PP-PicoDet_LCNet

PP-PicoDet 是 PaddleDetection 团队提出的超轻量级实时目标检测模型，在移动端具有良好的性能，其具有以下特点：

- 更高的 mAP: 第一个在1M参数量之内 mAP(0.5:0.95) 超越 30+（输入416像素时）；
- 更快的预测速度: 网络预测在 ARM CPU 下可达 150FPS；
- 部署友好: 支持 PaddleLite/MNN/NCNN/OpenVINO 等预测库，支持转出 ONNX，提供了 C++/Python/Android 的 demo；
- 先进的算法: 我们在现有 SOTA 算法中进行了创新, 包括：ESNet, CSP-PAN, SimOTA 等等。

模型及论文解析：

参照论文《PP-PicoDet: A Better Real-Time Object Detector on Mobile Devices》

摘要：

工作内容：提高目标检测的精度和效率。

1、研究无锚点（anchor-free）策略对轻量级目标检测模型的适用性，增强骨干结构并设计颈部轻量型结构，从而提高网络特征提取能力；

2、改进标签分配策略和损失函数，提高训练的稳定性和效率。

通过上述优化，项目创建了一个新的实时目标检测器系列：**PP-PicoDet**。

PP-PicoDet在模型的准确性和延迟之间做了更好的权衡，PicoDet-S在仅有0.99M个参数的基础上实现了30.6%的mAP，相较于YOLOX-Nano，mAP 绝对值提高了 4.8%，同时可移动 CPU 的推理延迟减少了55%，同时与NanoDet相比，mAP绝对值提高了7.1%。当输入大小为320时，其在可移动ARM CPU上达到了123 FPS（使用 Paddle Lite 时为 150 FPS）。PicoDet-L仅用了3.3M的参数量，实现了40.9% 的 mAP，比 NanoDet 的 mAP 绝对值提高了 3.7%，比 YOLOv5s 快 44%。

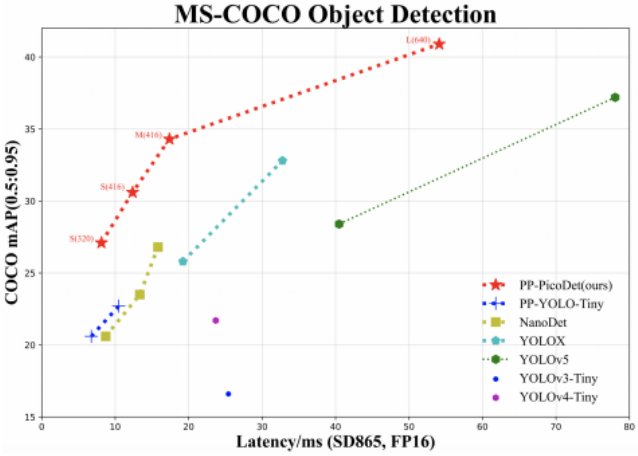


Figure 1. Comparison of the mAPs of different lightweight models. The latency of all models tested on Qualcomm Snapdragon® 865(4*A77+4*A55) Processor with batch size of 1. The details are presented in Table 1.

介绍：

文章的主要贡献：

1、使用CSP结构来构建CSP-PAN作为neck部分。CSP-PAN用1×1积将neck所有分支的输入通道数全部统一为相同的通道数，从而能够有效的增强网络的特征提取能力，且减少网络参数。同时，将3×3可分离卷积增加到5×5可分离卷积，从而增大感受野；

2、标签分配策略在目标检测中十分重要，文章使用SimOTA动态标签分配策略，并优化了一些计算细节；

3、ShuffleNetV2在移动端上计算是高效的，文章在此基础上改进了网络结构并提出了一种新的主干网络，称为 Enhanced ShuffleNet (ESNet)，性能优于ShuffleNetV2；

4、文章提出一种改进的用于检测的 One-Shot Neural Architecture Search (NAS)方法来自动找到目标检测的最优架构。文章在检测数据集上训练超网络，从而显着节省计算量并优化检测性能，NAS生成的模型实现了更好的效率精度均衡。

相关工作：

主要介绍了两类目标检测器：

- anchor-based检测器：Two-stage检测器通常都是anchor-based，会从图像中产生region-proposals，并从region-proposals产生最终的定位框。Two-stage检测器在目标定位上常常更加精确，然而很难在CPU或ARM设备上实现实时检测；
- anchor-free检测器：旨在消除锚框，是目标检测研究中的重要进展。Anchor-free检测器解决了一些anchor-based检测器存在的问题，降低了内存占用，并且对定位框的计算更加准确。

方法：

更好的主干网络：

新的主干网络ESNet，其网络模块ES-Block结构如下：

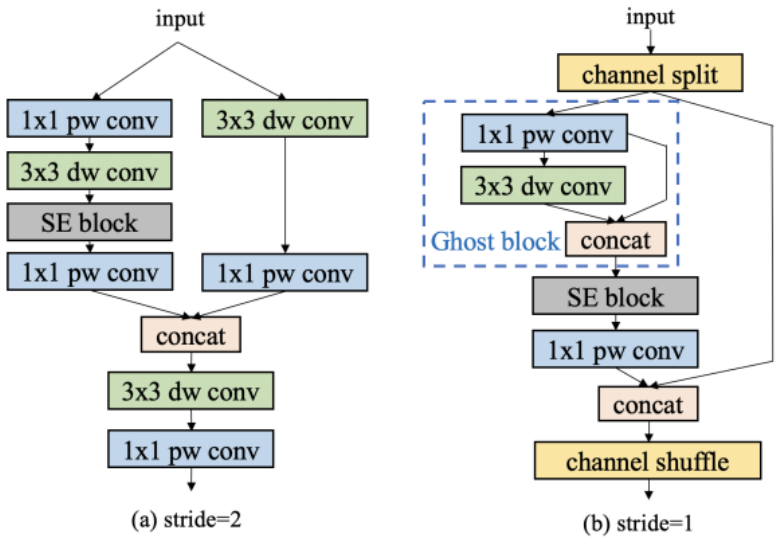


Figure 3. ES Block architecture. (a) ES Block with stride=2; (b) ES Block with stride=1.

SE模块能很好地对网络通道进行加权并获得更好的特征，因此在ES模块中加入了SE模块。在stride为2时加入了 depthwise convolution和 pointwise convolution来结合不同通道的信息，stride为2时加入Ghost模块，使其在较少参数量下产生更多特征图，从而提升网络的学习能力，进而增强ESNet的性能。

Neural Architecture Search（NAS）：针对目标检测器的one-shot搜索方法，共分为两步：（1）在检测数据集上训练one-shot超网；（2）使用EA（evolutionary algorithm，进化算法）算法对训练好的超网络进行架构搜索。

CSP-PAN和 detector head：

模型使用PAN结构来获得多层特征图以及CSP结构来进行相邻特征图间的特征连接和融合。对于原始的CSP-PAN，每个输出特征图的通道数与来自主干网络的输入特征图需保持相同，这对于移动设备来说成本较高。为此，模型使用1×1卷积使所有特征图中的通道数与最小的通道数相等，然后通过CSP结构实现top-down和bottom-up的特征融合，缩小的特征使得计算成本更低且不损失准确性。此外，模型在原有CSP-PAN的顶部加入了一个特征图尺度分支来检测更多物体。

所有除了1×1卷积之外的卷积层都使用Depthwise Separable Convolution（深度可分离卷积），这种结构可以扩大感受野，在使用很少参数的情况下给精度带来了大幅度提升。

相关模型的结构如下：

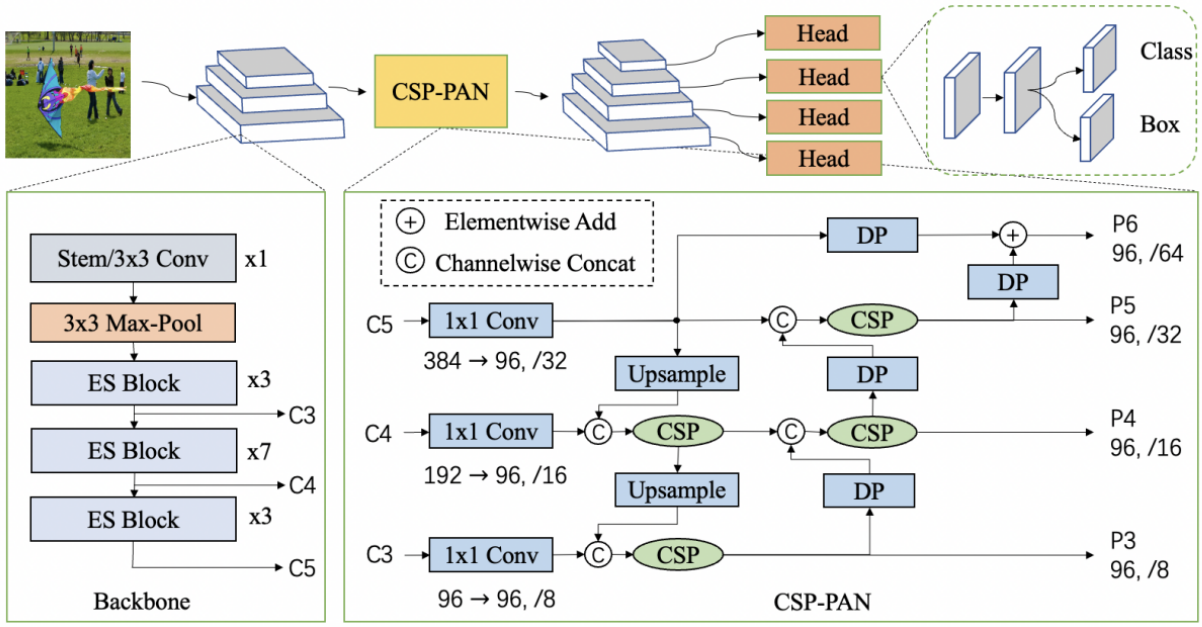


Figure 2. PP-PicoDet Architecture. The Backbone is ESNet, which outputs C3-C5 feature maps to the neck. The neck is CSP-PAN, which inputs three feature maps and outputs four feature maps. For PP-PicoDet-S, the input channel numbers are [96, 192, 384], and the output channel numbers are [96, 96, 96, 96]. DP module uses depthwise and pointwise convolution.

标签分配策略和损失函数：

模型使用SimOTA动态标签分配策略来优化训练过程。SimOTA是一种会随着训练进程持续变化的标签分配策略，首先SimOTA通过中心点先验信息决定候选区域，之后计算候选区内真值跟预测框的IoU，最后对每个真值框将n个最大的IoU求和获得参数κ。通过直接计算候选区内所有预测框跟真值框的损失作为代价矩阵，对于每个真值框，选最小κ的损失对应的锚点作为正样本。

模型使用Varifocal-Loss和GIoU-Loss的加权和作为代价矩阵，GIoU-loss的权重为λ 模型中设置为6。相关公式为：

$$\text{cost} = \text{loss}_{\text{vfl}} + \lambda \cdot \text{loss}_{\text{giou}}$$

在检测head中，对于分类，模型使用Varifocal-Loss来耦合分类预测和置信度预测。对于回归，模型使用GIoU-Loss和Distribution-Focal-Loss。相关公式为：

$$\text{loss} = \text{loss}_{\text{vfl}} + 2 \cdot \text{loss}_{\text{giou}} + 0.25 \cdot \text{loss}_{\text{dfl}}$$

上述公式中，loss_{vfl}为Varifocal-Loss，$\text{loss}_{\text{giou}}$指GIoU-loss，$\text{loss}_{\text{dfl}}$指Distribution-Focal-Loss。

其他策略：

模型将检测器中的激活函数从ReLU替换为H-Swish，在保持推理时延不变的情况下性能大幅度提升。

同时，不同于线性步长学习衰减，余弦学习衰减会以指数形式衰减学习率。当batch-size较大的时候，余弦学习率会平缓地下降，从而对训练过程更有利。

过多的数据增广会提高正则化效果而使得轻量模型更难以收敛，为此，模型只使用了随机翻转、随机裁剪和多尺度缩放作为训练中的数据增广。

实验：

实验细节：

训练时，模型使用随机梯度下降（SGD），并设置动量0.9，权重衰减为4e-5。同时，模型使用了余弦衰减的学习率策略，初始学习率为0.1，Batch-size默认为80x8。由于轻量型模型容易陷入局部最优而难以收敛，模型引入了一种类似于正则化的机制，称为Cycle-EMA，用于重置历史信息，由 forget step来控制。同时，模型使用L2-norm梯度裁剪来避免梯度爆炸。

消融实验：

Model	Params(M)	mAP(0.5:0.95)
Base	0.96	25.3
+CSP-PAN (3 feature maps)	1.12	28.1
+CSP-PAN (4 feature maps)	1.17	29.1
+Replace QFL with VFL	1.17	29.2
+Original SimOTA	1.17	29.2
+SimOTA with modified cost matrix	1.17	30.0
+Replace backbone with ESNet-0.75x	0.99	29.7
+Replace LeakyRelu with H-Swish	**0.99**	**30.6**

Table 2. Different configurations of ablation experiments in PP-PicoDet-S.

Base：主干网络使用ShuffleNetV2-1x，neck部分使用不包含卷积的PAN，损失函数使用标准GFL的损失函数，还有标签分配策略使用ATSS，所有的激活函数使用LeakyRelu。

模型进一步比较了Varifocal-Loss和GIoU-Loss使用不同权重时SimOTA的效果，通过改变公式中的λ值来寻找其最优值，相关结果如下：

λ	mAP(0.5:0.95)
5	29.8
6	30.0
7	29.8

可以看到，当GIoU-loss的权重为6时，获得了最佳的结果。

模型在ImageNet-1k上比较了ESNet-1x与原始ShuffleNetV2-1.5x网络，结果如下：

model	FLOPs (M)	Latency (ms)	Top-1 Acc (%)
ShuffleNetV2-1.5x	301	7.56	71.6
ESNet-1x	**197**	**7.35**	**73.9**

结果表明，在推理时间更低的情况下，ESNet实现了更高的精度。

模型比较了原始模型和搜索模型的性能，结果如图：

model	Params (M)	Latency (ms)	mAP (0.5:0.95)
original	2.35	24.6	34.5
searched	**2.15 (-9.3%)**	**17.39 (-41.5%)**	34.3 (-0.2)

搜索模型在实验限制下进降低了0.2%mAP，而移动端CPU推理时间提升了41.5%。

与SOTA算法对比：

Model	Size	Params(M)	FLOPs(G)	mAP(0.5:0.95)	mAP(0.5)	Latency(ms)
YOLOv3-Tiny	416	8.86	5.62	16.6	33.1	25.42
YOLOv4-Tiny	416	6.06	6.96	21.7	40.2	23.69
MobileDet-CPU	320	3.85	1.02	24.2	-	-
YOLObile	320	4.59	3.59	31.6	49.0	-
PP-YOLO-Tiny	320	1.08	0.58	20.6	-	6.75
PP-YOLO-Tiny	416	1.08	1.02	22.7	-	10.48
NanoDet-M	320	0.95	0.72	20.6	-	8.71
NanoDet-M	416	0.95	1.2	23.5	-	13.35
NanoDet-M-1.5x	416	2.08	2.42	26.8	-	15.83
YOLOX-Nano	416	0.91	1.08	25.8	-	19.23
YOLOX-Tiny	416	5.06	6.45	32.8	-	32.77
YOLOv5n	640	1.9	4.5	28.4	46.0	40.35
YOLOv5s	640	7.2	16.5	37.2	56.0	78.05
PP-PicoDet-ShuffleNetV2	416	1.17	1.53	30.0	44.6	15.06 | **10.63***
PP-PicoDet-MV3-large-1x	416	3.55	2.80	35.6	52.0	20.71 | **17.88***
PP-PicoDet-LCNet-1.5x	416	3.10	3.85	36.3	52.2	21.29 | **20.8***
PP-PicoDet-S	320	0.99	0.73	**27.1**	**41.4**	8.13 | **6.65***
PP-PicoDet-S	416	0.99	1.24	**30.6**	**45.5**	12.37 | **9.82***
PP-PicoDet-M	416	2.15	2.50	**34.3**	**49.8**	17.39 | **15.88***
PP-PicoDet-L	640	3.30	8.91	**40.9**	**57.6**	54.11 | **50.55***

从图表我们可以看到：模型在精度和速度上都大大超过了所有的YOLO模型。

上述效果主要归功于以下改进：

（1）模型的neck部分要比YOLO系列的neck要更加轻量，所以主干和head能分配更多参数。（2）模型使用针对类别不平衡的Varifocal-Loss、动态可学习的样本分配策略和基于FCOS的回归方法的组合，这种组合在轻量级模型中表现更好。在相同数量参数的情况下，PP-PicoDet-S在mAP和时延上都超越了YOLOX-Nano和NanoDet。PP-PicoDet-L的mAP和时延都优于YOLOv5s。由于使用汇编语言优化的卷积算子效率更高，模型在Paddle Lite上的推理时间甚至优于在NCNN上的推理时间。

源码解析：

所有代码来源于：GitHub - PaddlePaddle/PaddleDetection

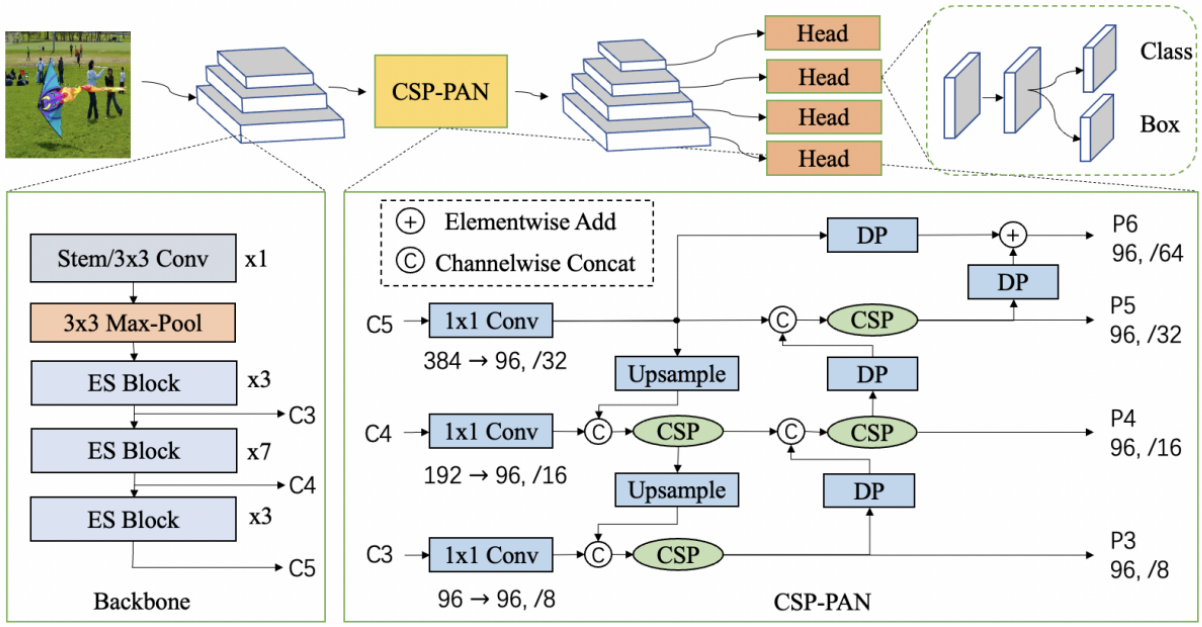


Figure 2. PP-PicoDet Architecture. The Backbone is ESNet, which outputs C3-C5 feature maps to the neck. The neck is CSP-PAN, which inputs three feature maps and outputs four feature maps. For PP-PicoDet-S, the input channel numbers are [96, 192, 384], and the output channel numbers are [96, 96, 96, 96]. DP module uses depthwise and pointwise convolution.

ES Block

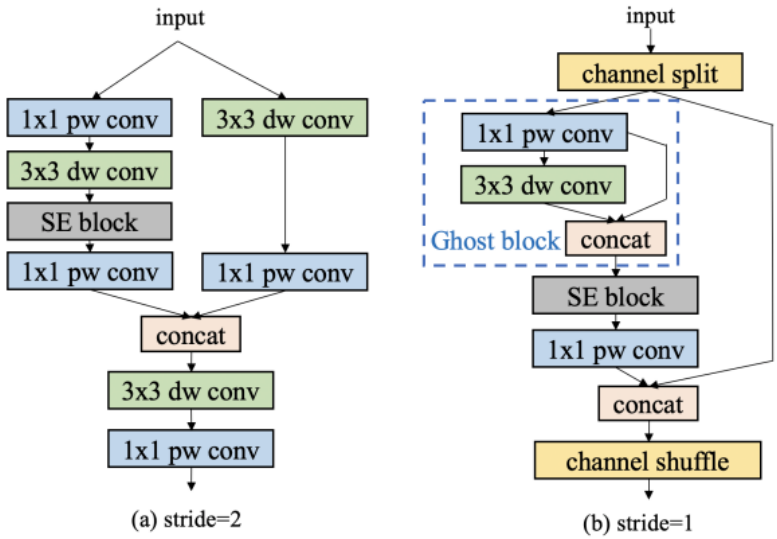
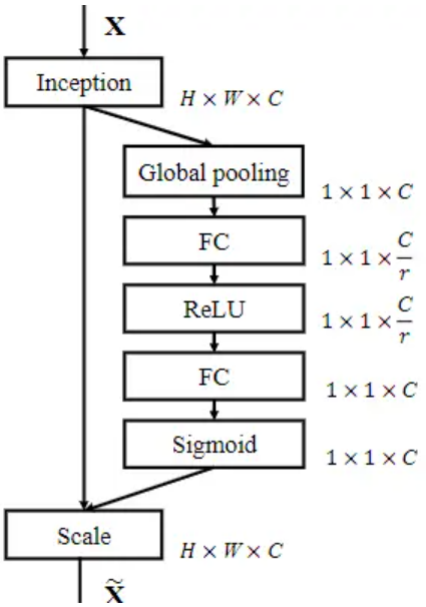


Figure 3. ES Block architecture. (a) ES Block with stride=2; (b) ES Block with stride=1.

ES模块中的主要部分是SE Block。



SE Block 结构

其核心思想是对通道的权重进行调整。对于右边分支，其通过对通道的下采集，RELU激活，上采样后，接一个simoid操作来得到各个通道的权重比例值，然后再将这个比例乘以原通道，实现对权重的调整。

```
class SEModule(nn.Layer):
    def __init__(self, channel, reduction=4):
        super(SEModule, self).__init__()
        self.avg_pool = AdaptiveAvgPool2D(1)
        self.conv1 = Conv2D(
            in_channels=channel,
            out_channels=channel // reduction,
            kernel_size=1,
            stride=1,
            padding=0,
            weight_attr=ParamAttr(),
            bias_attr=ParamAttr())

        self.conv2 = Conv2D(
            in_channels=channel // reduction,
            out_channels=channel,
            kernel_size=1,
            stride=1,
            padding=0,
            weight_attr=ParamAttr(),
            bias_attr=ParamAttr())

    def forward(self, inputs):
        outputs = self.avg_pool(inputs)
```

```
        outputs = self.conv1(outputs)
        outputs = F.relu(outputs)
        outputs = self.conv2(outputs)
        outputs = F.hardsigmoid(outputs)
        print('scale:',outputs)
        return paddle.multiply(x=inputs, y=outputs)
```

DW

深度可分离卷积，主要作用是减少卷积参数，提高模型效率。

```
self._conv_dw = ConvBNLayer(
    in_channels=mid_channels // 2,
    out_channels=mid_channels // 2,
    kernel_size=3,
    stride=stride,
    padding=1,
    groups=mid_channels // 2,
    act=None)
```

PW

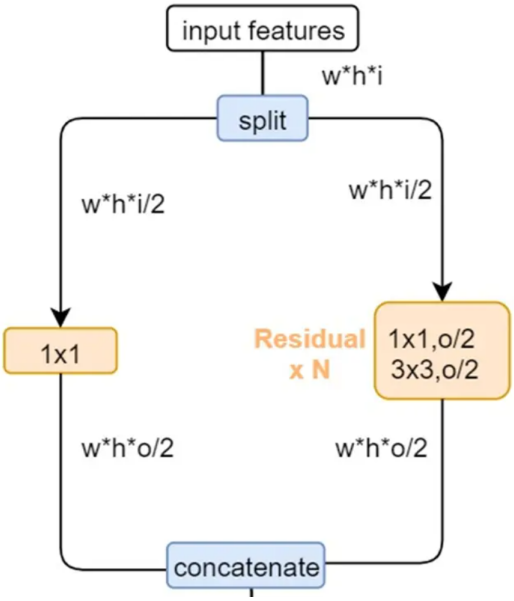
PW是1x1的卷积，主要配合DW来使用，来进行通道间的计算，实现对通道数的调整。同时，DW+PW的参数量与常规卷积相比较小。

Backbone

从结构图我们可以看到，Backbone就是ES Block的堆叠，然后取4，11，14层作为Neck层输入。

CSP

CSP的结构如下：



其核心思想是对通道进行切分，对于切分后的通道，CSP对右侧部分进行卷积层操作，然后与与左侧部分进行拼接，这样可以降低操作量。

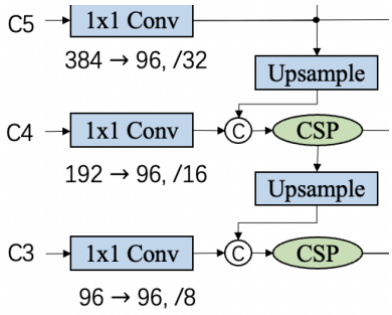
```
class CSPLayer(nn.Layer):
    """Cross Stage Partial Layer.

    Args:
        in_channels (int): The input channels of the CSP layer.
        out_channels (int): The output channels of the CSP layer.
        expand_ratio (float): Ratio to adjust the number of
channels of the
            hidden layer. Default: 0.5
        num_blocks (int): Number of blocks. Default: 1
        add_identity (bool): Whether to add identity in blocks.
            Default: True
        use_depthwise (bool): Whether to depthwise separable
convolution in
            blocks. Default: False
    """

    def __init__(self,
                 in_channels,
                 out_channels,
                 kernel_size=3,
                 expand_ratio=0.5,
                 num_blocks=1,
                 add_identity=True,
                 use_depthwise=False,
                 act="leaky_relu"):
```

```
        super().__init__()
        # *0.5,即通道数split成2份一样大小
        mid_channels = int(out_channels * expand_ratio)
        self.main_conv = ConvBNLayer(in_channels, mid_channels, 1,
act=act)
        self.short_conv = ConvBNLayer(in_channels, mid_channels, 1,
act=act)
        self.final_conv = ConvBNLayer(
            2 * mid_channels, out_channels, 1, act=act)

        self.blocks = nn.Sequential(* [
            DarknetBottleneck(
                mid_channels,
                mid_channels,
                kernel_size,
                1.0,
                add_identity,
                use_depthwise,
                act=act) for _ in range(num_blocks)
        ])

    def forward(self, x):
        # 左边路径中的split
        x_short = self.short_conv(x)
        # 右边路径中的split + 常规卷积
        x_main = self.main_conv(x)
        x_main = self.blocks(x_main)
        # concat 合并通道然后再接一层卷积
        x_final = paddle.concat((x_main, x_short), axis=1)
        return self.final_conv(x_final)
```

在代码中，CSP在进行通道切分并不是直接用split切分，而是通过1×1卷积来对通道数进行调整，这样可以融合所有通道的信息，在保证精度的情况下减少参数量。

FPN

FPN的结构如下，其实质上就是upsample 的过程。



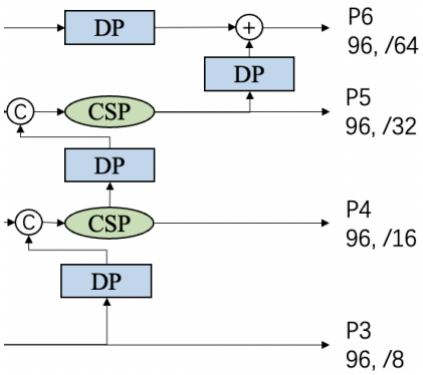
1、将backbone 输出的3层输出通道数通过1x1卷积改为96；

2、特征层从上到下（top-down），先将FM的尺寸进行upsample，然后和下一层进行concate连接，连接之后通道数变为96+96=192；

3、concate之后，经过CSP结构，通道数再次变为96,作为FPN的输出，PAN的输入。

PAN

FPN是top-down 的操作， PAN正好相反，是bottom-up操作，同时，将upsample的操作改成DP（DW+PW）进行卷积。



上述CSP、FPN、PAN部分的代码在`PaddleDetection\ppdet\modeling\necks\csp_pan.py`中。

Head

Head部分将将CSP-PAN的输出作为Head 的stage 输入。

模型训练及参数调整：

训练流程的代码：

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

Epochs：100；Batch Size：66；Learning Rate：0.1

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阈值取值为0.5-0.95，以0.05为步长， 值越大，表示模型性能越好）

mAP：0.736

测试结果：