mmdetection2.6的config文件

mmdetection合并了模块化与继承设计的思想来构成config系统，利用这种系统可以方便的执行多样化的实验。如果

想要检查配置文件，可以运行如下的代码来查看完整的配置文件：

```
python tools/print_config.py /PATH/TO/CONFIG
```

1. 配置文件的结构

在`config/_base_`中存在4中基本的组件类型，分别为：dataset(数据集), model(模型），schedule（学习率调整的粗略），default_runtime（默认运行时设置）。使用Faster R-CNN，Mask R-CNN，Cascade R-CNN，RPN，SSD中的一个可以轻松构造许多方法。来自`__base__`中的组件称为*promitive*.

对于同一文件夹下的所有配置，建议仅具有一个*原始（promitive）*配置。所有其他配置应从*原始（promitive）*配置继承。这样，继承级别的最大值为3。

便于理解，构建爱你模型推荐从现有的方法中继承。例如，如果有一些修改是基于faster rcnn的， 那么使用者可能通过指定`_base_ = ../faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py`首先继承基本的faster rcnn架构， 然后修改必要的config文件。

如果构建了一种没有基于任何架构的完全新方法，可以在configs文件夹下创建一个新的文件夹。

2. config文件命名风格

利用下面的文件命名方法命名：

```
{model}_[model setting]_{backbone}_{neck}_[norm setting]_[misc]_[gpu x
batch_per_gpu]_{schedule}_{dataset}
```

`{xxx}`是必填字段，`[yyy]`是可选字段。

- `{model}`：模型的类型， 例如`faster_rcnn`，`mask_rcnn`等。
- `[model setting]`：某些模型的特定设置，例如`without_semantic`for`htc`，`moment`for`reppoints`等。
- `{backbone}`：骨干类型，例如`r50`（ResNet-50），`x101`（ResNeXt-101）。
- `{neck}`：neck的类型， 例如`fpn`，`pafpn`，`nasfpn`，`c4`。
- `[norm_setting]`：如果不特别指定，默认使用`bn`，否则使用其他标准图层类型可以是`gn`（Group Normalization），`syncbn`（Synchronized Batch Normalization）。`gn-head`/`gn-neck`表示GN仅应用于头部/颈部，而`gn-all`表示GN应用于整个模型，例如，骨架，颈部，头部。
- `[misc]`：其他的设置/模型的插件，例如`dconv`，`gcb`，`attention`，`albu`，`mstrain`。
- `[gpu x batch_per_gpu]`：GPU数目和每个GPU的样本数目， 默认使用`8x2`。

- `{schedule}`：训练进度表，可选的有`1x`，`2x`，`20e`，`1x`和`2x`默认12与24 epoch。`20e`在级联模型中采用，表示20个epoch。对于`1x`/`2x`，初始学习率在第8/16和11/22个epoch衰减10倍。对于`20e`，初始学习率在第16和19个epoch衰减10倍。
- `{dataset}`：数据集，例如`coco`，`cityscapes`，`voc_0712`，`wider_face`。

3. mask rcnn的config文件

下边简单介绍maskrcnn + resnet50 + fpn的config文件介绍，文件存在于`configs/__base__/models/mask_rcnn_r50_fpn.py`。

```
model = dict(
    type='MaskRCNN',  # 检测器的名名称
    pretrained=
    'torchvision://resnet50',  # 加载的ImageNet预训练权重。
    backbone=dict(  # backbone的cofig文件
        type='ResNet',  # backbone的类型，更多的细节参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/backbones/resnet.py#L288。
        depth=50,  # backbone的深度，通常情况下resnet采用50， resnext采用101
        num_stages=4,  # backbone的段的数目.
        out_indices=(0, 1, 2, 3),  # 每个stage所产生的feature map的索引
        frozen_stages=1,  # 冻结第一个stage不训练
        norm_cfg=dict(  # 正则化层的config参数
            type='BN',  # 正则化层
            requires_grad=True),  # 是否训练BN中的gamma与beta
        norm_eval=True,  # 是否冻结BN中的统计量
        style='pytorch'),  # backbone的风格，可选的有pytorch与caffe，'pytorch' 参数表示stride为2的过程在3x3的卷积层中，而 'caffe' 表示stride 2的层在1x1的卷积层中）。
    neck=dict(
        type='FPN',  # 检测器的neck是FPN. 支持 'NASFPN', 'PAFPN'等. 更多细节可参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/necks/fpn.py#L10.
        in_channels=[256, 512, 1024, 2048],  # FPN的输入channels，这与neck的输出通道一致
        out_channels=256,  #  金字塔特征映射的每个级别的输出通道
        num_outs=5),  # 输出尺度的数目
    rpn_head=dict(
        type='RPNHead',  # RPN head is 'RPNHead', 框架同时支持 'GARPNHead'等. 更多细节参考https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/dense_heads/rpn_head.py#L12
        in_channels=256,  # 每个输入特征图的通道数， 这与neck的输出通道数一致。
        feat_channels=256,  # 卷积层的特征通道
        anchor_generator=dict(  # 生成anchor的配置
            type='AnchorGenerator',  # 大多数方法使用AnchorGenerator，而ssd检测器采用`SSDAnchorGenerator`. 更多细节参考https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/anchor/anchor_generator.py#L10
            scales=[8],  # anchor的基本尺度，一个featuremap的anchor的面积通过公式计算: scale * base_sizes
            ratios=[0.5, 1.0, 2.0],  # anchor的宽高比
            strides=[4, 8, 16, 32, 64]),  # anchor生成器的步长，这与FPN特征的步长一致。如果base_size没有设置，这个步长将会作为base_size.
        bbox_coder=dict(  # 边界框编码器的配置，在训练与测试时编码与解码边界框
```

```
            type='DeltaXYWHBBoxCoder',  # 边界框编码器的配置. 大多数的方法采
用'DeltaXYWHBBoxCoder'. 更多的细节参考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],  # 目标的均值，这个值用来编码与解码边界
框
            target_stds=[1.0, 1.0, 1.0, 1.0]),  # 目标的方差，这个值用来编码与解码边界
框
        loss_cls=dict(  # 分类分支的loss的配置
            type='CrossEntropyLoss',  # 分类分支loss的类型，除了交叉商损失外，也支持
focal loss.
            use_sigmoid=True,  # RPN曾通常为二分类任务，用来区分前景与背景，采用sigmoid激
活函数
            loss_weight=1.0),  # 分类分支loss的权重
        loss_bbox=dict(  # 回归分支lossfunction的config配置
            type='L1Loss',  # 回归损失的类型，除了l1 loss之外，也支持smooth l1 loss与各
种IOU loss. 更多的细节参考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封装了级联检测器或者两部法的第二个stage
        type='StandardRoIHead',  # ROIHead的类型.更多的细节参考
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config.
            type='SingleRoIExtractor',  # ROI特征提取其的类型,大多数方法采用
SingleRoIExtractor. 更多的细节参考https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/models/roi_heads/roi_extractors/single_level.
py#L10
            roi_layer=dict(  # ROI层的config
                type='RoIAlign',  # ROIAlign的类型，同时也支持DeformRoIPoolingPack
and ModulatedDeformRoIPoolingPack. Refer to https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/ops/roi_align/roi_align.py#L79 for details.
                output_size=7,  # featuremap的输出尺寸.
                sampling_ratio=0),  # 当提取ROI特征时的采样率， 如果采样率为0， 表示自适
应率
            out_channels=256,  # 提取得到的特征的输出通道.
            featmap_strides=[4, 8, 16, 32]),  # 多尺度特征图的步长， 应该与backbone的
输出一致
        bbox_head=dict(  # 在ROI head中 的box headconfig
            type='Shared2FCBBoxHead',  # box head的步长, Refer to
https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/models/roi_heads/bbox_heads/convfc_bbox_head.
py#L177 for implementation details.
            in_channels=256,  #bbox的输入通道，应该与roi特征提取器的输出一致.
            fc_out_channels=1024,  # 全链接层的输出通道.
            roi_feat_size=7,  # roi特征的尺寸
            num_classes=80,  # 分类的类别数目
            bbox_coder=dict(  # 在第二个stage中的bbox编码器的config
                type='DeltaXYWHBBoxCoder',  # bbox编码器的类型. 大多数方法采
用'DeltaXYWHBBoxCoder' .
                target_means=[0.0, 0.0, 0.0, 0.0],  # Means used to encode and
decode box
                target_stds=[0.1, 0.1, 0.2, 0.2]),  # 编解码的方差，这个方差更小，因为
此时的bbox已经西那个对比较准确. [0.1, 0.1, 0.2, 0.2] is a conventional setting.
            reg_class_agnostic=False,  # 是否回归分支是不知道类别的.
            loss_cls=dict(  # 分类分支的lossfunction
                type='CrossEntropyLoss',
                use_sigmoid=False,  # 是否采用sigmoid
                loss_weight=1.0),  # 分类loss的权重
```

```
            loss_bbox=dict(
                type='L1Loss',
                loss_weight=1.0)),
        mask_roi_extractor=dict(  # bbox回归的特征提取器.
            type='SingleRoIExtractor',  # ROI特征提取器的类型, 大多数方法采用
SingleRoIExtractor.
            roi_layer=dict(  # 用于实例分割的特征提取ROI层
                type='RoIAlign',  # Type of RoI Layer, DeformRoIPoolingPack and
ModulatedDeformRoIPoolingPack are also supported
                output_size=14,  # The output size of feature maps.
                sampling_ratio=0),  # Sampling ratio when extracting the RoI
features.
            out_channels=256,  # Output channels of the extracted feature.
            featmap_strides=[4, 8, 16, 32]),  # Strides of multi-scale feature
maps.
        mask_head=dict(  # mask预测head
            type='FCNMaskHead',  # Type of mask head, refer to
https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/models/roi_heads/mask_heads/fcn_mask_head.py#
L21 for implementation details.
            num_convs=4,  # mask head中卷积层的数目.
            in_channels=256,  # 输入通道数目,  应该与mask特征提取层的输出通道数目一致.
            conv_out_channels=256,  # 卷积层的输出通道数目.
            num_classes=80,  # 分割的类别数目.
            loss_mask=dict(  # Config of loss function for the mask branch.
                type='CrossEntropyLoss',  # Type of loss used for segmentation
                use_mask=True,  # 是否仅仅训练mask正确的位置
                loss_weight=1.0))))  # Loss weight of mask branch.


train_cfg = dict(  #rpn与rcnn的训练超参数设置。
    rpn=dict(  # rpn的训练config配置
        assigner=dict(  # assigner的config
            type='MaxIoUAssigner',  # assigner的类型, 大多数检测器采用MaxIoUAssigner.
Refer to https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/core/bbox/assigners/max_iou_assigner.py#L10
for more details.
            pos_iou_thr=0.7,  # IoU >= threshold 0.7 will be taken as positive
samples
            neg_iou_thr=0.3,  # IoU < threshold 0.3 will be taken as negative
samples
            min_pos_iou=0.3,  # The minimal IoU threshold to take boxes as
positive samples
            match_low_quality=True,  # Whether to match the boxes under low
quality (see API doc for more details).
            ignore_iof_thr=-1),  # IoF threshold for ignoring bboxes
        sampler=dict(  # Config of positive/negative sampler
            type='RandomSampler',  # Tsampler的类型, 也支持其他的PseudoSampler and
other samplers. Refer to https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/core/bbox/samplers/random_sampler.py#L8 for
implementation details.
            num=256,  # samples的数目
            pos_fraction=0.5,  # 所有的样本中正样本的比例
            neg_pos_ub=-1,  # The upper bound of negative samples based on the
number of positive samples.
            add_gt_as_proposals=False),  # 是否在采样后将GT作为正阳本加入samples
        allowed_border=-1,  # The border allowed after padding for valid anchors.
        pos_weight=-1,  # 训练过程中正样本的去
```

```
        debug=False),  # 是否设置debug模式
    rpn_proposal=dict(  # 训练过程中生成proposal的config。
        nms_across_levels=False,  # 是否跨层次执行nms
        nms_pre=2000,  # nms之前的proposal的数目
        nms_post=1000,  # nms保留的样本的数目
        max_num=1000,  # nms之后使用的proposal数目
        nms_thr=0.7,  # nms中使用的阈值
        min_bbox_size=0),  # 最小的box的尺寸
    rcnn=dict(  # The config for the roi heads.
        assigner=dict(  # Config of assigner for second stage, this is different
for that in rpn
            type='MaxIoUAssigner',  # Type of assigner, 现在所有的roi_head均使用
MaxIoUAssigner. Refer to https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/core/bbox/assigners/max_iou_assigner.py#L10
for more details.
            pos_iou_thr=0.5,  # IoU >= threshold 0.5 will be taken as positive
samples
            neg_iou_thr=0.5,  # IoU< threshold 0.5 will be taken as negative
samples
            min_pos_iou=0.5,  # The minimal IoU threshold to take boxes as
positive samples
            match_low_quality=False,  # Whether to match the boxes under low
quality (see API doc for more details).
            ignore_iof_thr=-1),  # IoF threshold for ignoring bboxes
        sampler=dict(
            type='RandomSampler',  # Type of sampler, PseudoSampler and other
samplers are also supported. Refer to https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/core/bbox/samplers/random_sampler.py#L8 for
implementation details.
            num=512,  # Number of samples
            pos_fraction=0.25,  # The ratio of positive samples in the total
samples.
            neg_pos_ub=-1,  # The upper bound of negative samples based on the
number of positive samples.
            add_gt_as_proposals=True
        ),  # Whether add GT as proposals after sampling.
        mask_size=28,  # Size of mask
        pos_weight=-1,  # The weight of positive samples during training.
        debug=False))  # Whether to set the debug mode


test_cfg = dict(  # Config for testing hyperparameters for rpn and rcnn
    rpn=dict(  # The config to generate proposals during testing
        nms_across_levels=False,  # Whether to do NMS for boxes across levels
        nms_pre=1000,  # The number of boxes before NMS
        nms_post=1000,  # The number of boxes to be kept by NMS
        max_num=1000,  # The number of boxes to be used after NMS
        nms_thr=0.7,  # The threshold to be used during NMS
        min_bbox_size=0),  # The allowed minimal box size
    rcnn=dict(  # The config for the roi heads.
        score_thr=0.05,  # Threshold to filter out boxes
        nms=dict(  # Config of nms in the second stage
            type='nms',  # Type of nms
            iou_thr=0.5),  # NMS threshold
        max_per_img=100,  # Max number of detections of each image
        mask_thr_binary=0.5))  # Threshold of mask prediction
```

```
dataset_type = 'CocoDataset'  # 数据集的类型
data_root = 'data/coco/'  # 数据的根目录
img_norm_cfg = dict(  # 图像正则化config
    mean=[123.675, 116.28, 103.53],  # Mean values used to pre-training the pre-trained backbone models
    std=[58.395, 57.12, 57.375],  # Standard variance used to pre-training the pre-trained backbone models
    to_rgb=True
)  # The channel orders of image used to pre-training the pre-trained backbone models


train_pipeline = [  # Training pipeline
    dict(type='LoadImageFromFile'),  # First pipeline to load images from file path
    dict(
        type='LoadAnnotations',  # Second pipeline to load annotations for current image
        with_bbox=True,  # Whether to use bounding box, True for detection
        with_mask=True,  # Whether to use instance mask, True for instance segmentation
        poly2mask=False),  # Whether to convert the polygon mask to instance mask, set False for acceleration and to save memory
    dict(
        type='Resize',  # Augmentation pipeline that resize the images and their annotations
        img_scale=(1333, 800),  # The largest scale of image
        keep_ratio=True
    ),  # whether to keep the ratio between height and width.
    dict(
        type='RandomFlip',  # Augmentation pipeline that flip the images and their annotations
        flip_ratio=0.5),  # The ratio or probability to flip
    dict(
        type='Normalize',  # Augmentation pipeline that normalize the input images
        mean=[123.675, 116.28, 103.53],  # These keys are the same of img_norm_cfg since the
        std=[58.395, 57.12, 57.375],  # keys of img_norm_cfg are used here as arguments
        to_rgb=True),
    dict(
        type='Pad',  # Padding config
        size_divisor=32),  # The number the padded images should be divisible
    dict(type='DefaultFormatBundle'),  # Default format bundle to gather data in the pipeline
    dict(
        type='Collect',  # Pipeline that decides which keys in the data should be passed to the detector
        keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks'])
]


test_pipeline = [
    dict(type='LoadImageFromFile'),  # First pipeline to load images from file path
    dict(
```

```
        type='MultiScaleFlipAug',  # An encapsulation that encapsulates the
testing augmentations
        img_scale=(1333, 800),  # Decides the largest scale for testing, used for
the Resize pipeline
        flip=False,  # Whether to flip images during testing
        transforms=[
            dict(type='Resize',  # Use resize augmentation
                 keep_ratio=True),  # Whether to keep the ratio between height
and width, the img_scale set here will be supressed by the img_scale set above.
            dict(type='RandomFlip'),  # Thought RandomFlip is added in pipeline,
it is not used because flip=False
            dict(
                type='Normalize',  # Normalization config, the values are from
img_norm_cfg
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(
                type='Pad',  # Padding config to pad images divisable by 32.
                size_divisor=32),
            dict(
                type='ImageToTensor',  # convert image to tensor
                keys=['img']),
            dict(
                type='Collect',  # Collect pipeline that collect necessary keys
for testing.
                keys=['img'])
        ])
]



data = dict(
    samples_per_gpu=2,  # 每个gpu上的images， 这里乘以gpu的数目即为batch size
    workers_per_gpu=2,  # Worker to pre-fetch data for each single GPU
    train=dict(  # Train dataset config
        type='CocoDataset',  # Type of dataset, refer to https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/datasets/coco.py#L19 for details.
        ann_file='data/coco/annotations/instances_train2017.json',  # Path of
annotation file
        img_prefix='data/coco/train2017/',  # Prefix of image path
        pipeline=[  # pipeline, this is passed by the train_pipeline created
before.
            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(  # Validation dataset config
        type='CocoDataset',
        ann_file='data/coco/annotations/instances_val2017.json',
        img_prefix='data/coco/val2017/',
        pipeline=[  # Pipeline is passed by test_pipeline created before
            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(  # Test dataset config, modify the ann_file for test-dev/test
submission
        type='CocoDataset',
        ann_file='data/coco/annotations/instances_val2017.json',
        img_prefix='data/coco/val2017/',
        pipeline=[  # Pipeline is passed by test_pipeline created before
            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  # Batch size of a single GPU used in testing
        ))

evaluation = dict(  # The config to build the evaluation hook, refer to
https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/core/evaluation/eval_hooks.py#L7 for more
details.
```

```
    interval=1,  #每1个epoch评估一次，这里也可以修改
    metric=['bbox', 'segm'])  # 评估标准
optimizer = dict(  # Config used to build optimizer, support all the optimizers
in PyTorch whose arguments are also the same as those in PyTorch
    type='SGD',  # Type of optimizers, refer to https://github.com/open-
mmlab/mmdetection/blob/master/mmdet/core/optimizer/default_constructor.py#L13 for
more details
    lr=0.02,  # Learning rate of optimizers, see detail usages of the parameters
in the documentaion of PyTorch
    momentum=0.9,  # Momentum
    weight_decay=0.0001)  # Weight decay of SGD
optimizer_config = dict(  # Config used to build the optimizer hook, refer to
https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/optimizer.py#L8
for implementation details.
    grad_clip=None)  # Most of the methods do not use gradient clip
lr_config = dict(  # Learning rate scheduler config used to register LrUpdater
hook
    policy='step',  # The policy of scheduler, also support CosineAnnealing,
Cyclic, etc. Refer to details of supported LrUpdater from
https://github.com/open-
mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9.
    warmup='linear',  # The warmup policy, also support `exp` and `constant`.
    warmup_iters=500,  # The number of iterations for warmup
    warmup_ratio=
    0.001,  # The ratio of the starting learning rate used for warmup
    step=[8, 11])  # Steps to decay the learning rate
total_epochs = 12  # Total epochs to train the model
checkpoint_config = dict(  # Config to set the checkpoint hook, Refer to
https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
for implementation.
    interval=1)  # The save interval is 1
log_config = dict(  # config to register logger hook
    interval=50,  # Interval to print the log
    hooks=[
        # dict(type='TensorboardLoggerHook')  # The Tensorboard logger is also
supported
        dict(type='TextLoggerHook')
    ])  # The logger used to record the training process.
dist_params = dict(backend='nccl')  # Parameters to setup distributed training,
the port can also be set.
log_level = 'INFO'  # The level of logging.
load_from = None  # 加载预训练模型权重，这里不是resume训练采用
resume_from = None  # 这里resume训练采用
workflow = [('train', 1)]  # Workflow for runner. [('train', 1)] means there is
only one workflow and the workflow named 'train' is executed once. The workflow
trains the model by 12 epochs according to the total_epochs.
work_dir = 'work_dir'  # 这个路径用来存储模型与日志文件
```

4. 一些问题：

在继承关系中有时可以设置 `_delete_=True` 来忽视在base config中的一些键。在mmcv中可以看到具体的一些指导细节。

在mmdetection中，在mask rcnn的config文件中来修改backbone。

```
model = dict(
    type='MaskRCNN',
    pretrained='torchvision://resnet50',
    backbone=dict(
        type='ResNet',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        frozen_stages=1,
        norm_cfg=dict(type='BN', requires_grad=True),
        norm_eval=True,
        style='pytorch'),
    neck=dict(...),
    rpn_head=dict(...),
    roi_head=dict(...))
```

`ResNet` and `HRNet` 使用不同的键来构建

```
_base_ = '../mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py'
model = dict(
    pretrained='open-mmlab://msra/hrnetv2_w32',
    backbone=dict(
        _delete_=True,
        type='HRNet',
        extra=dict(
            stage1=dict(
                num_modules=1,
                num_branches=1,
                block='BOTTLENECK',
                num_blocks=(4, ),
                num_channels=(64, )),
            stage2=dict(
                num_modules=1,
                num_branches=2,
                block='BASIC',
                num_blocks=(4, 4),
                num_channels=(32, 64)),
            stage3=dict(
                num_modules=4,
                num_branches=3,
                block='BASIC',
                num_blocks=(4, 4, 4),
                num_channels=(32, 64, 128)),
            stage4=dict(
                num_modules=3,
                num_branches=4,
                block='BASIC',
                num_blocks=(4, 4, 4, 4),
                num_channels=(32, 64, 128, 256)))),
    neck=dict(...))
```

`_delete_=True` 在 `backbone` 中利用新键来取代旧键。

4.1 在配置中使用中间变量

一些中间变量在配置文件中是非常重要的，例如，datasets中的train_pipeline与test_pipeline。值得注意的是，在子配置文件中修改中间变量时，用户需要再次将中间变量传递到相应的字段中。例如，我们想使用多尺度策略来训练Mask R-CNN。`train_pipeline`/`test_pipeline`是我们要修改的中间变量。

```
_base_ = './mask_rcnn_r50_fpn_1x_coco.py'
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(
        type='Resize',
        img_scale=[(1333, 640), (1333, 672), (1333, 704), (1333, 736),
                   (1333, 768), (1333, 800)],
        multiscale_mode="value",
        keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks']),
]
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', **img_norm_cfg),
            dict(type='Pad', size_divisor=32),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

这里我们首先定义了新的train_pipeline与test_pipeline并且将他们传给data。