PP-YOLOE_plus_crn_l_80e

PP-YOLOE基于PP-YOLOv2，而PP-YOLOE中主要的改进点是：anchor-free（无锚框），powerful backbone and neck（强大的骨干网络和特征提取器），TAL动态label assign，ET-head（高效Transformer头部）。

模型结构解析：

参照论文 ***PP-YOLOE: An evolved version of YOLO***

1.1 结构说明

PP-YOLOE使用宽度系数α和深度系数β控制网络中backbone和neck的结构。backbone的默认宽度设置为[64、128、256、512、1024]，深度设置为[3,6,6,3]。neck的默认宽度设置和深度设置分别为[192,384,768]和3。针对s、m、l和x模型的结构缩放系数如下所示：

	width multiplier α	depth multiplier β
s	0.50	0.33
m	0.75	0.67
l	1.00	1.00
x	1.25	1.33

Table 1: Width multiplier α and depth multiplier β specification for a series of networks

1.2powerful backbone and neck（强大的骨干网络和特征提取器）

CSPNet是YOLOv5和YOLOX中流行的backbone，基于该启发设计了RepResBlock集成了residual连接和dense连接的特点。Block的设计如下所示，先简化concat(连接)操作为add操作(a图->b图)，然后在推理阶段将b图结构转化为c图（RepResBlock的结构）。所设计的CSPRepResStage结构如图d所示，使用了ESE(Effective Squeeze and Extraction)进行通道上的attention，其中的激活函数为swish。

使用CSPPAN做Neck,其中的block也是CSPRepResStage(是一种由多个重复的CSPResidual块组成的结构。每个CSPResidual块包括一系列的卷积层、批量归一化层和激活函数。)，但是删除了shortcut和和ESE层（因为在Neck中特征已经提取的很极值了，不需要再度attention）。其中的激活函数为SiLU(Swish)=f (x) = x · sigmoid (x)

RepResBlock源码：

这段代码定义了一个名为`RepVggBlock`的类，表示`RepVGG`网络中的一个基本块。基本块接受输入特征图`x`，并根据训练或推理的不同情况执行不同的操作。

```
class RepVggBlock(nn.Layer):
    def __init__(self, ch_in, ch_out, act='relu'):
        pass
```

在初始化方法__init__中，基本块接受输入通道数ch_in和输出通道数ch_out，以及激活函数类型act作为参数。但是在给出的代码中，初始化方法没有被实现，只有一个pass语句，因此需要根据实际情况补充初始化操作。

在前向传播方法forward中，基本块首先根据是否存在conv属性来判断当前是训练还是推理阶段。如果存在conv属性，表示当前是推理阶段，会使用self.conv对输入进行卷积操作；否则，表示当前是训练阶段，会分别使用self.conv1和self.conv2对输入进行卷积操作，并将结果相加。然后，经过激活函数self.act的处理后，输出结果y被返回。

```
    def forward(self, x):
        # 推理。在拥有conv对象时使用不同的逻辑
        if hasattr(self, 'conv'):
            y = self.conv(x)
        else:
            y = self.conv1(x) + self.conv2(x)
        y = self.act(y)
        return y
```

convert_to_deploy方法用于将训练阶段的基本块转换为推理阶段的基本块。在该方法中，首先判断是否存在conv属性，如果不存在，则创建一个新的卷积层self.conv，其参数根据之前的ch_in和ch_out进行设置。接下来，通过调用self._fuse_bn_tensor方法获取等效的卷积核和偏置项，并将其赋值给self.conv的权重和偏置。最后，使用__delattr__方法删除conv1和conv2属性，完成转换。

```
    def convert_to_deploy(self):
        if not hasattr(self, 'conv'):
            self.conv = nn.Conv2D(
                in_channels=self.ch_in,
                out_channels=self.ch_out,
                kernel_size=3,
                stride=1,
                padding=1,
                groups=1)
        kernel, bias = self.get_equivalent_kernel_bias()
        self.conv.weight.set_value(kernel)
        self.conv.bias.set_value(bias)
        self.__delattr__('conv1')
        self.__delattr__('conv2')
```

get_equivalent_kernel_bias方法用于获取等效的卷积核和偏置项。在该方法中，首先对conv1和conv2属性调用_fuse_bn_tensor方法进行融合，得到对应的卷积核和偏置项。然后，通过调用_pad_1x1_to_3x3_tensor方法将kernel1x1的尺寸从1x1扩展到3x3，并返回融合后的卷积核和偏置项。

```
    def get_equivalent_kernel_bias(self):
        kernel3x3, bias3x3 = self._fuse_bn_tensor(self.conv1)
        kernel1x1, bias1x1 = self._fuse_bn_tensor(self.conv2)
        return kernel3x3 + self._pad_1x1_to_3x3_tensor(
            kernel1x1), bias3x3 + bias1x1
```

给出的代码片段中缺少了一些关键的实现细节，比如_fuse_bn_tensor和_pad_1x1_to_3x3_tensor方法的具体实现。这些方法的实现可能涉及到对Batch Normalization的处理和卷积核尺寸的调整。因此，为了完整理解和使用RepVggBlock类，需要查看完整的实现代码。

ET-head（高效Transformer头部）

为提升速度与性能，基于TOOD中的T-head进行改进，提出了ET-head。使用ESE层替换了T-head中的layer attention，并将回归分支的对齐简化为Distribution Focal Loss(DFL)层，其使用的的激活函数为SiLU(Swish)。通过这些改变，ET-Head在V100上提升了0.9ms。

PPYOLOE+

PPYOLOE+表示在object365中进行了预训练（其模型结构配置文件与PPYOLOE一模一样，只是在backbone中block分支中增加alpha参数）的PPYOLOE模型。两个模型在ATSSAssigner与TaskAlignedAssigner的epoch数上存在不同，ppyoloe的static_assigner_epoch为100，ppyoloe+的为30[经过预训练后ppyoloe+对ATSSAssigner的依赖降低]。预训练可以使模型在更大规模的数据上进行学习，从而提取更丰富和泛化的特征表示。

模型特点

2.1 锚框机制

Anchor-free（Anchor-base模型引入超参数，依赖手工设计，对不同的数据集需要单独聚类），在每个像素上放置一个锚点，为三个检测头设置GT尺寸的上届和下界。计算GT的中心，选择最近的锚点做正样本。Anchor-free方式使mAP比Anchor-base下降0.3，但是速度有所提升，具体如下表所示。

Model	mAP(%)	Parameters(M)	GFLOPs	Latency(ms)	FPS
PP-YOLOv2 baseline model	49.1	54.58	115.77	14.5	68.9
+Anchor-free	48.8 (−0.3)	54.27	114.78	14.3	69.8
+CSPRepResNet	49.5 (+0.7)	47.42	101.87	11.7	85.5
+TAL	50.4 (+0.9)	48.32	104.75	11.9	84.0
+ET-head	**50.9** (+0.5)	52.20	110.07	12.8	78.1

Table 2: Ablation study of PP-YOLOE-l on COCO *val*. We use 640×640 resolution as input with FP32-precision, and test on Tesla V100 without post-processing.

2.2label assign方式

使用TOOD中的TAL(Task Aligned Learing),显性的对齐分类最优点和位置回归最优点。TOOD论文中，提出任务对齐头部（T-Head）和任务对齐学习（TAL）。T-head在学习任务交互特征和任务特定特征之间提供了更好的平衡，并通过任务对齐预测器学习对齐的灵活性提高,TAL通过设计的样本分配方案和任务对齐损失，明确地拉近（甚至统一）两个任务的最优锚点 TAL包含任务对齐指标、正样本选择标准，并将任务对齐指标与原先的的分类损失、回归损失进行联立[其本质就是根据任务对齐指标调节不同样本在反向传播时loss的权重，PP-YOLOE其实也尝试过多种标签对齐方式，具体如下所示，可见TAL效果是最佳的。

Method	mAP(0.5:0.95)
ATSS[34]	43.1
SimOTA[6]	44.3
TAL[5]	**45.2**

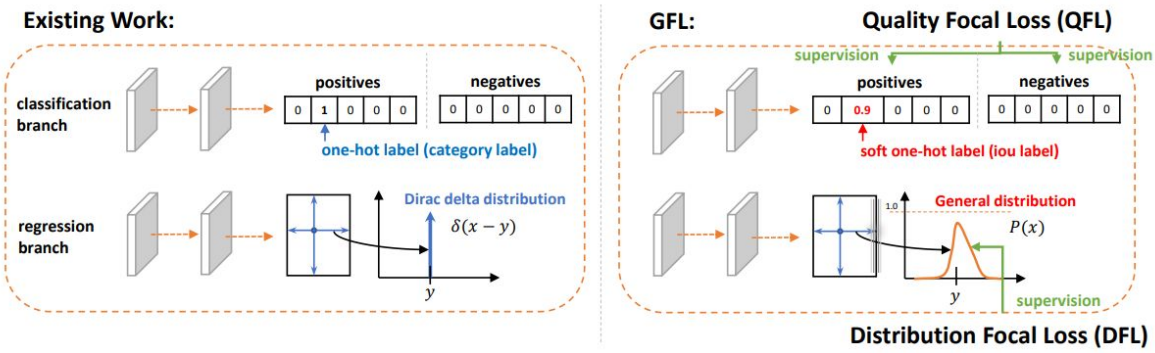
Table 3: Different label assignment on base model. We use CSPRepResStage as backbone and neck, one 1×1 conv layer as head, and only train 36 epochs on COCO *train2017*.

2.3 loss设计

对于分类和定位任务，分别选择了varifocal loss（VFL）和distribution focal loss（DFL）。PP-Picodet成功将VFL和DFL引入到目标检测中。VFL与quality focal(QFL)不同，VFL使用目标评分来衡量正样本loss的权重（可提升正样本loss的贡献，使模型更多关注高质量正样本，解决了NMS过程中classification score 和 IoU/centerness score 训练测试不一致[训练时两个孤立，nms时两个联立]），两者都使用带有IOU感知的分类评分作为预测目标。整体loss设计如下所示，使用ET-head提升了0.5的map。

$$Loss = \frac{\alpha \cdot \text{loss}_{VFL} + \beta \cdot \text{loss}_{GIoU} + \gamma \cdot \text{loss}_{DFL}}{\sum_{i}^{N_{pos}} \hat{t}}$$

DFL（distribution focal loss）：为了解决bbox不灵活的问题，提出使用distribution[将迪克拉分布转化为一般分布]预测bbox[预测top、left、right、bottom]。



2.4 训练细节

使用带动量的SGD，其中momentum为0.9，权重衰减为5e-4，使用余弦学习率调度器，总共epoch为300，预热epoch为5。学习率为0.01，batchsize为64,8个32G的V100多卡训练。训练过程中使用decay=0.9998的EMA策略。只使用了基本的数据增强，包括随机裁剪、随机水平翻转、颜色失真和多尺度（尺度范围为320到768，步长为32），测试尺度为640。具体性能对比如下标所示。

Method	Backbone	Size	FPS (v100)		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
			w/o TRT	with TRT						
YOLOv3 + ASFF* [17]	Darknet-53	320	60	-	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF* [17]	Darknet-53	416	54	-	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv4 [2]	CSPDarknet-53	416	96	-	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4 [2]	CSPDarknet-53	512	83	-	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4-CSP [27]	Modified CSPDarknet-53	512	97	-	46.2%	64.8%	50.2%	24.6%	50.4%	61.9%
YOLOv4-CSP [27]	Modified CSPDarknet-53	640	73	-	47.5%	66.2%	51.7%	28.2%	51.2%	59.8%
EfficientDet-D0 [25]	Efficient-B0	512	98.0	-	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1 [25]	Efficient-B1	640	74.1	-	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2 [25]	Efficient-B2	768	56.5	-	43.0%	62.3%	46.2%	22.5%	47.0%	58.4%
EfficientDet-D2 [25]	Efficient-B3	896	34.5	-	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
PP-YOLO [18]	ResNet50-vd-dcn	320	132.2^+	242.2^+	39.3%	59.3%	42.7%	16.7%	41.4%	57.8%
PP-YOLO [18]	ResNet50-vd-dcn	416	109.6^+	215.4^+	42.5%	62.8%	46.5%	21.2%	45.2%	58.2%
PP-YOLO [18]	ResNet50-vd-dcn	512	89.9^+	188.4^+	44.4%	64.6%	48.8%	24.4%	47.1%	58.2%
PP-YOLO [18]	ResNet50-vd-dcn	608	72.9^+	155.6^+	45.9%	65.2%	49.9%	26.3%	47.8%	57.2%
PP-YOLOv2 [13]	ResNet50-vd-dcn	320	123.3	152.9	43.1%	61.7%	46.5%	19.7%	46.3%	61.8%
PP-YOLOv2 [13]	ResNet50-vd-dcn	416	102^+	145.1^+	46.3%	65.1%	50.3%	23.9%	50.2%	62.2%
PP-YOLOv2 [13]	ResNet50-vd-dcn	512	93.4^+	141.2^+	48.2%	67.1%	52.7%	27.7%	52.1%	62.1%
PP-YOLOv2 [13]	ResNet50-vd-dcn	640	68.9^+	106.5^+	49.5%	68.2%	54.4%	30.7%	52.9%	61.2%
PP-YOLOv2 [13]	ResNet101-vd-dcn	640	50.3^+	87.0^+	50.3%	69.0%	55.3%	31.6%	53.9%	62.4%
YOLOv5-s [14]	Modified CSP v6	640	156.2^+	454.5*	37.4%	56.8%	-	-	-	-
YOLOv5-m [14]	Modified CSP v6	640	121.9^+	263.1*	45.4%	64.1%	-	-	-	-
YOLOv5-l [14]	Modified CSP v6	640	99.0^+	172.4*	49.0%	67.3%	-	-	-	-
YOLOv5-x [14]	Modified CSP v6	640	82.6^+	117.6*	50.7%	68.9%	-	-	-	-
YOLOX-s [6]	Modified CSP v5	640	119.0* | 102.0^+	246.9*	40.5%	-	-	-	-	-
YOLOX-m [6]	Modified CSP v5	640	96.1* | 81.3^+	177.3*	47.2%	-	-	-	-	-
YOLOX-l [6]	Modified CSP v5	640	62.5* | 68.9^+	120.1*	50.1%	-	-	-	-	-
YOLOX-x [6]	Modified CSP v5	640	40.3* | 57.8^+	87.4*	51.5%	-	-	-	-	-
PP-YOLOE-s	CSPRepResNet	640	208.3	333.3	**43.1%**	**60.5%**	**46.6%**	**23.2%**	**46.4%**	**56.9%**
PP-YOLOE-m	CSPRepResNet	640	123.4	208.3	**48.9%**	**66.5%**	**53.0%**	**28.6%**	**52.9%**	**63.8%**
PP-YOLOE-l	CSPRepResNet	640	78.1	149.2	**51.4%**	**68.9%**	**55.6%**	**31.4%**	**55.3%**	**66.1%**
PP-YOLOE-x	CSPRepResNet	640	45.0	95.2	**52.2%**	**69.9%**	**56.5%**	**33.3%**	**56.3%**	**66.4%**
PP-YOLOE+-s	CSPRepResNet	640	208.3	333.3	**43.7%**	**60.6%**	**47.9%**	**26.5%**	**47.5%**	**59.0%**
PP-YOLOE+-m	CSPRepResNet	640	123.4	208.3	**49.8%**	**67.1%**	**54.5%**	**31.8%**	**53.9%**	**66.2%**
PP-YOLOE+-l	CSPRepResNet	640	78.1	149.2	**52.9%**	**70.1%**	**57.9%**	**35.2%**	**57.5%**	**69.1%**
PP-YOLOE+-x	CSPRepResNet	640	45.0	95.2	**54.7%**	**72.0%**	**59.9%**	**37.9%**	**59.3%**	**70.4%**

关键步骤实现

3.1 网络细节与TODO的差异

ppyoloe的下采样倍数为32, 16, 8，可以修改backbone、neck、head的输出chanel，增加输出级别以适应不同尺度的目标检测。其所设计的backbone、neck、head本质上是可以替换的，只是区别于TODO（其在resnetXt101上最佳map为48.3，在resnetXt101-dcn上最佳map为51.1，预计是没有使用PAN[PAN论文中可以将MAsk R-CNN提升3-5个点]，且resnetXt101与CSPReResnet存在差异[CSPReResNet在参数量上分别有CSP和REP上的优势]，然后再bbox回归中ppyoloe使用的dfl和GIOU），在backbone上做了修改，在T-head上做了轻量化。

```
architecture: YOLOv3
norm_type: sync_bn
use_ema: true
ema_decay: 0.9998
ema_black_list: ['proj_conv.weight']
custom_black_list: ['reduce_mean']

YOLOv3:
  backbone: CSPResNet
  neck: CustomCSPPAN
  yolo_head: PPYOLOEHead
  post_process: ~

CSPResNet:
  layers: [3, 6, 6, 3]
  channels: [64, 128, 256, 512, 1024]
  return_idx: [1, 2, 3]
  use_large_stem: True
```

```
CustomCSPPAN:
  out_channels: [768, 384, 192]
  stage_num: 1
  block_num: 3
  act: 'swish'
  spp: true

PPYOLOEHead:
  fpn_strides: [32, 16, 8]
  grid_cell_scale: 5.0
  grid_cell_offset: 0.5
  static_assigner_epoch: 100
  use_varifocal_loss: True
  loss_weight: {class: 1.0, iou: 2.5, dfl: 0.5}
  static_assigner:
    name: ATSSAssigner
    topk: 9
  assigner:
    name: TaskAlignedAssigner
    topk: 13
    alpha: 1.0
    beta: 6.0
  nms:
    name: MultiClassNMS
    nms_top_k: 1000
    keep_top_k: 300
    score_threshold: 0.01
    nms_threshold: 0.7
```

3.2 SMLX版本

PPYOLOE存在S、M、L、X等版本，是由depth_mult和width_mult两个参数控制模型的深度，故此可以通过修改depth_mult和width_mult得到其他版本的PPYOLOE模型。

在以下配置文件中，depth_mult用于控制layers内(backbone中stage的深度，默认为[3,6,6,3])的参数(故其最小值为1/3)，width_mult用于控制channels内(stem和backbone中stages的宽度)的参数。

```
CSPResNet:
  layers: [3, 6, 6, 3]
  channels: [64, 128, 256, 512, 1024]
  return_idx: [1, 2, 3]
  use_large_stem: True
  use_alpha: True

CustomCSPPAN:
  out_channels: [768, 384, 192]
  stage_num: 1
  block_num: 3
  act: 'swish'
  spp: true
  use_alpha: True

depth_mult: 0.33
width_mult: 0.50
```

模型训练及参数调整：

使用百度aistudio的PaddleX来进行数据校验、模型训练、评估测试等。
数据校验：导入标注好的数据和原数据。
模型训练：完成数据校验后对模型进行训练，需要调整的参数：Epochs，Batch Size，Learning Rate

1. **Epochs（训练轮数）**：Epochs指的是将整个训练数据集完整地通过神经网络训练一次的次数。调整Epochs的值可以影响模型的训练时间和性能。较小的Epochs可能导致模型欠拟合，而较大的Epochs可能导致模型过拟合。通常建议进行实验并观察验证集上的性能来确定合适的Epochs值。如果验证集上的性能不再提升，可以提前停止训练。
2. **Batch Size（批量大小）**：Batch Size是指每次输入给模型的样本数量。调整Batch Size可以影响模型的收敛速度和内存使用情况。较小的Batch Size可以提高模型的收敛速度，但可能导致训练过程中的噪声较大。较大的Batch Size可以减少训练过程中的噪声，但可能需要更多的内存。一般来说，较常用的Batch Size取值是32、64、128等。需要根据具体情况进行实验和调整。
3. **Learning Rate（学习率）**：Learning Rate控制模型参数在每次迭代中更新的步长大小。调整Learning Rate可以影响模型的收敛速度和准确度。较大的Learning Rate可能导致模型无法收敛或发散，而较小的Learning Rate可能导致模型收敛速度较慢。常用的策略是从一个较大的Learning Rate开始训练，然后随着训练的进行逐渐降低Learning Rate。可以使用学习率衰减（Learning Rate Decay）或自适应学习率算法（如Adam、Adagrad等）来调整学习率。

在调整这些参数时，进行实验和观察模型在验证集上的性能，并进行适当的调整，以找到最佳的参数组合。还可以使用一些自动化的调参工具或采用网格搜索、随机搜索等方法来寻找最佳参数组合。此外，根据具体问题和数据集的特点，可能还需要考虑其他参数的调整，例如正则化项、优化器选择等。

Epochs：50；Batch Size：8；Learning Rate：0.0001

模型评估：指标mAP（根据多个交并比（IoU）阈值计算出的精度平均值， 综合反映算法在不同IoU阈值下的性能， 其中IoU阈值取值0.5-0.95，以0.05为步长， 值越大，表示模型性能越好）

mAP：0.902

测试结果：


overflow 0.98
overflow 0.95
garbage_bigarbage_bin 0.96
可回收物
请将垃圾
分类入桶
其它垃圾
Other waste