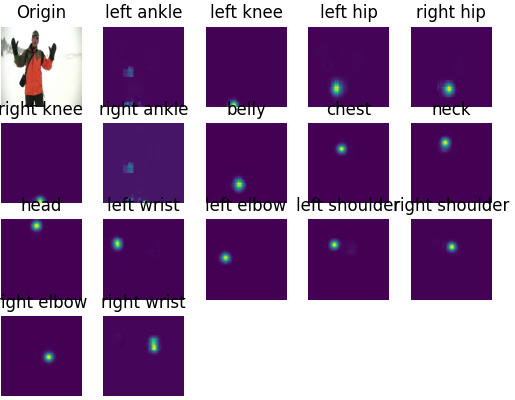


图2-3 姿态估计模型复现

在组内学习的过程中，我对目标检测领域作了更多探索，学习了YOLO系列、R-CNN系列，之后在对小目标和行人检测的探索过程中，我学习了RetinaNet、FCOS、FSAF、AutoAssign、CenterNet、CornerNet等模型，并在其中了解到Focal Loss、FPN、多级学习、多尺度训练和测试等崭新的策略和方法，并自我复现了FCOS模型：<https://github.com/Koorye/fcos>

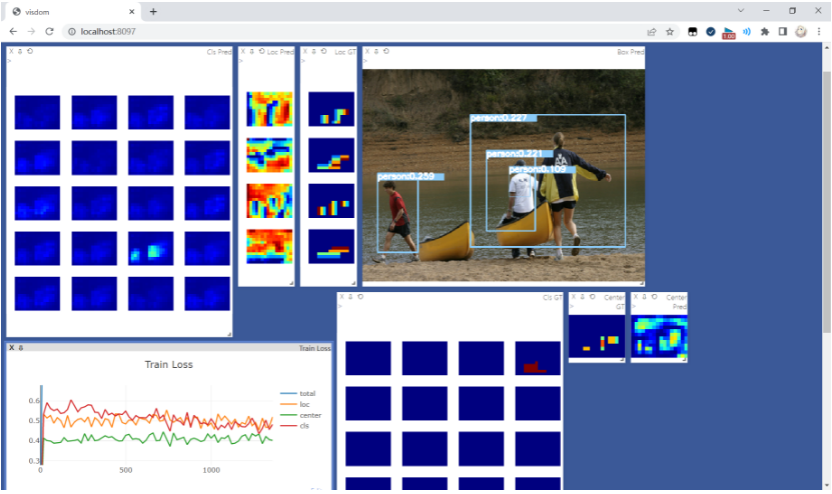


图2-4 FCOS

之后在组内我对小目标和行人检测领域作了针对性探索，整理对小目标和行人检测性能有针对性提升的策略，如在样本选择和NMS阶段使用NWD代替IoU、采用上下文探测模块、对小目标更多的采样等策略都可以提升小目标检测性能；而采用排斥损失、聚集损失，利用显式上下文信息如OR-CNN、Double Pair、PDM等模型，优化NMS等策略都可以提升密集人群的检测性能，通过在组内作报告我受益颇多。

1. **深度学习框架的使用和理解**

通过Pytorch Tutorial我学习了更多关于Pytorch的用法，熟练掌握了张量的变换、科学计算等代码的实现，在工程上对深度学习有了更深入的理解。

在各种模型复现的过程中，我对种种工程问题，诸如如何防止梯度爆炸、如何调整和自适应学习率、如何设定合理的训练次数、如何选择合适的优化器、如何选择损失函数等有了更深入的理解。例如，在损失函数的编写中，要常常注意除法的分母是否接近0，如果过于接近则可能导致梯度溢出，需要继续你clamp或为分母加一个很小的数字来防止这种情况。

组内指导我接触MMDetection2这一目标检测框架，其提供了丰富的模型可供使用，通过对MMDetection2的学习，我在工程上对模块化的编程设计有了更深入的理解，并且学会了使用MMDetection2训练自己的模型、数据集，修改源码等。

1. **深度学习环境的搭建和使用**

对于离线的服务器环境，我在深度学习环境的搭建上遇到了很多困难，由于服务器不能连接公网，所有依赖都需要离线安装，在离线配置环境的过程中，我对Linux系统的操作、Conda的使用、Python依赖的编译等都有了更深入的理解，这对我未来接触正式工作有很大的帮助。

# 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：**

1. 学习机器学习的基础知识，包括：

* 吴恩达公开课<https://www.coursera.org/learn/machine-learning>
* 李宏毅机器学习中文课程<https://www.bilibili.com/video/av10590361?from=search&seid=2567563418955556168>
* 李航统计机器学习

1. 学习深度学习的基础知识，包括：

* 斯坦福公开课CS231n <http://cs231n.stanford.edu/>
* 基于numpy实现常见的神经网络结构，如全连接、卷积、BN等

1. 学习和使用深度学习框架，包括：

* 学习和运行Pytorch官方教程代码 <https://pytorch.org/tutorials/>
* 使用Pytorch复现常见的深度学习模型，如ResNet、GAN、Style Transfer等

1. 学习MMDetection框架的使用，包括：

* 选择MMDetection中的一个或几个目标检测算法进行复现
* 选择MMPose或MMAction2中的一个或几个行为识别和姿态估计算法进行复现

1. 探索小目标检测和行人检测领域

* 阅读小目标检测和行人检测的相关论文
* 学习目标检测模型，如RetinaNet、FCOS、FSAF等，探索其中的各种提升检测性能的策略，寻找对小目标和密集目标检测的启发

1. **未完成：**
2. 针对One-Stage模型，在FashionPedia数据集上研究提升性能的策略
3. 研究样本严重不均衡、遮挡严重的数据集的训练提升策略
4. 研究上下文相关的目标检测推理策略
5. 探索目标检测在模型压缩、弱监督、迁移学习等领域的策略

## 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 | 1. 针对One-Stage模型，在FashionPedia数据集上研究提升性能的策略 | 2022-3-30 | 2022-4-30 |
| 2 | 1. 研究样本严重不均衡、遮挡严重的数据集的训练提升策略 | 2022-5-1 | 2022-5-31 |
| 3 | 1. 研究上下文相关的目标检测推理策略 | 2022-6-1 | 2022-6-30 |
| 4 | 1. 探索目标检测在模型压缩、弱监督、迁移学习等领域的策略 | 2022-7-1 | 2022-8-31 |

# 参考文献

[1] Lin T Y , Goyal P , Girshick R , et al. Focal Loss for Dense Object Detection[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2017, PP(99):2999-3007.

[2] Tian Z , Shen C , Chen H , et al. FCOS: Fully Convolutional One-Stage Object Detection[C]// 2019 IEEE/CVF International Conference on Computer Vision (ICCV). IEEE, 2020.

[3] Zhu C , He Y , Savvides M . Feature Selective Anchor-Free Module for Single-Shot Object Detection[C]// 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2019.

[4] Zhang S , Chi C , Yao Y , et al. Bridging the Gap Between Anchor-Based and Anchor-Free Detection via Adaptive Training Sample Selection[C]// 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2020.

[5] Sun K , Xiao B , Liu D , et al. Deep High-Resolution Representation Learning for Human Pose Estimation[J]. arXiv e-prints, 2019.

[6] Newell A , Yang K , Jia D . Stacked Hourglass Networks for Human Pose Estimation[J]. Springer International Publishing, 2016.

[7] Sun K , Xiao B , Liu D , et al. Deep High-Resolution Representation Learning for Human Pose Estimation[J]. arXiv e-prints, 2019.

[8] Wang J , Xu C , Yang W , et al. A Normalized Gaussian Wasserstein Distance for Tiny Object Detection[J]. 2021.

[9] Zhang S , Wen L , Bian X , et al. Occlusion-aware R-CNN: Detecting Pedestrians in a Crowd[J]. Springer, Cham, 2018.

[10] Cui Y , Jia M , Lin T Y , et al. Class-Balanced Loss Based on Effective Number of Samples[J]. arXiv, 2019.

[11] Huang C , Li Y , Loy C C , et al. Learning Deep Representation for Imbalanced Classification[C]// Computer Vision & Pattern Recognition. IEEE, 2016:5375-5384.

[12] Elhagry A , Saeed M . Investigating the Challenges of Class Imbalance and Scale Variation in Object Detection in Aerial Images[J]. 2022.

[13] Huang X , Ge Z , Jie Z , et al. NMS by Representative Region: Towards Crowded Pedestrian Detection by Proposal Pairing[J]. 2020.