基于MMdetection3D配置BEV3d检测代码重现分析

目录

[简介 3](#_Toc176269540)

[环境搭建 4](#_Toc176269541)

[MMDetection3d框架 5](#_Toc176269542)

[PointPillar 11](#_Toc176269543)

[Model定义 12](#_Toc176269544)

[检测实际代码分析 16](#_Toc176269545)

[pts\_voxel\_layer(Voxelization) 16](#_Toc176269546)

[pts\_voxel\_encoder(HardVFE) 17](#_Toc176269547)

[pts\_middle\_encoder(PointPillarsScatter) 17](#_Toc176269548)

[主干网络pts\_backbone(SECOND) 18](#_Toc176269549)

[颈部网络neck(FPN) 19](#_Toc176269550)

[检测头pts\_bbox\_head(Anchor3dHead) 19](#_Toc176269551)

[Data定义 20](#_Toc176269552)

[Schedule定义 23](#_Toc176269553)

[Runtime定义 24](#_Toc176269554)

[训练可视化 24](#_Toc176269555)

[结果可视化 24](#_Toc176269556)

[Detr 25](#_Toc176269557)

[Model定义 25](#_Toc176269558)

[检测实际代码分析 28](#_Toc176269559)

[img\_backbone (Resnet） 28](#_Toc176269560)

[Bbox\_head (Transformer） 28](#_Toc176269561)

[DetrTransformerEncoder 30](#_Toc176269562)

[DetrTransformerDecoder 30](#_Toc176269563)

[HuangarianAssigner 31](#_Toc176269564)

[Detr3d 31](#_Toc176269565)

[Model定义 33](#_Toc176269566)

[检测实际代码分析 38](#_Toc176269567)

[img\_backbone (Resnet） 38](#_Toc176269568)

[Img\_neck(FPN) 39](#_Toc176269569)

[pts\_bbox\_head(Detr3dHead) 40](#_Toc176269570)

[Detr3DTransformer 42](#_Toc176269571)

[Transformer解码器Detr3DTransformerDecoder 42](#_Toc176269572)

[交叉注意力机制Detr3DCrossAtten 43](#_Toc176269573)

[特征采样函数feature\_sampling 43](#_Toc176269574)

[Data定义 44](#_Toc176269575)

[Schedule定义 44](#_Toc176269576)

[Futr3d Lidar Only 模式 44](#_Toc176269577)

[Model定义 44](#_Toc176269578)

[检测实际代码分析 44](#_Toc176269579)

[Backbone 44](#_Toc176269580)

[Neck 44](#_Toc176269581)

[Data定义 44](#_Toc176269582)

[Schedule定义 44](#_Toc176269583)

[Runtime定义 44](#_Toc176269584)

[Futr3d 激光视觉融合模式 45](#_Toc176269585)

[Model定义 45](#_Toc176269586)

[检测实际代码分析 45](#_Toc176269587)

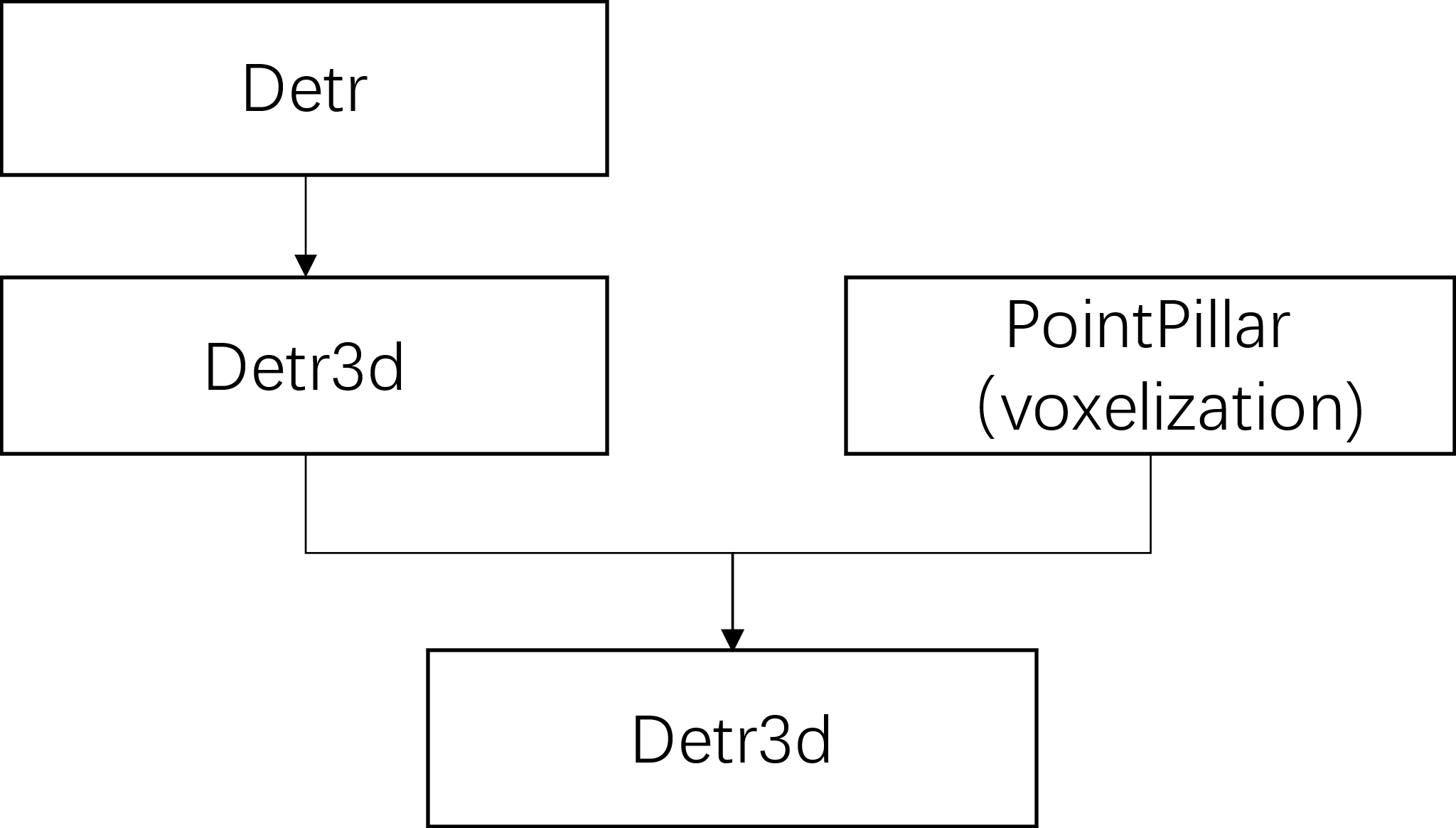
[Data定义 45](#_Toc176269588)

[Schedule定义 45](#_Toc176269589)

[Runtime定义 45](#_Toc176269590)

# 简介

本文档适用于MMdetection3d 1.0.0版本，首先以PointPillar的实现为例子，通过整个代码的解释，分析整个MMdetection3d的框架与互相调用关系，之后再以detr3d为例分析自己如何构建一个plugin代码，以及BEV3d检测的一般流程。最后说明了集合detr3d代码。



图表 1

# 环境搭建

为了运行mmdetection3d 1.0.0框架下的代码，需要进行一定的环境配置，

# MMDetection3d框架

MMdetection3d通过config配置文件，实现对整个三维检测框架的实现，配置文件一般由以下4个部分组成

1 model:核心组件，通过脚本定义了模型的参数

2 data: 定义了数据集及其调用的相关设定

3 schedule: 训练optimizer相关设定

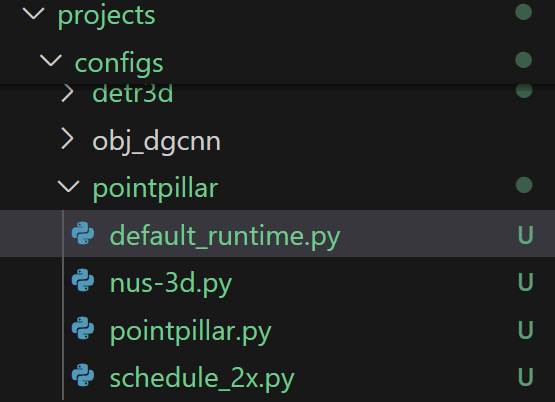
4 default\_runtime:运行环境设定，一般直接采用default\_runtime.py

图表 2展示了三个主要模块和其中包含的主要内容



图表 2 MMdetection3d配置文件主要构成部分

上文的4个部件都需要在config的配置文件中设定，虽然我们也可以将所有配置写在一个文件中，但是分成四个文件更有助于我们对框架不同部分的解耦与理解。图表 3展示了以构建pointpillar的几个配置文件，其中pointpillar设置了model，并引入了其他配置文件，作为主配置文件



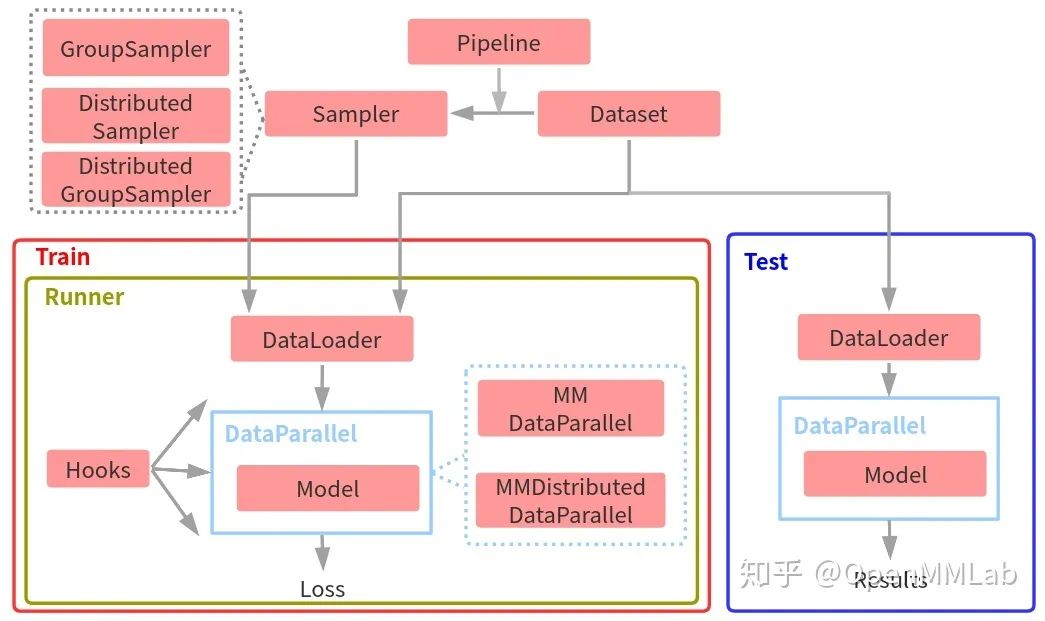
图表 3 configs文件组成

如果我们将所有的配置文件写在一起，则一般包含了下表中的几个部分

|  |
| --- |
| # 1. 模型配置(models) =========================================  model = dict(  type='FastRCNN', # 模型名称是FastRCNN  backbone=dict( # BackBone是ResNet  type='ResNet',  ...,  ),  neck=dict( # Neck是FPN  type='FPN',  ...,  ),  roi\_head=dict( # Head是StandardRoIHead  type='StandardRoIHead',  ...,  loss\_cls=dict(...), # 分类损失函数  loss\_bbox=dict(...), # 回归损失函数  ),  train\_cfg=dict( # 训练参数配置  assigner=dict(...), # BBox Assigner  sampler=dict(...), # BBox Sampler  ...  ),  test\_cfg =dict( # 测试参数配置  nms=dict(...), # NMS后处理  ...,  )  )  # 2. 数据集配置(datasets) =========================================  dataset\_type = '...' # 数据集名称  data\_root = '...' # 数据集根目录  img\_norm\_cfg = dict(...) # 图像归一化参数  train\_pipeline = [ # 训练数据处理Pipeline  ...,  dict(type='Normalize', \*\*img\_norm\_cfg),  ...  ]  test\_pipeline = [...] # 测试数据处理Pipeline  data = dict(  samples\_per\_gpu=2, # batch\_size  workers\_per\_gpu=2, # GPU数量  train=dict( # 训练集配置  type=dataset\_type,  ann\_file=data\_root + 'annotations/instances\_train2017.json', # 标注问加你  img\_prefix=data\_root + 'train2017/', # 图像前缀  pipline=trian\_pipline, # 数据预处理pipeline  ),  val=dict( # 验证集配置  ...,  pipline=test\_pipline,  ),  test=dict( # 测试集配置  ...,  pipline=test\_pipline,  )  )  # 3. 训练策略配置(schedules) =========================================  evaluation = dict(interval=1, metric='bbox')  optimizer = dict(type='SGD', lr=0.02, momentum=0.9, weight\_decay=0.0001)  optimizer\_config = dict(grad\_clip=None)  lr\_config = dict(  policy='step',  warmup='linear',  warmup\_iters=500,  warmup\_ratio=0.001,  step=[8, 11])  runner = dict(type='EpochBasedRunner', max\_epochs=12)  # 4. 运行配置(runtime) =========================================  checkpoint\_config = dict(interval=1)  log\_config = dict(interval=50, hooks=[dict(type='TextLoggerHook')])  custom\_hooks = [dict(type='NumClassCheckHook')]  dist\_params = dict(backend='nccl')  log\_level = 'INFO'  load\_from = None  resume\_from = None  workflow = [('train', 1)] |

配置文件构建完毕之后，运行mmdetection3d框架实现检测代码时，流程如图表 4所示

1. Pipeline通过dataloader遍历数据集中的数据，并根据train\_pipeline和test\_pipeline中的设定，进行数据增强等预处理
2. 读取到的批数据输入到网络中，开始训练或者测试
3. 训练模式下通过schedule的设定参数进行训练



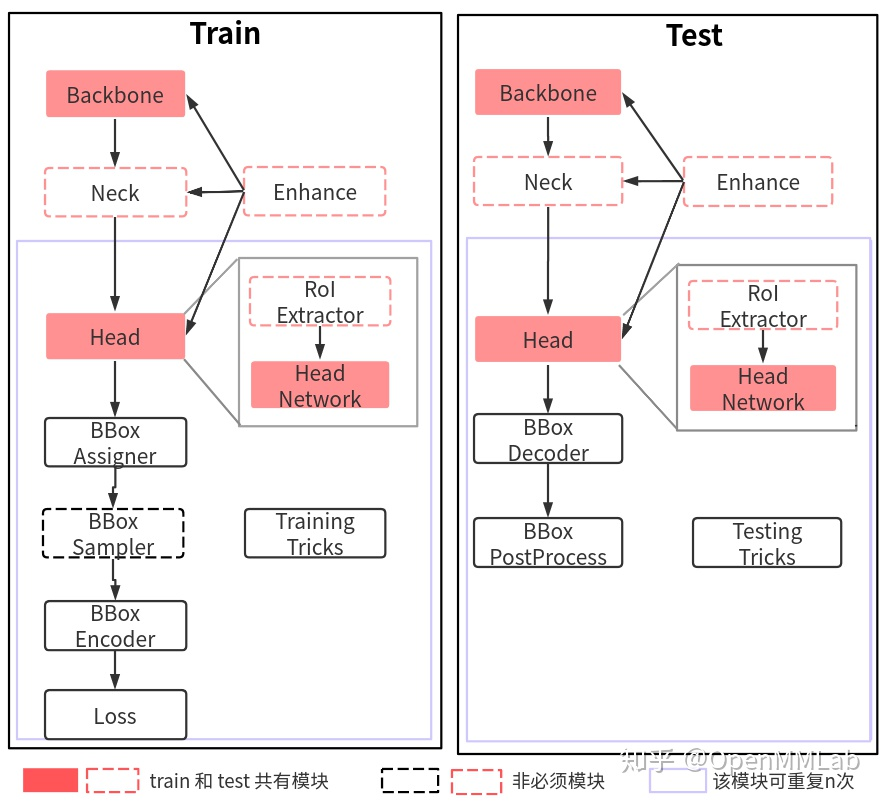
图表 4 MMdetection3d运行框架

在mmdetection3d框架中，一般图表 5所示一样，规定了一套基本框架，一般由以下几个部分组成

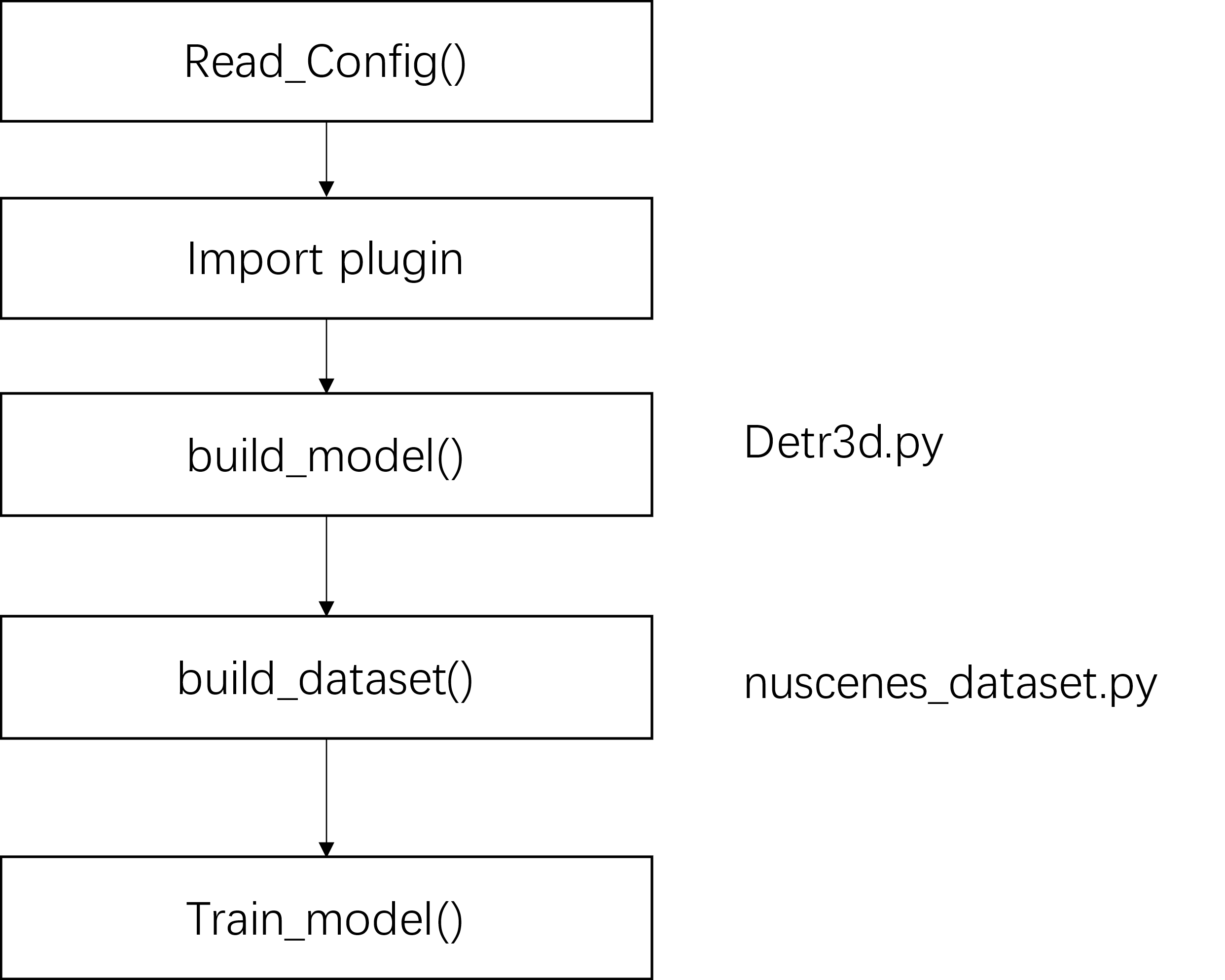
* Backbone:骨干网络，用于从输入文件中提取特征信息，比如图片常用ResNet, 点云常用SECOND
* Neck: 深度学习neck是模型的核心部分，通常由多组卷积层、池化层、全连接层和其他类型的层组成。 它的主要作用是连接深度学习head和深度学习backbone，将前者提取的特征进行非线性变换和降维，以减少计算量和提高模型性能。
* Head: 检测头，用于从neck的信息提取出检测框，常用的由Anchor3dHead锚框，detr3dHead等检测头

特别的，在训练阶段，由于需要将网络输出的BBox和真值的BBox关联后计算loss，需要以下几个部件

* BBox Assigner:
* BBox Sampler:
* BBox Encoder



图表 5 MMdetection3d训练与运行代码中各个所需组件



图表 6 tool/train.py主要代码流程

# PointPillar

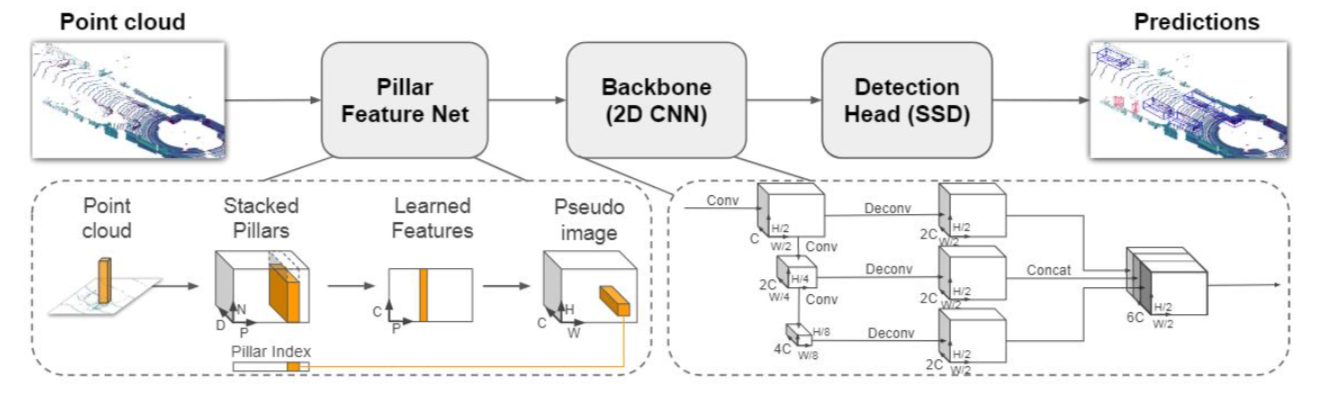
在这一部分，我们以较为简单的PointPillar为例，数据集采用Nuscenes数据集，介绍一个典型的3D检测任务的框架。

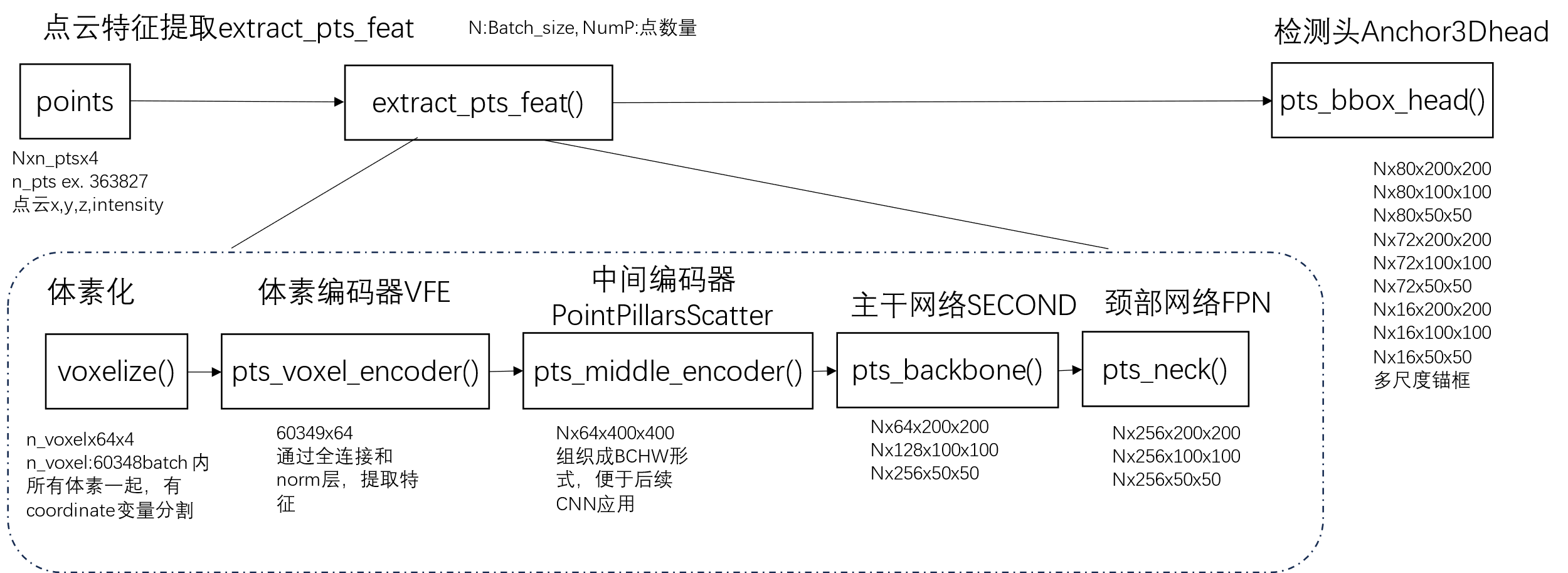
在训练过程中，我们采用命令

Python tools/train.py projects/configs/pointpillar/pointpillar.py –gpus 1

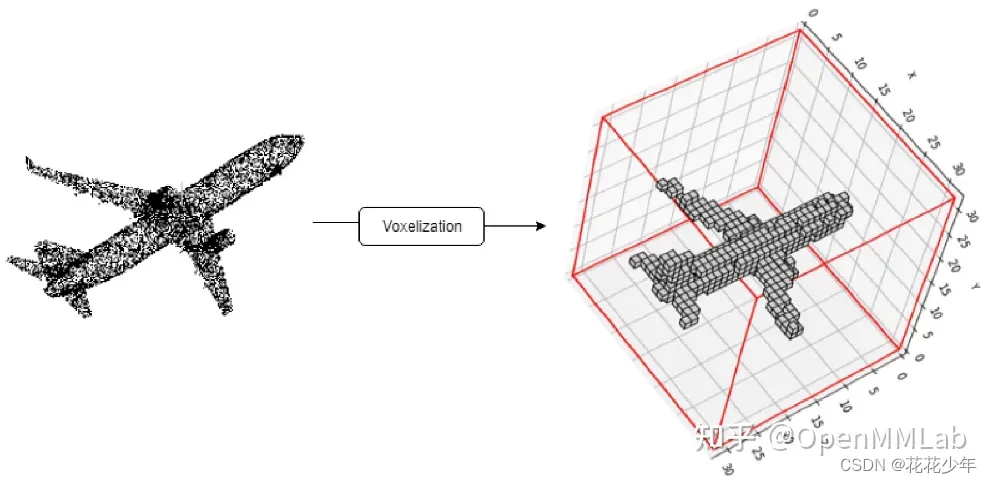
其中tools/train.py 是mmdetection3d模块的入口，而pointpillar.py指向了配置文件。

PointPillar的特点是先将点云数据经过一个Pillar Feature Network（PFN）处理为伪图片的形式，然后再用后续的基于CNN的卷积神经网络backbone提取出特征信息，最后接上检测头，从而完成





图表 7 pointpillar模型代码流程



tools/train.py里有以下几个模块

## Model定义

Model是整个config文件中，最核心的部分，也是我们主要建立的部分，它指定了模型的各个环节，从而调用不同的对象并进行初始化，从而构建网络。针对激光pointpillar检测网络，需要设定图表 8中的几个部分，

pts\_voxel\_layer:体素化层，对点云数据转换为体素

pts\_voxel\_encoder:体素编码HardVFE，通过全连接从体素中提取出特征信息

pts\_middle\_encoder:中间编码'PointPillarsScatter'，将体素特征转换为BCHW的形式

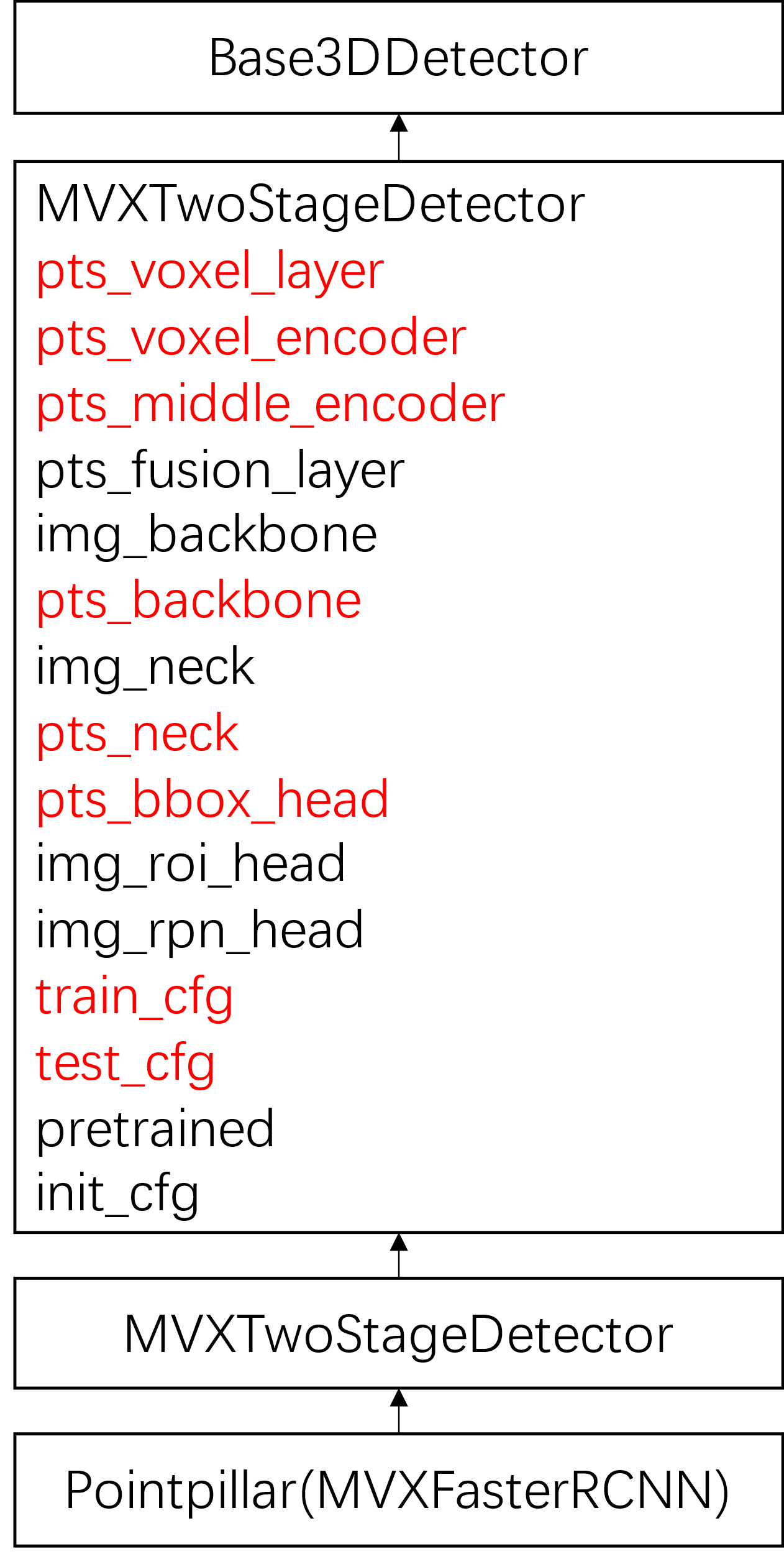
pts\_backbone:特征提取Second，通过CNN为核心的网络提取特征

pts\_neck: 颈部网络'FPN'

pts\_bbox\_head: 检测头

train\_cfg: 训练配置

test\_cfg: 测试配置



图表 8 pointpillar模型为MVXFasterRCNN，以及其需要初始化的部件

注意，在单卡模式下，需要对pointpillar.py做一定的修改，防止出现多线程错误。需要将代码内所有的norm\_cfg修改成

norm\_cfg=dict(type='BN1d'

一共有三处

norm\_cfg=dict(type='naiveSyncBN1d', eps=1e-3, momentum=0.01))改为

norm\_cfg=dict(type='BN1d', eps=1e-3, momentum=0.01))

norm\_cfg=dict(type='naiveSyncBN2d', eps=1e-3, momentum=0.01),改为

norm\_cfg=dict(type='BN2d', eps=1e-3, momentum=0.01),

norm\_cfg=dict(type='naiveSyncBN1d', eps=1e-3, momentum=0.01))改为

norm\_cfg=dict(type='BN1d', eps=1e-3, momentum=0.01))

pointpillar.py的config代码如下

|  |
| --- |
| #  \_base\_ = [      './nus-3d.py', './schedule\_2x.py',      './default\_runtime.py'  ]  # model settings  # Voxel size for voxel encoder  # Usually voxel size is changed consistently with the point cloud range  # If point cloud range is modified, do remember to change all related  # keys in the config.  voxel\_size = [0.25, 0.25, 8]  model = dict(      type='MVXFasterRCNN',      pts\_voxel\_layer=dict(          max\_num\_points=64,          point\_cloud\_range=[-50, -50, -5, 50, 50, 3],          voxel\_size=voxel\_size,          max\_voxels=(30000, 40000)),      pts\_voxel\_encoder=dict(          type='HardVFE',          in\_channels=4,          feat\_channels=[64, 64],          with\_distance=False,          voxel\_size=voxel\_size,          with\_cluster\_center=True,          with\_voxel\_center=True,          point\_cloud\_range=[-50, -50, -5, 50, 50, 3],          norm\_cfg=dict(type='BN1d', eps=1e-3, momentum=0.01)),      pts\_middle\_encoder=dict(          type='PointPillarsScatter', in\_channels=64, output\_shape=[400, 400]),      pts\_backbone=dict(          type='SECOND',          in\_channels=64,          norm\_cfg=dict(type='BN2d', eps=1e-3, momentum=0.01),          layer\_nums=[3, 5, 5],          layer\_strides=[2, 2, 2],          out\_channels=[64, 128, 256]),      pts\_neck=dict(          type='FPN',          norm\_cfg=dict(type='BN2d', eps=1e-3, momentum=0.01),          act\_cfg=dict(type='ReLU'),          in\_channels=[64, 128, 256],          out\_channels=256,          start\_level=0,          num\_outs=3),      pts\_bbox\_head=dict(          type='Anchor3DHead',          num\_classes=10,          in\_channels=256,          feat\_channels=256,          use\_direction\_classifier=True,          anchor\_generator=dict(              type='AlignedAnchor3DRangeGenerator',              ranges=[[-50, -50, -1.8, 50, 50, -1.8]],              scales=[1, 2, 4],              sizes=[                  [0.8660, 2.5981, 1.],  # 1.5/sqrt(3)                  [0.5774, 1.7321, 1.],  # 1/sqrt(3)                  [1., 1., 1.],                  [0.4, 0.4, 1],              ],              custom\_values=[0, 0],              rotations=[0, 1.57],              reshape\_out=True),          assigner\_per\_size=False,          diff\_rad\_by\_sin=True,          dir\_offset=0.7854,  # pi/4          dir\_limit\_offset=0,          bbox\_coder=dict(type='DeltaXYZWLHRBBoxCoder', code\_size=9),          loss\_cls=dict(              type='FocalLoss',              use\_sigmoid=True,              gamma=2.0,              alpha=0.25,              loss\_weight=1.0),          loss\_bbox=dict(type='SmoothL1Loss', beta=1.0 / 9.0, loss\_weight=1.0),          loss\_dir=dict(              type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=0.2)),      # model training and testing settings      train\_cfg=dict(          pts=dict(              assigner=dict(                  type='MaxIoUAssigner',                  iou\_calculator=dict(type='BboxOverlapsNearest3D'),                  pos\_iou\_thr=0.6,                  neg\_iou\_thr=0.3,                  min\_pos\_iou=0.3,                  ignore\_iof\_thr=-1),              allowed\_border=0,              code\_weight=[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.2, 0.2],              pos\_weight=-1,              debug=False)),      test\_cfg=dict(          pts=dict(              use\_rotate\_nms=True,              nms\_across\_levels=False,              nms\_pre=1000,              nms\_thr=0.2,              score\_thr=0.05,              min\_bbox\_size=0,              max\_num=500))) |

## 检测实际代码分析

Model部分实际上只是通过脚本的形式，调用了实际代码并进行了初始化，对于实际运行的model代码，需要自己写py文件，一般保存在plugin文件夹中，通过在config文件中引用，继而调用对应的各个对象，完成model的构建

Pointpillar相关文件为mmdection3d预实现，其中调用了几个类，我们针对model中各个部分关联解析

### pts\_voxel\_layer(Voxelization)

pts\_voxel\_layer 调用了Voxelization类，位置在mmdet3d/ops/voxel/voxelize.py其作用是体素化是将点云数据划分为二维平面的柱体（pillars）。每个柱体包含一组点，之后将这些点用于特征提取。因此该部分主要起一个数据转换作用，并没有科可学习的参数

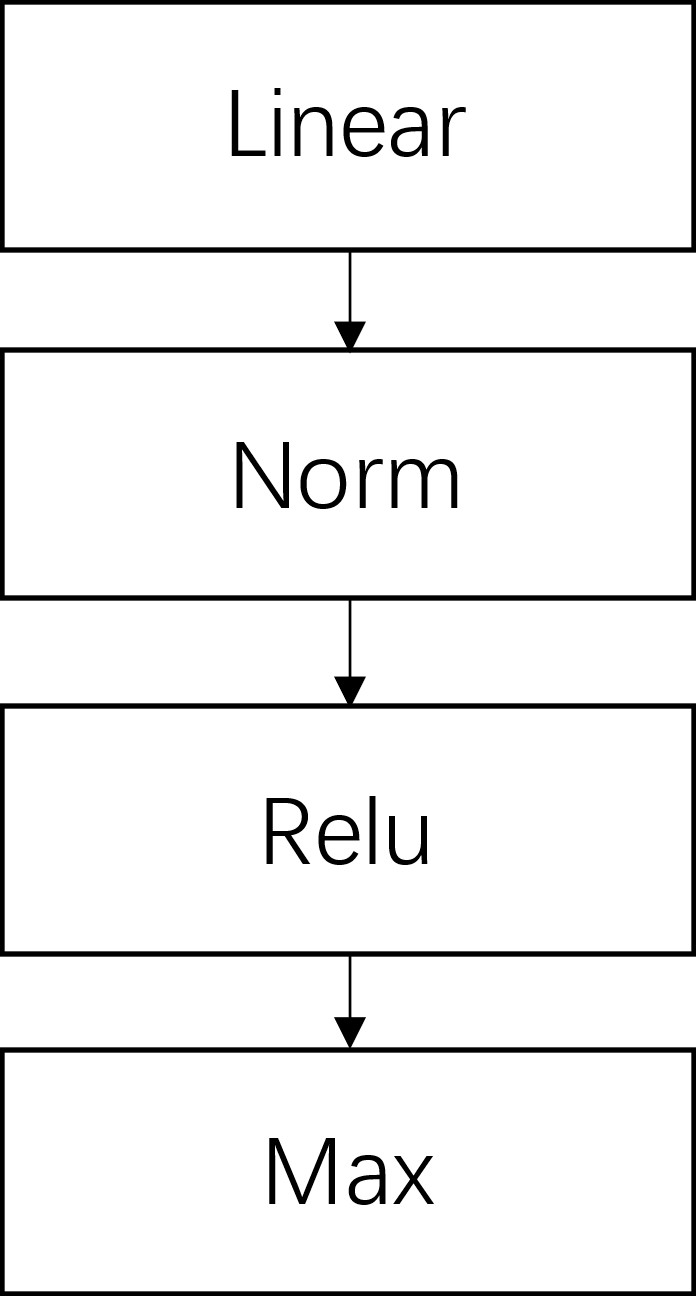
C:点云特征大小，一般为4，即x,y,z,intensity

表格 1 Voxelization forward函数输入输出数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| input | List(Tensor(float)) |  | 一个batch的点云 | 列表长为batch\_size，其中第i个元素大小为(ni,p,C) |
| voxels\_out | Tensor(float) | 返回值1 | 每个体素内的点 | (N,M,C),其中N为整个batch的总非空体素数，即N= |
| coors\_out | Tensor(int) | 返回值2 | 每个体素的batch索引和3维位置索引 | (N,4)，其中4表示batch\_id,z\_id,y\_id,x\_id |
| num\_points\_  per\_voxel\_out | Tensor(int) | 返回值3 | 每个体素内的点数 | 大小为(N,) |

### pts\_voxel\_encoder(HardVFE)

pts\_voxel\_encoder调用了HardVFE，位置在mmdet3d/models/voxel\_encoders/voxel\_encoders.py。VFE 是一种将每个柱体中的点特征编码为一个柱体全局特征的方式。



图表 9

表格 2HardVFE forward函数输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| features | Tensor(float) |  | 每个体素内的点 | (N,M,C) |
| num\_points | Tensor(int) |  | 每个体素内的点数 | (N,) |
| coors | Tensor(int) |  | 每个体素的batch索引和3维位置索引 | (N,4) |
| voxel\_feats | Tensor(float) | 返回值 | 非空体素特征 | （N,C） |

### pts\_middle\_encoder(PointPillarsScatter)

pts\_middle\_encoder采用了'PointPillarsScatter'，其作用为将维度为（N，C）的数据转化为（BCHW）形式

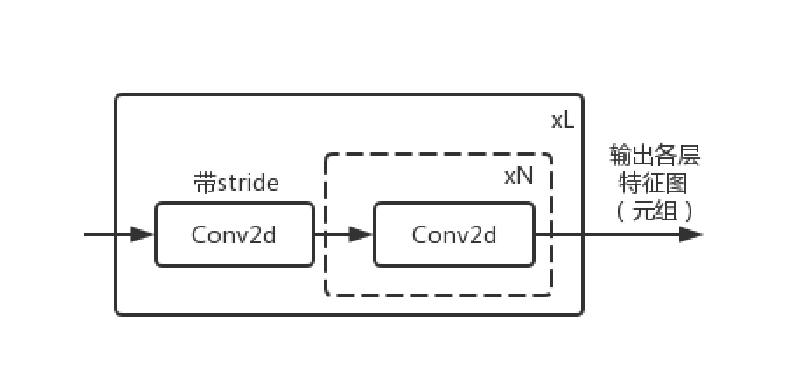
表格 3 PointPillarsScatter输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| voxel\_features | Tensor(float) |  | 非空体素特征 | (N,C),N为体素数；该参数通常是VOXEL\_ENCODERS的输出 |
| coors | Tensor(int) |  | 非空体素坐标 | (N,4)，该参数通常是体素化函数的输出 |
| Batch\_size | int |  | batch size | Batch大小 |
| Canvas | Tensor(float) | 返回值 | BEV特征图 | 大小为( B , C ′ , H , W )其中H为宽度（y\_size），W 长度（x\_size） |

### 主干网络pts\_backbone(SECOND)

pts\_backbone采用了Second网络，位置在mmdet3d/models/backbones/second.py

Second的特点是



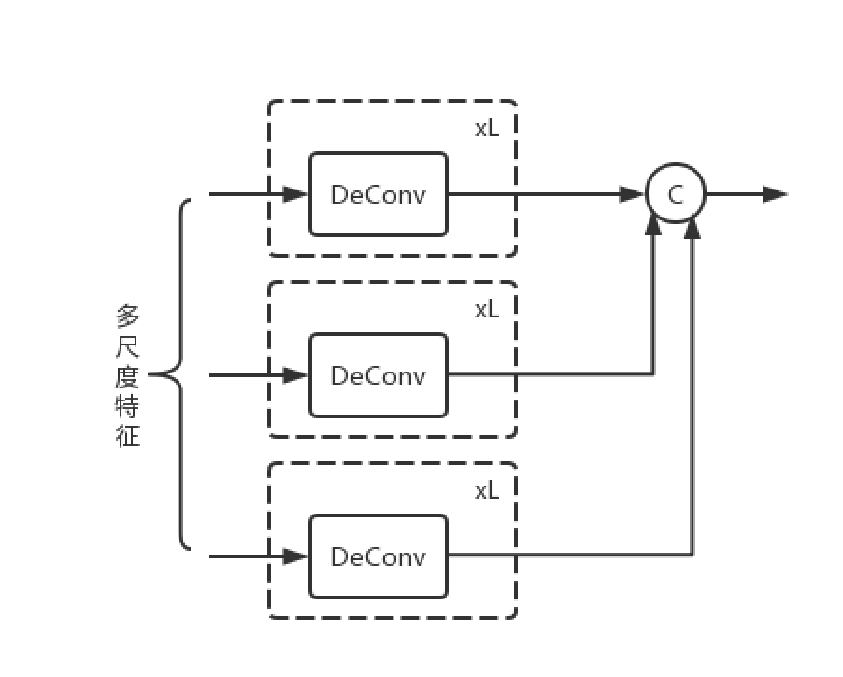
图表 10

表格 4 backbone的SECOND输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| x | Tensor(float) |  | 点云的BEV表达 | (B,C,H,W) 通常为MIDDLE\_ENCODERS的输出 |
| 返回值 | Tuple(Tensor(float)) | 返回值 | 输出为点云的多尺度BEV特征 | 列表长度等于尺度数，每个尺度特征图大小为(B,Ci,Hi,Wi) |

### 颈部网络neck(FPN)

颈部网络采用了一个FPN金字塔网络，如图表 11所示，对backbone的多尺度输出进行反卷积，并concate成单一输出



图表 11 颈部网络的多尺度FPN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| x | List(Tensor(float)) |  | 点云多尺度BEV特征 | 列表长度为尺度数，可直接使用SECOND的输出 |
| 返回值 | List(Tensor(float)) | 返回值 | 多尺度融合后的BEV特征 | 列表长度为1 |

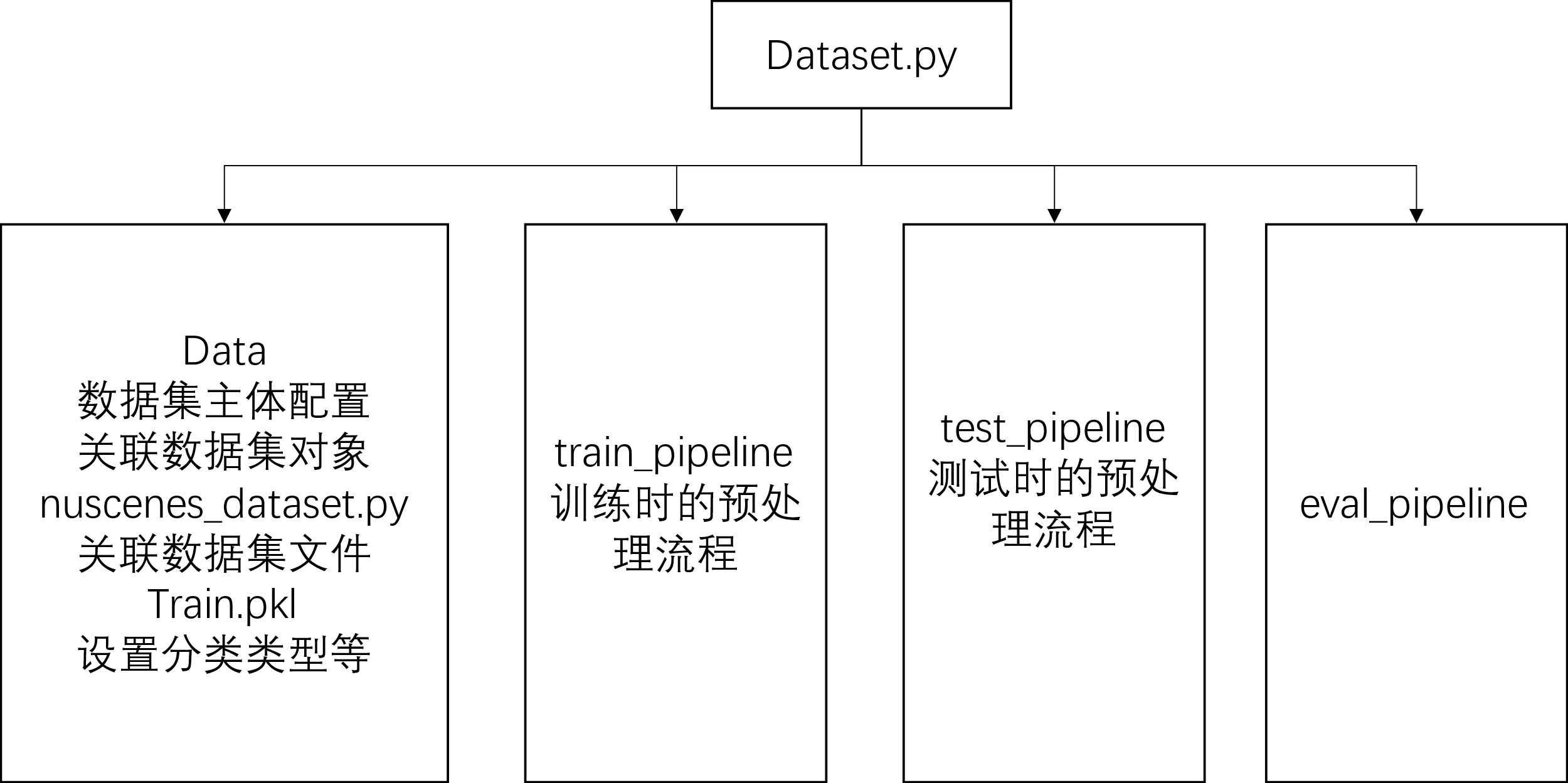
### 检测头pts\_bbox\_head(Anchor3dHead)

检测头pts\_bbox\_head采用了Anchor3DHead，位置在mmdet3d/models/dense\_heads/anchor3d\_head.py

表格 5 检测头的输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| feats | list[torch.Tensor] |  | 多尺度特征 |  |
| 返回值 | list[torch.Tensor] | 返回值 |  |  |

## Data定义



图表 12 mmdetection3d中data部分的配置模块

数据直接采用了nuscenes数据集，因而直接引入

实际上数据集对象文件的定义在

mmdetection3d/mmdet3d/datasets/pipelines/nuscenes\_dataset.py

在config文件中，我们需要进行以下几个定义，如图表 12所示

1. data=dict()

数据集本身的定义，包括数据集文件、处理数据集的代码如nuscenes\_dataset.py

1. train\_pipeline=dict()

train模式下对数据集数据的预处理

1. test\_pipeline=dic()

test模式下对数据的预处理

|  |
| --- |
| # If point cloud range is changed, the models should also change their point  # cloud range accordingly  point\_cloud\_range = [-50, -50, -5, 50, 50, 3]  # For nuScenes we usually do 10-class detection  class\_names = [      'car', 'truck', 'trailer', 'bus', 'construction\_vehicle', 'bicycle',      'motorcycle', 'pedestrian', 'traffic\_cone', 'barrier'  ]  dataset\_type = 'NuScenesDataset'  data\_root = 'data/nuscenes/'  # Input modality for nuScenes dataset, this is consistent with the submission  # format which requires the information in input\_modality.  input\_modality = dict(      use\_lidar=True,      use\_camera=False,      use\_radar=False,      use\_map=False,      use\_external=False)  file\_client\_args = dict(backend='disk')  # Uncomment the following if use ceph or other file clients.  # See https://mmcv.readthedocs.io/en/latest/api.html#mmcv.fileio.FileClient  # for more details.  # file\_client\_args = dict(  #     backend='petrel',  #     path\_mapping=dict({  #         './data/nuscenes/': 's3://nuscenes/nuscenes/',  #         'data/nuscenes/': 's3://nuscenes/nuscenes/'  #     }))  train\_pipeline = [      dict(          type='LoadPointsFromFile',          coord\_type='LIDAR',          load\_dim=5,          use\_dim=5,          file\_client\_args=file\_client\_args),      dict(          type='LoadPointsFromMultiSweeps',          sweeps\_num=10,          file\_client\_args=file\_client\_args),      dict(type='LoadAnnotations3D', with\_bbox\_3d=True, with\_label\_3d=True),      dict(          type='GlobalRotScaleTrans',          rot\_range=[-0.3925, 0.3925],          scale\_ratio\_range=[0.95, 1.05],          translation\_std=[0, 0, 0]),      dict(type='RandomFlip3D', flip\_ratio\_bev\_horizontal=0.5),      dict(type='PointsRangeFilter', point\_cloud\_range=point\_cloud\_range),      dict(type='ObjectRangeFilter', point\_cloud\_range=point\_cloud\_range),      dict(type='ObjectNameFilter', classes=class\_names),      dict(type='PointShuffle'),      dict(type='DefaultFormatBundle3D', class\_names=class\_names),      dict(type='Collect3D', keys=['points', 'gt\_bboxes\_3d', 'gt\_labels\_3d'])  ]  test\_pipeline = [      dict(          type='LoadPointsFromFile',          coord\_type='LIDAR',          load\_dim=5,          use\_dim=5,          file\_client\_args=file\_client\_args),      dict(          type='LoadPointsFromMultiSweeps',          sweeps\_num=10,          file\_client\_args=file\_client\_args),      dict(          type='MultiScaleFlipAug3D',          img\_scale=(1333, 800),          pts\_scale\_ratio=1,          flip=False,          transforms=[              dict(                  type='GlobalRotScaleTrans',                  rot\_range=[0, 0],                  scale\_ratio\_range=[1., 1.],                  translation\_std=[0, 0, 0]),              dict(type='RandomFlip3D'),              dict(                  type='PointsRangeFilter', point\_cloud\_range=point\_cloud\_range),              dict(                  type='DefaultFormatBundle3D',                  class\_names=class\_names,                  with\_label=False),              dict(type='Collect3D', keys=['points'])          ])  ]  # construct a pipeline for data and gt loading in show function  # please keep its loading function consistent with test\_pipeline (e.g. client)  eval\_pipeline = [      dict(          type='LoadPointsFromFile',          coord\_type='LIDAR',          load\_dim=5,          use\_dim=5,          file\_client\_args=file\_client\_args),      dict(          type='LoadPointsFromMultiSweeps',          sweeps\_num=10,          file\_client\_args=file\_client\_args),      dict(          type='DefaultFormatBundle3D',          class\_names=class\_names,          with\_label=False),      dict(type='Collect3D', keys=['points'])  ]  data = dict(      samples\_per\_gpu=4,      workers\_per\_gpu=4,      train=dict(          type=dataset\_type,          data\_root=data\_root,          ann\_file=data\_root + 'nuscenes\_infos\_train.pkl',          pipeline=train\_pipeline,          classes=class\_names,          modality=input\_modality,          test\_mode=False,          # we use box\_type\_3d='LiDAR' in kitti and nuscenes dataset          # and box\_type\_3d='Depth' in sunrgbd and scannet dataset.          box\_type\_3d='LiDAR'),      val=dict(          type=dataset\_type,          data\_root=data\_root,          ann\_file=data\_root + 'nuscenes\_infos\_val.pkl',          pipeline=test\_pipeline,          classes=class\_names,          modality=input\_modality,          test\_mode=True,          box\_type\_3d='LiDAR'),      test=dict(          type=dataset\_type,          data\_root=data\_root,          ann\_file=data\_root + 'nuscenes\_infos\_val.pkl',          pipeline=test\_pipeline,          classes=class\_names,          modality=input\_modality,          test\_mode=True,          box\_type\_3d='LiDAR'))  # For nuScenes dataset, we usually evaluate the model at the end of training.  # Since the models are trained by 24 epochs by default, we set evaluation  # interval to be 24. Please change the interval accordingly if you do not  # use a default schedule.  evaluation = dict(interval=24, pipeline=eval\_pipeline) |

## Schedule定义

Schecule主要设定了训练过程中optimizer的相关参数，例如learning rate，decay等，一般选取一个mmdetection3d给出的schedule中的一个即可

schedule\_2x.py

|  |
| --- |
| # optimizer  # This schedule is mainly used by models on nuScenes dataset  optimizer = dict(type='AdamW', lr=0.001, weight\_decay=0.01)  # max\_norm=10 is better for SECOND  optimizer\_config = dict(grad\_clip=dict(max\_norm=35, norm\_type=2))  lr\_config = dict(      policy='step',      warmup='linear',      warmup\_iters=1000,      warmup\_ratio=1.0 / 1000,      step=[20, 23])  momentum\_config = None  # runtime settings  runner = dict(type='EpochBasedRunner', max\_epochs=24) |

## Runtime定义

default\_runtime.py

该部分主要完成了在训练、运行过程中，打log、读取checkpoint训练结果等设定，一般直接引用default\_runtime.py即可

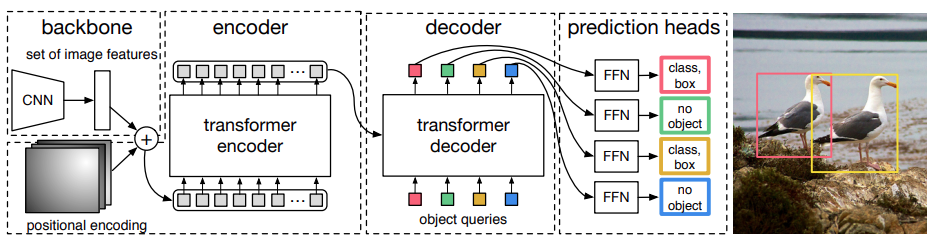
|  |
| --- |
| checkpoint\_config = dict(interval=1)  # yapf:disable push  # By default we use textlogger hook and tensorboard  # For more loggers see  # https://mmcv.readthedocs.io/en/latest/api.html#mmcv.runner.LoggerHook  log\_config = dict(      interval=50,      hooks=[          dict(type='TextLoggerHook'),          dict(type='TensorboardLoggerHook')      ])  # yapf:enable  dist\_params = dict(backend='nccl')  log\_level = 'INFO'  work\_dir = None  load\_from = None  resume\_from = None  workflow = [('train', 1)] |

## 训练可视化

## 结果可视化

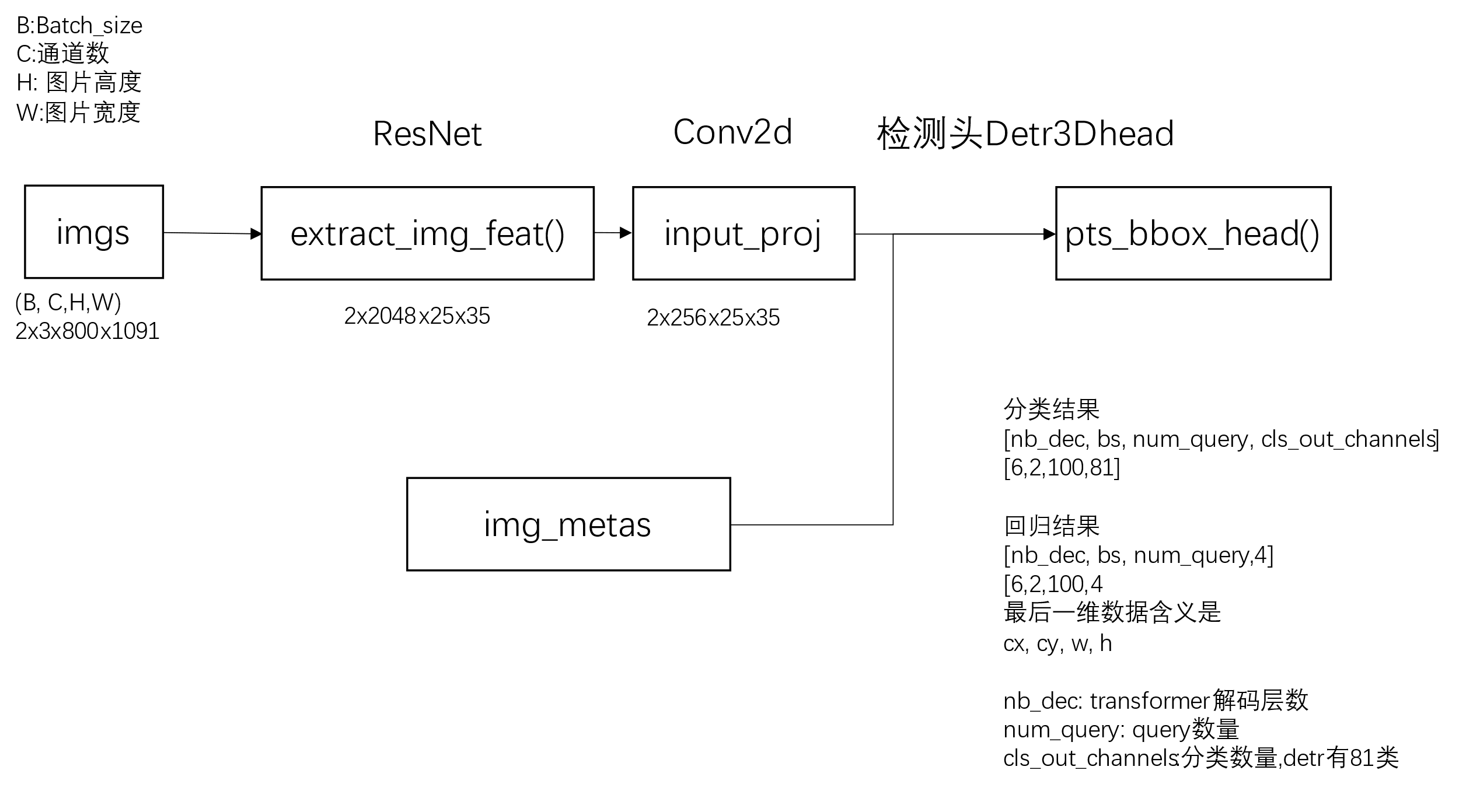
# Detr

为了更好地研究Detr3d框架，我们首先引入在二维检测工作上的著名的工作Detr，因为Detr3d很大程度上借鉴了Detr的做法，并将其推广到了3D平面上。Detr的总体架构图表 13所示，是一个端到端的结构，首先通过Resnet backbone提取出特征信息后，和位置编码一起进入transformer encoder，再在transformer decoder中结合object query编码，输出100个候选检测对象，训练时通过huangarian匹配，实现端到端的对象检测。



图表 13 Detr总体架构

图表 14更详细地给出了detr各个主要部分的数据维度与数据流



图表 14 Detr算法数据流

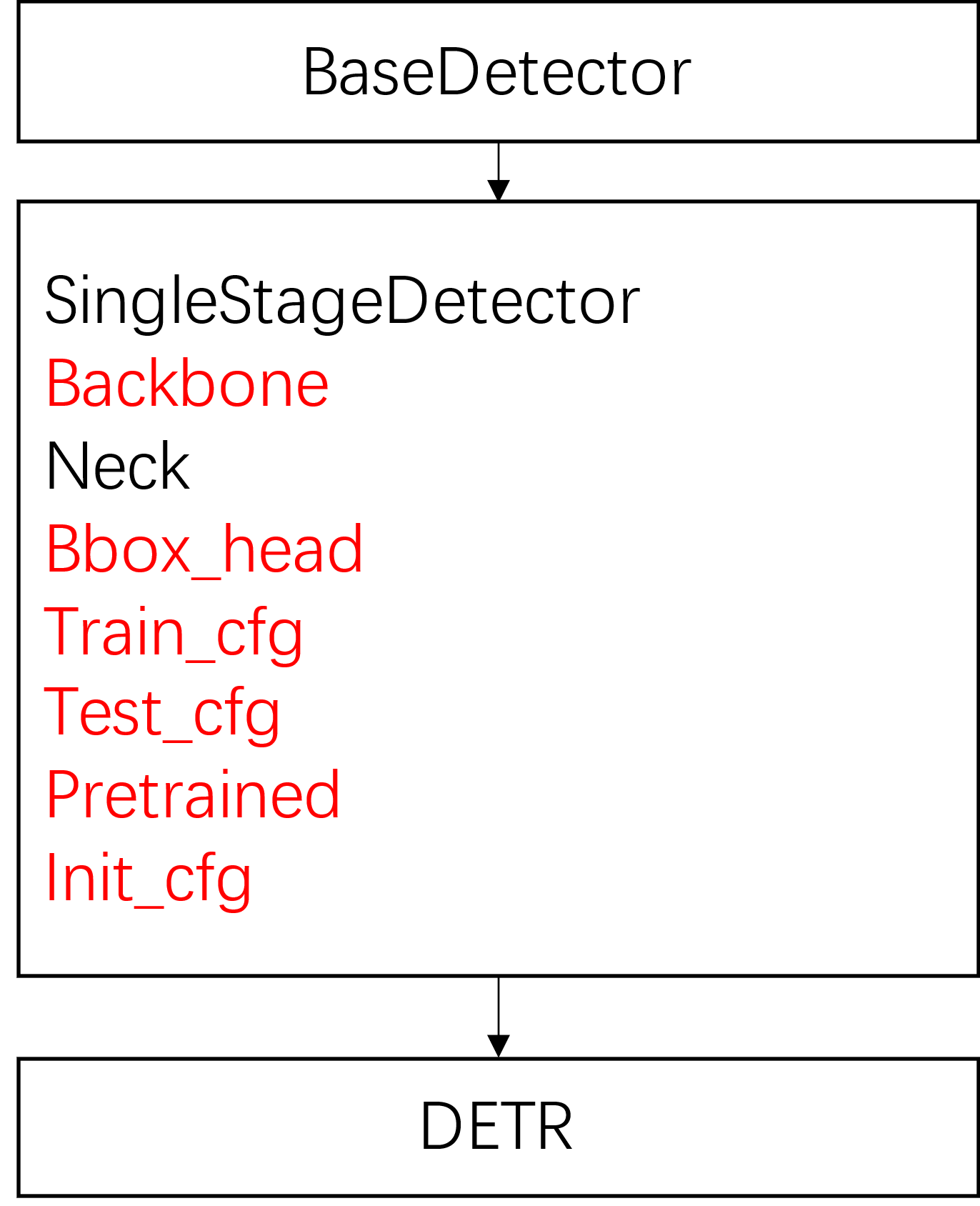
## Model定义

我们从model的定义中，可以看到，detr由以下几个部分组成

backbone:ResNet50

bbox\_head:DETRHead，采用了transformer架构，分为encoder和decoder部分

|  |
| --- |
| model = dict(      type='DETR',      backbone=dict(          type='ResNet',          depth=50,          num\_stages=4,          out\_indices=(3, ),          frozen\_stages=1,          norm\_cfg=dict(type='BN', requires\_grad=False),          norm\_eval=True,          style='pytorch',          init\_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')),      bbox\_head=dict(          type='DETRHead',          num\_classes=80,          in\_channels=2048,          transformer=dict(              type='Transformer',              encoder=dict(                  type='DetrTransformerEncoder',                  num\_layers=6,                  transformerlayers=dict(                      type='BaseTransformerLayer',                      attn\_cfgs=[                          dict(                              type='MultiheadAttention',                              embed\_dims=256,                              num\_heads=8,                              dropout=0.1)                      ],                      feedforward\_channels=2048,                      ffn\_dropout=0.1,                      operation\_order=('self\_attn', 'norm', 'ffn', 'norm'))),              decoder=dict(                  type='DetrTransformerDecoder',                  return\_intermediate=True,                  num\_layers=6,                  transformerlayers=dict(                      type='DetrTransformerDecoderLayer',                      attn\_cfgs=dict(                          type='MultiheadAttention',                          embed\_dims=256,                          num\_heads=8,                          dropout=0.1),                      feedforward\_channels=2048,                      ffn\_dropout=0.1,                      operation\_order=('self\_attn', 'norm', 'cross\_attn', 'norm',                                       'ffn', 'norm')),              )),          positional\_encoding=dict(              type='SinePositionalEncoding', num\_feats=128, normalize=True),          loss\_cls=dict(              type='CrossEntropyLoss',              bg\_cls\_weight=0.1,              use\_sigmoid=False,              loss\_weight=1.0,              class\_weight=1.0),          loss\_bbox=dict(type='L1Loss', loss\_weight=5.0),          loss\_iou=dict(type='GIoULoss', loss\_weight=2.0)),      # training and testing settings      train\_cfg=dict(          assigner=dict(              type='HungarianAssigner',              cls\_cost=dict(type='ClassificationCost', weight=1.),              reg\_cost=dict(type='BBoxL1Cost', weight=5.0, box\_format='xywh'),              iou\_cost=dict(type='IoUCost', iou\_mode='giou', weight=2.0))),      test\_cfg=dict(max\_per\_img=100)) |



图表 15 Detr设定的参数

通过图表 15，我们可以更清楚地看到，对于detr，我们设定了backbone，bbox\_head，train\_cfg,test\_cfg几个主要部分

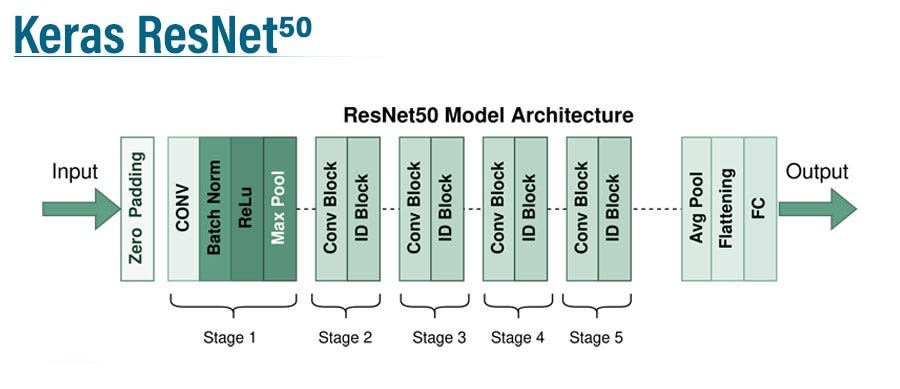
## 检测实际代码分析

### img\_backbone (Resnet）

Det的backbone采用了Resnet提取特征信息，位置在mmdet/models/backbones/resnet.py，设置采用了4 stage 50层的resnet

表格 6 Resnet 50输入输出

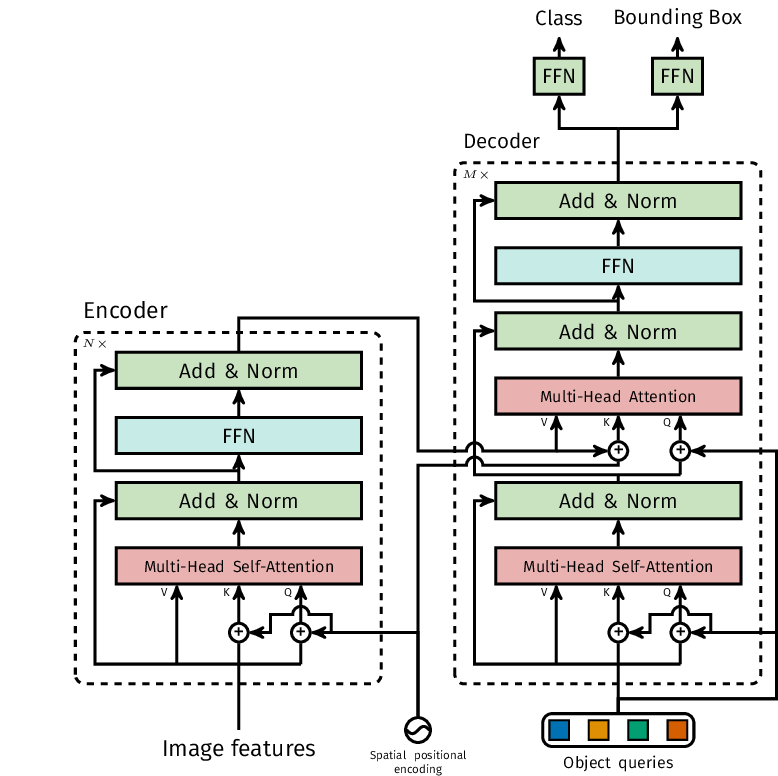
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| Img | tensor(float) |  | 一个batch的图片 | BCHW，如[2,3,800,1091] |
| Img\_feats | tensor(float) | 返回值 | 一个batch提取出的特征图 | BCHW [2,2048,25,35] |



图表 16 ResNet50 架构

### Bbox\_head (Transformer）

Detr的bbox\_head采用了transformer架构，分为encoder和decoder两部分，架构如图表 17所示。其中decoder输入了特征图和一个用sin进行的位置编码，二者维度相同，decoder一共有六层，其特殊之处在于每一层的输入都介入了位置编码。Decoder输入了encoder的结果和一个可学习的query object，其维度是100x256，最终输出了检测对象的位置和分类



图表 17 Detr中使用的Transformer具体架构

表格 7 Transformer输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| x | Tensor(tensor(float) |  | 特征图 | BCHW，如[2,3,800,1091] |
| masks | tensor(float) |  | 为了应用于多分辨率的补0掩码 | BCHW [2,2048,25,35] |
| query\_embed | tensor(float) |  | Query objects，可训练，实际就是embedding | [100，256] |
| pos\_embed | tensor(float) |  | 特征图的位置编码 | BCHW [2,2048,25,35] |
| out\_dec |  |  |  | [6,2,100,256] |
| memory |  |  |  | [2,256,25,35] |

### DetrTransformerEncoder

DetrTransformer的Encoder只用到了self attention，一共有六层，第一层输入时backbone的特征图，其他五层都是上一层的输出。注意以下两点

1. 每一层的输入都用到了spatial positional encoding
2. 只在Query和Key上面使用了位置编码，也就是Key=Query=Query+pos\_encoding

表格 8 DetrTransformerEncoder输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| query | Tensor(tensor(float) |  | Transformer的输入 | [875,2,256]为了适应multihead,经过换维[bs, c, h, w] -> [h\*w, bs, c]  875\*25\*35 |
| key | tensor(float) | None | Key | 同上 |
| value | tensor(float) | None | value | 同上 |
| query\_pos | tensor(float) |  | 特征图的位置编码 | [h\*w, bs, c] [875,2,256] |
| query\_key\_padding\_mask | tensor(float) |  | 补0掩码 | [2,875] |
| Memory | tensor(float) | 返回值 | Encoder的输出 | [h\*w, bs, c] [875,2,256] |

### DetrTransformerDecoder

DetrTransformer的decoder部分输入了，其输入输出如表格 9所示

1. 第一层输入时，用的输入数据是全0的输入，其他层用之前曾的输出
2. 在self Attention层，对key和query加上object query，value不变，而在cross attention层，对query加上object query，而对于key则需要加上spatial encoding
3. Cross attention层的query是上之前self attention的归一化结果，而key和value是encoder的输出

表格 9 DetrTransformerDecoder输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| query | tensor(float) |  | Transformer的输入 | Target，[100,bz,c]第一层全0，之后需用之前层的输出  [100,2,256] |
| key | tensor(float) |  | Key | Encoder的输出Memory，[h\*w, bs, c] [875,2,256] |
| value | tensor(float) |  | value | 同上 |
| Key\_pos |  |  | 特征图的位置编码 | pos\_embed  [875,2,256] |
| query\_pos | tensor(float) |  |  | query\_embed  [100,2,256] |
| key\_padding\_mask |  |  | 补0掩码 | Mask  [2,875] |
| out\_dec | 返回值 |  | Decoder多层叠加 | [6,100,2,256] |

### HuangarianAssigner

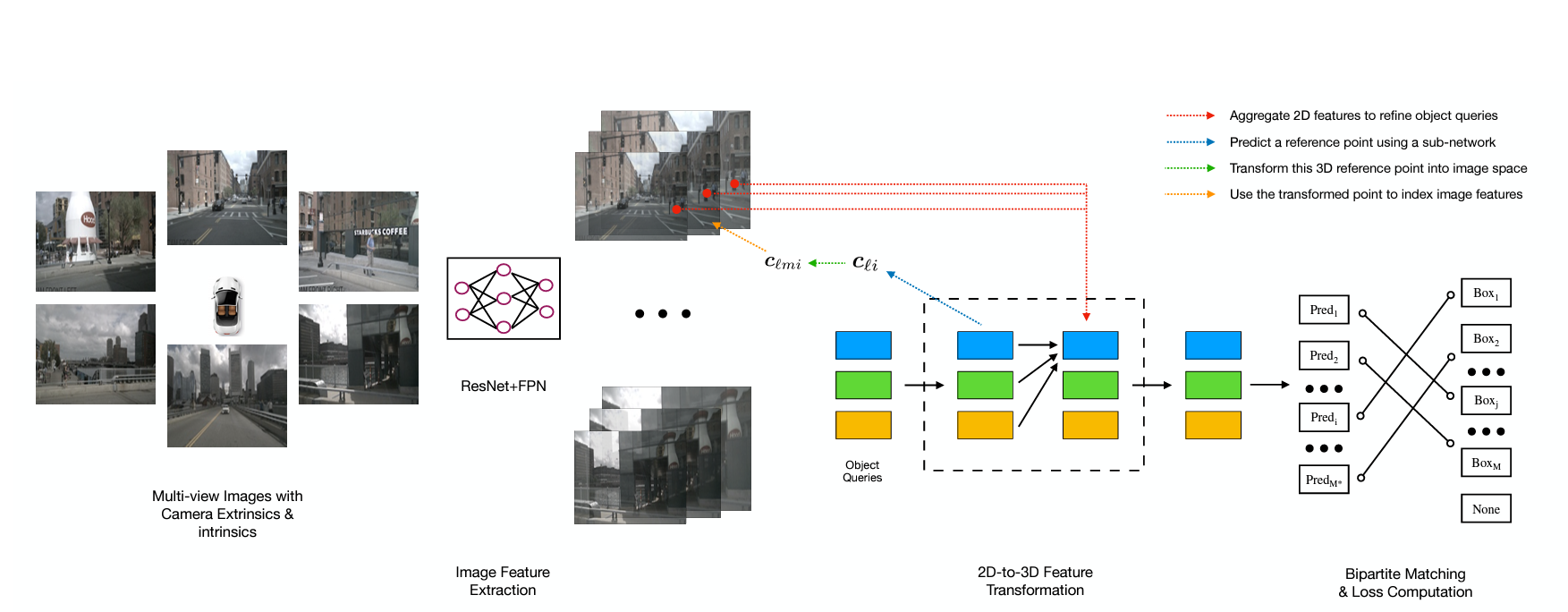
位置在mmdetection/mmdet/core/bbox/assigners/Hungarian\_assigner.py中HungarianAssigner类中assign函数

表格 10 assign函数输入输出

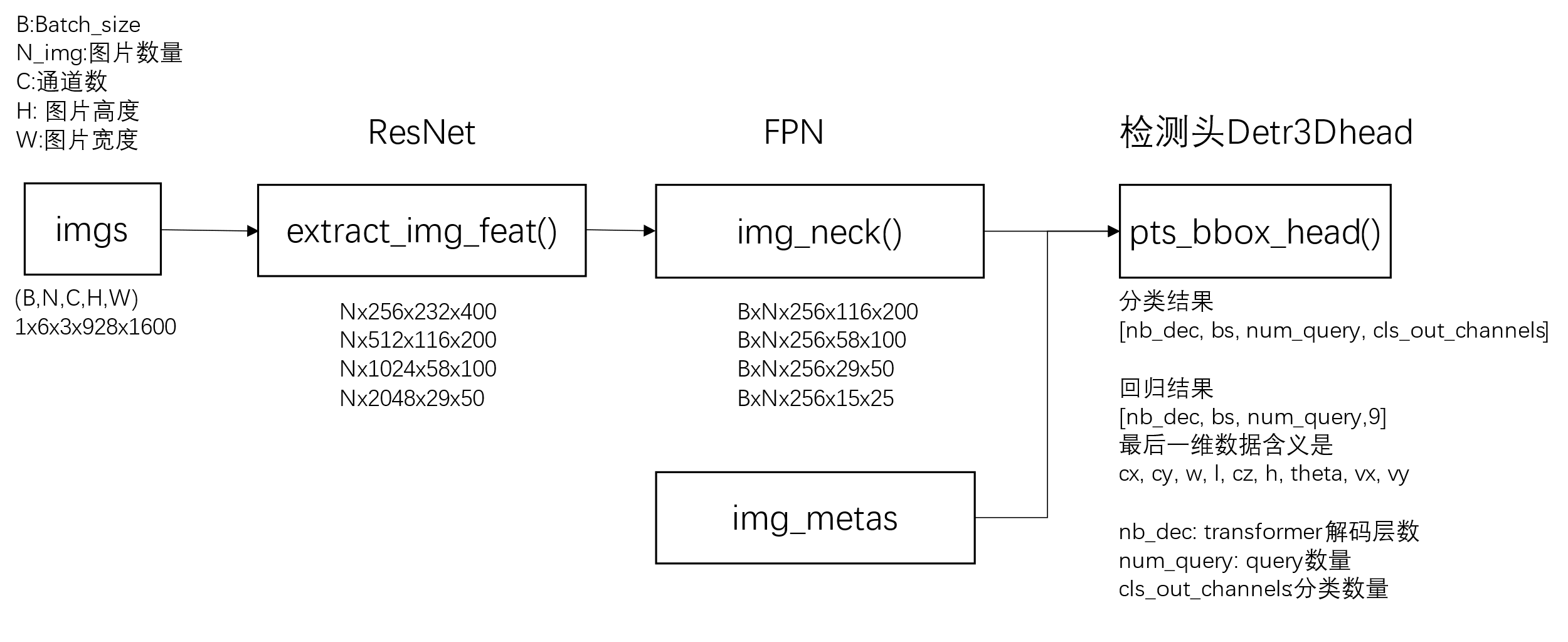
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| bbox\_pred | Tensor(float) |  |  | [100,4] |
| cls\_pred | Tensor(float) |  |  | [100,81] |
| gt\_bboxes | Tensor(float) |  |  | [10,4] |
| gt\_labels | Tensor(float) |  |  | [10,81] |
| img\_meta |  |  |  |  |
| gt\_bboxes\_ignore |  |  |  |  |
| eps | Double | 1e-7 |  |  |
| assign\_result |  |  |  | gt\_inds,tensor(int)  一百个，[0,0,0,0,2] |

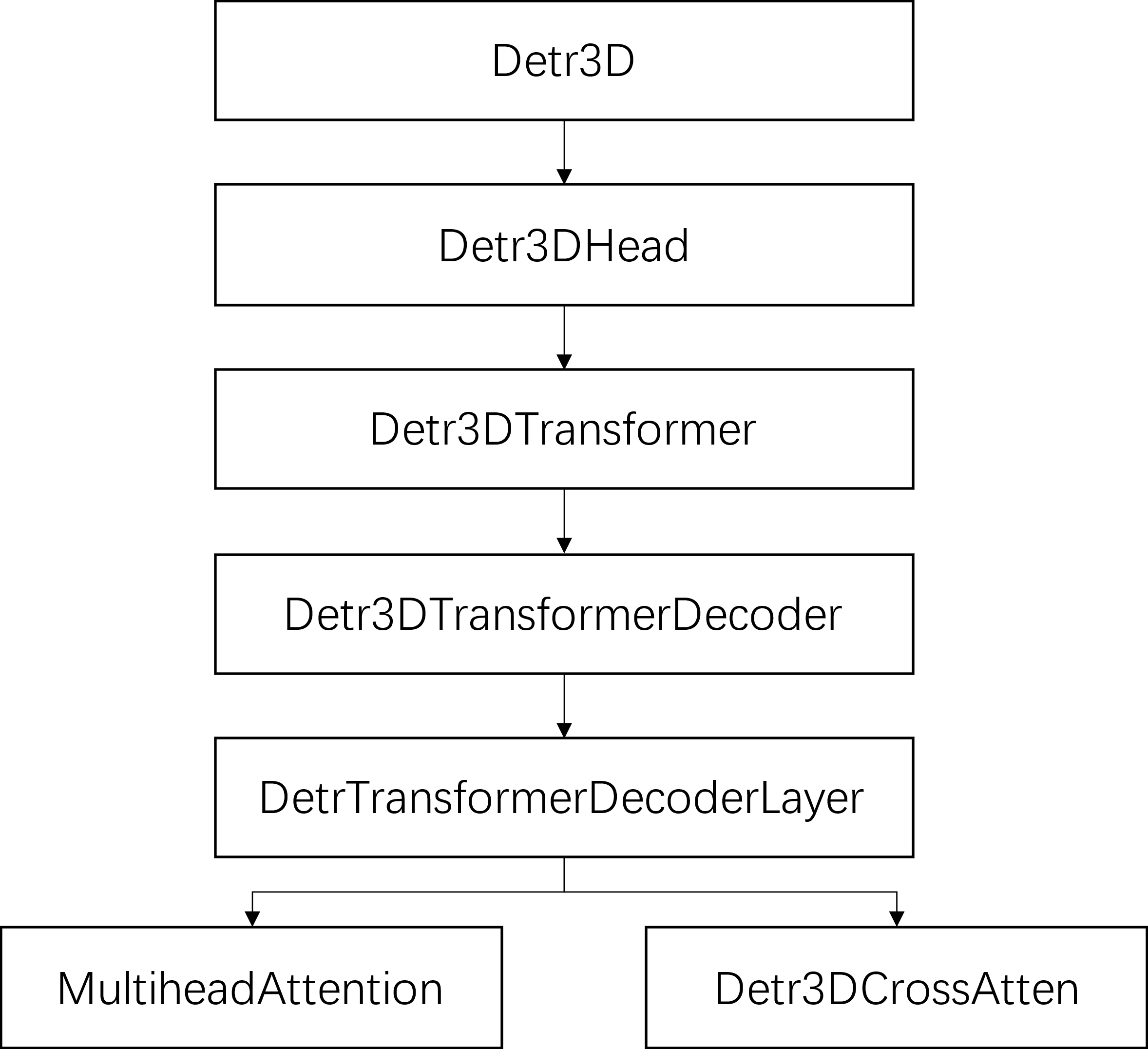
# Detr3d

Detr3d是一个基于纯视觉的3D检测网络，它类似于Detr，利用transformer进行检测，训练时利用3D的Hungarian BBox计算误差，其基本思想是在BEV坐标系下产生900个ref，经过外参投影后，进而通过Query到输入数据的六个相机中进行计算。



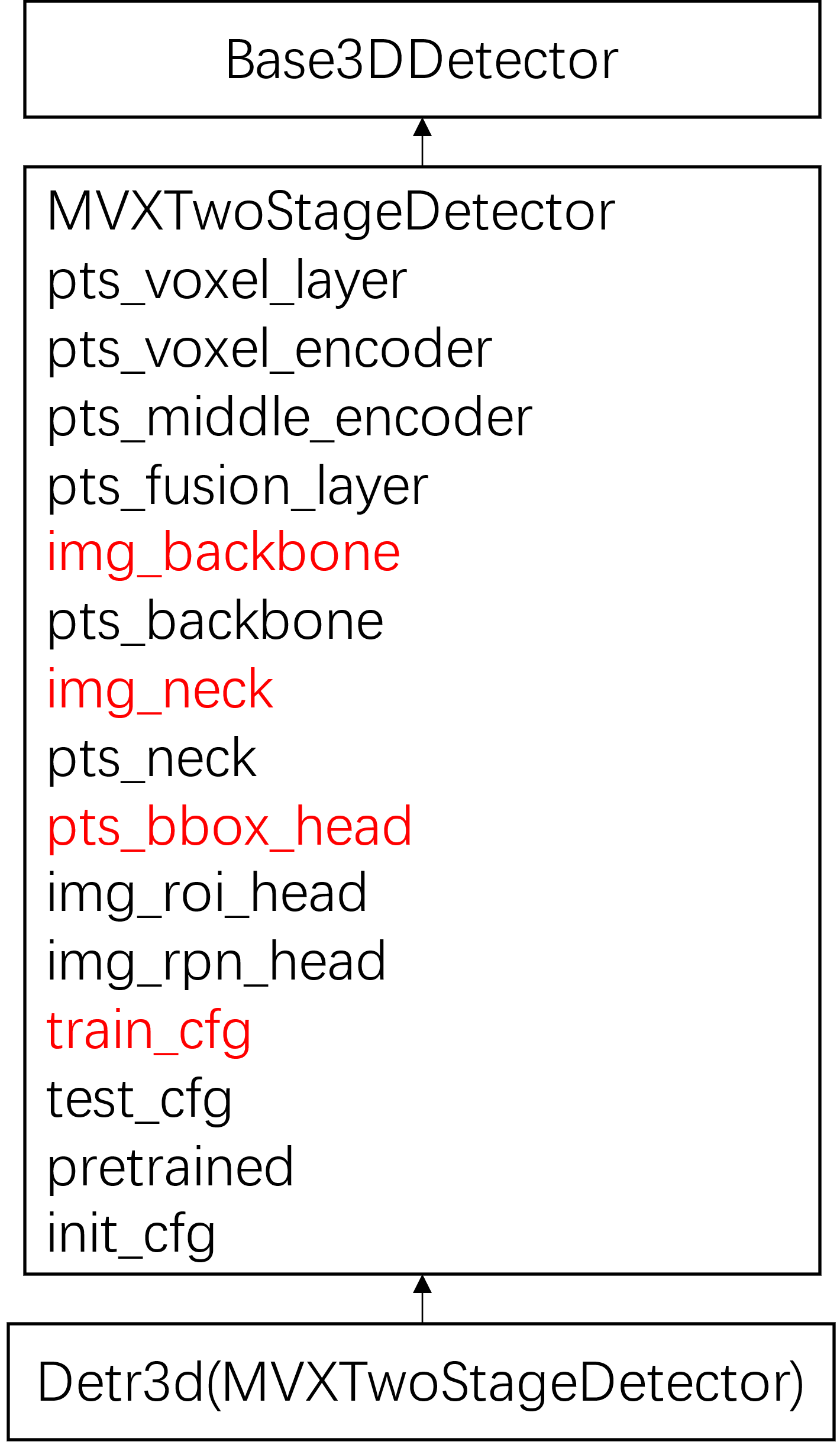
图表 18 Detr3D算法流程，通过3d找2D的方式，对BEV坐标系下的object detection query到图片上查询





图表 19 Detr3D中将Detr3DHead作为检测头，检测头中运用了Detr3dTransformer

## Model定义



图表 20 Detr3d模型为MVXTwoStageDetector，以及其需要初始化的部件

Detr3D继承了MVXTwoStageDetector,是一个两段式的检测器，分为特征提取和基于特征的对象检测两部分，因此model定义时定义了以下几个部分

img\_backbone:

img\_neck:

pts\_bbox\_head:

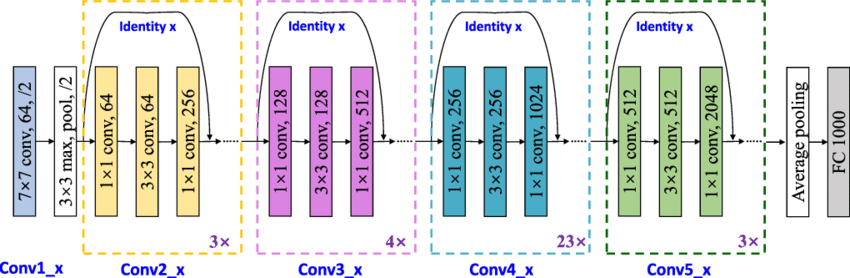
train\_cfg:

|  |
| --- |
| \_base\_ = [      '../../../mmdetection3d/configs/\_base\_/datasets/nus-3d.py',      '../../../mmdetection3d/configs/\_base\_/default\_runtime.py'  ]  plugin=True  plugin\_dir='projects/mmdet3d\_plugin/'  # If point cloud range is changed, the models should also change their point  # cloud range accordingly  point\_cloud\_range = [-51.2, -51.2, -5.0, 51.2, 51.2, 3.0]  voxel\_size = [0.2, 0.2, 8]  img\_norm\_cfg = dict(      mean=[103.530, 116.280, 123.675], std=[1.0, 1.0, 1.0], to\_rgb=False)  # For nuScenes we usually do 10-class detection  class\_names = [      'car', 'truck', 'construction\_vehicle', 'bus', 'trailer', 'barrier',      'motorcycle', 'bicycle', 'pedestrian', 'traffic\_cone'  ]  input\_modality = dict(      use\_lidar=False,      use\_camera=True,      use\_radar=False,      use\_map=False,      use\_external=False)  model = dict(      type='Detr3D',      use\_grid\_mask=True,      img\_backbone=dict(          type='ResNet',          depth=101,          num\_stages=4,          out\_indices=(0, 1, 2, 3),          frozen\_stages=1,          norm\_cfg=dict(type='BN2d', requires\_grad=False),          norm\_eval=True,          style='caffe',          dcn=dict(type='DCNv2', deform\_groups=1, fallback\_on\_stride=False),          stage\_with\_dcn=(False, False, True, True)),      img\_neck=dict(          type='FPN',          in\_channels=[256, 512, 1024, 2048],          out\_channels=256,          start\_level=1,          add\_extra\_convs='on\_output',          num\_outs=4,          relu\_before\_extra\_convs=True),      pts\_bbox\_head=dict(          type='Detr3DHead',          num\_query=900,          num\_classes=10,          in\_channels=256,          sync\_cls\_avg\_factor=True,          with\_box\_refine=True,          as\_two\_stage=False,          transformer=dict(              type='Detr3DTransformer',              decoder=dict(                  type='Detr3DTransformerDecoder',                  num\_layers=6,                  return\_intermediate=True,                  transformerlayers=dict(                      type='DetrTransformerDecoderLayer',                      attn\_cfgs=[                          dict(                              type='MultiheadAttention',                              embed\_dims=256,                              num\_heads=8,                              dropout=0.1),                          dict(                              type='Detr3DCrossAtten',                              pc\_range=point\_cloud\_range,                              num\_points=1,                              embed\_dims=256)                      ],                      feedforward\_channels=512,                      ffn\_dropout=0.1,                      operation\_order=('self\_attn', 'norm', 'cross\_attn', 'norm',                                       'ffn', 'norm')))),          bbox\_coder=dict(              type='NMSFreeCoder',              post\_center\_range=[-61.2, -61.2, -10.0, 61.2, 61.2, 10.0],              pc\_range=point\_cloud\_range,              max\_num=300,              voxel\_size=voxel\_size,              num\_classes=10),          positional\_encoding=dict(              type='SinePositionalEncoding',              num\_feats=128,              normalize=True,              offset=-0.5),          loss\_cls=dict(              type='FocalLoss',              use\_sigmoid=True,              gamma=2.0,              alpha=0.25,              loss\_weight=2.0),          loss\_bbox=dict(type='L1Loss', loss\_weight=0.25),          loss\_iou=dict(type='GIoULoss', loss\_weight=0.0)),      # model training and testing settings      train\_cfg=dict(pts=dict(          grid\_size=[512, 512, 1],          voxel\_size=voxel\_size,          point\_cloud\_range=point\_cloud\_range,          out\_size\_factor=4,          assigner=dict(              type='HungarianAssigner3D',              cls\_cost=dict(type='FocalLossCost', weight=2.0),              reg\_cost=dict(type='BBox3DL1Cost', weight=0.25),              iou\_cost=dict(type='IoUCost', weight=0.0), # Fake cost. This is just to make it compatible with DETR head.              pc\_range=point\_cloud\_range))))  dataset\_type = 'NuScenesDataset'  data\_root = 'data/nuscenes/'  file\_client\_args = dict(backend='disk')  db\_sampler = dict(      data\_root=data\_root,      info\_path=data\_root + 'nuscenes\_dbinfos\_train.pkl',      rate=1.0,      prepare=dict(          filter\_by\_difficulty=[-1],          filter\_by\_min\_points=dict(              car=5,              truck=5,              bus=5,              trailer=5,              construction\_vehicle=5,              traffic\_cone=5,              barrier=5,              motorcycle=5,              bicycle=5,              pedestrian=5)),      classes=class\_names,      sample\_groups=dict(          car=2,          truck=3,          construction\_vehicle=7,          bus=4,          trailer=6,          barrier=2,          motorcycle=6,          bicycle=6,          pedestrian=2,          traffic\_cone=2),      points\_loader=dict(          type='LoadPointsFromFile',          coord\_type='LIDAR',          load\_dim=5,          use\_dim=[0, 1, 2, 3, 4],          file\_client\_args=file\_client\_args))  train\_pipeline = [      dict(type='LoadMultiViewImageFromFiles', to\_float32=True),      dict(type='PhotoMetricDistortionMultiViewImage'),      dict(type='LoadAnnotations3D', with\_bbox\_3d=True, with\_label\_3d=True, with\_attr\_label=False),      dict(type='ObjectRangeFilter', point\_cloud\_range=point\_cloud\_range),      dict(type='ObjectNameFilter', classes=class\_names),      dict(type='NormalizeMultiviewImage', \*\*img\_norm\_cfg),      dict(type='PadMultiViewImage', size\_divisor=32),      dict(type='DefaultFormatBundle3D', class\_names=class\_names),      dict(type='Collect3D', keys=['gt\_bboxes\_3d', 'gt\_labels\_3d', 'img'])  ]  test\_pipeline = [      dict(type='LoadMultiViewImageFromFiles', to\_float32=True),      dict(type='NormalizeMultiviewImage', \*\*img\_norm\_cfg),      dict(type='PadMultiViewImage', size\_divisor=32),      dict(          type='MultiScaleFlipAug3D',          img\_scale=(1333, 800),          pts\_scale\_ratio=1,          flip=False,          transforms=[              dict(                  type='DefaultFormatBundle3D',                  class\_names=class\_names,                  with\_label=False),              dict(type='Collect3D', keys=['img'])          ])  ]  data = dict(      samples\_per\_gpu=1,      workers\_per\_gpu=4,      train=dict(          type=dataset\_type,          data\_root=data\_root,          ann\_file=data\_root + 'nuscenes\_infos\_train.pkl',          pipeline=train\_pipeline,          classes=class\_names,          modality=input\_modality,          test\_mode=False,          use\_valid\_flag=True,          # we use box\_type\_3d='LiDAR' in kitti and nuscenes dataset          # and box\_type\_3d='Depth' in sunrgbd and scannet dataset.          box\_type\_3d='LiDAR'),      val=dict(pipeline=test\_pipeline, classes=class\_names, modality=input\_modality),      test=dict(pipeline=test\_pipeline, classes=class\_names, modality=input\_modality))  optimizer = dict(      type='AdamW',      lr=2e-4,      paramwise\_cfg=dict(          custom\_keys={              'img\_backbone': dict(lr\_mult=0.1),          }),      weight\_decay=0.01)  optimizer\_config = dict(grad\_clip=dict(max\_norm=35, norm\_type=2))  # learning policy  lr\_config = dict(      policy='CosineAnnealing',      warmup='linear',      warmup\_iters=500,      warmup\_ratio=1.0 / 3,      min\_lr\_ratio=1e-3)  total\_epochs = 24  evaluation = dict(interval=2, pipeline=test\_pipeline)  runner = dict(type='EpochBasedRunner', max\_epochs=total\_epochs)  load\_from='ckpts/fcos3d.pth' |

## 检测实际代码分析

### img\_backbone (Resnet）

Detr3d的backbone采用了Resnet提取特征信息，位置在mmdet/models/backbones/resnet.py，设置采用了4 stage 101层的resnet，因此输出会有四个尺度的特征图



图表 21 4 stage 101层restnet网络架构

每一个stage的不同尺度、通道数的结构都进行了输出

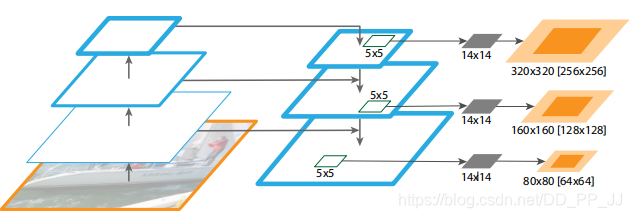
表格 11 Resnet输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| Img | Tensor(tensor(float) |  | 一个batch的图片 | (B\*N)CHW，对一个Batch内多个摄像头的摄像头维度上拉平，形成B\*N并行处理 |
| Img\_feats | List[tensor(float)] | 返回值 |  | 多尺度的特征信息,每个层次都是(B\*N)CHW，例如  [[6,256,232,400],  [6,512,116,200],  [6,1024,58,100],  [6,2048,29,50]] |

### Img\_neck(FPN)

颈部网络采用了FPN(feature pyramid network)，其作用是把上游backbone输出的多尺度特征信息的channel数变成一致，以生成适合于目标检测的输出，适配detection head。图表 22展示了FPN网络，其流程是

1. 左侧的输入多尺度信息就是上游的backbone输出的多尺度特征信息，
2. 每个尺度的信息都会被对应的lateral\_conv卷积再处理一遍，
3. 插值放大后
4. 和backbone输出的特征信息相加
5. 再经过fpn\_conv，输出多尺度信息，此时的多尺度信息的channel数量是想通的

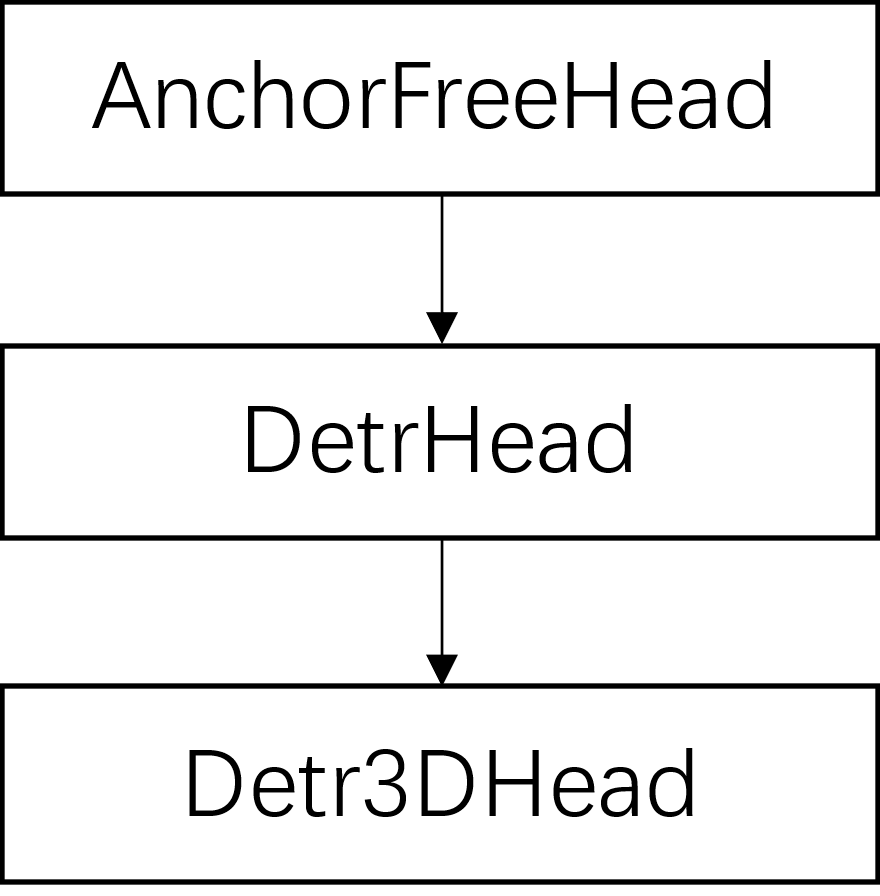


图表 22 FPN的网络架构与其中Lateral conv与fpn\_conv进行多尺度特征处理

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| Img\_feats | List[tensor(float)] |  | Backbone处理后的多尺度特征信息 | 多尺度的特征信息,每个层次都是(B\*N)CHW，例如  [[6,256,232,400],  [6,512,116,200],  [6,1024,58,100],  [6,2048,29,50]] |
| 返回值 | List[tensor(float)] | 返回值 | 颈部网络处理后的多尺度统一通道数特征信息 | 多尺度的特征信息,每个层次都是(B\*N)CHW，C都是一样的例如  [[6,256,116,200],  [6,256,58,100],  [6,256,29,50],  [6,256,15,25]] |

### pts\_bbox\_head(Detr3dHead)

Detr3dHead是detr3d中的核心部分，作为一个插件的形式自行构建，位置在mmdet3d\_plugin/models/dense\_heads/detr3d\_head.p，实现了从多个图片的特征信息中提取出三维检测对象



图表 23 Detr3DHead 继承关系

表格 12 Detr3DHead初始化参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| num\_query | Int |  | Reference point query的数量 | 代码中取了900，也就是900个候选 |
| num\_classes | Int |  | 分类数量 | 10类 |
| in\_channels | Int |  | 输入的特征通道数 | 上游的feats是256通道 |
| sync\_cls\_avg\_factor | Bool |  |  |  |
| with\_box\_refine | Bool |  |  |  |
| as\_two\_stage | Bool |  |  |  |
| transformer | Dict |  |  |  |
| bbox\_coder | Dict |  |  |  |
| positional\_encoding | Dict |  |  |  |
| loss\_cls | Dict |  |  |  |
| loss\_bbox | Dict |  |  |  |
| loss\_iou | Dict |  |  |  |

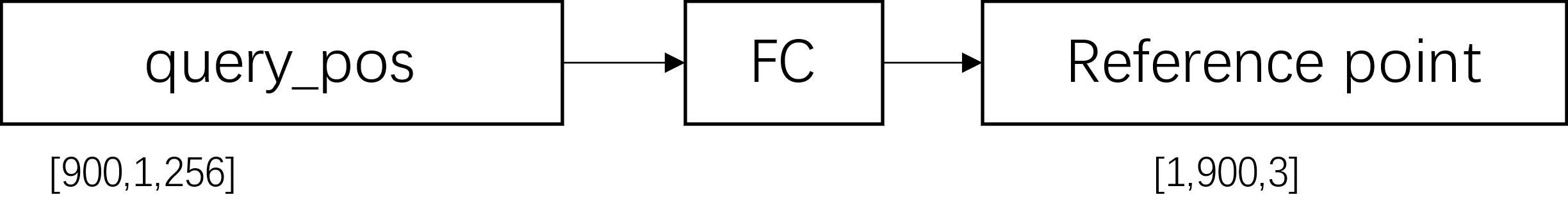
表格 13 Detr3d forward()函数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| pts\_feats | List[tensor(float) |  | 颈部网络处理后的多尺度统一通道数特征信息 | 多尺度的特征信息,每个层次都是BNCHW，C都是一样的例如  [[1,6,256,116,200],  [1,6,256,58,100],  [1,6,256,29,50],  [1,6,256,15,25]] |
| img\_metas | List[dict()] |  | 图片属性，包括外参等 | Filename:list[str]  Lidar2img:list[array]外参 |
| outs | Dict{all\_cls\_scores  : all\_bbox\_preds  ,} | 返回值 | 检测到的bbox和分类信息 | all\_cls\_scores  :outputs\_classes  all\_bbox\_preds  :outputs\_coords |

### Detr3DTransformer

检测头的实现采用了Detr3DTransformer，Detr3DTransformer和DetrTransformer有几个不同之处

1. 只有decoder而没有encoder
2. 对于query\_pos，需要先通过一个全连接层，将特征维度换成3，对应空间中的点，形成reference point， 如图表 24所示



图表 24 Detr3dTransformer对于query\_pos的变换操作

表格 14 Detr3dTransformer初始化参数

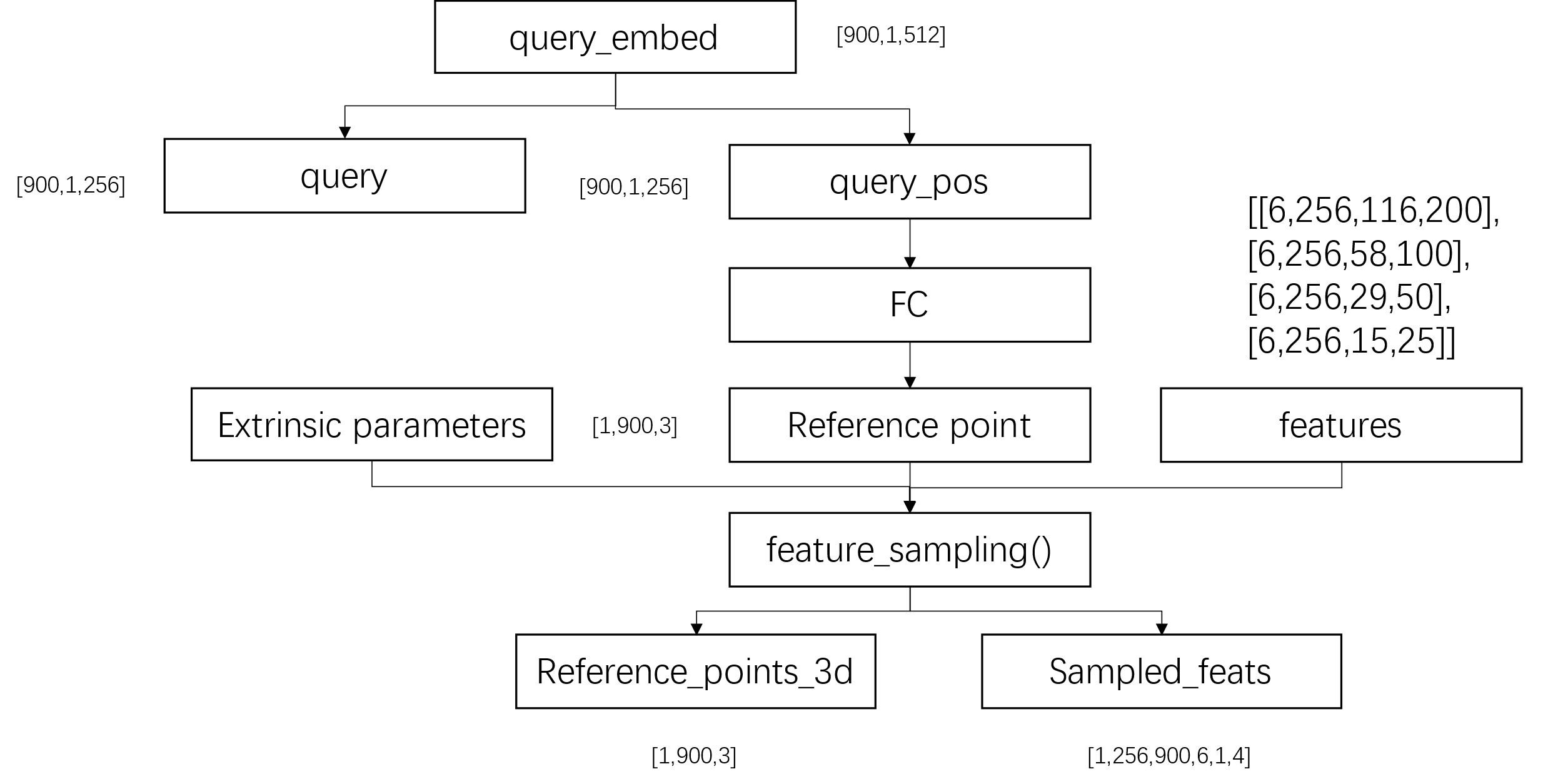
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| num\_feature\_levels | List[tensor(float) | 4 | 颈部网络处理后的多尺度统一通道数特征信息 | 多尺度的特征信息,每个层次都是BNCHW，C都是一样的例如  [[1,6,256,116,200],  [1,6,256,58,100],  [1,6,256,29,50],  [1,6,256,15,25]] |
| num\_cams | int |  |  | 6 |
| two\_stage\_num\_proposals |  |  |  |  |
| decoder |  |  |  | Transformer中具体用的解码器 |

表格 15 Detr3dTransformer forward函数输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| mlvl\_feats |  |  |  | 多尺度的特征信息,每个层次都是BNCHW，C都是一样的例如  [[1,6,256,116,200],  [1,6,256,58,100],  [1,6,256,29,50],  [1,6,256,15,25]] |
| query\_embed |  |  | 可学习的query编码，就是个embedding | [900,512] |
| reg\_branches |  |  |  | Regression heads for  feature maps from each decoder layer |
| img\_metas |  |  | 图片属性信息 |  |
|  |  |  |  |  |
|  |  |  |  |  |

### 特征采样函数feature\_sampling

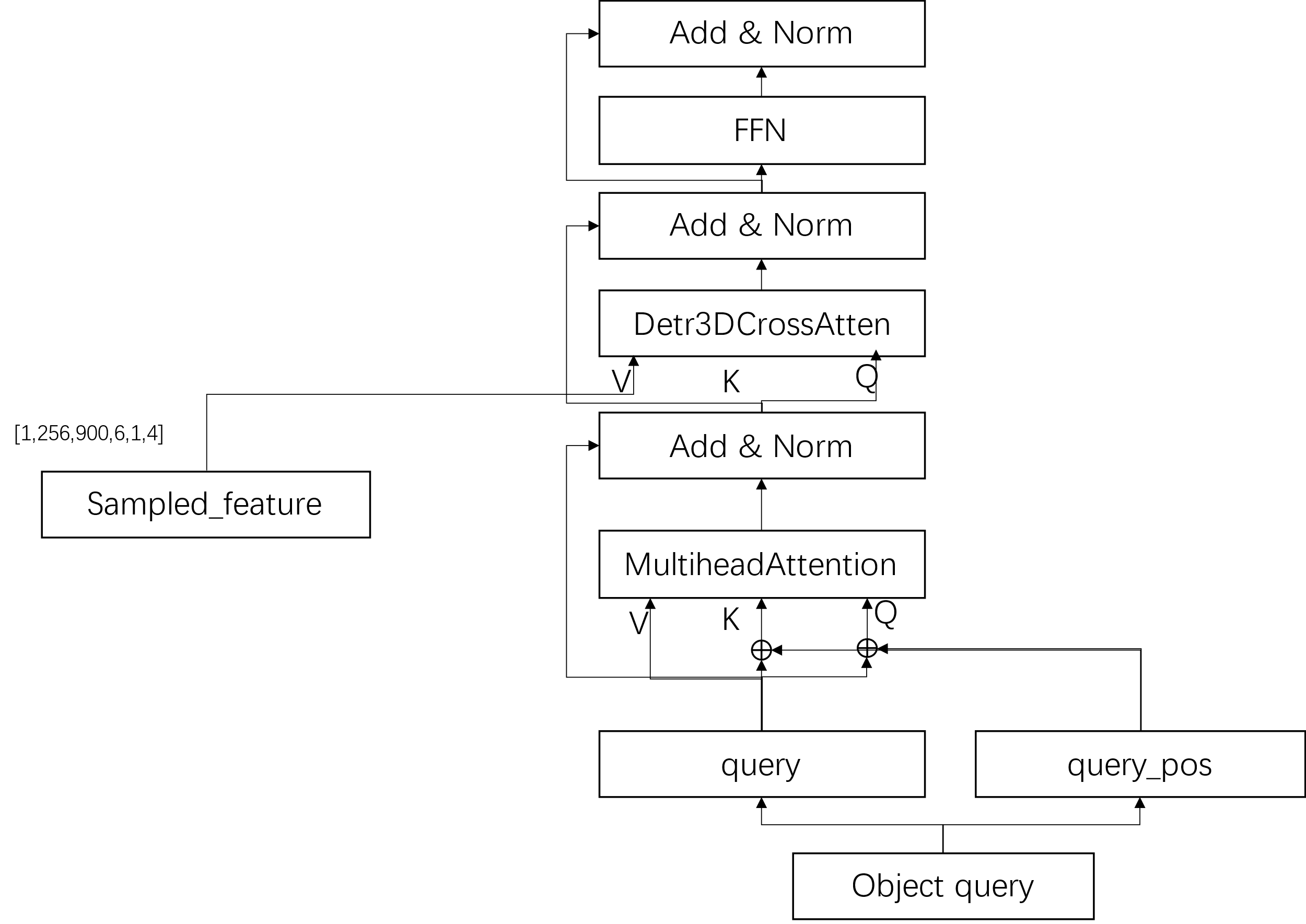
在detr3d代码中，feature\_sampling函数起到了3d到2d的转换作用，通过将3d坐标系下的reference\_point利用相机外参投影到相机平面上，进而提取出对应区域的特征信息



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| mlvl\_feats | List[tensor(float) |  | 多尺度的特征图， | 每个图BNCH，N是摄像头数量6 |
| reference\_points |  |  | 参考点信息 |  |
| pc\_range |  |  | 点云取值范围 |  |
| img\_metas |  |  | 图片外参 |  |
| reference\_points\_3d |  |  |  |  |
| sampled\_feats |  |  |  |  |
| mask |  |  |  |  |

1. 从图片外参中提取出lidar2img投影矩阵，4x4
2. 将reference\_point从归一化坐标恢复到物理坐标，然后加一维度，做成齐次
3. 对每个reference\_point维度expand之后，计算在每个相机的投影
4. 把投影到镜头后的mask掉
5. 归一化二维坐标到[0,1]之间
6. 对四个特征层分别求出线性插值后的feature
7. 返回原始的3d point和提取到的sampled feats

### Transformer解码器Detr3DTransformerDecoder



表格 16 Detr3dTransformerDecoder输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| query |  |  |  |  |
| reference\_points |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

### 交叉注意力机制Detr3DCrossAtten

3D交叉注意力机制是Detr3DTransformer进行3D到2D转换的核心构建，其基本原理是

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 数据类型 | 默认值 | 含义 | 取值说明 |
| mlvl\_feats | List[tensor(float) |  |  | 多尺度的特征图，每个图BNCH，N是摄像头数量6 |
| reference\_points |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Data定义

Data部分可参阅pointpillar，采用了nuscene数据集

## Schedule定义

# Futr3d Lidar Only 模式

Futr3d中激光部分与pillarpoint不同，检测头是通过transformer完成的，其流程如下图所示

## Model定义

## 检测实际代码分析

### Backbone

### Neck

## Data定义

## Schedule定义

## Runtime定义

# Futr3d 激光视觉融合模式

## Model定义

## 检测实际代码分析

## Data定义

## Schedule定义

## Runtime定义