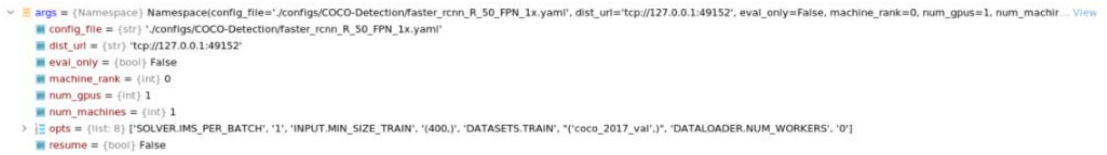
参考示例 1:

单步调试 Faster R-CNN 算法

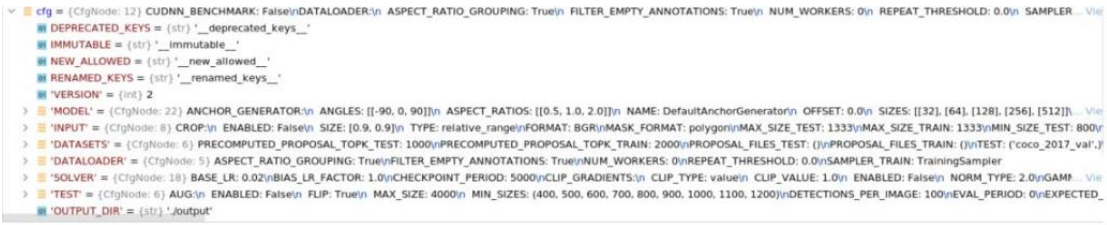
train_net.py

首先读取 cfg 参数:

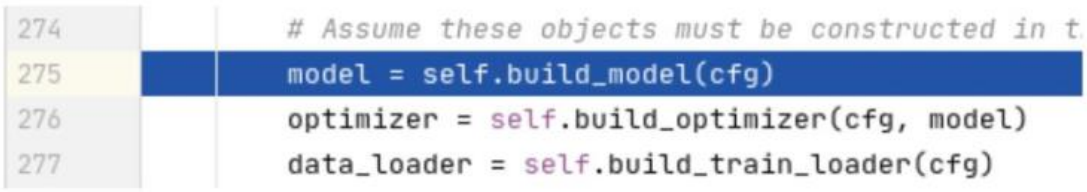
args = default_argument_parser().parse_args()



cfg = setup(args) #加载网络参数



进入 Trainer 函数， defaults.py 文件中 275、 276、 277 为模型构建、优化器构建、数据读取构建部分:



模型构建部分:

特征提取网络

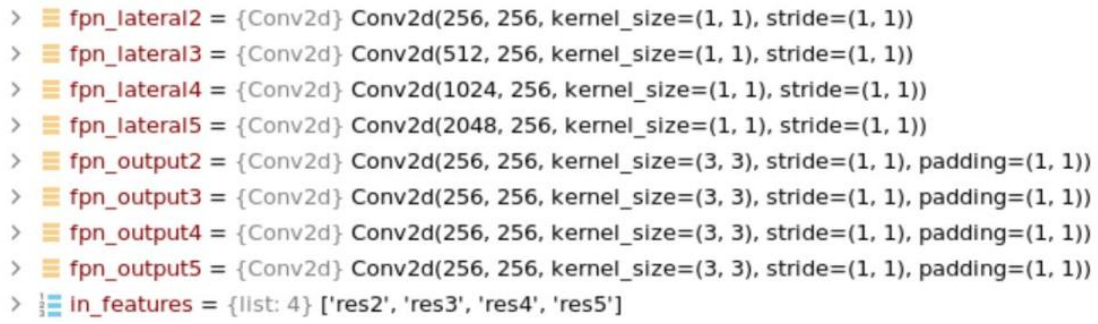
resnet.py

210:bottom_up = build_resnet_backbone(cfg,input_shape) # 建立 resnet 的 backbone， 此处选用 resnet50，根据代码可知：它包括四组 bottleneck，每组分别为 3，4，6，3 个，每组里的具体结构如下所示，用于提取语义信息。

211: in_features = ['res2', 'res3', 'res4', 'res5']#fpn 的输入特征,即采用 resnet 的此 4 个卷积组中的特征图，组内特征图大小相同，组间递减。

fpn.py

213：建立 FPN 网络结构，它主要包括自下而上网络、自上而下网络、横向连接与卷积融合四个部分得到输出。 自下而上结构采用 resnet50 中的 res2,res3,res4,res5 各组中的特征图作为输入，输入通道数依次为 256， 512，1024， 2048， 经 1*1 卷积降低通道数使输出通道数为固定的 256， 自上而下的对特征图上采样（最近邻插值）， 从而进行特征图的融合，而后经 3*3 卷积对融合后的特征图再融合，这样可以消除上采样带来的重叠效应，从而生成最终的特征图即检测层， 好处是使用 FPN 方法融合了不同层的特征， 较好地改善了多尺度检测问题。



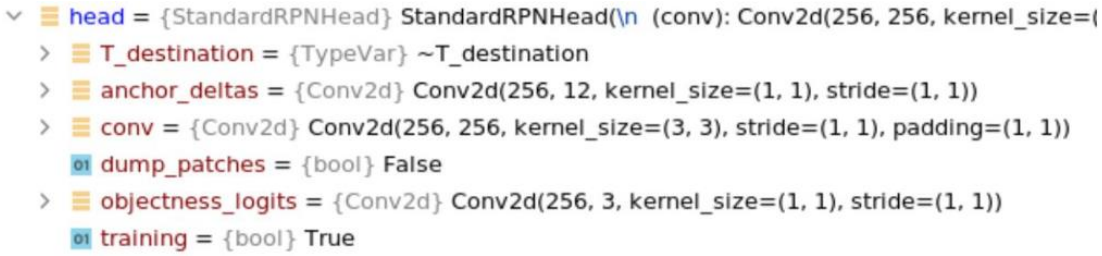
RPN 模块

rpn.py

201： in_features = ['p2', 'p3', 'p4', 'p5', 'p6'] #输入 5 个检测层。

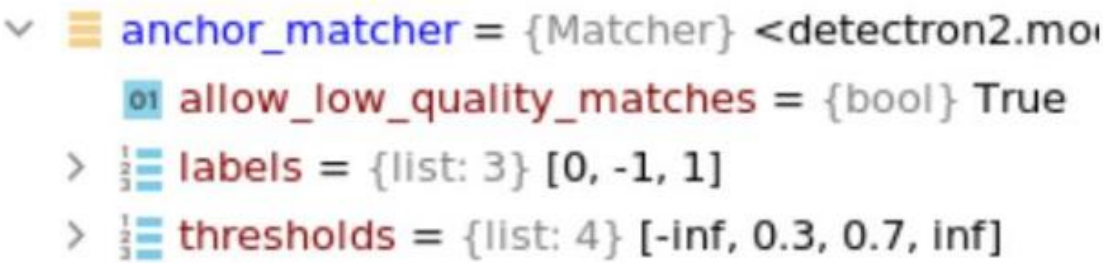
202： self.rpn_head = head #定义 rpn 的 head，用于提取特征，单步调试时可

看到 anchor_deltas 和 objectness_logits 中的 12 和 3, 从而可知每个特征点铺设了三个锚框。



203： self.anchor_generator = anchor_generator #定义生成 anchor 的函数， 4 个通道。

204: self.anchor_matcher = anchor_matcher #定义对 标签和 anchor 进行匹配的函数，可知 iou<0.3 时为负样本， iou>0.7 时为正样本。

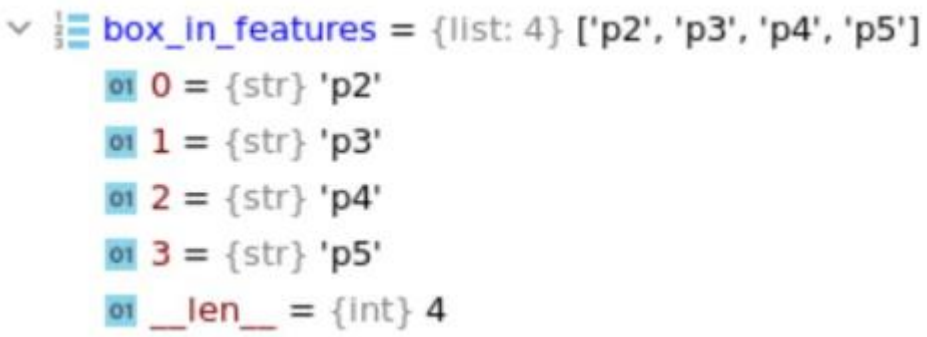


205： self.box2box_transform = box2box_transform #定义对 anchor 和 ground truth 求取回归的真值函数。

Fast-RCNN 模块

roi_heads.py

525: in_features = ['p2', 'p3', 'p4', 'p5']#输入的特征，RoI Pooling 模块接受卷积网络提取的特征图和对应的 RPN 的 RoI，此处指卷积网络提取的 4 种特征图。



526： box_pooler = ROIPooler((level_poolers): ModuleList(

(0): ROIAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0,

aligned=True)

(1): ROIAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0,

aligned=True)

(2): ROIAlign(output_size=(7, 7), spatial_scale=0.0625, sampling_ratio=0,

aligned=True)

(3): ROIAlign(output_size=(7, 7), spatial_scale=0.03125, sampling_ratio=0,

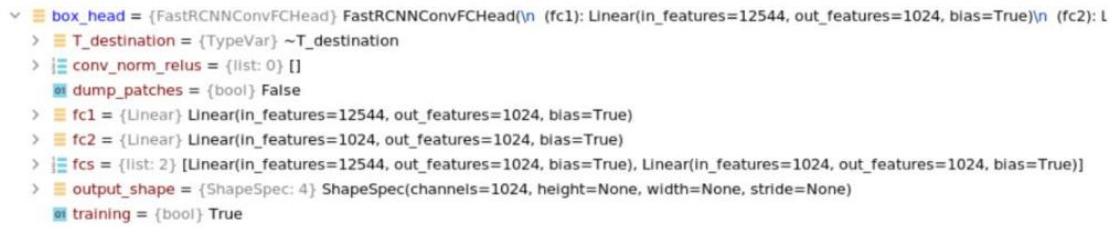
aligned=True)

))#到 RoI Pooling 阶段，将 RPN 产生的不同维度特征变换到相同维度，满足后续全连接神经网络要求，此处应是用 RoI Align 实现此操作，最终得到固定 7*7 大小区域的特征。

527： box_head = FastRCNNConvFCHead(

(fc1): Linear(in_features=12544, out_features=1024, bias=True)

(fc2): Linear(in_features=1024, out_features=1024, bias=True))

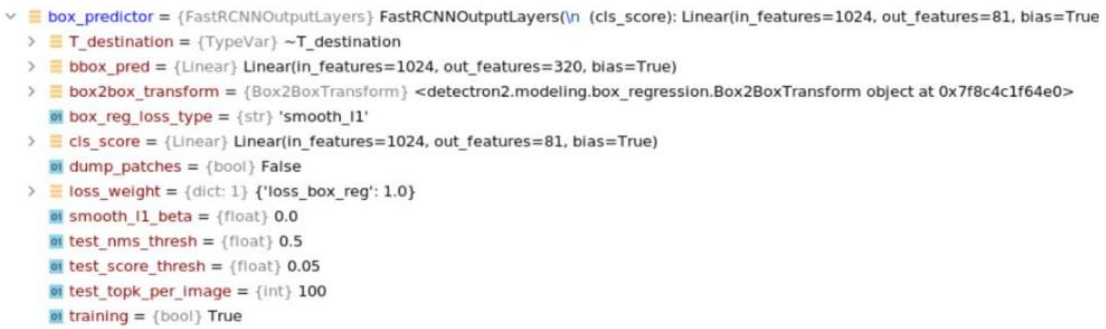


此处指每个 RoI 经池化后得到固定维度为 256*7*7=12544 的特征，经全连接神经网络得到长度为 1024 的 RoI 特征。

528： box_predictor = FastRCNNOutputLayers(

(cls_score): Linear(in_features=1024, out_features=81, bias=True)

(bbox_pred): Linear(in_features=1024, out_features=320, bias=True)

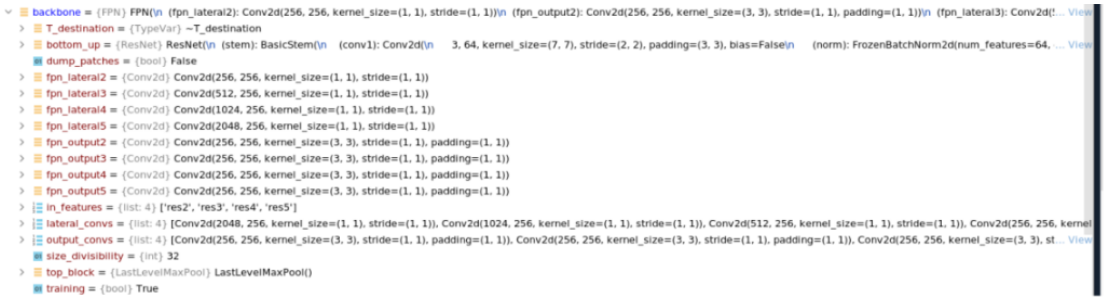


在此有 80 类物体+1 类背景，因此分类网络输出维度应为 81；回归网络预测的输出维度为 80*4。

rcnn.py（整个 faster-rcnn 的框架）

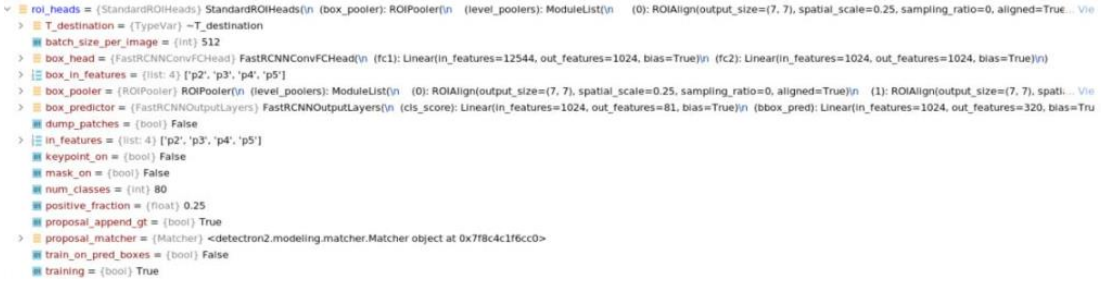
```
58        self.backbone = backbone
59        self.proposal_generator = proposal_generator
60        self.roi_heads = roi_heads
```

58： #此处为 Fast-RCNN 的 backbone， 即 resnet50+FPN， 如下图所示：



59：RPN 定义生成 proposal，结构如下图所示：

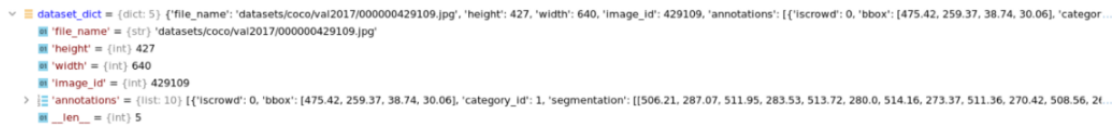
60： Faster-RCNN 第二阶段定义， ROIAlign 后输出大小都为 7*7， 如下图所示：



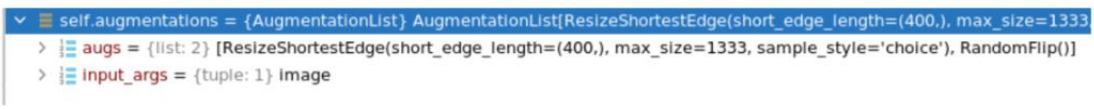
模型训练部分：

5） dataset_dict.py

121：读入图片及一些必要信息，如下图所示：



数据增广，此处的处理是短边到 400，长边不大于 1333 等，如下图所示：

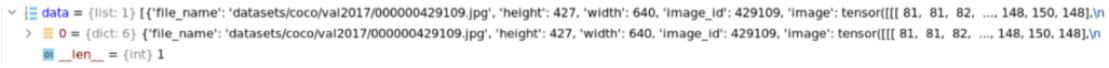


157：对数据增广后的图片相应的标注进行处理。

185：得到处理好的图片和标注。

train_loop.py

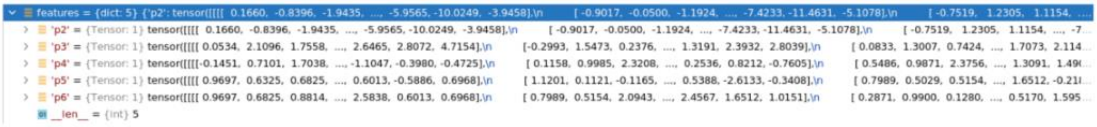
读入要进行处理的图片及数据



rcnn.py 151:#对输入数据先进行前处理

```
images = [x["image"].to(self.device) for x in batched_inputs]  x: {'file_name
images = [(x - self.pixel_mean) / self.pixel_std for x in images]
images = ImageList.from_tensors(images, self.backbone.size_divisibility)
return images
```

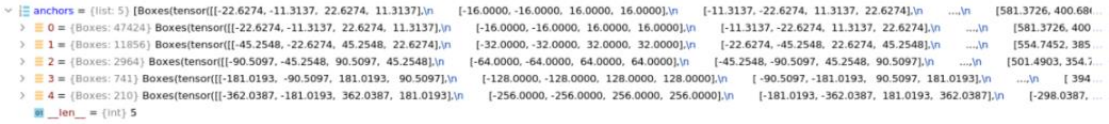
rpn.py： ResNet 和 FPN 特征提取得到五个检测层，该图片的检测层如下图所示：



生成 anchor：

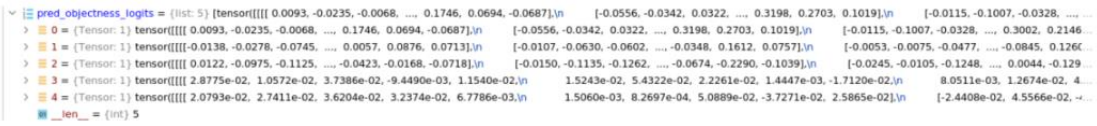
```
221     grid_sizes = [feature_map.shape[-2:] for feature_map in features]
222     anchors_over_all_feature_maps = self._grid_anchors(grid_sizes)
223     return [Boxes(x) for x in anchors_over_all_feature_maps]
```

5 个检测层生成 5 个对应的 anchor:

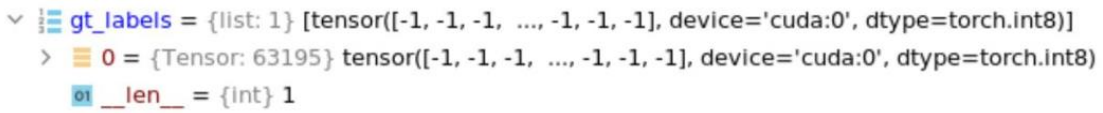


把特征输入 rpn 网络预测锚框的类别和偏差预测的锚框的偏差：

预测的锚框的类别：

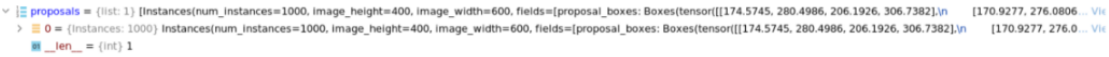


将锚框与真实标注匹配，锚框的真实类别：

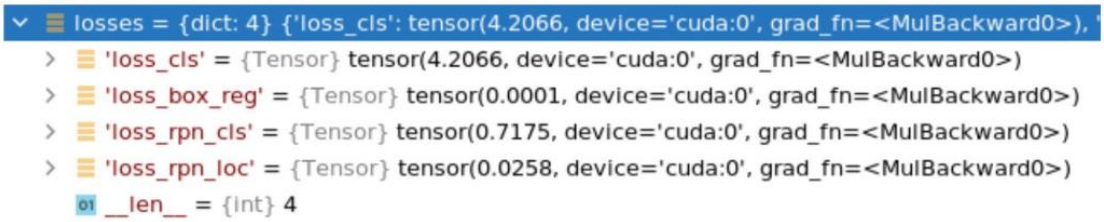


根据锚框真实偏差、真实类别和预测偏差、预测类别可以算 loss， 根据代码可知回归用 smooth l1 loss， 分类用 cross entropy。

448：用 NMS 生成 1000 个 proposal 做进一步的分类与回归得到 proposal：



roi_heads.py 661： 把 proposal 和真实标注混合得到正负样本并得到最终的 RoI 便于求第二阶段的分类和回归，经 RoI Pooling 得到相同维度 7*7 的特征 rcnn.py 对候选区域分类和回归预测经损失函数得到 4 个 losses， 则一张图片处理完成。



至此， 对一张图片前传训练完成， 而后进行反传。

Training 过程：

构建模型：build_model(cfg)、build_optimizer(cfg, model)、build_train_loader(cfg)。

迭代过程： train_loop： run_step()——>rcnn.py： forward()

图片预处理： rcnn： forward()： self.precess_image()

rcnn.py： forward()：返回四个 losses：二分类+回归 & rpn 的多分类+回归

参考示例 2:

训练程序从 train_net.py 开始，首先读入配置文件参数 args，main 函数中读入所有参数 cfg。

然后跳转到 Trainer 函数。

首先是 Trainer 的初始化过程:

初始化过程包括构建模型、优化器、数据读取三个部分，对应函数分别为 build_model(cfg)，build_optimizer(cfg, model), build_train_loader(cfg)。

build_model(cfg)函数:

build_model 包括构建 backbone、proposal 生成、构建 roi_heads 三个主要函数。

1. build_backbone(cfg)函数:

首先构建 resnet_backbone，对应函数为 build_resnet_backbone(cfg, input_shape)，构建 resnet_backbone 包括从配置文件中读取 resnet 深度，out_features 等，以及从默认 cfg 文件中读取 resnet 默认设置参数，最终返回 resnet model。

接着构建 FPN 模块，构建 fpn 模块使用 FPN 类来构建，输入参数包括 bottom_up(resnet50 模型), in_features (res2, res3, res4, res5) , out_channels(256) 等, 构建 fpn 时使用横向连接卷积和上采样卷积完成。最终 out_features 为[p2, p3, p4, p5, p6]五个层，每层的通道数为 256。最后返回 FPN 模块。

2. bulid_proposal_generator(cfg, backbone.output_shape())函数:

包括构建 rpn 网络和计算 rpn loss 两个主要内容。

构建 rpn：

其中 rpn 构建使用 RPN 类来完成，利用 cfg 设置 rpn 的输入特征 in_features 为[res2，res3，res4，res5]，nms 阈值为 0.7，正样本阈值 0.5，训练和测试时的 proposal 最大数量（训练 2000，测试 1000）等参数。之后使用 build_anchor_generator 构建 5 个特征层上的 anchor。五个特征层上的 anchor 尺寸为[32, 64, 128, 256, 512]，同时长宽比使用[0.5, 1, 2]来生成 3 个长宽比的 anchor，利用不同特征图的 h 和 w 和下采样 stride 倍数，来 shift 生成整个特征图的 anchor。

接着使用 Box2BoxTransfrom 来构建坐标转换类，其中包括成员函数 get_deltas 为计算预测 box 和 target box 的偏差值(dx, dy, dw, dh)，计算公式为 Faster R-CNN 论文中的偏差归一化计算公式。构建 anchor 匹配类 Matcher，Matcher 的输入参数为 iou 阈值（0.3，0.7），匹配时每个 gt 的最大 iou 的 anchor 为正样本，iou 大于 0.7 的 anchor 为正样本，小于 0.3 的为负样本，中间区间的为忽略样本。最后构建 rpn_head 模块,包括一个 3x3 特征增强卷积和两路 1x1 卷积，一路用于 rpn 分类预测，一路用于 rpn 回归预测。

3. bulid_roi_heads(cfg, backbone.output_shape())函数

初始化构建包括 box_head、box_pooler、box_predictor 三个模块构建。Box_head 包含两个全连接层（7*7*256 –> 1024, 1024 -> 1024）。box_pooler 为 roi_pooling 或 roi_align 模块，代码中构建 4 个 roi_align 模块分别对应从[p2, p3, p4, p5]上扣取特征。Box_predictor 包含两个全连接层（1024 -> 81, 1024 -> 320），分别负责类别预测和坐标回归。

build_optimizer(cfg, model)函数：

包括初始学习率的设置，weight_decay 设置，模型参数 params 的 weight 和 bias 中学习率和 weight_decay 的设置，优化器的设置（SGD）。

build_train_loader(cfg)函数：

构建训练数据集，包括数据预处理（几何变换、光学变换等（程序中使用 Resize 和 HFlip）），并将数据集转换为 pytorch 的 data_loader 形式。

完成 Trainer 的初始化构建后，开始进入训练过程，主要使用 run_step()函数来完成：

run_step()函数

完成 Trainer 的构建后，进入 run_step()函数开始训练。

首先使用 data = next(data_loader)迭代器来读取数据，接着 data 进入之前构建的 model 开始训练过程：

1. 首先对 data 进行预处理（均值方差归一化，数据增强变换等）

2. data 进入 backbone 提取特征。首先经过 resnet 得到[res2, res3, res4, res5]的特征，然后经过 FPN 模块得到[p2, p3, p4, p5, p6]的特征。

3. 调用 proposal_generator 函数生成 proposals 和 proposal_loss。proposals 的生成定义在 RPN_outputs 类中，包括 anchor 的匹配，正负样本划分，分类与回归 loss 的计算等）

4. 将 proposal 送入 roi_heads 网络得到 roi_head_loss。

5. loss 更新 proposal_loss 和 roi_head_loss。

6. 返回 loss，退出 model 的 forward 函数。

返回至 run_step 函数，对 loss 字典的 4 种 loss 求和。

然后进行 pytorch 训练的 4 步经典过程（计算 loss，优化器梯度清 0， loss 反

传， 优化器单次优化）

losses = sum（losses）

optimizer.zero_grad()

losses.backward()

optimizer.step()

一步训练完成后。train_loop.py 文件中的 train（）函数循环调用 run_step()函数完成训练的迭代，最终完成模型的训练。

训练 Faster R-CNN_R50_FPN

训练环境：ubuntu18.04，1 块 GTX 1080Ti 显卡

训练集：coco_2017_trian，测试集 coco_2017_val

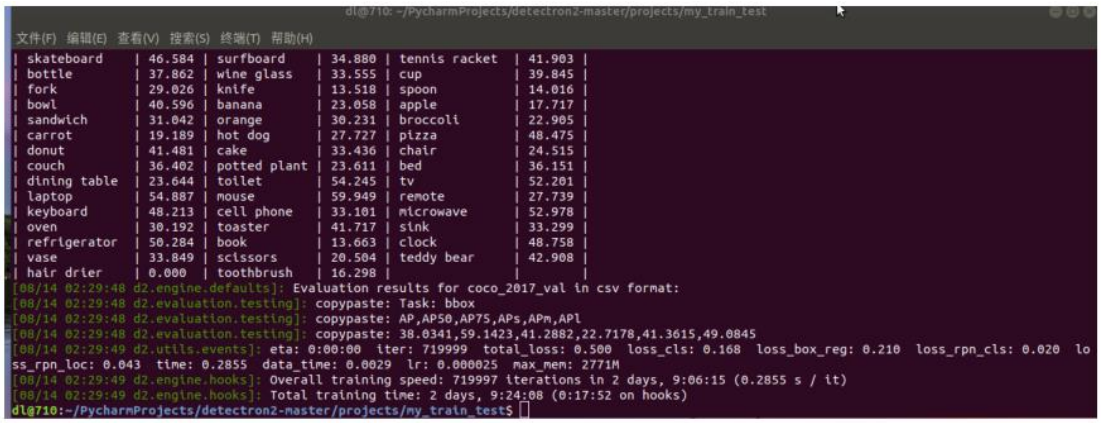
训练参数：

batch_size: 2

由于使用单卡训练，因此修改 iter、lr 这两个参数。

将 max_iter 由 90000 变为 720000，相应的 step_iter 由（60000， 80000）变为（480000， 640000），lr 由 0.02 变为 0.0025。

训练结果：



单卡训练总时间：57 小时

模型精度：ap：38.03 ap50：59.14 aps：22.71 apm：41.36 apl：49.08

官方精度：ap：37.93 ap50：58.84 aps：22.44 apm：41.14 apl：49.10

训练结果与官方结果相比基本相同。