FCOS3D损失函数计算流程

- 真值：
  - 训练/验证/测试数据的ann_file分别加载数据集中的
    - nuscenes_infos_train_mono3d.coco.json
    - nuscenes_infos_val_mono3d.coco.json
    - nuscenes_infos_test_mono3d.coco.json(这块mmdetection3d源码中测试集来源仍然是nuscenes_infos_val_mono3d.coco.json)
  - 从以上文件中读取制定训练数据：
    - img（验证与测试时仅需读此key）
    - gt_bboxes
    - gt_labels
    - attr_labels
    - gt_bboxes_3d
    - gt_labels_3d
    - centers2d
    - depth
  - 数据路径在configs/*base*/datasets/nus-mono3d.py中配置
- 预测值：
  - 在对应模型的dense_head中生成
  - fcos_mono3d_head继承anchor_free_mono3d_head
  - fcos3d中有5个输出特征图对应5个输入特征图，每个输出特征图有一个检测头，每个检测头有两个分支：分类分枝和回归分枝，每个分枝包括4个卷积层，且参数不共享。但是5个输出层的检测头的模块权重共享。
  - 初始化分类分枝中的卷积

```
    def _init_cls_convs(self):
        """Initialize classification conv layers of the head."""
        self.cls_convs = nn.ModuleList()
        for i in range(self.stacked_convs):
            chn = self.in_channels if i == 0 else self.feat_channels
            if self.dcn_on_last_conv and i == self.stacked_convs - 1:
                conv_cfg = dict(type='DCNv2')
            else:
                conv_cfg = self.conv_cfg
            self.cls_convs.append(
                ConvModule(
                    chn,
                    self.feat_channels,
                    3,
                    stride=1,
                    padding=1,
                    conv_cfg=conv_cfg,
                    norm_cfg=self.norm_cfg,
                    bias=self.conv_bias))
```

- 初始化回归分枝中的卷积

```
    def _init_reg_convs(self):
        """Initialize bbox regression conv layers of the head."""
        self.reg_convs = nn.ModuleList()
        for i in range(self.stacked_convs):
            chn = self.in_channels if i == 0 else self.feat_channels
            if self.dcn_on_last_conv and i == self.stacked_convs - 1:
                conv_cfg = dict(type='DCNv2')
            else:
                conv_cfg = self.conv_cfg
            self.reg_convs.append(
                ConvModule(
                    chn,
                    self.feat_channels,
                    3,
                    stride=1,
                    padding=1,
                    conv_cfg=conv_cfg,
                    norm_cfg=self.norm_cfg,
                    bias=self.conv_bias))
```

- 初始化分类分枝和回归分枝的预测模块

```
        """Initialize predictor layers of the head."""
        self.conv_cls_prev = self._init_branch(
            conv_channels=self.cls_branch,
            conv_strides=(1, ) * len(self.cls_branch))
        self.conv_cls = nn.Conv2d(self.cls_branch[-1], self.cls_out_channels,
                                  1)
        self.conv_reg_prevs = nn.ModuleList()
        self.conv_regs = nn.ModuleList()
        for i in range(len(self.group_reg_dims)):
            reg_dim = self.group_reg_dims[i]
            reg_branch_channels = self.reg_branch[i]
            out_channel = self.out_channels[i]
            if len(reg_branch_channels) > 0:
                self.conv_reg_prevs.append(
                    self._init_branch(
                        conv_channels=reg_branch_channels,
                        conv_strides=(1, ) * len(reg_branch_channels)))
                self.conv_regs.append(nn.Conv2d(out_channel, reg_dim, 1))
            else:
                self.conv_reg_prevs.append(None)
                self.conv_regs.append(
                    nn.Conv2d(self.feat_channels, reg_dim, 1))
        if self.use_direction_classifier:
            self.conv_dir_cls_prev = self._init_branch(
                conv_channels=self.dir_branch,
                conv_strides=(1, ) * len(self.dir_branch))
            self.conv_dir_cls = nn.Conv2d(self.dir_branch[-1], 2, 1)
        if self.pred_attrs:
            self.conv_attr_prev = self._init_branch(
                conv_channels=self.attr_branch,
                conv_strides=(1, ) * len(self.attr_branch))
            self.conv_attr = nn.Conv2d(self.attr_branch[-1], self.num_attrs, 1)
```

- group_reg_dims的长度代表回归分枝中不同预测功能模块的个数，对应位置的值代表该功能模块需要回归的值的个数，该list的求和值应该要和bbox_coder中的code_size值保持一致
- nchor_free_mono3d_head中的回归分枝的预测模块有5个，分别是offset, depth, size, rot, velocity, 总共预测sum(2,1,3,1,2) = 9个值，实际运行中不预测velo上的两个值

- 分层调用forward_single函数
  - 计算分类分枝特征和分类值
    -

```
cls_feat = x
reg_feat = x

for cls_layer in self.cls_convs:
    cls_feat = cls_layer(cls_feat)
# clone the cls_feat for reusing the feature map afterwards
clone_cls_feat = cls_feat.clone()
for conv_cls_prev_layer in self.conv_cls_prev:
    clone_cls_feat = conv_cls_prev_layer(clone_cls_feat)
cls_score = self.conv_cls(clone_cls_feat)
```

- 计算回归分枝特征和对应9个bbox属性的预测值

```
for reg_layer in self.reg_convs:
    reg_feat = reg_layer(reg_feat)
bbox_pred = []
for i in range(len(self.group_reg_dims)):
    # clone the reg_feat for reusing the feature map afterwards
    clone_reg_feat = reg_feat.clone()
    if len(self.reg_branch[i]) > 0:
        for conv_reg_prev_layer in self.conv_reg_prevs[i]:
            clone_reg_feat = conv_reg_prev_layer(clone_reg_feat)
    bbox_pred.append(self.conv_regs[i](clone_reg_feat))
bbox_pred = torch.cat(bbox_pred, dim=1)
```

- 在回归分枝中预测方向类别和在分类分枝中预测属性值

```
dir_cls_pred = None
if self.use_direction_classifier:
    clone_reg_feat = reg_feat.clone()
    for conv_dir_cls_prev_layer in self.conv_dir_cls_prev:
        clone_reg_feat = conv_dir_cls_prev_layer(clone_reg_feat)
    dir_cls_pred = self.conv_dir_cls(clone_reg_feat)
```

```
attr_pred = None
if self.pred_attrs:
    # clone the cls_feat for reusing the feature map afterwards
    clone_cls_feat = cls_feat.clone()
    for conv_attr_prev_layer in self.conv_attr_prev:
        clone_cls_feat = conv_attr_prev_layer(clone_cls_feat)
    attr_pred = self.conv_attr(clone_cls_feat)
```

- fcos_mono3d_head子类中额外实现在回归分枝的特征图上预测中心度

```
        cls_score, bbox_pred, dir_cls_pred, attr_pred, cls_feat, reg_feat = \
            super().forward_single(x)

        if self.centerness_on_reg:
            clone_reg_feat = reg_feat.clone()
            for conv_centerness_prev_layer in self.conv_centerness_prev:
                clone_reg_feat = conv_centerness_prev_layer(clone_reg_feat)
            centerness = self.conv_centerness(clone_reg_feat)
        else:
            clone_cls_feat = cls_feat.clone()
            for conv_centerness_prev_layer in self.conv_centerness_prev:
                clone_cls_feat = conv_centerness_prev_layer(clone_cls_feat)
            centerness = self.conv_centerness(clone_cls_feat)

        bbox_pred = self.bbox_coder.decode(bbox_pred, scale, stride,
                                           self.training, cls_score)

        return cls_score, bbox_pred, dir_cls_pred, attr_pred, centerness, \
            cls_feat, reg_feat
```

- 调用loss函数利用真是值和预测值计算损失
  - 损失函数的配置在backbbone中

```
        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=1.0),
        loss_attr=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0),
        loss_centerness=dict(
            type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0),
```

  - 计算分类损失

```
        loss_cls = self.loss_cls(
            flatten_cls_scores,
            flatten_labels_3d,
            avg_factor=num_pos + num_imgs)  # avoid num_pos is 0
```

  - 分别计算bbox上5个模块的损失，前两个为偏移，第三个为深度，四五六为大小，七位角度的sin值，八九为两个方向的速度当然速度默认是不预测的，当然在这之前会挑选出正样本
    - 挑选正样本，标签在0-9上的都是正样本，否则是负样本

```
# FG cat_id: [0, num_classes -1], BG cat_id: num_classes
bg_class_ind = self.num_classes
pos_inds = ((flatten_labels_3d >= 0)
            & (flatten_labels_3d < bg_class_ind)).nonzero().reshape(-1)
num_pos = len(pos_inds)
```

- 计算损失

```
loss_offset = self.loss_bbox(
    pos_bbox_preds[:, :2],
    pos_bbox_targets_3d[:, :2],
    weight=bbox_weights[:, :2],
    avg_factor=equal_weights.sum())
loss_depth = self.loss_bbox(
    pos_bbox_preds[:, 2],
    pos_bbox_targets_3d[:, 2],
    weight=bbox_weights[:, 2],
    avg_factor=equal_weights.sum())
loss_size = self.loss_bbox(
    pos_bbox_preds[:, 3:6],
    pos_bbox_targets_3d[:, 3:6],
    weight=bbox_weights[:, 3:6],
    avg_factor=equal_weights.sum())
loss_rotsin = self.loss_bbox(
    pos_bbox_preds[:, 6],
    pos_bbox_targets_3d[:, 6],
    weight=bbox_weights[:, 6],
    avg_factor=equal_weights.sum())
loss_velo = None
if self.pred_velo:
    loss_velo = self.loss_bbox(
        pos_bbox_preds[:, 7:9],
        pos_bbox_targets_3d[:, 7:9],
        weight=bbox_weights[:, 7:9],
        avg_factor=equal_weights.sum())
```

- 计算中心度损失

```
loss_centerness = self.loss_centerness(pos_centerness,
                                       pos_centerness_targets)
```

- 计算方向分类损失

```
# direction classification loss
loss_dir = None
# TODO: add more check for use_direction_classifier
if self.use_direction_classifier:
    loss_dir = self.loss_dir(
        pos_dir_cls_preds,
        pos_dir_cls_targets,
        equal_weights,
        avg_factor=equal_weights.sum())
```

- 计算属性值损失

```
# attribute classification loss
loss_attr = None
if self.pred_attrs:
    loss_attr = self.loss_attr(
        pos_attr_preds,
        pos_attr_targets,
        pos_centerness_targets,
        avg_factor=pos_centerness_targets.sum())
```

- 返回损失字典

```
loss_dict = dict(
    loss_cls=loss_cls,
    loss_offset=loss_offset,
    loss_depth=loss_depth,
    loss_size=loss_size,
    loss_rotsin=loss_rotsin,
    loss_centerness=loss_centerness)

if loss_velo is not None:
    loss_dict['loss_velo'] = loss_velo

if loss_dir is not None:
    loss_dict['loss_dir'] = loss_dir

if loss_attr is not None:
    loss_dict['loss_attr'] = loss_attr

return loss_dict
```