

单步调试 Faster R-CNN 算法

首先读取 cfg 参数:

```
args = {'Namespace': Namespace(config_file='./configs/COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml', dist_url='tcp://127.0.0.1:49152', eval_only=False, machine_rank=0, num_gpus=1, num_machir=1, config_file='./configs/COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml', dist_url='tcp://127.0.0.1:49152', eval_only=False, machine_rank=0, num_gpus=1, num_machines=1, opts=[('SOLVER.IMS_PER_BATCH', '1'), ('INPUT.MIN_SIZE_TRAIN', '400'), ('DATASETS.TRAIN', 'coco_2017_val'), ('DATALOADER.NUM_WORKERS', '0')], resume=False)}
```

```
> cfg = {CfgNode: 12} CUDNN_BENCHMARK: False\nDATALOADER:\n ASPECT_RATIO_GROUPING: True\n FILTER_EMPTY_ANNOTATIONS: True\n NUM_WORKERS: 0\n REPEAT_THRESHOLD: 0\n SAMPLER: V\n DEPRECATED_KEYS = (str) '__deprecated_keys__'\n IMMUTABLE = (str) '__immutable__'\n NEW_ALLOWED = (str) '__new_allowed__'\n RENAMED_KEYS = (str) '__renamed_keys__'\n VERSION = (int) 2\n> MODEL = (CfgNode: 22) ANCHOR_GENERATOR:\n ANGLES: [[-90, 0, 90]]\n ASPECT RATIOS: [[0.5, 1.0, 2.0]]\n NAME: DefaultAnchorGenerator\n OFFSET: 0.0\n SIZES: [[32], [64], [128], [256], [512]]\n> INPUT = (CfgNode: 8) CROP:\n ENABLED: False\n SIZE: [0.0, 0.9]\n TYPE: relative_range\nFORMAT: BGR\nMASK_FORMAT: polygon\nMAX_SIZE_TEST: 1333\nMIN_SIZE_TRAIN: 1333\nMIN_SIZE_TEST: 800\n> DATASETS = (CfgNode: 6) PRECOMPUTED_PROPOSAL_TOPK_TEST: 1000\nPRECOMPUTED_PROPOSAL_TOPK_TRAIN: 2000\nPROPOSAL_FILES_TEST: (list) ('coco_2017_val')\n> DATALOADER = (CfgNode: 5) ASPECT_RATIO_GROUPING: True\nFILTER_EMPTY_ANNOTATIONS: True\nNUM_WORKERS: 0\nREPEAT_THRESHOLD: 0\nSAMPLER: TrainingSampler\n> SOLVER = (CfgNode: 18) BASE_LR: 0.02\nBIAS_LR_FACTOR: 1.0\nCHECKPOINT_PERIOD: 5000\nCLIP_GRADIENTS_CLIP_VALUE: 1.0\nENABLED: False\nNORM_TYPE: 2.0\nGAMMA: 1.0\n> TEST = (CfgNode: 6) AUG:\n ENABLED: False\n FLIP: True\n MAX_SIZE: 4000\n MIN_SIZES: (400, 500, 600, 700, 800, 900, 1000, 1100, 1200)\nDETECTIONS_PER_IMAGE: 100\nEVAL_PERIOD: 0\nEXPECTED_OUTPUT_DIR = (str) 'output'
```

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：

```
274 # Assume these objects must be constructed in t
275 model = self.build_model(cfg)
276 optimizer = self.build_optimizer(cfg, model)
277 data_loader = self.build_train_loader(cfg)
```

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。

```

bottom_up = {ResNet} ResNet(n stem: BasicStem(n (conv1: Conv2d(n 3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False) (norm: FrozenBatchNorm2d(num_features=64, eps=1e-05))n ))n
> T_destination = (TypeVar) ~T_destination
dump_patches = {bool} False
num_classes = {NoneType} None
> res2 = {Sequential: 3} Sequential(n (0: BottleneckBlock(n (shortcut: Conv2d(n 64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False) (norm: FrozenBatchNorm2d(num_features=256, eps=1e-05))n ))n
> res3 = {Sequential: 4} Sequential(n (0: BottleneckBlock(n (shortcut: Conv2d(n 256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False) (norm: FrozenBatchNorm2d(num_features=512, eps=1e-05))n ))n
> res4 = {Sequential: 6} Sequential(n (0: BottleneckBlock(n (shortcut: Conv2d(n 512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False) (norm: FrozenBatchNorm2d(num_features=1024, eps=1e-05))n ))n
> res5 = {Sequential: 3} Sequential(n (0: BottleneckBlock(n (shortcut: Conv2d(n 1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False) (norm: FrozenBatchNorm2d(num_features=2048, eps=1e-05))n ))n
size_divisibility = {int} 0
> stages_and_names = {list: 4} [(Sequential(n (0: BottleneckBlock(n (shortcut: Conv2d(n 64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False) (norm: FrozenBatchNorm2d(num_features=256, eps=1e-05))n ))n
> stem = {BasicStem} BasicStem(n (conv1: Conv2d(n 3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False) (norm: FrozenBatchNorm2d(num_features=64, eps=1e-05))n ))n
training = {bool} True

```

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。

```

> fpn_lateral2 = {Conv2d} Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
> fpn_lateral3 = {Conv2d} Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
> fpn_lateral4 = {Conv2d} Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
> fpn_lateral5 = {Conv2d} Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
> fpn_output2 = {Conv2d} Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
> fpn_output3 = {Conv2d} Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
> fpn_output4 = {Conv2d} Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
> fpn_output5 = {Conv2d} Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
> in_features = {list: 4} ['res2', 'res3', 'res4', 'res5']

```

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。

```
✓ head = {StandardRPNHead} StandardRPNHead(\n (conv): Conv2d(256, 256, kernel_size=(  
> T_destination = {TypeVar} ~T_destination  
> anchor_deltas = {Conv2d} Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))  
> conv = {Conv2d} Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
01 dump_patches = {bool} False  
> objectness_logits = {Conv2d} Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))  
01 training = {bool} True
```

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。

```
✓ anchor_matcher = {Matcher} <detectron2.moi  
01 allow_low_quality_matches = {bool} True  
> labels = {list: 3} [0, -1, 1]  
> thresholds = {list: 4} [-inf, 0.3, 0.7, inf]
```

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。

```

    box_in_features = {llst: 4} ['p2', 'p3', 'p4', 'p5']
    01 0 = {str} 'p2'
    01 1 = {str} 'p3'
    01 2 = {str} 'p4'
    01 3 = {str} 'p5'
    01 __len__ = {int} 4

```

```

526: box_pooler = ROIAlign((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))

```


59: RPN 定义生成 proposal, 结构如下图所示:

```
proposal_generator = (RPN) RPN(in (rpn_head: StandardRPNHead(in (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))\n  (objectness_logits: Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))\n    \n  )\n  T_destination = (TypeVar) ~T_destination\n  anchor_boundary_thresh = (int) -1\n  anchor_generator = (DefaultAnchorGenerator) DefaultAnchorGenerator(in (cell_anchors: BufferList(j)\n  anchor_matcher = (Matcher) <detectron2.modeling.matcher.Matcher object at 0x7f8c4c1ddb00>\n  batch_size_per_image = (int) 256\n  box2box_transform = (Box2BoxTransform) <detectron2.modeling.box_regression.Box2BoxTransform object at 0x7f8c4c1edd68>\n  box_reg_loss_type = (str) 'smooth_l1'\n  dump_patches = (bool) False\n  in_features = (list: 5) ['p2', 'p3', 'p4', 'p5', 'p6']\n  loss_weight = (dict: 2) {'loss_rpn_cls': 1.0, 'loss_rpn_loc': 1.0}\n  min_box_size = (float) 0.0\n  rms_thresh = (float) 0.7\n  positive_fraction = (float) 0.5\n  post_nms_topk = (dict: 2) {True: 1000, False: 1000}\n  pre_nms_topk = (dict: 2) {True: 2000, False: 1000}\n  rpn_head = (StandardRPNHead) StandardRPNHead(in (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))\n  smooth_l1_beta = (float) 0.0\n  training = (bool) True
```

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

```
roi_heads = (StandardROIHeads) StandardROIHeads(in (box_pooler: ROIPointer(in (level_poolers: ModuleList(in (0: ROIAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0, aligned=True)\n  T_destination = (TypeVar) ~T_destination\n  batch_size_per_image = (int) 512\n  box_head = (FastRCNNConvFCHead) FastRCNNConvFCHead(in (fc1: Linear(in_features=12544, out_features=1024, bias=True)\n  box_in_features = (list: 4) ['p2', 'p3', 'p4', 'p5']\n  box_pooler = (ROIPointer) ROIPointer(in (level_poolers: ModuleList(in (0: ROIAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0, aligned=True)\n  box_predictor = (FastRCNNOutputLayers) FastRCNNOutputLayers(in (cls_score: Linear(in_features=1024, out_features=81, bias=True)\n  dump_patches = (bool) False\n  in_features = (list: 4) ['p2', 'p3', 'p4', 'p5']\n  keypoint_on = (bool) False\n  mask_on = (bool) False\n  num_classes = (int) 80\n  positive_fraction = (float) 0.25\n  proposal_append_gt = (bool) True\n  proposal_matcher = (Matcher) <detectron2.modeling.matcher.Matcher object at 0x7f8c4c1f6cc0>\n  train_on_pred_boxes = (bool) False\n  training = (bool) True
```

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:

```
dataset_dict = (dict: 5) {'file_name': 'datasets/coco/val2017/000000429109.jpg', 'height': 427, 'width': 640, 'image_id': 429109, 'annotations': [{'iscrowd': 0, 'bbox': [475.42, 259.37, 38.74, 30.06], 'category': 'person', 'id': 1, 'segmentation': [[506.21, 287.07, 511.95, 283.53, 513.72, 280.0, 514.16, 273.37, 511.36, 270.42, 508.56, 270.42, 506.21]]}]\n  'file_name' = (str) 'datasets/coco/val2017/000000429109.jpg'\n  'height' = (int) 427\n  'width' = (int) 640\n  'image_id' = (int) 429109\n  'annotations' = (list: 10) [{'iscrowd': 0, 'bbox': [475.42, 259.37, 38.74, 30.06], 'category_id': 1, 'segmentation': [[506.21, 287.07, 511.95, 283.53, 513.72, 280.0, 514.16, 273.37, 511.36, 270.42, 508.56, 270.42, 506.21]]}]\n  __len__ = (int) 5
```

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:

```
self.augmentations = (AugmentationList) AugmentationList([ResizeShortestEdge(short_edge_length=(400,), max_size=1333, sample_style='choice'), RandomFlip()]\n  augs = (list: 2) [ResizeShortestEdge(short_edge_length=(400,), max_size=1333, sample_style='choice'), RandomFlip()]\n  input_args = (tuple: 1) Image
```

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

```
> data = [list: {}] {'file_name': 'datasets/coco/val2017/000000429109.jpg', 'height': 427, 'width': 640, 'image_id': 429109, 'image': tensor([[[[ 81, 81, 82, ..., 148, 150, 148]]]])}
> 0 = {dict: {} 'file_name': 'datasets/coco/val2017/000000429109.jpg', 'height': 427, 'width': 640, 'image_id': 429109, 'image': tensor([[[[ 81, 81, 82, ..., 148, 150, 148]]]])}
> _len_ = {int: 1}
```

```
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': '000000000000.jpg', 'image': tensor([[[[ 0.0000e+00,  0.0000e+00,  0.0000e+00, ...,  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, ...,  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, ...,  0.0000e+00,  0.0000e+00,  0.0000e+00], ...,
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, ...,  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, ...,  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, ...,  0.0000e+00,  0.0000e+00,  0.0000e+00]]], dtype=torch.float32, device='cuda:0')}
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:

[illegible]

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]
224

```

5 个检测层生成 5 个对应的 anchor:

```

> anchors = [list: 5] Boxes(tensor[[-22.6274, -1.1317, 22.6274, 11.3137]]\n [-16.0000, -16.0000, 16.0000, 16.0000]]\n [-11.3137, -22.6274, 11.3137, 22.6274]]\n ...,n] [581.3726, 400.608]
> 0 = (Boxes: 47424) Boxes(tensor[[-22.6274, -1.1317, 22.6274, 11.3137]]\n [-16.0000, -16.0000, 16.0000, 16.0000]]\n [-11.3137, -22.6274, 11.3137, 22.6274]]\n ...,n] [581.3726, 400]
> 1 = (Boxes: 11856) Boxes(tensor[[-45.2548, -22.6274, 45.2548, 22.6274]]\n [-32.0000, -32.0000, 32.0000, 32.0000]]\n [-22.6274, -45.2548, 22.6274, 45.2548]]\n ...,n] [554.7452, 385]
> 2 = (Boxes: 2964) Boxes(tensor[[-90.5097, -45.2548, 90.5097, 45.2548]]\n [-64.0000, -64.0000, 64.0000, 64.0000]]\n [-45.2548, -90.5097, 45.2548, 90.5097]]\n ...,n] [501.4903, 354.7]
> 3 = (Boxes: 741) Boxes(tensor[[-181.0193, -90.5097, 181.0193, 90.5097]]\n [-128.0000, -128.0000, 128.0000, 128.0000]]\n [-90.5097, -181.0193, 90.5097, 181.0193]]\n ...,n] [394]
> 4 = (Boxes: 210) Boxes(tensor[[-362.0387, -181.0193, 362.0387, 181.0193]]\n [-256.0000, -256.0000, 256.0000, 256.0000]]\n [-181.0193, -362.0387, 181.0193, 362.0387]]\n ...,n] [-298.0387, ...]
len = (int) 5

```

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:

[illegible]

预测的锚框的类别:

```

> pred_objectness_logits = (list 5) [tensor([[[[ 0.0093, -0.0235, -0.0068, ..., 0.1746, 0.0694, -0.0687]]], [[-0.0556, -0.0342, 0.0322, ..., 0.3198, 0.2703, 0.1019]], [[-0.0115, -0.1007, -0.0328, ...], [ 0. (Tensor 1) tensor([[[[ 0.0093, -0.0235, -0.0068, ..., 0.1746, 0.0694, -0.0687]]], [[-0.0556, -0.0342, 0.0322, ..., 0.3198, 0.2703, 0.1019]], [[-0.0115, -0.1007, -0.0328, ...], [ 1. (Tensor 1) tensor([[[[ 0.0138, -0.0278, -0.0745, ..., 0.0057, 0.0876, 0.0613]]], [[-0.0107, -0.0630, -0.0602, ..., -0.0348, 0.1612, 0.0757]]], [[-0.0053, -0.0075, -0.0477, ..., 0.0845, 0.1266], [ 2. (Tensor 1) tensor([[[[ 0.0122, -0.0975, -0.1125, ..., -0.0423, -0.0168, -0.0718]]], [[-0.0150, -0.1135, -0.1262, ..., -0.0676, -0.2290, -0.1039]]], [[-0.0245, -0.0105, -0.1248, ..., 0.0040, -0.1219], [ 3. (Tensor 1) tensor([[[[ 0.2875e-02, 1.0572e-02, 3.7386e-02, -9.4490e-03, 1.1540e-02], [1.5243e-03, 5.4322e-02, 2.2261e-02, 1.4447e-03, 1.7120e-02], [ 8.0511e-03, 1.2674e-02, 4. (Tensor 1) tensor([[[[ 2.0793e-02, 2.7411e-02, 3.6204e-02, 3.2374e-02, 6.7786e-03], [1.5060e-03, 8.2697e-04, 5.0889e-02, -3.7271e-02, 2.5865e-02]], [[-2.4408e-02, 4.5566e-02, ...], [
len = (int) 5

```

将锚框与真实标注匹配，锚框的真实类别：

```

gt_labels = {list: 1} [tensor([-1, -1, -1, ..., -1, -1, -1], device='cuda:0', dtype=torch.int8)]
0 = {Tensor: 63195} tensor([-1, -1, -1, ..., -1, -1, -1], device='cuda:0', dtype=torch.int8)
__len__ = {int} 1

```

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。

```

losses = {dict: 2} {'loss_rpn_cls': tensor(0.7006, device='cuda:0', grad_fn=<MulBackward0>),
'loss_rpn_loc': tensor(0.3026, device='cuda:0', grad_fn=<MulBackward0>)}
__len__ = {int} 2

```

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

```

proposals = {list: 1} [Instances(num_instances=1000, image_height=400, image_width=600, fields=[proposal_boxes: Boxes(tensor([[174.5745, 280.4986, 206.1926, 306.7382]], device='cuda:0', dtype=torch.float64), [170.9277, 276.0806, ..., 170.9277, 276.0806], device='cuda:0', dtype=torch.float64), proposal_classes: tensor([1, 1, 1, ..., 1, 1, 1], device='cuda:0', dtype=torch.int64), proposal_regression_targets: tensor([0.0001, 0.0001, 0.0001, ..., 0.0001, 0.0001, 0.0001], device='cuda:0', dtype=torch.float64), proposal_regression_targets_weights: tensor([1, 1, 1, ..., 1, 1, 1], device='cuda:0', dtype=torch.float64))]
__len__ = {int} 1

```

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。

```

losses = {dict: 4} {'loss_cls': tensor(4.2066, device='cuda:0', grad_fn=<MulBackward0>),
'loss_box_reg': tensor(0.0001, device='cuda:0', grad_fn=<MulBackward0>),
'loss_rpn_cls': tensor(0.7175, device='cuda:0', grad_fn=<MulBackward0>),
'loss_rpn_loc': tensor(0.0258, device='cuda:0', grad_fn=<MulBackward0>)}
__len__ = {int} 4

```

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.preprocess_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 Box2BoxTransform 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_train, 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:

```
dl@710: ~/PycharmProjects/detectron2-master/projects/my_train_test
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
| skateboard | 46.584 | surfboard | 34.880 | tennis racket | 41.903 |
| bottle | 37.062 | wine glass | 33.555 | cup | 39.045 |
| fork | 29.026 | knife | 13.518 | spoon | 14.016 |
| bowl | 40.596 | banana | 23.058 | apple | 17.717 |
| sandwich | 31.042 | orange | 30.231 | broccoli | 22.905 |
| carrot | 19.189 | hot dog | 27.727 | pizza | 48.475 |
| donut | 41.481 | cake | 33.436 | chair | 24.515 |
| couch | 36.402 | potted plant | 23.611 | bed | 36.151 |
| dining table | 23.644 | toilet | 54.245 | tv | 52.201 |
| laptop | 54.887 | mouse | 59.949 | remote | 27.739 |
| keyboard | 48.213 | cell phone | 33.101 | microwave | 52.978 |
| oven | 30.192 | toaster | 41.717 | sink | 33.299 |
| refrigerator | 50.284 | book | 13.663 | clock | 48.758 |
| vase | 33.849 | scissors | 20.504 | teddy bear | 42.908 |
| hair drier | 0.000 | toothbrush | 16.298 |
[08/14 02:29:48 d2.engine.defaults]: Evaluation results for coco_2017_val in csv format:
[08/14 02:29:48 d2.evaluation.testing]: cypypaste: Task: bbox
[08/14 02:29:48 d2.evaluation.testing]: cypypaste: AP,AP50,AP75,APr,APm,AP1
[08/14 02:29:48 d2.evaluation.testing]: cypypaste: 38.0341,59.1423,41.2882,22.7178,41.3615,49.0845
[08/14 02:29:49 d2.utils.events]: eta: 0:00:00 iter: 719999 total_loss: 0.500 loss_cls: 0.168 loss_box_reg: 0.210 loss_rpn_cls: 0.020 loss_rpn_loc: 0.043 time: 0.2855 data_time: 0.0029 lr: 0.000025 max_mem: 2771M
[08/14 02:29:49 d2.engine.hooks]: Overall training speed: 719997 iterations in 2 days, 9:06:15 (0.2855 s / it)
[08/14 02:29:49 d2.engine.hooks]: Total training time: 2 days, 9:24:08 (0:17:52 on hooks)
dl@710:~/PycharmProjects/detectron2-master/projects/my_train_test$
```

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。