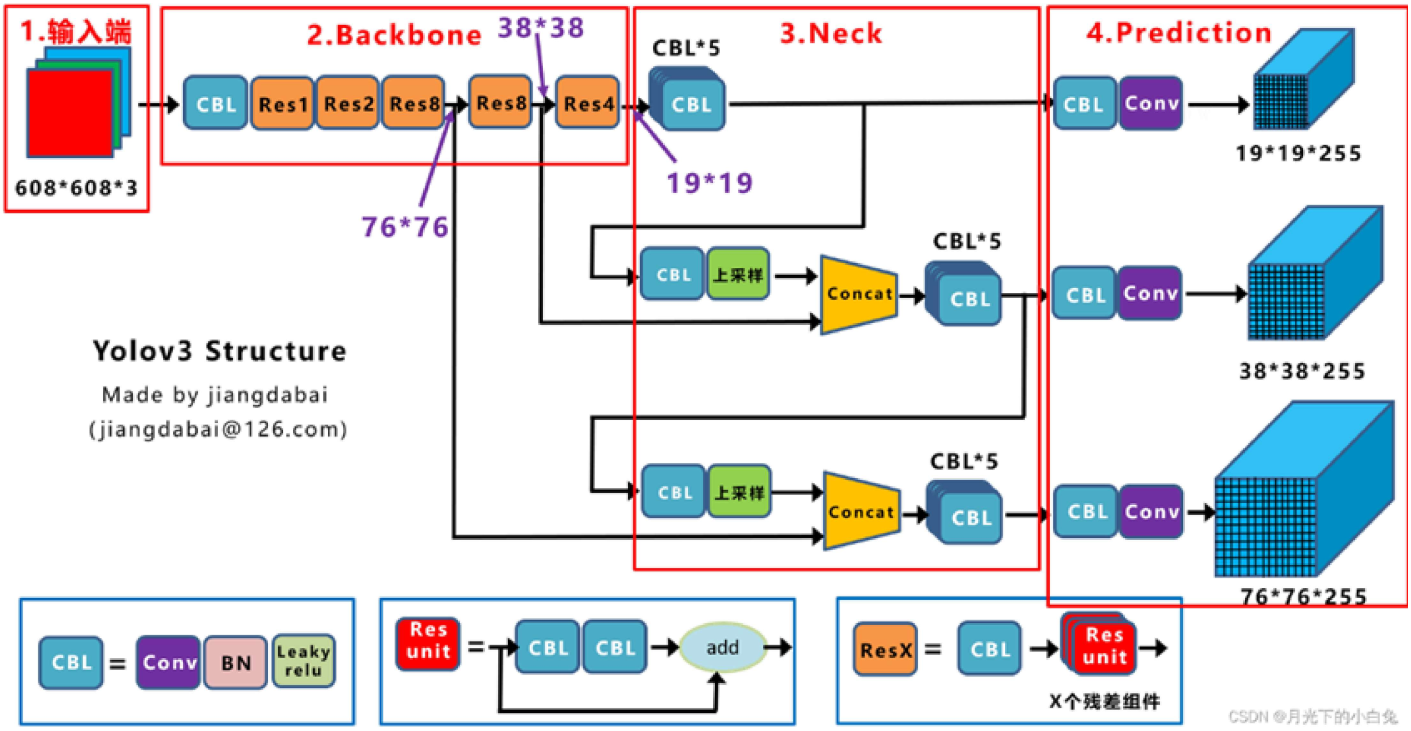
yolov3_mobilenet_v3_large_ssld_270e_voc

网络结构：

YOLOv3_mobilenet_v3_large:
backbone: MobileNetV3
neck: YOLOv3FPN
yolo_head: YOLOv3Head

网络结构图：（backbone为mobilenetv3_large）



Backbone:mobilev3

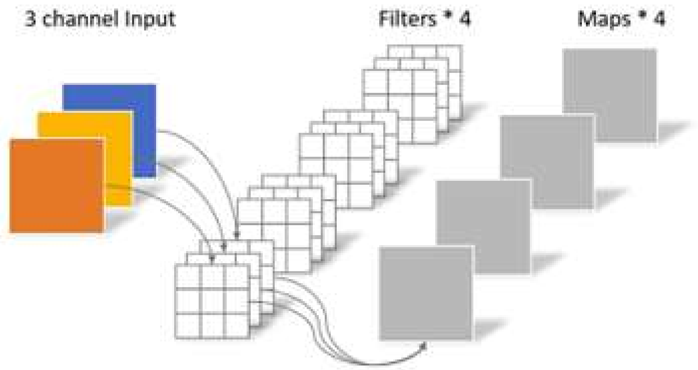
Mobilenetv1

传统卷积神经网络，内存需求大、运算量大，导致无法在移动设备以及嵌入式设备上运行。
专注于移动端或者嵌入式设备中的轻量级CNN网络

亮点：

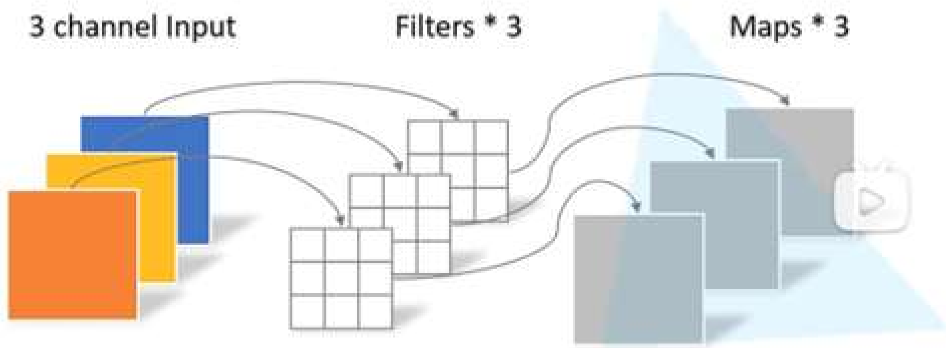
1. Depthwise Conbolution(大大减少运算量和参数量)
2. 增加超参数α、β

传统卷积：



卷积核channel=输入特征矩阵channel
输出特征矩阵channel=卷积核个数

DW卷积：



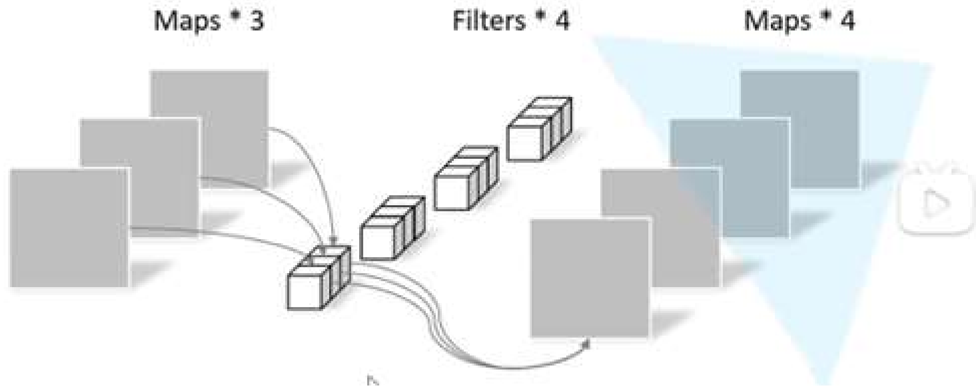
卷积核channel=1
输入特征矩阵channel=卷积核个数=输出特征矩阵channel
也就是说，经过DW卷积后，输出的特征矩阵channel是不会改变的

Depthwise Separable Conv(深度可分离卷积)：

由DW卷积和PW卷积组合而成

Pointwise Conv：

其实就是普通卷积，只不过卷积核的大小为1



普通卷积与depthwise separable conv的计算量比较：
Df:输入特征矩阵的高和宽
Dk:卷积核的大小
M:输入特征矩阵的channel
N:输出特征矩阵的channel，也就是卷积核的个数
默认卷积stride=1
普通卷积的计算量：DkDkMNDfDf
depthwise separable conv的计算量：DkDkMDfDf+MNDfDf
所以，理论上普通卷积计算量是depthwise separable conv的八到九倍
Mobilenetv1的结构图：

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

举例：Conv/s2表示普通卷积，stride=2
333*32：卷积核的高和宽为3，输入特征矩阵的深度为3，采用32个卷积核
α：width multiplier卷积核个数的倍率
β：resolution multiplier输入图像尺寸
DW部分的卷积核容易废掉，即卷积核参数大部分为零（在mobilenetv2会优化）

Mobilenetv2

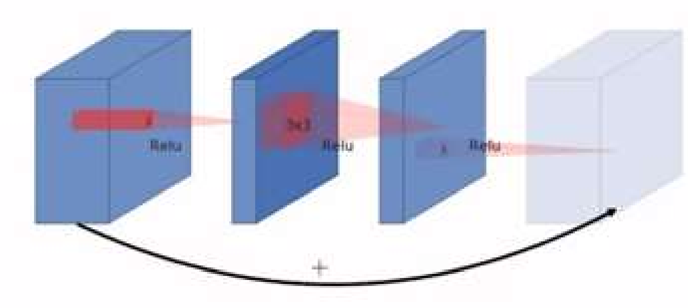
相比v1准确率更高，模型更小

亮点：

1. inverted residuals（倒残差结构）
2. linear bottlenecks

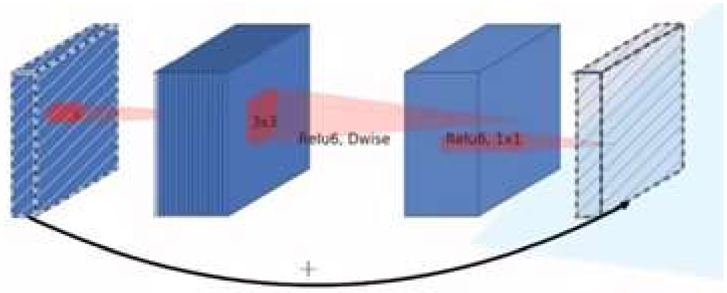
inverted residuals（倒残差结构）：

residual block： （激活函数：ReLU）



采用1×1的卷积核对特征矩阵进行压缩，减少特征矩阵的channel
再采用3×3的卷积核进行卷积处理
采用1×1的卷积核扩充channel

Inverted residuals block： （激活函数：ReLU6）



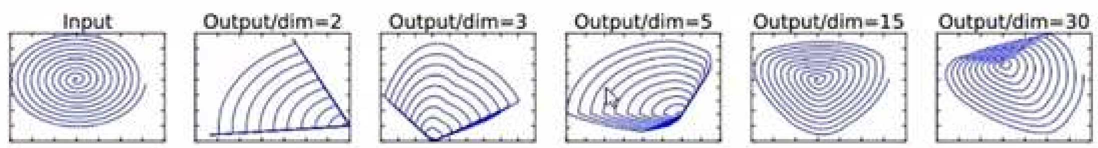
采用1×1卷积核升维操作
采用3×3卷积核（DW）
采用1×1卷积核降维
y=ReLU6(x)=min(max(x,0),6)
代码：(这里是mobilenetv3的倒残差结构，增加了SE模块，下文会讲)

```python
class ExtraBlockDW(nn.Layer):
    def __init__(self,
                 in_c,
                 ch_1,
                 ch_2,
                 stride,
                 lr_mult,
                 conv_decay=0.,
                 norm_type='bn',
                 norm_decay=0.,
                 freeze_norm=False,
                 name=None):
        super(ExtraBlockDW, self).__init__()
        self.pointwise_conv = ConvBNLayer(
            in_c=in_c,
            out_c=ch_1,
            filter_size=1,
            stride=1,
            padding='SAME',
            act='relu6',
            lr_mult=lr_mult,
            conv_decay=conv_decay,
            norm_type=norm_type,
            norm_decay=norm_decay,
            freeze_norm=freeze_norm,
            name=name + "_extra1")
        self.depthwise_conv = ConvBNLayer(
            in_c=ch_1,
            out_c=ch_2,
            filter_size=3,
            stride=stride,
            padding='SAME',
            num_groups=int(ch_1),
            act='relu6',
            lr_mult=lr_mult,
            conv_decay=conv_decay,
            norm_type=norm_type,
            norm_decay=norm_decay,
            freeze_norm=freeze_norm,
            name=name + "_extra2_dw")
        self.normal_conv = ConvBNLayer(
            in_c=ch_2,
            out_c=ch_2,
            filter_size=1,
            stride=1,
            padding='SAME',
```

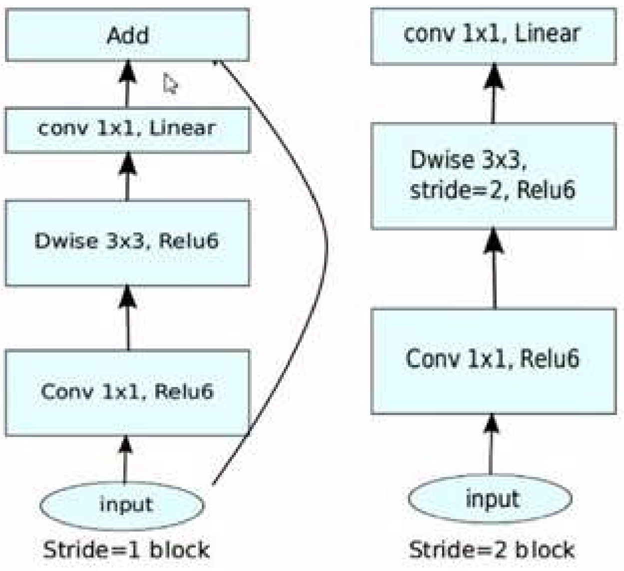
```
            act='relu6',
            lr_mult=lr_mult,
            conv_decay=conv_decay,
            norm_type=norm_type,
            norm_decay=norm_decay,
            freeze_norm=freeze_norm,
            name=name + "_extra2_sep")

    def forward(self, inputs):
        x = self.pointwise_conv(inputs)
        x = self.depthwise_conv(x)
        x = self.normal_conv(x)
        return x
```

linear bottlenecks:(针对倒残差结构最后一个1*1的卷积层使用了线性激活函数)

ReLU激活函数对低维特征信息造成大量损失，对高维特征信息造成的损失比较小



倒残差结构图：



在mobilenetv2中并不是每个倒残差结构都有shortcut
如上结构图，当stride=1时有shortcut，当stride=2时没有shortcut

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

t是扩展因子，对应的第一层1*1卷积核采用的扩展倍率t
n是bottleneck的重复次数
s是步距（针对第一层，其他为1）

Mobilenetv3

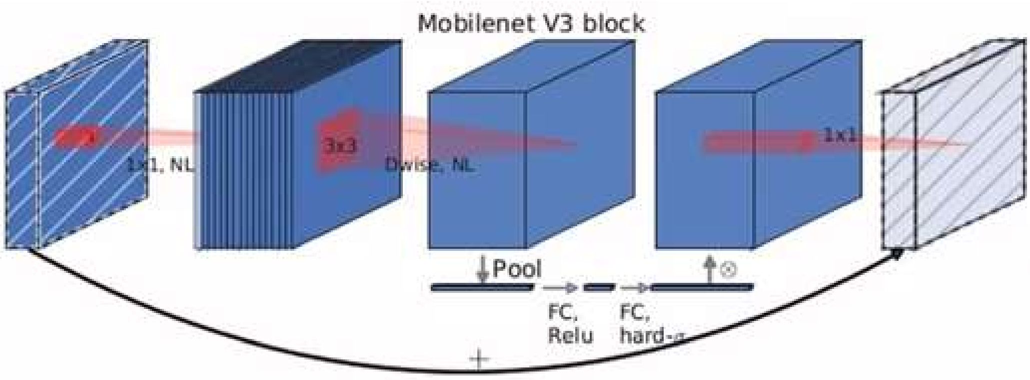
亮点：

1.更新block（bneck）在原来v2倒残差结构上进行了简单的改动
2.使用NAS搜索参数（neural architecture search）（不讲）
3.重新设计耗时层结构

更新block：

1.加入SE模块

2.更新激活函数



SE模块是注意力机制
先进行池化处理，在经过两个全连接层
第一个全连接层节点个数为特征矩阵channel的1/4
第二个全连接层节点个数与特征矩阵channel保持一致
代码：

```
class SEModule(nn.Layer):
    def __init__(self, channel, lr_mult, conv_decay, reduction=4, name=""):
        super(SEModule, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2D(1)
        mid_channels = int(channel // reduction)
        self.conv1 = nn.Conv2D(
            in_channels=channel,
            out_channels=mid_channels,
            kernel_size=1,
            stride=1,
            padding=0,
            weight_attr=ParamAttr(
                learning_rate=lr_mult, regularizer=L2Decay(conv_decay)),
            bias_attr=ParamAttr(
                learning_rate=lr_mult, regularizer=L2Decay(conv_decay)))
        self.conv2 = nn.Conv2D(
            in_channels=mid_channels,
            out_channels=channel,
            kernel_size=1,
            stride=1,
            padding=0,
            weight_attr=ParamAttr(
                learning_rate=lr_mult, regularizer=L2Decay(conv_decay)),
            bias_attr=ParamAttr(
                learning_rate=lr_mult, regularizer=L2Decay(conv_decay)))

    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, slope=0.2, offset=0.5)
        return paddle.multiply(x=inputs, y=outputs)
```

重新设计耗时层结构：

1.减少第一个卷积层的卷积核个数（32->16）

2.精简last stage

重新设计激活函数

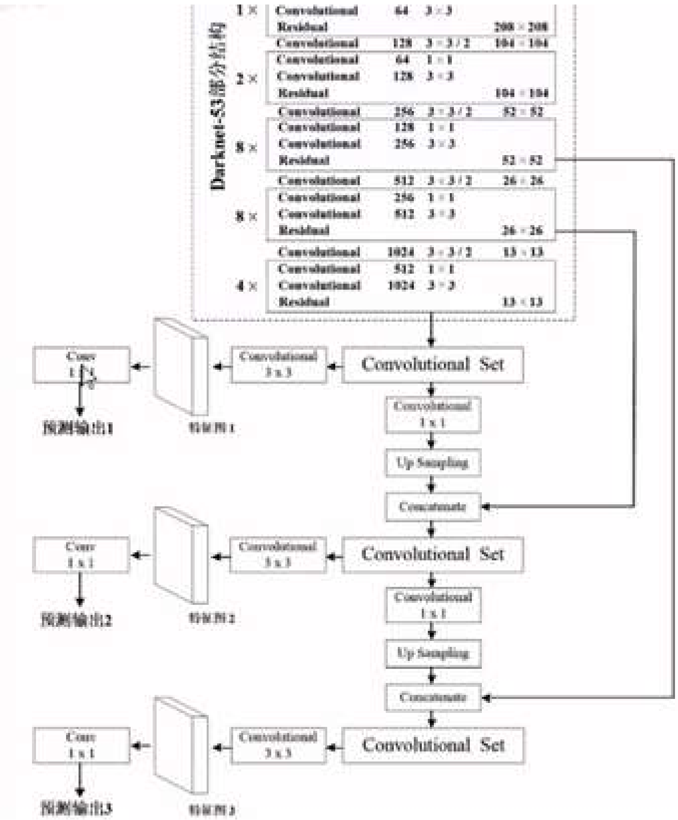
Swishx=x·σ(x)
计算、求导复杂，对量化过程不友好
h-swish[x]=x·ReLU6(x+3)/6
量化过程就是精简数据结构（不损失太多精度的情况下），节约内存，比如int32量化成int8
Mobilenetv3结构图：

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

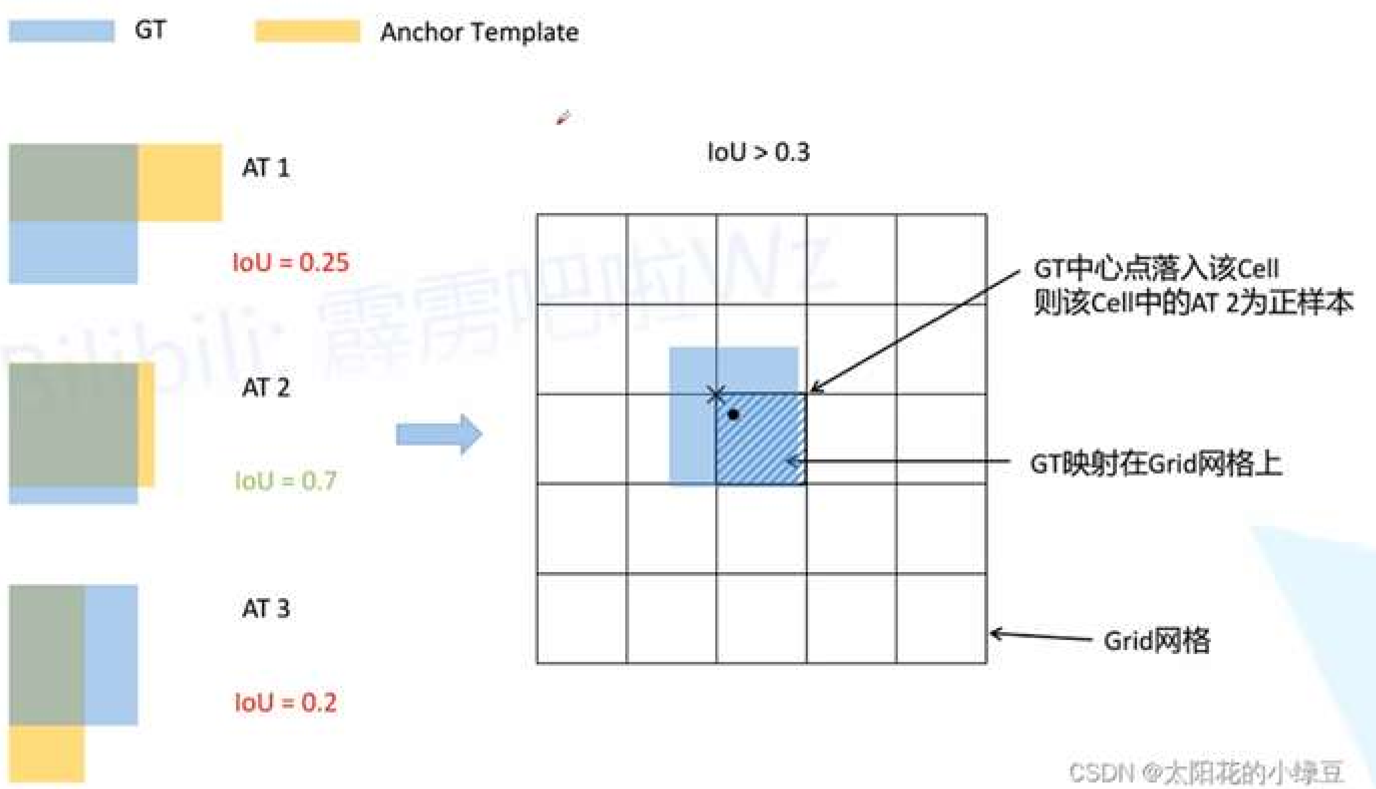
Input:输入特征矩阵
Operateor：操作
Out：输出特征矩阵
SE：是否使用se模块
NL：非线性激活函数（HS：h-swish，RE：ReLU）
Exp size：表示使用block时，通过第一层卷积升维后的特征矩阵channel
S：步距
NBN就是不使用NB
注意：可以看到蓝色框里的部分，我们发现输入的特征矩阵的channel和exp size是一样的，也就是说原本第一层11*的卷积应该升维的，但是却没有升维，所以在代码里，没有进行卷积操作，直接对特征矩阵进行了dwise convolution操作，同时也没有SE模块，所以直接通过11*的卷积层处理

neck:FPN(feature pyramid networks for object detection)

FPN结构图



目标检测框预测



损失函数

损失的计算

$$L(o,c,O,C,l,g)=\lambda_1 \underbrace{\boxed{L_{conf}(o,c)}}_{\text{置信度损失}}+\lambda_2 \underbrace{\boxed{L_{cla}(O,C)}}_{\text{分类损失}}+\lambda_3 \underbrace{\boxed{L_{loc}(l,g)}}_{\text{定位损失}}$$

$\lambda_1,\lambda_2,\lambda_3$为平衡系数

置信度损失：二值交叉熵损失

置信度损失

Binary Cross Entropy

$$L_{conf}(o,c)=-\frac{\sum_i (o_i \ln(\hat{c}_i)+(1-o_i)\ln(1-\hat{c}_i))}{N}$$

$$\hat{c}_i=Sigmoid(c_i)$$

可能和原文有出入

分类损失：二值交叉熵损失

类别损失

Binary Cross Entropy

$$L_{cla}(O,C)=-\frac{\sum_{i\in pos}\sum_{j\in cla}(O_{ij}\ln(\hat{C}_{ij})+(1-O_{ij})\ln(1-\hat{C}_{ij}))}{N_{pos}}$$

$$\hat{C}_{ij}=Sigmoid(C_{ij})$$

定位损失：sum of squared error loss

定位损失

$$L_{loc}(t,g)=\frac{\sum_{i\in pos}(\sigma(t_x^i)-\hat{g}_x^i)^2+(\sigma(t_y^i)-\hat{g}_y^i)^2+(t_w^i-\hat{g}_w^i)^2+(t_h^i-\hat{g}_h^i)^2}{N_{pos}}$$

$$\hat{g}_x^i=g_x^i-c_x^i$$

$$\hat{g}_y^i=g_y^i-c_y^i$$

$$\hat{g}_w^i=\ln(g_w^i/p_w^i)$$

$$\hat{g}_h^i=\ln(g_h^i/p_h^i)$$

t_x,t_y,t_w,t_h：为网络预测的回归参数

g_x,g_y,g_w,g_h：为GT 中心点的坐标 x, y以及宽度和高度

（映射在Grid网格中的）

Du
ground
dient i
truth b