The Simplest Faster - RCNN

by cmd23333

github : cmd23333

e-mail : 623074850@qq.com

opt._parse(kwargs) 更新并打印config信息

dataset = Dataset(opt) 导入数据集

dataloader = torch.utils.data.Dataloader
(dataset, batch_size=1, shuffle=True, num_workers=0)

faster_rcnn = FasterRCNNVGG16() 构建模型 (默认 n_fg_class = 20)

→ extractor, classifier = decom_vgg16()
(提特征, ~~好像没用到~~)

rpn = RegionProposalNetwork(512, 512, ratios=ratios, anchor_scales=…, feat_stride=16)
(512: input通道数; 512: 中间层通道数; feat_stride: vgg的conv5把原图变小了16倍.)

→ rpn的anchor_base = generate_anchor_base(anchor_scales, ratios)
(anchor_scales: 8, 16, 32; ratios: 0.5, 1, 2)

以(0,0)为中心, 产生9个anchors. 基础尺寸=16

ratio	w, h			
0.5	$w=2^{6.5}$, $h=2^{7.5}$			
1	$(16\times8)^2$	$(16\times16)^2$	$(16\times32)^2$	
		$w=256, h=256$		
2				

$16\times8\times16\times8=2^{14}$

$ratio=\frac{x}{y}$

$0.5y\cdot y=2^{14}$

$y=2^{7.5}$

$x=2^{6.5}$

↳ rpn的 proposal_layer = ProposalCreator(self, ..)

输入：loc、score和对2W个anchor的坐标和原图尺寸scale

输出：~2000个训练样本rois (~2000, 4) 坐标. 无gt

9个 n_anchor = self.anchor_base.shape[0]

self.conv1 = nn.Conv2d(in_channel, mid_channel, 3, 1, 1) 'SAME' kernel_size pad=1. $\frac{h-3+2\times1}{1}+1=h$.

self.score = nn.Conv2d(mid_channel, n_anchor*2, 1, 1, 0) stride

self.loc = nn.Conv2d(mid_channel, n_anchor*4, 1, 1, 0) padding

然后用 normal_init() 初始化上面的层.

head = VGG16RoIHead(n_class = n_fg_class + 1, 前景=20
roi_size = 7,
spatial_scale = $\frac{1}{\text{self.feat\_stride}}$,
classifier = classifier)

↳ head的 classifier = classifier

同样.初始化这2层.

cls_loc = nn.Linear(4096, n_class * 4)

score = nn.Linear(4096, n_class)

head的 roi = ROIPooling2D(roi_size, roi_size, spatial_scale)

再初始化父类 FasterRCNN

self.loc_normalize_mean = loc_normalize_mean (0, 0, 0, 0)

self.loc_normalize_std = loc_normalize_std (0.1, 0.1, 0.2, 0.2)

self.use_preset('evaluate')

设置 nms_thresh = 0.3

score_thresh = 0.05

这样. model construct completed.

trainer = FasterRCNNTrainer(faster_rcnn).cuda()

↳

self.faster_rcnn = faster_rcnn

self.rpn_sigma = opt.rpn_sigma 3

self.roi_sigma = opt.roi_sigma 1

正负样本尽量一半.

↑

pos_ratio = 0.5

默认 n_sample = 256. neg_iou_thresh = 0.3 pos_iou_thresh = 0.7

trainer的 anchor_target_creator = AnchorTargetCreator()

输入: ~2W个anchor坐标 和图中bbox真实坐标

输出: (~2W, 1) label: 128个1, 128个0. 其余-1

(~2w, 4) 的回归目标. (所有anchor的坐标都有)

trainer的 proposal_target_creator = ProposalTargetCreator()
默认 n_sample = 128

从RPN产生的~2000个Roi中挑128个训练.

Roi和gt的iou > 0.5 (pos_iou_thres) 选32个, < 0.1 选 128-32=96个

输入: ~2000个roi, 一张图中所有bbox gt (R,4). 对应类(R,1)
(pos_ratio)

输出: 128个sample roi (128,4)

gt_roi_loc (128,4). gt_roi_label (128,1)

trainer的 optimizer = self.faster_rcnn.get_optimizer()

t.optim.SGD(params, momentum=0.9)

trainer的 meters = {k: AvergageValueMeter() for k in LossTuple._fields}

加载保存的权重 如有.

lr_ = opt.lr 0.001

下面就开始训练了.

trainer.reset_meters()

img, bbox_, label_, scale = next(iter(dataloader))
就看一下一张图的训练过程量

全部 .cuda()
- img: [1, 3, 600(H), 901(W)]
- bbox: [1, 2, 4] (317, 472, 529, 755) (..)
- label_: [1, 2] 9, 9
- scale 转化为 float 1.801

trainer.train_step(img, bbox, label, scale)

首先. optimizer.zero_grad()

loss = self.forward(img, bbox, label, scale)

确认 bbox.shape[0] == 1.

img_size = (600, 901)

features = self.faster_rcnn.extractor(img)

(1, 512, 37, 56)

rpn_locs, rpn_scores, rois, roi_indices, anchor =

self.faster_rcnn.rpn(features, img_size, scale)

600, 901 1, 801

RPN

1 37 56

n, _, hh, ww = features.shape

37×56×9

anchor = _enumerate_shifted_anchor

(18648, 4)

(np.array(self.anchor_base), self.feat_stride, hh, ww)

16

(原图尺寸)

n_anchor = anchor.shape[0] / (hh * ww) = 9

h = F.relu(self.conv1(features))

(1, 512, 37, 56)

512, 512, 3, 1, 1

f s p

rpn_locs = self.loc(h)

(1, 36, 37, 56)

Conv2D 512, 36, 1, 1

f s

rpn_locs = rpn_locs.permute(0,2,3,1).contiguous()
(1,18648,4) (1,37,56,36)
.view(n,-1,4)

rpn_scores = self.score(h)
(1,18,37,56) ↓ Conv2D (512,18,1,1)

rpn_scores = rpn_scores.permute(0,2,3,1).contiguous()
(1,37,56,18)

rpn_softmax_scores = F.softmax(
(1,37,56,9,2) rpn_scores.view(n,hh,ww,n_anchor,2)
dim=4)

(1,37,56,9) 前景得分.
rpn_fg_scores = rpn_softmax_scores[:,:,:,:,1].contiguous()
rpn_fg_scores = rpn_fg_scores.view(n,-1)
(1,18648)
rpn_scores = rpn_scores.view(n,-1,2)
(1,18648,2)

rois = list(), roi_indices = list()
for i in range(n): 我们这batch_size=1
为检测网络提供2000个训练样本
roi = self.proposal_layer
(rpn_locs[i].cpu().data.numpy(), (18648,4)
rpn_fg_scores[i].cpu().data.numpy(), (18648)
anchor, img_size, scale=scale)
(18648,4) 600,901 1.801

n_pre_nms = 12000

n_post_nms = 2000

(18648,4)

rpn_locs和anchor怎么转换.

roi = loc2bbox(anchor, rpn_locs[i]) (18648,4)

把x,y,h,w → y1, x1, y2, x2

$\hat{G}_x = P_w \cdot d_x(P) + P_x$ P_*为anchor的

$\hat{G}_w = P_w \exp(d_w(P))$ $d_*(P)$为rpn_locs的.

y小于高度, >0 roi[:, slice(0,4,2)] = np.clip

(roi[:, slice(0,4,2)], 0, img_size[0]) H=600

x小于宽度, >0 roi[:, slice(1,4,2)] = np.clip

(roi[:, slice(1,4,2)], 0, img_size[1]) W=901

大小的预测框不要. 16 1.801

28.82 min_size = self.min_size * scale

hs = roi[:,2] - roi[:,0]

ws = roi[:,3] - roi[:,1]

keep = np.where((hs >= min_size) & (ws >= min_size))[0]

要求预测框h,w都比min_size大

可能是用了载入的权重,没有坏的不要.

(18648,4)

roi = roi[keep, :]

score = score[keep] → rpn_fg_scores[i]

(18648,1)

排序 order = score.ravel().argsort()[::-1]

前12000个 roi = roi[order[:n_pre_nms], :]

12000

表示保留的index

keep = non_maximum_suppression (cp.asarray(roi), thresh=self.nms_thresh)
(917,) 0.7

roi = roi[keep[:n_post]] (2000)

RPN

返回3 roi (917,4) 原图尺寸的bbox (y1, x1, y2, x2)坐标

batch_index = i * np.ones((len(roi),), dtype=np.int32)
[917,] 都是0

rois.append(roi)
roi_indices.append(batch_index)

(roi1, roi2, ...) 不过我们batch_size=1

917,4 rois = np.concatenate(rois, axis=0) 这两条语句 无用.
917, roi_idces = np.concatenate(roi_indices, axis=0)

上面的rpn网络返回3

rpn_locs, rpn_scores, rois, roi_indices, anchor

下一步, 因为batch_size为1.

bbox = bbox[0] label = label[0]
[2,4] [1,2,4] [2] [1,2]

18648,2 rpn_score = rpn_scores[0]
18648,4 rpn_loc = rpn_locs[0]
917,4 roi = rois

下面开始ROI3.

sample_roi, gt_roi_loc, gt_roi_label

= self.proposal_target_creator(roi, bbox, label,

self.loc_normalize_mean, self.loc_normalize_std)

(0.0, 0.0, 0.0, 0.0) (0.1, 0.1, 0.2, 0.2)

从上面生成的 ~2000个(917个) roi中，选择128个用来训练.

32 ~ IoU(roi, gt) > 0.5

128-32 ~ IoU(roi, gt) < 0.1

给这128个赋予 ground truth.

roi

n_bbox, _ = bbox.shape (2, 4)

roi = np.concatenate((roi, bbox), axis=0)

(919, 4). 不是很懂这操作？作弊式训练？

pos_roi_per_image = np.round(self.n_sample * self.pos_ratio)

32.0 = 128 x 0.25

(919, 2) 每个预测和每个gt的iou

iou = bbox_iou(roi, bbox)

(919,) 每个预测和图中哪个gt的iou大

gt_assignment = iou.argmax(axis=1)

(919,) 每个预测和gt_bbox的iou最大值

max_iou = iou.max(axis=1)

[9 9]

gt_roi_label = label[gt_assignment] + 1

把类别从 0~19 换到1~20 因为0用作bg

(45,)

pos_index = np.where(max_iou >= self.pos_iou_thresh)[0]

pos_roi_per_this_image = int(min(pos_roi_per_image, 32.0

pos_index.size)) 45

32

if pos_index.size > 0 : 从过多iou>0.5的抽32个. 不放回

pos_index = np.random.choice(..)

neg_index = np.where((max_iou < neg_iou_thresh_hi) & (max_iou > neg_iou_thresh_low))[0]

0.5 0 原文0.1

同样从neg_index中选取 n_sample - pos_roi_per_this_img 个

keep_index = np.append(pos_index, neg_index) (128,)

gt_roi_label = gt_roi_label[keep_index]

gt_roi_label[pos_roi_per_this_img:] = 0 负样本为0 不计算回归损失

sample_roi = roi[keep_index] (128,4)

(128,4) 真实的 gt. offset

gt_roi_loc = bbox2loc(sample_roi, bbox[gt_assignment[keep_index]])

(128,4) (128,4)

$t_x = \frac{G_x - P_x}{P_w}$ $t_w = \log\left(\frac{G_w}{P_w}\right)$

gt_roi_loc = (gt_roi_loc - loc_norm_mean) / loc_norm_std (128,4)

上面返回 sample_roi, gt_roi_loc, gt_roi_label 原图尺寸

128个0，因为 batch_size=1

```
sample_roi_index = t.zeros(len(sample_roi))
roi_cls_loc, roi_score = self.faster_rcnn.head
    (features, sample_roi, sample_roi_index)
```

(128,)

```
roi_indices = at.totensor(sample_roi_index).float()
```

(128,4)

```
rois = at.totensor(sample_roi).float()
indices_and_rois = t.cat([roi_indices[:, None], rois],
    dim=1)
```

(128,5)

yx → xy

```
xy_indices_and_rois = indices_and_rois[:, [0, 2, 1, 4, 3]]
```

1,512,37,56

```
pool = self.roi(x, xy_indices_and_rois)
```

features　　roi

```
B, C, H, W = x.size
N = roi.size(0)
output = t.zeros(N, C, 7, 7).cuda()
```

128　512

(128,25088)

```
pool = pool.view(pool.size(0), -1)
```

(128, 512×7×7)

```
fc7 = self.classifier(pool)
```

(128,4096)

in_　out_

25088, 4096 Linear

Relu

FC (4096, 4096)

Relu

roi_cls_locs (128,84) = self.cls_loc(fc7) FC(4096,84)

roi_scores (128,21) = self.score(fc7) FC(4096,21)

再往下，计算 loss.

trainer 为所有任务分配 gt

gt_rpn_loc, gt_rpn_label = self.anchor_target_creator
(bbox, anchor, img_size)
[2,4] [18648,4] (600,901)

输出 (~2w,1) label. 128个1, 128个0, 其余 -1

(~2w,4) 回归目标. 所有anchors都有

选择方式:

对每个 gt_bbox, 和它 iou 最大的正样本

剩下的, iou > 0.7 的正样本, 补足128个

iou < 0.3 的负样本. (256 - 正样本个数)个.

600 901
img_H, img_W = img_size

inside_index = _get_inside_index(anchor, img_H, img_W)

在图片内部的 anchor

anchor = anchor[inside_index]

(6928,4)

argmax_ious, label = self._create_label

(6928,) (6928,) (inside_index, anchor, bbox)

每个anchor对应 iou 最大的 bbox 是第几个.

(6928,4) 转换成 offset

loc = bbox2loc(anchor, bbox[argmax_ious])

把不全在图里的anchor恢复过来，label为-1，loc为0。

(18648)
label = _unmap(label, n_anchor, inside_index, fill=-1)

(18648,4)
loc = _unmap(label, n_anchor, inside_index, fill=0)

gt_rpn_label (18648,)

gt_rpn_loc (18648,4)

计算 rpn_loc_loss

0.0444
rpn_loc_loss = _fast_rcnn_loc_loss(
rpn_loc, gt_rpn_loc,
gt_rpn_label.data,
self.rpn_sigma)

label为-1的不参与计算

smooth_L1_loss

3

0.3984
rpn_cls_loss = F.cross_entropy(rpn_score, gt_rpn_label,
ignore_index=-1)

计算 roi_loss

128 128,84
n_sample = roi_cls_loc.shape[0]

roi_cls_loc = roi_cls_loc.view(n_sample, -1, 4)
128,21,4

roi_loc = roi_cls_loc[t.arange(0, n_sample),
128,4
gt_roi_label]

0.5763
roi_loc_loss = _fast_rcnn_loc_loss

(roi_loc, gt_roi_loc, gt_roi_label.data,
128,4 128,4 1
self.roi_sigma)
1

1.8238
roi_cls_loss = nn.CrossEntropy()(roi_score, gt_roi_label)

losses = [rpn_loc_loss, rpn_cls_loss, roi_loc_loss,
roi_cls_loss]
losses = losses + [sum(losses)] 总loss

return LossTuple(*losses)

至此，train_step的 forward 结束.

losses.total_loss.backward()
self.optimizer.step()

trainer

self.update_meters(losses)

至此，train_step 结束