6、YOLOX骨干网backbone-PAFPN网络结构示意图（结合代码）

LayH
I'm still learning

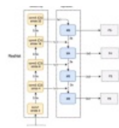
1、前置知识

FPN：Feature pyramid network，特征金字塔网络。

PAN: Path Aggregation network，路径聚合网络。

关于FPN和PAN可以参考：

《YOLOV4&5原理与源代码解析之七：PANet模块》 - JackRuiYu - 博客园
www.cnblogs.com/winslam/p/14486803.h···



YOLO的骨干网主要是借助PA-FPN的结构将不同层次的特状图进行高效融合。

PA（Path Aggregation）的策略使得不同层次的特征在传递时需要“穿越”的网络层次数量大大减少。

关于基本的网络blocks和作用的解释：

LayH：4、Focus模块-in YOLO
6 赞同 · 4 评论 文章

CSP模块：将feature map拆成两个部分，一部分进行卷积操作，另一部分和上一部分卷积操作的结果进行concate。

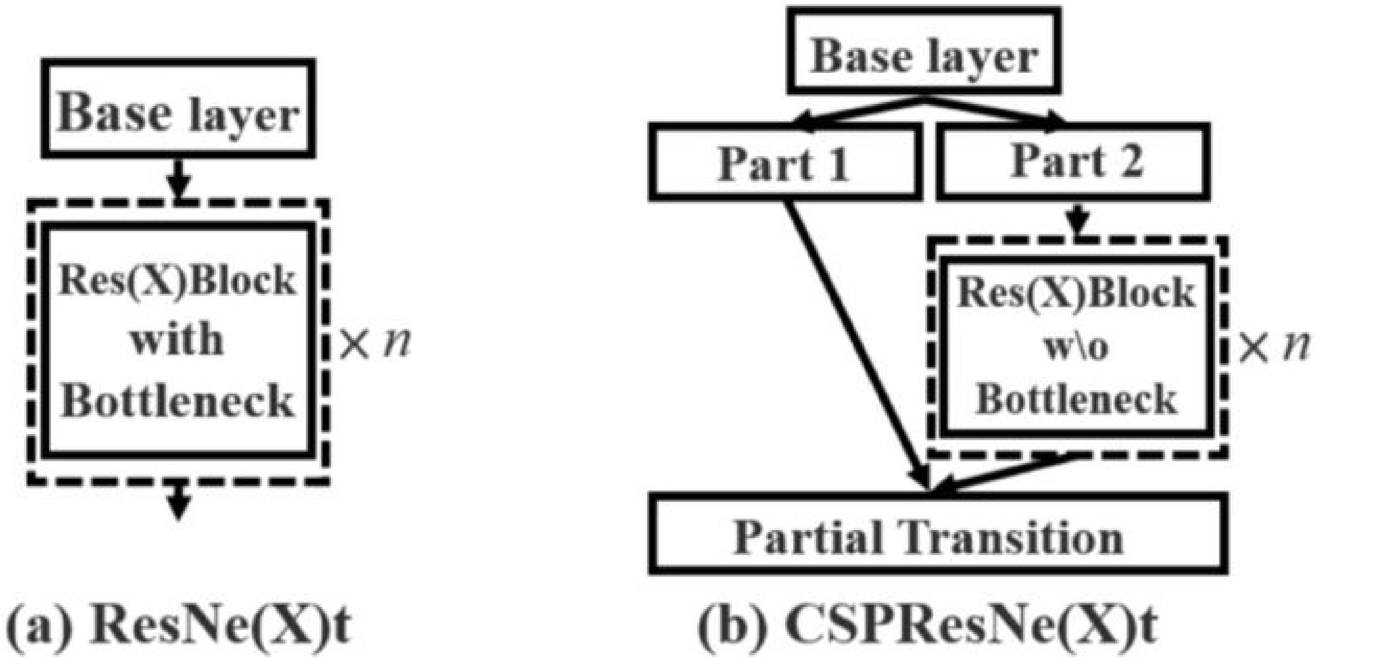


Figure 5: Applying CSPNet to ResNe(X)t.

深度学习_经典网络_CSPNet网络

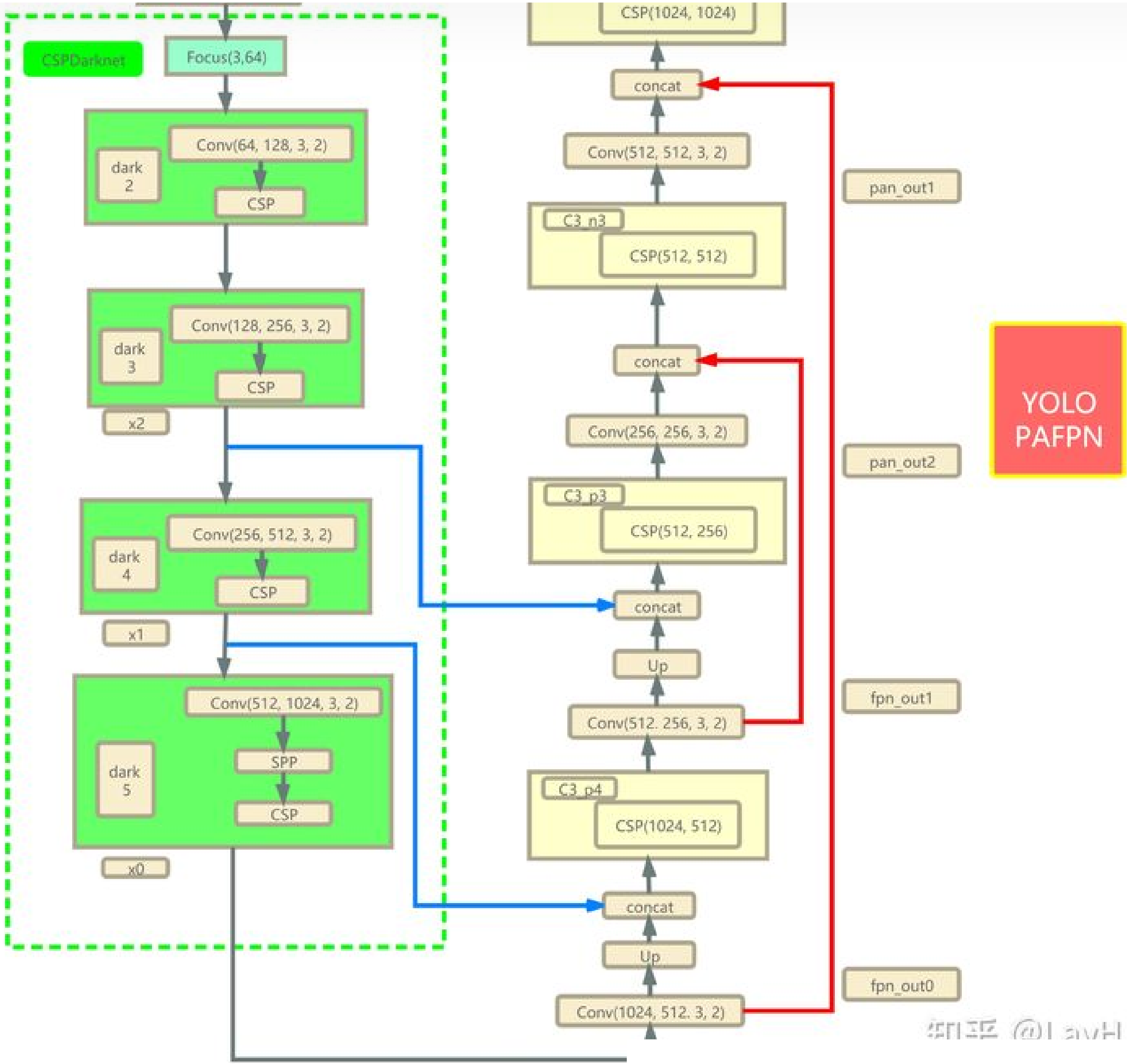
YOLO的CSP层包括了Bottleneck层，

下文网络结构的Conv是指Depthwise Separable Conv，深度可分离卷积。

关于Bottleneck和Depthwise Separable Conv的详细说明：

LayH：3、Pointwise Conv, bottleneck, depthwise Conv原理解释
2 赞同 · 1 评论 文章

2、YOLO-PAFPN的网络结构


CSPDarknet
Focus(3,64)
dark 2
Conv(64, 128, 3, 2)
CSP
dark 3
Conv(128, 256, 3, 2)
CSP
x2
dark 4
Conv(256, 512, 3, 2)
CSP
x1
dark 5
Conv(512, 1024, 3, 2)
SPP
CSP
x0
CSP(1024, 1024)
concat
Conv(512, 512, 3, 2)
C3_n3
CSP(512, 512)
concat
Conv(256, 256, 3, 2)
C3_p3
CSP(512, 256)
concat
Up
Conv(512, 256, 3, 2)
C3_p4
CSP(1024, 512)
concat
Up
Conv(1024, 512, 3, 2)
pan_out1
pan_out2
fpn_out1
fpn_out0
YOLO PAFPN

输入是Batch*3*640*640尺寸的图像。

输出是经过PAFPN网络之后的不同层次的特征图：(pan_out2, pan_out1, pan_out0)。

左边绿色的CSPDarknet，右边红色的线表示Path Aggregation。

具体的代码如下：

```
class YOLOPAFPN(nn.Module):
    """
    YOLOv3 model. Darknet 53 is the default backbone of this model.
    """

    def __init__(
        self,
        depth=1.0,
        width=1.0,
        in_features=("dark3", "dark4", "dark5"),
        in_channels=[256, 512, 1024],
        depthwise=False,
        act="silu",
    ):
        super().__init__()
        self.backbone = CSPDarknet(depth, width, depthwise=depthwise, act=act)
        self.in_features = in_features
        self.in_channels = in_channels
        Conv = DWConv if depthwise else Bas
```

```
    int(in_channels[2] * width), int(in_channels[1] * width), 1, 1, act
)
self.C3_p4 = CSPLayer(
    int(2 * in_channels[1] * width),
    int(in_channels[1] * width),
    round(3 * depth),
    False,
    depthwise=depthwise,
    act=act,
)  # cat

self.reduce_conv1 = BaseConv(
    int(in_channels[1] * width), int(in_channels[0] * width), 1, 1, act
)
self.C3_p3 = CSPLayer(
    int(2 * in_channels[0] * width),
    int(in_channels[0] * width),
    round(3 * depth),
    False,
    depthwise=depthwise,
    act=act,
)

# bottom-up conv
self.bu_conv2 = Conv(
    int(in_channels[0] * width), int(in_channels[0] * width), 3, 2, act
)
self.C3_n3 = CSPLayer(
    int(2 * in_channels[0] * width)
```

```
            False,
            depthwise=depthwise,
            act=act,
        )

        # bottom-up conv
        self.bu_conv1 = Conv(
            int(in_channels[1] * width), int(in_channels[1] * width), 3, 2, act
        )
        self.C3_n4 = CSPLayer(
            int(2 * in_channels[1] * width),
            int(in_channels[2] * width),
            round(3 * depth),
            False,
            depthwise=depthwise,
            act=act,
        )

    def forward(self, input):
        """
        Args:
            inputs: input images.

        Returns:
            Tuple[Tensor]: FPN feature.
        """

        #  backbone
        out_features = self.backbone(input)
```

```
fpn_out0 = self.lateral_conv0(x0)  # 1024->512/32
f_out0 = self.upsample(fpn_out0)  # 512/16
f_out0 = torch.cat([f_out0, x1], 1)  # 512->1024/16
f_out0 = self.C3_p4(f_out0)  # 1024->512/16

fpn_out1 = self.reduce_conv1(f_out0)  # 512->256/16
f_out1 = self.upsample(fpn_out1)  # 256/8
f_out1 = torch.cat([f_out1, x2], 1)  # 256->512/8
pan_out2 = self.C3_p3(f_out1)  # 512->256/8

p_out1 = self.bu_conv2(pan_out2)  # 256->256/16
p_out1 = torch.cat([p_out1, fpn_out1], 1)  # 256->512/16
pan_out1 = self.C3_n3(p_out1)  # 512->512/16

p_out0 = self.bu_conv1(pan_out1)  # 512->512/32
p_out0 = torch.cat([p_out0, fpn_out0], 1)  # 512->1024/32
pan_out0 = self.C3_n4(p_out0)  # 1024->1024/32

outputs = (pan_out2, pan_out1, pan_out0)
return outputs
```

发布于 2021-08-06 15:24

YOLO 目标检测