# 对比详细

## 3D目标检测对比

### 运行环境

#### 硬件

GPU：2080Ti

System：gcc version 7.5.0 (ubuntu1~18.04)

#### 软件

**评测工具**

采用mmdetection3D提供的get FLOPs工具tools/analysis\_tools/get\_flops.py，目前暂未发现其他平台提供专门计算open-mmlab的模型的工具。该工具可以计算模型FLOPs以及参数量。其内部调用[mmcv.cnn.get\_model\_complexity\_info()](https://github.com/open-mmlab/mmcv/blob/master/mmcv/cnn/utils/flops_counter.py)去计算相关值，其可以打印每一层的参数量以及FLOPs，大部分的常见op其都支持。具体官方说明如下：

1. FLOPs值和输入尺寸有关，如batchsize，input size等，但是参数量和以上无关。

1. 一些GN操作和自定义操作没有计算到FLOPs中。但是大部分常见op都支持。

1. 目前只支持single-stage的以及single-modality的FLOPs计算。后续会支持多模态输入。

不过后续可以手动更改让该工具支持多模态输入。

**代码框架**

mmdetection3d

### 评估详情

#### 对比模型

* LSSDet-efficientnet-b3-camera

* FCOS3D-efficient-b3

#### Backbone调整

由于原版FCOS3Dbackbone是resnet-101，其较efficientnet-b3较重，故调整。

两个模型训练代码都基于mmdetection3d修改。FCOS3D的efficient-b3来自于mmdet模块，训练时加载pretrained模型，并锁定first stage（efficient-b3共有6 stages）参数。其他和LSSDet训练环境不同的是：

* 两者训练的mmdetection3d包版本有细微差异。LSSDet是0.17.3的版本，FCOS3D是最新的，不过代码上时间差别不远，差别较细微。

* FCOS3D模型Neck处的输入特性尺寸上有更改。Efficientnet-b3和resnet-101的每个stage的数量以及stage的输出尺寸不一样。所以FCOS3D-Efficientnet-b3版本的neck input size有变更in\_channels=[256, 512, 1024, 2048]->in\_channels=[256, 136, 384, 1536].

type='FCOSMono3D',  
backbone=dict(  
 type='EfficientNet',  
 arch='b3',  
 out\_indices=(3, 4, 5, 6),  
 frozen\_stages=1,  
 init\_cfg=dict(  
 type='Pretrained', prefix='backbone', checkpoint='./checkpoints/'  
 'efficientnet-b3\_3rdparty\_8xb32-aa\_in1k\_20220119-5b4887a0.pth')  
),  
neck=dict(  
 type='FPN',  
 # in\_channels=[256, 512, 1024, 2048],  
 in\_channels=[256, 136, 384, 1536],  
 out\_channels=256,  
 start\_level=1,  
 add\_extra\_convs='on\_output',  
 num\_outs=5,  
 relu\_before\_extra\_convs=True),

#### 模型算力统计

以下分析input统一到相同大小进行评价，LSSDet:(1,6,3,256,704), FCOS3D:(6,3,704, 256)。模型的算力分析可见log文件：[📎fcos3d\_profile.txt](https://yuque.antfin.com/attachments/lark/0/2022/txt/23156489/1655717489239-e9a97ea0-0486-4104-8dcf-d16e2b1d3b53.txt)[📎lssdet\_profile.txt](https://yuque.antfin.com/attachments/lark/0/2022/txt/23156489/1655717489234-e79e88c9-0494-40a5-aec5-a95dc6165b02.txt)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | LSSDet-efficientnet-camera | | | FCOS3D-efficientnet | | |
| 模块 | 结构 | GFLOPs | 比重 | 结构 | GFLOPs | 比重 |
| Img backbone | EfficientNetLSS | all:20.884  efficientnet:0.556  up:20.1 | all:24.646%  efficientnet:0.657%  up:23.721% | EfficientNet-b3 | 0.678 | 1.586% |
| Img Neck | LiftSplit | 0 | 0% | FPN | 4.602 | 10.800% |
| BEV encoder | SECOND | 36.346 | 42.893% |  |  |  |
| BEV neck | SECONDFPN | 1.361 | 1.606% |  |  |  |
| Head | CenterHead | 26.145 | 30.855% | FCOSMono3DHead | 37.356 | 87.567% |
| 总计 | / | 84.74 | / | / | 42.66 | / |

根据以上表格，分析得到算力占用较高的模块，可为后期优化提供依据。现总结如下：

1. 总体上，LSSDet的所需算力是FCOS3D-eff版的2倍。因为LSSDet的Img Neck是用来进行2D向BEV的转换，这部分算力并没有统计到，所以总体算力还会更高。

1. LSSDet和FCOS3D的img backbone里，efficientnet-b3的算力相当，但是LSSDet backbone部分有上采样操作Up，所需算力较大，后期可尝试算力较小的算子，如PixelShuffle。

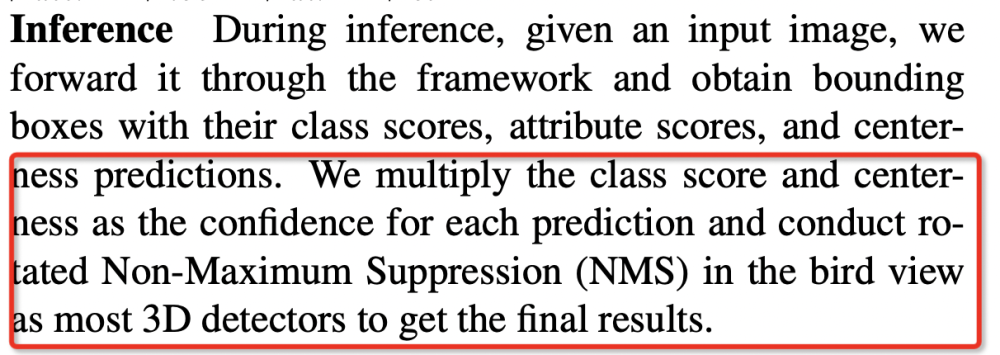
1. LSSDet的BEV encoder部分所需算力较大。这部分可能是encoder的backbone较重。BEV encoder+BEV neck+Head部分的整体算力和之前的评估相当。

1. 之前主要的算力瓶颈都在image backbone，更换较轻量的backbone后，backbone的算力大幅下降，算力大头集中到了BEV encoder和head部分。

#### 3D目标检测的对比

**FCOS3D的指标的评价方式**

从论文中看，作者强调，在inference阶段，是在鸟瞰图中做rotated NMS，获得最终结果。和其他的3D检测框保持一致。



在代码实现方面，mmdet3d遵循统一的NuScenes数据集计算方式进行评价，即收集6相机的图像数据的结果，变换到BEV视角下，再统一做NMS，最后将结果返回，去计算mAP和NDS。

def \_format\_bbox(self, results, jsonfile\_prefix=None):  
 """Convert the results to the standard format.  
  
 Args:  
 results (list[dict]): Testing results of the dataset.  
 jsonfile\_prefix (str): The prefix of the output jsonfile.  
 You can specify the output directory/filename by  
 modifying the jsonfile\_prefix. Default: None.  
  
 Returns:  
 str: Path of the output json file.  
 """  
 nusc\_annos = {}  
 mapped\_class\_names = self.CLASSES  
  
 print('Start to convert detection format...')  
 # 指定camera的数量为6  
 CAM\_NUM = 6  
 # 遍历结果里的3D框  
 for sample\_id, det in enumerate(mmcv.track\_iter\_progress(results)):  
 # 每6张图片组合为一个BEV结果  
 if sample\_id % CAM\_NUM == 0:  
 boxes\_per\_frame = []  
 attrs\_per\_frame = []  
  
 # need to merge results from images of the same sample  
 # 合并同一样本下的图片  
 annos = []  
 boxes, attrs = output\_to\_nusc\_box(det)  
 sample\_token = self.data\_infos[sample\_id]['token']  
 boxes, attrs = cam\_nusc\_box\_to\_global(self.data\_infos[sample\_id],  
 boxes, attrs,  
 mapped\_class\_names,  
 self.eval\_detection\_configs,  
 self.eval\_version)  
  
 boxes\_per\_frame.extend(boxes)  
 attrs\_per\_frame.extend(attrs)  
 # Remove redundant predictions caused by overlap of images  
 # 删除大量有冗余重叠的预测框  
 if (sample\_id + 1) % CAM\_NUM != 0:  
 continue  
  
 boxes = global\_nusc\_box\_to\_cam(  
 self.data\_infos[sample\_id + 1 - CAM\_NUM], boxes\_per\_frame,  
 mapped\_class\_names, self.eval\_detection\_configs,  
 self.eval\_version)  
 cam\_boxes3d, scores, labels = nusc\_box\_to\_cam\_box3d(boxes)  
 # box nms 3d over 6 images in a frame  
 # 在6张图片组成的单帧里进行NMS  
 nms\_cfg = dict(  
 use\_rotate\_nms=True,  
 nms\_across\_levels=False,  
 nms\_pre=4096,  
 nms\_thr=0.05,  
 score\_thr=0.01,  
 min\_bbox\_size=0,  
 max\_per\_frame=500)  
 from mmcv import Config  
 nms\_cfg = Config(nms\_cfg)  
 cam\_boxes3d\_for\_nms = xywhr2xyxyr(cam\_boxes3d.bev)  
 boxes3d = cam\_boxes3d.tensor  
 # generate attr scores from attr labels  
 # 从attr标签中生成attr得分  
 attrs = labels.new\_tensor([attr for attr in attrs\_per\_frame])  
 boxes3d, scores, labels, attrs = box3d\_multiclass\_nms(  
 boxes3d,  
 cam\_boxes3d\_for\_nms,  
 scores,  
 nms\_cfg.score\_thr,  
 nms\_cfg.max\_per\_frame,  
 nms\_cfg,  
 mlvl\_attr\_scores=attrs)  
 cam\_boxes3d = CameraInstance3DBoxes(boxes3d, box\_dim=9)  
 det = bbox3d2result(cam\_boxes3d, scores, labels, attrs)  
 boxes, attrs = output\_to\_nusc\_box(det)  
 boxes, attrs = cam\_nusc\_box\_to\_global(  
 self.data\_infos[sample\_id + 1 - CAM\_NUM], boxes, attrs,  
 mapped\_class\_names, self.eval\_detection\_configs,  
 self.eval\_version)  
  
 for i, box in enumerate(boxes):  
 name = mapped\_class\_names[box.label]  
 attr = self.get\_attr\_name(attrs[i], name)  
 nusc\_anno = dict(  
 sample\_token=sample\_token,  
 translation=box.center.tolist(),  
 size=box.wlh.tolist(),  
 rotation=box.orientation.elements.tolist(),  
 velocity=box.velocity[:2].tolist(),  
 detection\_name=name,  
 detection\_score=box.score,  
 attribute\_name=attr)  
 annos.append(nusc\_anno)  
 # other views results of the same frame should be concatenated  
 if sample\_token in nusc\_annos:  
 nusc\_annos[sample\_token].extend(annos)  
 else:  
 nusc\_annos[sample\_token] = annos  
  
 nusc\_submissions = {  
 'meta': self.modality,  
 'results': nusc\_annos,  
 }  
  
 mmcv.mkdir\_or\_exist(jsonfile\_prefix)  
 res\_path = osp.join(jsonfile\_prefix, 'results\_nusc.json')  
 print('Results writes to', res\_path)  
 mmcv.dump(nusc\_submissions, res\_path)  
 return res\_path

原后融合方式



更改后的后融合方式



#### 指标对比

**后融合处理方式的更改**

1. 在单目相机的情况下，先进行单目结果的Rotated NMS

single\_camera\_boxes, single\_camera\_attrs = global\_nusc\_box\_to\_cam(  
 self.data\_infos[sample\_id - (sample\_id % CAM\_NUM)], boxes,  
 mapped\_class\_names, self.eval\_detection\_configs,  
 self.eval\_version, single\_camera\_attrs)  
cam\_boxes3d, scores, labels = nusc\_box\_to\_cam\_box3d(single\_camera\_boxes)  
  
# 设置nms相关参数  
nms\_cfg = dict(  
 use\_rotate\_nms=True,  
 nms\_across\_levels=False,  
 nms\_pre=4096,  
 nms\_thr=0.05,  
 score\_thr=0.01,  
 min\_bbox\_size=0,  
 max\_per\_frame=500)  
from mmcv import Config  
nms\_cfg = Config(nms\_cfg)  
cam\_boxes3d\_for\_nms = xywhr2xyxyr(cam\_boxes3d.bev)  
boxes3d = cam\_boxes3d.tensor  
# 进行单目相机结果的nms  
single\_camera\_tensor\_attrs = labels.new\_tensor([attr for attr in single\_camera\_attrs])  
boxes3d, scores, labels, attrs = box3d\_multiclass\_nms(  
 boxes3d,  
 cam\_boxes3d\_for\_nms,  
 scores,  
 nms\_cfg.score\_thr,  
 nms\_cfg.max\_per\_frame,  
 nms\_cfg,  
 mlvl\_attr\_scores=single\_camera\_tensor\_attrs)

1. 每6个相机的结果进行合并并转换到相机坐标系下进行后续计算

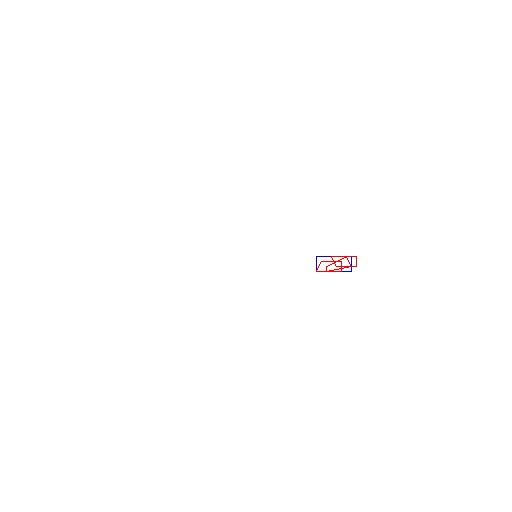
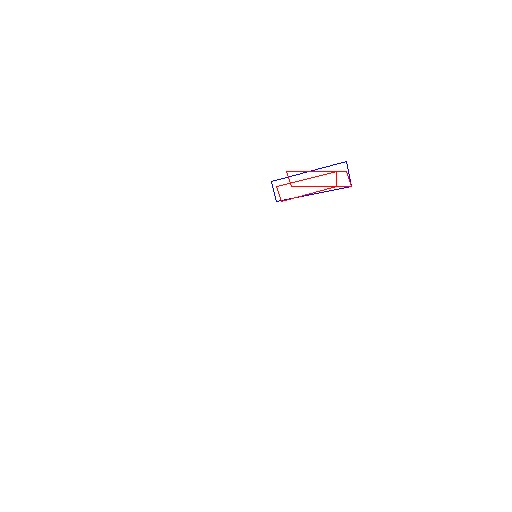
CAM\_NUM = 6  
# 每张图对应一个‘token’和‘image id’  
token\_per\_frame.extend([{'token':sample\_token,  
 'id':img\_id} for i in range(boxes3d.shape[0])])  
corners\_per\_frame.append(cam\_boxes3d.corners)  
boxes\_bev\_name\_per\_frame.append(cam\_boxes3d.bev)  
boxes\_per\_frame.extend(boxes3d)  
attrs\_per\_frame.extend(attrs)  
scores\_per\_frame.extend(scores)  
labels\_per\_frame.extend(labels)  
# 每6个相机进行相机边界处同一物体的框的合并计算  
if (sample\_id + 1) % CAM\_NUM != 0:  
 continue  
# 转换到相机坐标系下的框格式  
cam\_boxes3d\_for\_merge = CameraInstance3DBoxes(torch.stack(boxes\_per\_frame), box\_dim=9)  
# 合并在相机边界处的框  
boxes\_per\_frame, scores\_per\_frame, labels\_per\_frame, attrs\_per\_frame = \  
 self.\_merge\_boxes\_at\_camera\_edge(  
 token\_per\_frame,  
 boxes\_per\_frame,  
 cam\_boxes3d\_for\_merge.corners,  
 cam\_boxes3d\_for\_merge.bev,  
 scores\_per\_frame,  
 labels\_per\_frame,  
 attrs\_per\_frame)

1. 只有在同token下的不同相机图片内的同类别物体，且不是'traffic\_cone','barrier' 物体，且IOU超过阈值则合并删除冗余框。

if tokens[sample\_id]['token'] == tokens[moving\_id]['token'] and \  
 tokens[sample\_id]['id'] != tokens[moving\_id]['id'] and \  
 labels[sample\_id].item() == labels[moving\_id].item() and \  
 labels[moving\_id].item() not in filter\_classes:  
 # calculate iou of two boxes  
 box1 = boxes\_for\_bev\_merge[sample\_id].unsqueeze(0)  
 box2 = boxes\_for\_bev\_merge[moving\_id].unsqueeze(0)  
 ious = box\_iou\_rotated(box1, box2, mode='iou', aligned=True)  
   
 iou = torch.max(ious)  
 if iou.item() > iou\_thr:  
 ......

1. 采用bev视角下，框的4个角点作为框的表示，求所有框的最小外接矩形来得到合并框的中心点(x,y)、宽高(w,h)和偏航角(yaw)

x0;在雷达坐标系下，绘制BEV视角下的框，合并效果如下所示。蓝色为合并框。



以下指标测评方法如下：

1. Flops，Params，MemoryUsage跑10次取平均。

1. FPS统计2000张图的平均运行帧率作为结果。

1. NDS和mAP统一在NuScenes数据集的val数据上做评价，后融合规则是更改后的后融合方式。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 算法 | LSSDet Camera Only | FCOS3D Eff B3 | FCOS3D Eff B3 | FCOS3D Resnet101 |
| Input | 1,6,3,256,704 | 6,3,704, 256 | 6,3,1600, 900 | 1,3,1600, 900 |
| Flops(GFLOPs) | 84.74 | 42.66 | 58.0 | 334.7 |
| Params(Mb) | 23.03 | 20.95 | 20.95 | 54.62 |
| GPU MemoryUsage(Gb) | 5.75 | 5.97 | 14.550 | 4.87 |
| FPS | 6.1 | 5.5 | 1.47 | 0.65(6帧) |
| Epoch | 20 | 20 | 20 | 12 |
| NDS | 33.572 | To be continued | To be continued | 37.2 |
| mAP | 27.281 | To be continued | To be continued | 29.5 |

\*单目3D检测合并iou\_thr=0.2

\*为和其他模型训练策略拉齐，FCOS3D-r101版的指标，是未进行finetune的结果。

在后融合方案评估后，总结几个点：

1. 合并采用后融合处理的框时，合并的框越多指标越低。

分析了下，正如上边合并框效果所示，当不同相机预测的框预测的框合并后，采用合并的方式合并后，一定会存在长/宽度变大，yaw角/中心点偏差很大的情况。所以在后融合方案中，无论采用何种合并方式，都会引入较大误差，导致指标下降。

1. 在单目3D目标检测input size比基于BEV的检测input size大，性能更低的情况下，从训练结果看两项精度指标均低于后者。新后融合处理的方式的评估正在进行中。