**Swin使用方法**

一：查看configs配置：

python tools/misc/print\_config.py configs/swin/mask\_rcnn\_swin\_small\_patch4\_window7\_mstrain\_480-800\_adamw\_3x\_coco.py > config.txt

输出configs文件内容：

1. models
2. data
3. Schedule

Model部分：

Config:

model = dict(

# 模型类型

type='MaskRCNN',

pretrained=None,

# backbone类型与参数

backbone=dict(

type='SwinTransformer',

embed\_dim=96,

depths=[2, 2, 18, 2],

num\_heads=[3, 6, 12, 24],

window\_size=7,

mlp\_ratio=4.0,

qkv\_bias=True,

qk\_scale=None,

drop\_rate=0.0,

attn\_drop\_rate=0.0,

drop\_path\_rate=0.2,

ape=False,

patch\_norm=True,

out\_indices=(0, 1, 2, 3),

use\_checkpoint=False),

neck=dict(

type='FPN',

in\_channels=[96, 192, 384, 768],

out\_channels=256,

num\_outs=5),

rpn\_head=dict(

type='RPNHead',

in\_channels=256,

feat\_channels=256,

anchor\_generator=dict(

type='AnchorGenerator',

scales=[8],

ratios=[0.5, 1.0, 2.0],

strides=[4, 8, 16, 32, 64]),

bbox\_coder=dict(

type='DeltaXYWHBBoxCoder',

target\_means=[0.0, 0.0, 0.0, 0.0],

target\_stds=[1.0, 1.0, 1.0, 1.0]),

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0),

loss\_bbox=dict(type='L1Loss', loss\_weight=1.0)),

roi\_head=dict(

type='StandardRoIHead',

bbox\_roi\_extractor=dict(

type='SingleRoIExtractor',

roi\_layer=dict(type='RoIAlign', output\_size=7, sampling\_ratio=0),

out\_channels=256,

featmap\_strides=[4, 8, 16, 32]),

bbox\_head=dict(

type='Shared2FCBBoxHead',

in\_channels=256,

fc\_out\_channels=1024,

roi\_feat\_size=7,

num\_classes=13,

bbox\_coder=dict(

type='DeltaXYWHBBoxCoder',

target\_means=[0.0, 0.0, 0.0, 0.0],

target\_stds=[0.1, 0.1, 0.2, 0.2]),

reg\_class\_agnostic=False,

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=1.0),

loss\_bbox=dict(type='L1Loss', loss\_weight=1.0)),

mask\_roi\_extractor=dict(

type='SingleRoIExtractor',

roi\_layer=dict(type='RoIAlign', output\_size=14, sampling\_ratio=0),

out\_channels=256,

featmap\_strides=[4, 8, 16, 32]),

mask\_head=dict(

type='FCNMaskHead',

num\_convs=4,

in\_channels=256,

conv\_out\_channels=256,

num\_classes=13,

loss\_mask=dict(

type='CrossEntropyLoss', use\_mask=True, loss\_weight=1.0))),

train\_cfg=dict(

rpn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.7,

neg\_iou\_thr=0.3,

min\_pos\_iou=0.3,

match\_low\_quality=True,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=256,

pos\_fraction=0.5,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=False),

allowed\_border=-1,

pos\_weight=-1,

debug=False),

rpn\_proposal=dict(

nms\_pre=2000,

max\_per\_img=1000,

nms=dict(type='nms', iou\_threshold=0.7),

min\_bbox\_size=0),

rcnn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.5,

neg\_iou\_thr=0.5,

min\_pos\_iou=0.5,

match\_low\_quality=True,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=512,

pos\_fraction=0.25,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=True),

mask\_size=28,

pos\_weight=-1,

debug=False)),

test\_cfg=dict(

rpn=dict(

nms\_pre=1000,

max\_per\_img=1000,

nms=dict(type='nms', iou\_threshold=0.7),

min\_bbox\_size=0),

rcnn=dict(

score\_thr=0.05,

nms=dict(type='nms', iou\_threshold=0.5),

max\_per\_img=100,

mask\_thr\_binary=0.5)))

dataset\_type = 'CocoDataset'

data\_root = 'data/coco/'

# 数据正则

img\_norm\_cfg = dict(

mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to\_rgb=True)

###训练数据处理流

train\_pipeline = [

dict(type='LoadImageFromFile'),

dict(type='LoadAnnotations', with\_bbox=True, with\_mask=True),

dict(type='RandomFlip', flip\_ratio=0.5),

###多尺度

dict(

type='AutoAugment',

policies=[[{

'type':

'Resize',

'img\_scale': [(480, 1333), (512, 1333), (544, 1333), (576, 1333),

(608, 1333), (640, 1333), (672, 1333), (704, 1333),

(736, 1333), (768, 1333), (800, 1333)],

'multiscale\_mode':

'value',

'keep\_ratio':

True

}],

[{

'type': 'Resize',

'img\_scale': [(400, 1333), (500, 1333), (600, 1333)],

'multiscale\_mode': 'value',

'keep\_ratio': True

}, {

'type': 'RandomCrop',

'crop\_type': 'absolute\_range',

'crop\_size': (384, 600),

'allow\_negative\_crop': True

}, {

'type':

'Resize',

'img\_scale': [(480, 1333), (512, 1333), (544, 1333),

(576, 1333), (608, 1333), (640, 1333),

(672, 1333), (704, 1333), (736, 1333),

(768, 1333), (800, 1333)],

'multiscale\_mode':

'value',

'override':

True,

'keep\_ratio':

True

}]]),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='DefaultFormatBundle'),

dict(type='Collect', keys=['img', 'gt\_bboxes', 'gt\_labels', 'gt\_masks'])

]

###测试验证数据流

test\_pipeline = [

dict(type='LoadImageFromFile'),

dict(

type='MultiScaleFlipAug',

img\_scale=(1333, 800),

flip=False,

transforms=[

dict(type='Resize', keep\_ratio=True),

dict(type='RandomFlip'),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='ImageToTensor', keys=['img']),

dict(type='Collect', keys=['img'])

])

]

data部分：

#train数据类型与数据路径(附加数据信息)

data = dict(

#batch size of each GPU.

samples\_per\_gpu=2,

#How many subprocesses to use for data loading for each GPU

workers\_per\_gpu=2,

train=dict(

# 数据集类型默认coco 在./mmdet/datasets下添加自定义 数据类型同dataset\_type

type='CocoDataset',

# 数据根路径,存放原始数据和标注文件

ann\_file='data/coco/annotations/instances\_train2017.json',

img\_prefix='data/coco/train2017/',

# 数据处理流

pipeline=[

dict(type='LoadImageFromFile'),

dict(type='LoadAnnotations', with\_bbox=True, with\_mask=True),

dict(type='RandomFlip', flip\_ratio=0.5),

dict(

type='AutoAugment',

policies=[[{

'type':

'Resize',

'img\_scale': [(480, 1333), (512, 1333), (544, 1333),

(576, 1333), (608, 1333), (640, 1333),

(672, 1333), (704, 1333), (736, 1333),

(768, 1333), (800, 1333)],

'multiscale\_mode':

'value',

'keep\_ratio':

True

}],

[{

'type': 'Resize',

'img\_scale': [(400, 1333), (500, 1333),

(600, 1333)],

'multiscale\_mode': 'value',

'keep\_ratio': True

}, {

'type': 'RandomCrop',

'crop\_type': 'absolute\_range',

'crop\_size': (384, 600),

'allow\_negative\_crop': True

}, {

'type':

'Resize',

'img\_scale': [(480, 1333), (512, 1333),

(544, 1333), (576, 1333),

(608, 1333), (640, 1333),

(672, 1333), (704, 1333),

(736, 1333), (768, 1333),

(800, 1333)],

'multiscale\_mode':

'value',

'override':

True,

'keep\_ratio':

True

}]]),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='DefaultFormatBundle'),

dict(

type='Collect',

keys=['img', 'gt\_bboxes', 'gt\_labels', 'gt\_masks'])

]),

val=dict(

type='CocoDataset',

ann\_file='data/coco/annotations/instances\_val2017.json',

img\_prefix='data/coco/val2017/',

pipeline=[

dict(type='LoadImageFromFile'),

dict(

type='MultiScaleFlipAug',

img\_scale=(1333, 800),

flip=False,

transforms=[

dict(type='Resize', keep\_ratio=True),

dict(type='RandomFlip'),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='ImageToTensor', keys=['img']),

dict(type='Collect', keys=['img'])

])

]),

test=dict(

type='CocoDataset',

ann\_file='data/coco/annotations/instances\_val2017.json',

img\_prefix='data/coco/val2017/',

pipeline=[

dict(type='LoadImageFromFile'),

dict(

type='MultiScaleFlipAug',

img\_scale=(1333, 800),

flip=False,

transforms=[

dict(type='Resize', keep\_ratio=True),

dict(type='RandomFlip'),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='ImageToTensor', keys=['img']),

dict(type='Collect', keys=['img'])

])

]))

# 评估周期与评估指标

evaluation = dict(interval=1, metric= ' bbox ')

#evaluation = dict(metric=['bbox', 'segm'])

# 优化器类型,初始lr, momentum, weight\_decay

optimizer = dict(

type='AdamW',

lr=0.0001,

betas=(0.9, 0.999),

weight\_decay=0.05,

paramwise\_cfg=dict(

custom\_keys=dict(

absolute\_pos\_embed=dict(decay\_mult=0.0),

relative\_position\_bias\_table=dict(decay\_mult=0.0),

norm=dict(decay\_mult=0.0))))

optimizer\_config = dict(

grad\_clip=None,

type='DistOptimizerHook',

update\_interval=1,

coalesce=True,

bucket\_size\_mb=-1,

use\_fp16=True)

#学习率schedule

#step表示lr drop的epoch

lr\_config = dict(

policy='step',

warmup='linear',

warmup\_iters=500,

warmup\_ratio=0.001,

step=[27, 33])

# 训练总周期

runner = dict(type='EpochBasedRunnerAmp', max\_epochs=36)

# 保存checkpoints间隔

checkpoint\_config = dict(interval=1)

# 打印log间隔

log\_config = dict(interval=50, hooks=[dict(type='TextLoggerHook')])

custom\_hooks = [dict(type='NumClassCheckHook')]

dist\_params = dict(backend='nccl')

log\_level = 'INFO'

# 加载参数

load\_from = None

# 重新加载(包含epoch等信息，会覆盖load\_from)

resume\_from = None

#工作流 train val test

workflow = [('train', 1)]

fp16 = None

#保存目录

work\_dir =“\_”

二：训练

python tools/train.py

configs/swin/mask\_rcnn\_swin\_small\_patch4\_window7\_mstrain\_480-800\_adamw\_3x\_coco.py --gpus 1 --work-dir swin\_training

model = dict(

type='MaskRCNN', # 检测器(detector)名称

backbone=dict( # 主干网络的配置文件

type='ResNet', # 主干网络的类别，可用选项请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/backbones/resnet.py#L308

depth=50, # 主干网络的深度，对于 ResNet 和 ResNext 通常设置为 50 或 101。

num\_stages=4, # 主干网络状态(stages)的数目，这些状态产生的特征图作为后续的 head 的输入。

out\_indices=(0, 1, 2, 3), # 每个状态产生的特征图输出的索引。

frozen\_stages=1, # 第一个状态的权重被冻结

norm\_cfg=dict( # 归一化层(norm layer)的配置项。

type='BN', # 归一化层的类别，通常是 BN 或 GN。

requires\_grad=True), # 是否训练归一化里的 gamma 和 beta。

norm\_eval=True, # 是否冻结 BN 里的统计项。

style='pytorch', # 主干网络的风格，'pytorch' 意思是步长为2的层为 3x3 卷积， 'caffe' 意思是步长为2的层为 1x1 卷积。

init\_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')), # 加载通过 ImageNet 与训练的模型

neck=dict(

type='FPN', # 检测器的 neck 是 FPN，我们同样支持 'NASFPN', 'PAFPN' 等，更多细节可以参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/necks/fpn.py#L10。

in\_channels=[256, 512, 1024, 2048], # 输入通道数，这与主干网络的输出通道一致

out\_channels=256, # 金字塔特征图每一层的输出通道

num\_outs=5), # 输出的范围(scales)

rpn\_head=dict(

type='RPNHead', # RPN\_head 的类型是 'RPNHead', 我们也支持 'GARPNHead' 等，更多细节可以参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/dense\_heads/rpn\_head.py#L12。

in\_channels=256, # 每个输入特征图的输入通道，这与 neck 的输出通道一致。

feat\_channels=256, # head 卷积层的特征通道。

anchor\_generator=dict( # 锚点(Anchor)生成器的配置。

type='AnchorGenerator', # 大多是方法使用 AnchorGenerator 作为锚点生成器, SSD 检测器使用 `SSDAnchorGenerator`。更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/anchor/anchor\_generator.py#L10。

scales=[8], # 锚点的基本比例，特征图某一位置的锚点面积为 scale \* base\_sizes

ratios=[0.5, 1.0, 2.0], # 高度和宽度之间的比率。

strides=[4, 8, 16, 32, 64]), # 锚生成器的步幅。这与 FPN 特征步幅一致。 如果未设置 base\_sizes，则当前步幅值将被视为 base\_sizes。

bbox\_coder=dict( # 在训练和测试期间对框进行编码和解码。

type='DeltaXYWHBBoxCoder', # 框编码器的类别，'DeltaXYWHBBoxCoder' 是最常用的，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/bbox/coder/delta\_xywh\_bbox\_coder.py#L9。

target\_means=[0.0, 0.0, 0.0, 0.0], # 用于编码和解码框的目标均值

target\_stds=[1.0, 1.0, 1.0, 1.0]), # 用于编码和解码框的标准方差

loss\_cls=dict( # 分类分支的损失函数配置

type='CrossEntropyLoss', # 分类分支的损失类型，我们也支持 FocalLoss 等。

use\_sigmoid=True, # RPN通常进行二分类，所以通常使用sigmoid函数。

los\_weight=1.0), # 分类分支的损失权重。

loss\_bbox=dict( # 回归分支的损失函数配置。

type='L1Loss', # 损失类型，我们还支持许多 IoU Losses 和 Smooth L1-loss 等，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/losses/smooth\_l1\_loss.py#L56。

loss\_weight=1.0)), # 回归分支的损失权重。

roi\_head=dict( # RoIHead 封装了两步(two-stage)/级联(cascade)检测器的第二步。

type='StandardRoIHead', # RoI head 的类型，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/roi\_heads/standard\_roi\_head.py#L10。

bbox\_roi\_extractor=dict( # 用于 bbox 回归的 RoI 特征提取器。

type='SingleRoIExtractor', # RoI 特征提取器的类型，大多数方法使用 SingleRoIExtractor，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/roi\_heads/roi\_extractors/single\_level.py#L10。

roi\_layer=dict( # RoI 层的配置

type='RoIAlign', # RoI 层的类别, 也支持 DeformRoIPoolingPack 和 ModulatedDeformRoIPoolingPack，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/ops/roi\_align/roi\_align.py#L79。

output\_size=7, # 特征图的输出大小。

sampling\_ratio=0), # 提取 RoI 特征时的采样率。0 表示自适应比率。

out\_channels=256, # 提取特征的输出通道。

featmap\_strides=[4, 8, 16, 32]), # 多尺度特征图的步幅，应该与主干的架构保持一致。

bbox\_head=dict( # RoIHead 中 box head 的配置.

type='Shared2FCBBoxHead', # bbox head 的类别，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/roi\_heads/bbox\_heads/convfc\_bbox\_head.py#L177。

in\_channels=256, # bbox head 的输入通道。 这与 roi\_extractor 中的 out\_channels 一致。

fc\_out\_channels=1024, # FC 层的输出特征通道。

roi\_feat\_size=7, # 候选区域(Region of Interest)特征的大小。

num\_classes=80, # 分类的类别数量。

bbox\_coder=dict( # 第二阶段使用的框编码器。

type='DeltaXYWHBBoxCoder', # 框编码器的类别，大多数情况使用 'DeltaXYWHBBoxCoder'。

target\_means=[0.0, 0.0, 0.0, 0.0], # 用于编码和解码框的均值

target\_stds=[0.1, 0.1, 0.2, 0.2]), # 编码和解码的标准方差。因为框更准确，所以值更小，常规设置时 [0.1, 0.1, 0.2, 0.2]。

reg\_class\_agnostic=False, # 回归是否与类别无关。

loss\_cls=dict( # 分类分支的损失函数配置

type='CrossEntropyLoss', # 分类分支的损失类型，我们也支持 FocalLoss 等。

use\_sigmoid=False, # 是否使用 sigmoid。

loss\_weight=1.0), # 分类分支的损失权重。

loss\_bbox=dict( # 回归分支的损失函数配置。

type='L1Loss', # 损失类型，我们还支持许多 IoU Losses 和 Smooth L1-loss 等。

loss\_weight=1.0)), # 回归分支的损失权重。

mask\_roi\_extractor=dict( # 用于 mask 生成的 RoI 特征提取器。

type='SingleRoIExtractor', # RoI 特征提取器的类型，大多数方法使用 SingleRoIExtractor。

roi\_layer=dict( # 提取实例分割特征的 RoI 层配置

type='RoIAlign', # RoI 层的类型，也支持 DeformRoIPoolingPack 和 ModulatedDeformRoIPoolingPack。

output\_size=14, # 特征图的输出大小。

sampling\_ratio=0), # 提取 RoI 特征时的采样率。

out\_channels=256, # 提取特征的输出通道。

featmap\_strides=[4, 8, 16, 32]), # 多尺度特征图的步幅。

mask\_head=dict( # mask 预测 head 模型

type='FCNMaskHead', # mask head 的类型，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/roi\_heads/mask\_heads/fcn\_mask\_head.py#L21。

num\_convs=4, # mask head 中的卷积层数

in\_channels=256, # 输入通道，应与 mask roi extractor 的输出通道一致。

conv\_out\_channels=256, # 卷积层的输出通道。

num\_classes=80, # 要分割的类别数。

loss\_mask=dict( # mask 分支的损失函数配置。

type='CrossEntropyLoss', # 用于分割的损失类型。

use\_mask=True, # 是否只在正确的类中训练 mask。

loss\_weight=1.0)))) # mask 分支的损失权重.

train\_cfg = dict( # rpn 和 rcnn 训练超参数的配置

rpn=dict( # rpn 的训练配置

assigner=dict( # 分配器(assigner)的配置

type='MaxIoUAssigner', # 分配器的类型，MaxIoUAssigner 用于许多常见的检测器，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/bbox/assigners/max\_iou\_assigner.py#L10。

pos\_iou\_thr=0.7, # IoU >= 0.7(阈值) 被视为正样本。

neg\_iou\_thr=0.3, # IoU < 0.3(阈值) 被视为负样本。

min\_pos\_iou=0.3, # 将框作为正样本的最小 IoU 阈值。

match\_low\_quality=True, # 是否匹配低质量的框(更多细节见 API 文档).

ignore\_iof\_thr=-1), # 忽略 bbox 的 IoF 阈值。

sampler=dict( # 正/负采样器(sampler)的配置

type='RandomSampler', # 采样器类型，还支持 PseudoSampler 和其他采样器，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/bbox/samplers/random\_sampler.py#L8。

num=256, # 样本数量。

pos\_fraction=0.5, # 正样本占总样本的比例。

neg\_pos\_ub=-1, # 基于正样本数量的负样本上限。

add\_gt\_as\_proposals=False), # 采样后是否添加 GT 作为 proposal。

allowed\_border=-1, # 填充有效锚点后允许的边框。

pos\_weight=-1, # 训练期间正样本的权重。

debug=False), # 是否设置调试(debug)模式

rpn\_proposal=dict( # 在训练期间生成 proposals 的配置

nms\_across\_levels=False, # 是否对跨层的 box 做 NMS。仅适用于 `GARPNHead` ，naive rpn 不支持 nms cross levels。

nms\_pre=2000, # NMS 前的 box 数

nms\_post=1000, # NMS 要保留的 box 的数量，只在 GARPNHHead 中起作用。

max\_per\_img=1000, # NMS 后要保留的 box 数量。

nms=dict( # NMS 的配置

type='nms', # NMS 的类别

iou\_threshold=0.7 # NMS 的阈值

),

min\_bbox\_size=0), # 允许的最小 box 尺寸

rcnn=dict( # roi head 的配置。

assigner=dict( # 第二阶段分配器的配置，这与 rpn 中的不同

type='MaxIoUAssigner', # 分配器的类型，MaxIoUAssigner 目前用于所有 roi\_heads。更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/bbox/assigners/max\_iou\_assigner.py#L10。

pos\_iou\_thr=0.5, # IoU >= 0.5(阈值)被认为是正样本。

neg\_iou\_thr=0.5, # IoU < 0.5(阈值)被认为是负样本。

min\_pos\_iou=0.5, # 将 box 作为正样本的最小 IoU 阈值

match\_low\_quality=False, # 是否匹配低质量下的 box(有关更多详细信息，请参阅 API 文档)。

ignore\_iof\_thr=-1), # 忽略 bbox 的 IoF 阈值

sampler=dict(

type='RandomSampler', #采样器的类型，还支持 PseudoSampler 和其他采样器，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/bbox/samplers/random\_sampler.py#L8。

num=512, # 样本数量

pos\_fraction=0.25, # 正样本占总样本的比例。.

neg\_pos\_ub=-1, # 基于正样本数量的负样本上限。.

add\_gt\_as\_proposals=True

), # 采样后是否添加 GT 作为 proposal。

mask\_size=28, # mask 的大小

pos\_weight=-1, # 训练期间正样本的权重。

debug=False)) # 是否设置调试模式。

test\_cfg = dict( # 用于测试 rnn 和 rnn 超参数的配置

rpn=dict( # 测试阶段生成 proposals 的配置

nms\_across\_levels=False, # 是否对跨层的 box 做 NMS。仅适用于`GARPNHead`，naive rpn 不支持做 NMS cross levels。

nms\_pre=1000, # NMS 前的 box 数

nms\_post=1000, # NMS 要保留的 box 的数量，只在`GARPNHHead`中起作用。

max\_per\_img=1000, # NMS 后要保留的 box 数量

nms=dict( # NMS 的配置

type='nms', # NMS 的类型

iou\_threshold=0.7 # NMS 阈值

),

min\_bbox\_size=0), # box 允许的最小尺寸

rcnn=dict( # roi heads 的配置

score\_thr=0.05, # bbox 的分数阈值

nms=dict( # 第二步的 NMS 配置

type='nms', # NMS 的类型

iou\_thr=0.5), # NMS 的阈值

max\_per\_img=100, # 每张图像的最大检测次数

mask\_thr\_binary=0.5)) # mask 预处的阈值

dataset\_type = 'CocoDataset' # 数据集类型，这将被用来定义数据集。

data\_root = 'data/coco/' # 数据的根路径。

img\_norm\_cfg = dict( #图像归一化配置，用来归一化输入的图像。

mean=[123.675, 116.28, 103.53], # 预训练里用于预训练主干网络模型的平均值。

std=[58.395, 57.12, 57.375], # 预训练里用于预训练主干网络模型的标准差。

to\_rgb=True

) # 预训练里用于预训练主干网络的图像的通道顺序。

train\_pipeline = [ # 训练流程

dict(type='LoadImageFromFile'), # 第 1 个流程，从文件路径里加载图像。

dict(

type='LoadAnnotations', # 第 2 个流程，对于当前图像，加载它的注释信息。

with\_bbox=True, # 是否使用标注框(bounding box)， 目标检测需要设置为 True。

with\_mask=True, # 是否使用 instance mask，实例分割需要设置为 True。

poly2mask=False), # 是否将 polygon mask 转化为 instance mask, 设置为 False 以加速和节省内存。

dict(

type='Resize', # 变化图像和其注释大小的数据增广的流程。

img\_scale=(1333, 800), # 图像的最大规模。

keep\_ratio=True

), # 是否保持图像的长宽比。

dict(

type='RandomFlip', # 翻转图像和其注释大小的数据增广的流程。

flip\_ratio=0.5), # 翻转图像的概率。

dict(

type='Normalize', # 归一化当前图像的数据增广的流程。

mean=[123.675, 116.28, 103.53], # 这些键与 img\_norm\_cfg 一致，因为 img\_norm\_cfg 被

std=[58.395, 57.12, 57.375], # 用作参数。

to\_rgb=True),

dict(

type='Pad', # 填充当前图像到指定大小的数据增广的流程。

size\_divisor=32), # 填充图像可以被当前值整除。

dict(type='DefaultFormatBundle'), # 流程里收集数据的默认格式捆。

dict(

type='Collect', # 决定数据中哪些键应该传递给检测器的流程

keys=['img', 'gt\_bboxes', 'gt\_labels', 'gt\_masks'])

]

test\_pipeline = [

dict(type='LoadImageFromFile'), # 第 1 个流程，从文件路径里加载图像。

dict(

type='MultiScaleFlipAug', # 封装测试时数据增广(test time augmentations)。

img\_scale=(1333, 800), # 决定测试时可改变图像的最大规模。用于改变图像大小的流程。

flip=False, # 测试时是否翻转图像。

transforms=[

dict(type='Resize', # 使用改变图像大小的数据增广。

keep\_ratio=True), # 是否保持宽和高的比例，这里的图像比例设置将覆盖上面的图像规模大小的设置。

dict(type='RandomFlip'), # 考虑到 RandomFlip 已经被添加到流程里，当 flip=False 时它将不被使用。

dict(

type='Normalize', # 归一化配置项，值来自 img\_norm\_cfg。

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(

type='Pad', # 将配置传递给可被 32 整除的图像。

size\_divisor=32),

dict(

type='ImageToTensor', # 将图像转为张量

keys=['img']),

dict(

type='Collect', # 收集测试时必须的键的收集流程。

keys=['img'])

])

]

data = dict(

samples\_per\_gpu=2, # 单个 GPU 的 Batch size

workers\_per\_gpu=2, # 单个 GPU 分配的数据加载线程数

train=dict( # 训练数据集配置

type='CocoDataset', # 数据集的类别, 更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/datasets/coco.py#L19。

ann\_file='data/coco/annotations/instances\_train2017.json', # 注释文件路径

img\_prefix='data/coco/train2017/', # 图片路径前缀

pipeline=[ # 流程, 这是由之前创建的 train\_pipeline 传递的。

dict(type='LoadImageFromFile'),

dict(

type='LoadAnnotations',

with\_bbox=True,

with\_mask=True,

poly2mask=False),

dict(type='Resize', img\_scale=(1333, 800), keep\_ratio=True),

dict(type='RandomFlip', flip\_ratio=0.5),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='DefaultFormatBundle'),

dict(

type='Collect',

keys=['img', 'gt\_bboxes', 'gt\_labels', 'gt\_masks'])

]),

val=dict( # 验证数据集的配置

type='CocoDataset',

ann\_file='data/coco/annotations/instances\_val2017.json',

img\_prefix='data/coco/val2017/',

pipeline=[ # 由之前创建的 test\_pipeline 传递的流程。

dict(type='LoadImageFromFile'),

dict(

type='MultiScaleFlipAug',

img\_scale=(1333, 800),

flip=False,

transforms=[

dict(type='Resize', keep\_ratio=True),

dict(type='RandomFlip'),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='ImageToTensor', keys=['img']),

dict(type='Collect', keys=['img'])

])

]),

test=dict( # 测试数据集配置，修改测试开发/测试(test-dev/test)提交的 ann\_file

type='CocoDataset',

ann\_file='data/coco/annotations/instances\_val2017.json',

img\_prefix='data/coco/val2017/',

pipeline=[ # 由之前创建的 test\_pipeline 传递的流程。

dict(type='LoadImageFromFile'),

dict(

type='MultiScaleFlipAug',

img\_scale=(1333, 800),

flip=False,

transforms=[

dict(type='Resize', keep\_ratio=True),

dict(type='RandomFlip'),

dict(

type='Normalize',

mean=[123.675, 116.28, 103.53],

std=[58.395, 57.12, 57.375],

to\_rgb=True),

dict(type='Pad', size\_divisor=32),

dict(type='ImageToTensor', keys=['img']),

dict(type='Collect', keys=['img'])

])

],

samples\_per\_gpu=2 # 单个 GPU 测试时的 Batch size

))

evaluation = dict( # evaluation hook 的配置，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/evaluation/eval\_hooks.py#L7。

interval=1, # 验证的间隔。

metric=['bbox', 'segm']) # 验证期间使用的指标。

optimizer = dict( # 用于构建优化器的配置文件。支持 PyTorch 中的所有优化器，同时它们的参数与 PyTorch 里的优化器参数一致。

type='SGD', # 优化器种类，更多细节可参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/optimizer/default\_constructor.py#L13。

lr=0.02, # 优化器的学习率，参数的使用细节请参照对应的 PyTorch 文档。

momentum=0.9, # 动量(Momentum)

weight\_decay=0.0001) # SGD 的衰减权重(weight decay)。

optimizer\_config = dict( # optimizer hook 的配置文件，执行细节请参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/optimizer.py#L8。

grad\_clip=None) # 大多数方法不使用梯度限制(grad\_clip)。

lr\_config = dict( # 学习率调整配置，用于注册 LrUpdater hook。

policy='step', # 调度流程(scheduler)的策略，也支持 CosineAnnealing, Cyclic, 等。请从 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr\_updater.py#L9 参考 LrUpdater 的细节。

warmup='linear', # 预热(warmup)策略，也支持 `exp` 和 `constant`。

warmup\_iters=500, # 预热的迭代次数

warmup\_ratio=

0.001, # 用于热身的起始学习率的比率

step=[8, 11]) # 衰减学习率的起止回合数

runner = dict(

type='EpochBasedRunner', # 将使用的 runner 的类别 (例如 IterBasedRunner 或 EpochBasedRunner)。

max\_epochs=12) # runner 总回合数， 对于 IterBasedRunner 使用 `max\_iters`

checkpoint\_config = dict( # Checkpoint hook 的配置文件。执行时请参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py。

interval=1) # 保存的间隔是 1。

log\_config = dict( # register logger hook 的配置文件。

interval=50, # 打印日志的间隔

hooks=[

# dict(type='TensorboardLoggerHook') # 同样支持 Tensorboard 日志

dict(type='TextLoggerHook')

]) # 用于记录训练过程的记录器(logger)。

dist\_params = dict(backend='nccl') # 用于设置分布式训练的参数，端口也同样可被设置。

log\_level = 'INFO' # 日志的级别。

load\_from = None # 从一个给定路径里加载模型作为预训练模型，它并不会消耗训练时间。

resume\_from = None # 从给定路径里恢复检查点(checkpoints)，训练模式将从检查点保存的轮次开始恢复训练。

workflow = [('train', 1)] # runner 的工作流程，[('train', 1)] 表示只有一个工作流且工作流仅执行一次。根据 total\_epochs 工作流训练 12个回合。

work\_dir = 'work\_dir' # 用于保存当前实验的模型检查点和日志的目录文件地址。

三：测试结果

训练过程的可视化：

python tools/ analysis\_tools /analyze\_logs.py plot\_curve zuixin/20200805\_020309.log.json --keys acc loss\_cls loss\_bbox loss\_mask --out out.pdf

用权重进行检测：

python tools/test.py xianjin\_workdir/mask\_rcnn\_r101\_fpn\_2x\_coco.py zuixin/epoch\_12.pth --show