

# Alexnet

多个CNN层堆叠而成的经典卷积神经网络，用于图像分类的任务

- 使用了ReLU激活函数，防止深层次的网络导致梯度消失
- 使用Dropout正则化，解决过拟合问题
- 使用池化层进行降维

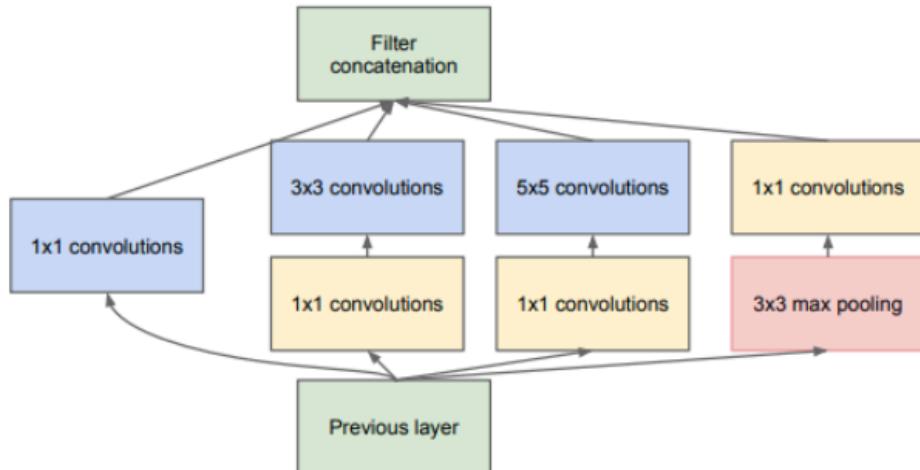
# GoogleNet

传统CNN通过多堆叠卷积层可以学习到图像更多特征，但会导致

- 计算量爆炸
- 梯度消失
- 过拟合

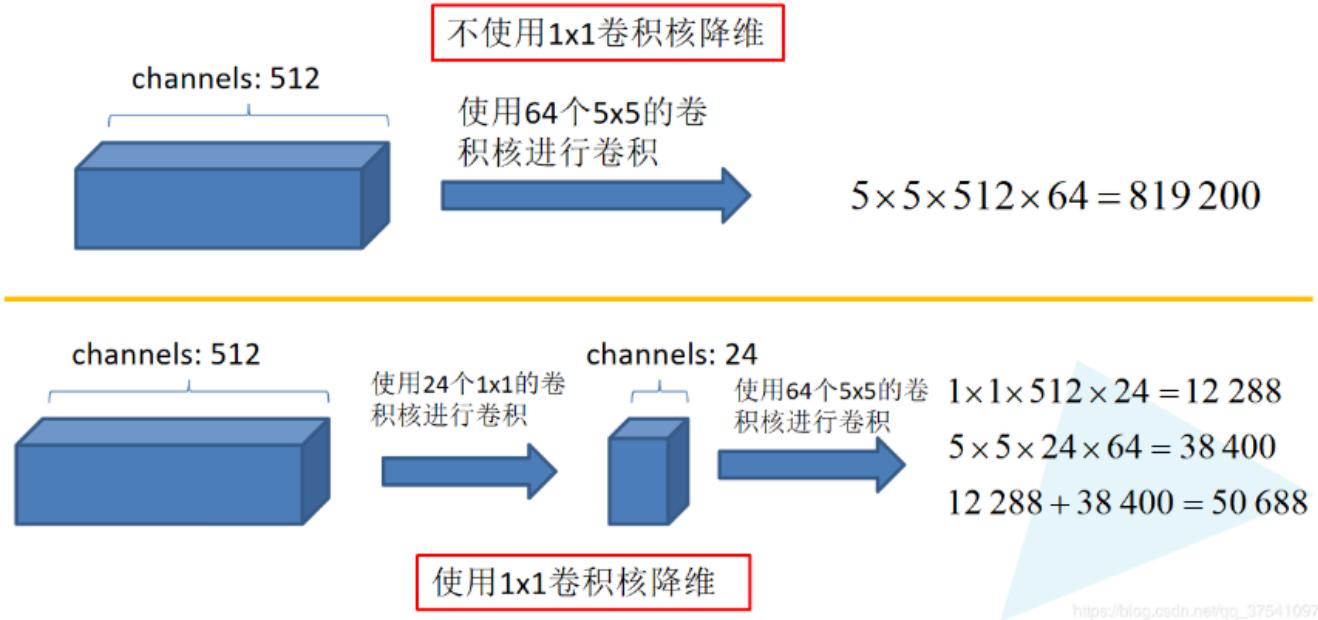
Googlenet设计了宽而深的结构，创造性引入了Inception模块和辅助分类器模块

## Inception模块



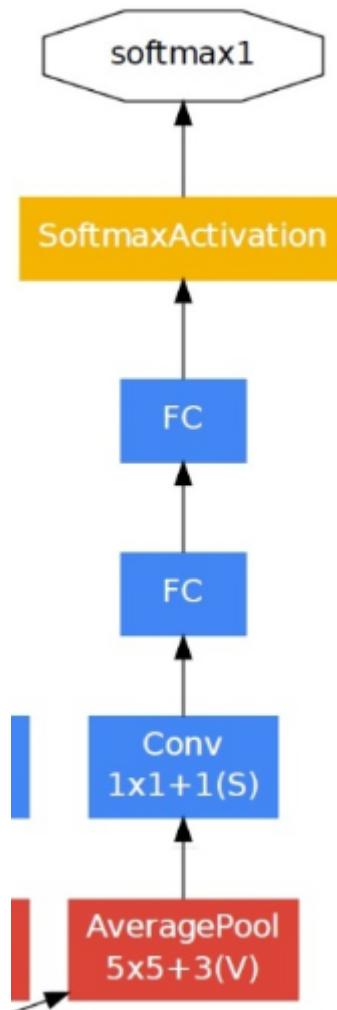
(b) Inception module with dimensionality reduction

- 并行多尺度卷积：在同一层并行使用 $1 \times 1$ 、 $3 \times 3$ 、 $5 \times 5$ 和 $3 \times 3$ 最大池化，融合不同感受野的特征
- 先使用 $1 \times 1$ 卷积进行降维，压缩通道数，同时加上ReLU，可以显著降低参数量



辅助分类器模块

## 辅助分类器



- GoogleNet有3个输出层，其中2个是辅助分类层
- 训练模型时，将2个辅助分类器的损失乘以权重加到总损失上

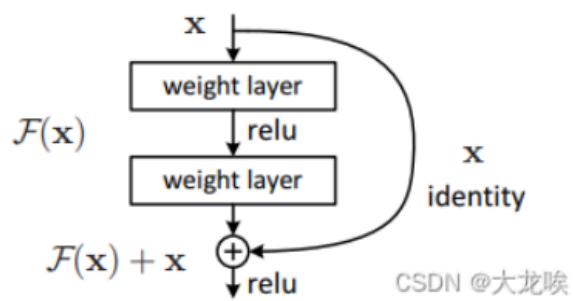
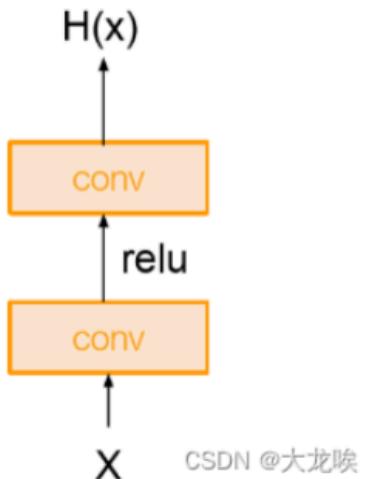
- 辅助分类器也能预测类别，在整个网络中起到一个调整的作用，可以防止网络发生过拟合
- 也可以加速梯度传递，防止梯度消失

# Resnet

2015年由何凯明团队提出，利用短路连接解决深度卷积神经网络中梯度消失的问题

$$H(x) = x + F(x)$$

这样使得网络在最差情况下也能获得和输入一样的输出，不会出现网络退化的问题



## ResNet Block

`BasicBlock` 不会对每一个block的输出进行升维

`BottleNeck` 会对每个layer的第一个block的输出进行升维，其输出通道数是输入中间通道数的4倍

二者结构如下所示

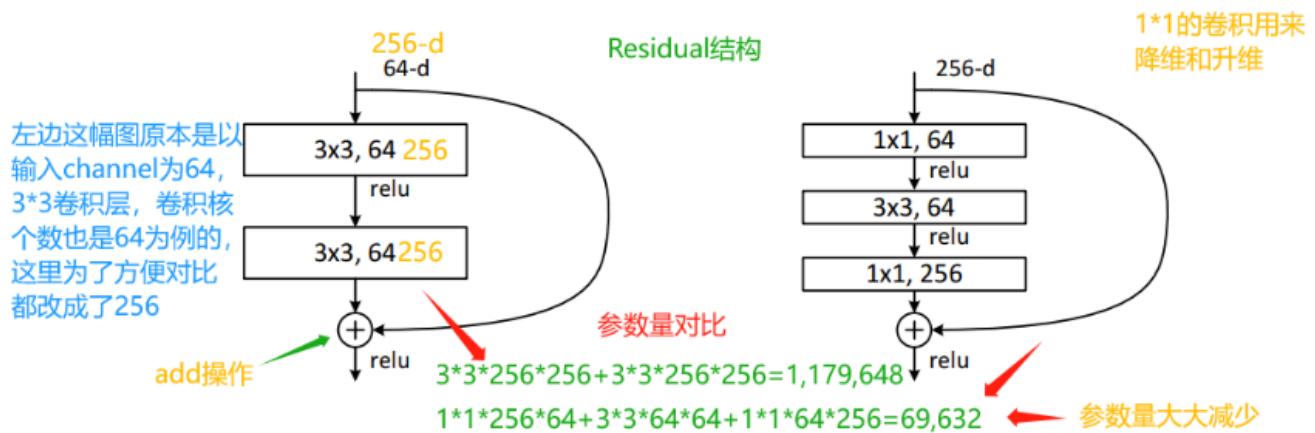


Figure 5. A deeper residual function  $\mathcal{F}$  for ImageNet. Left: a building block (on  $56 \times 56$  feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

注意：主分支与shortcut的输出特征矩阵shape必须相同

CSDN @大龙唉

- `BasicBlock`

- 从Layer2开始，每个 layer 的第一个 Block 都会升维，而后续 Block 只是保持这个维度不变
- 两个 $3 \times 3$ 的卷积核，输出通道为64
- block的输入输出通道数相同
- 重点

1. 在第1个layer中通道数不增加

2. 进入到第2个layer的第一个block，通道数增加，但在后面的block（相同layer）中通道数不变

#### • BottleNeck

- 每个Layer的第一个 Block 都会升维，而后续 Block 只是保持这个维度不变
- 将 $3 \times 3$ 的卷积层替换为 $1 \times 1 + 3 \times 3 + 1 \times 1$
- 先通过 $1 \times 1$ 的卷积核进行通道降维，巧妙扩张或缩减特征图的维度，一般降到输入维度的四分之一
- 再用 $3 \times 3$ 进行主卷积
- 最后用 $1 \times 1$ 进行通道升维，一般升到输入中间通道数的4倍
- 重点

- 经过每个layer的第一个block之后通道数都会上升，但在后面的block（相同layer）中通道数不变
- 通道数增加至输入中间通道数，也就是经过 $1 \times 1$ 的卷积的输出通道数的4倍
- 输入中间通道数会比输入通道数小二分之一

可以减小计算量，同时保证输入输出维度一致，可进行残差连接

注意：

- 当输入输出尺寸或者通道数不一致的时候，需要下采样
- 只有每个layer的第一个block会使用 stride=2 进行下采样，后续block必须保持 stride=1

各类别ResNet一览

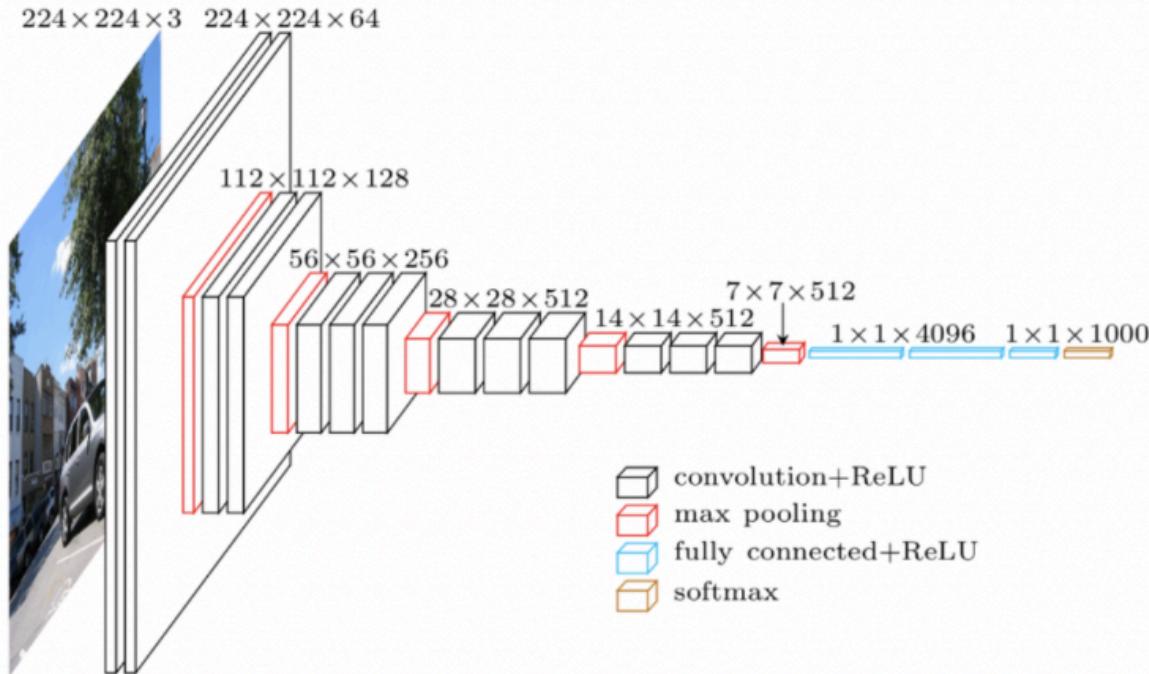
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	$112 \times 112$			$7 \times 7, 64, \text{stride } 2$		
conv2_x	$56 \times 56$	$3 \times 3 \text{ max pool, stride } 2$				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$28 \times 28$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	$14 \times 14$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	$7 \times 7$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	$1 \times 1$	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$ 暂未算

## VGG

深度卷积神经网络，固定卷积核大小为 $3 \times 3$ ，一个非常经典的架构

使用多个小卷积核 ( $3 \times 3$ ) 替代大卷积核 (如  $5 \times 5, 7 \times 7$ )

- 每层使用固定大小的  $3 \times 3$  卷积核
- 步幅为 1 (stride = 1), **padding=1**, 保证特征图大小不变
- 通过堆叠多个  $3 \times 3$  卷积, 获得更大的感受野



## MobileNet

轻量级神经网络, 可部署在边缘设备上

其计算时间95%都花费在 $1 \times 1$ 卷积上

## 深度可分离卷积

由深度卷积和逐点卷积组成 (显著减少计算量和参数量), 最后进行归一化和非线性激活

- 深度卷积用于滤波
  - 对每个输入通道单独进行空间滤波, 提取局部特征
  - 与普通卷积不一样, 每个输入通道独立使用一个卷积核, 不与其他通道交互
- 逐点卷积用于合并
  - 混合通道信息, 生成新的特征图

### 宽度乘子

通过系数 $\alpha$ 等比例减少所有层的通道数, 进一步压缩模型

### 分辨率乘子

通过系数 $\rho$ 降低输入图像分辨率, 减少计算量

### 激活函数

使用了**ReLU6**, 防止激活值过大导致量化时精度损失

$$ReLU_6(x) = \min(ReLU(x), 6)$$

## 倒残差结构 (Inverted Residuals)

借鉴了ResNet里面的残差结构

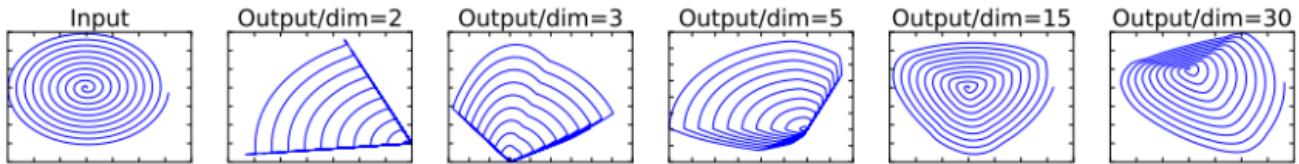
- 先降维再升维：ResNet里面的BottleNeck是先使用 $1 \times 1$ 的卷积进行降维，再使用 $3 \times 3$ 的卷积进行特征提取，最后使用 $1 \times 1$ 的卷积进行升维
- 先升维再降维：**MobileNetv2**里面先使用 $1 \times 1$ 的卷积进行升维，再使用 $3 \times 3$ 的逐通道卷积（深度可分离卷积）进行特征提取，最后使用 $1 \times 1$ 的卷积进行降维
  - 可以激活更多特征通道，让模型学习到更加丰富的特征
  - 深度可分离卷积在低维空间学习不到更多特征

## 线性瓶颈

低维空间使用ReLU会丢失部分特征信息（如负值被置零），因此在最后一层移除ReLU，改用线性激活

原因：

- 在变换过程中，需要将低维空间的信息转换到高维空间，再从高维空间转换为低维空间。如果输出维度高，信息损失小，输出维度低，信息损失大，如下图所示：



- 在MobileNetV1中，深度可分离卷积的卷积核中的数值大部分都为0，作者猜想是ReLU会把小于0的部分截断取0，容易造成信息的丢失，因此把ReLU更换为线性激活函数

## Vit (Vision Transformer)

贡献：首次将Transformer应用于图像分类，证明自注意力机制可完全替代传统卷积操作

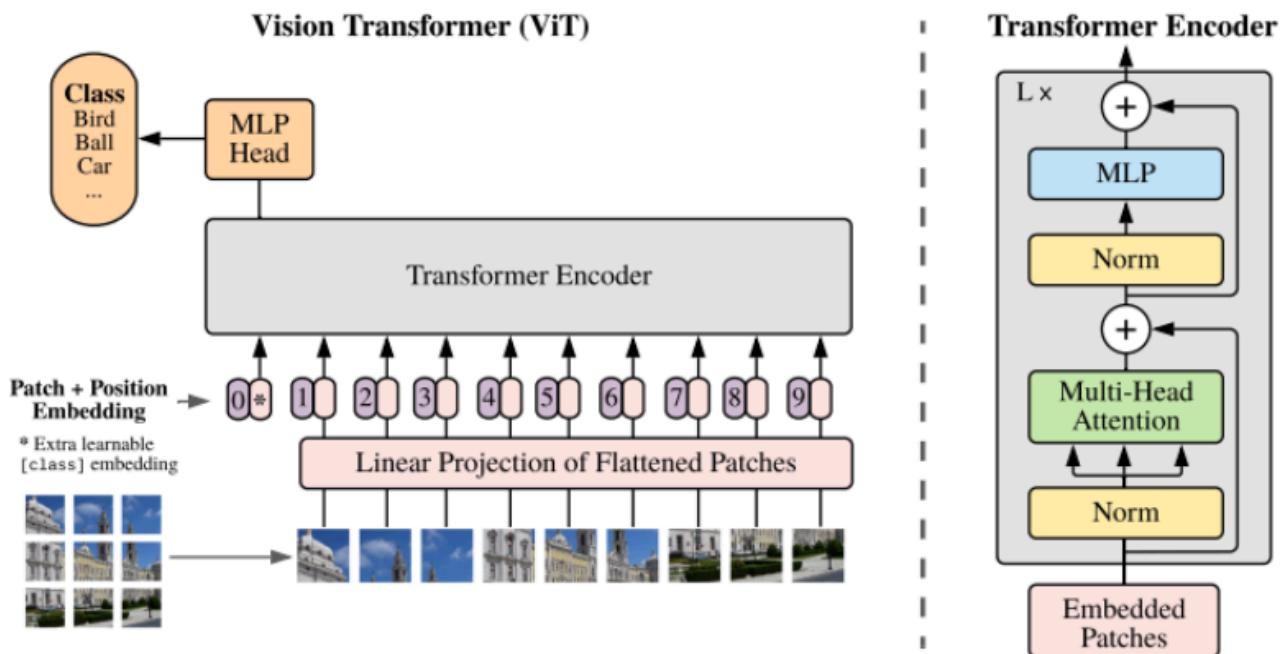
工作流程：

- 将图像进行分块，分成多个patches，将图像展平为序列
- 送入线性投影层，将patches投影成D（通常是768）维向量，并且加入位置编码（绝对位置编码），再加上一个[CLS] Token（类似Bert）
- 送入transformer的encoder层，[CLS] Token会与所有patch交互，但其他patch之间也互相计算注意力
- 从encoder层出来后，将[CLS] Token送入全连接层进行分类

局限性

- 依赖大规模数据集，在小规模数据集上泛化能力不佳
- 不像传统CNN具有针对图像的归纳偏置（Instructive Bias），也就是模型没有潜意识要去怎么处理图像，在小的数据集上鲁棒性差

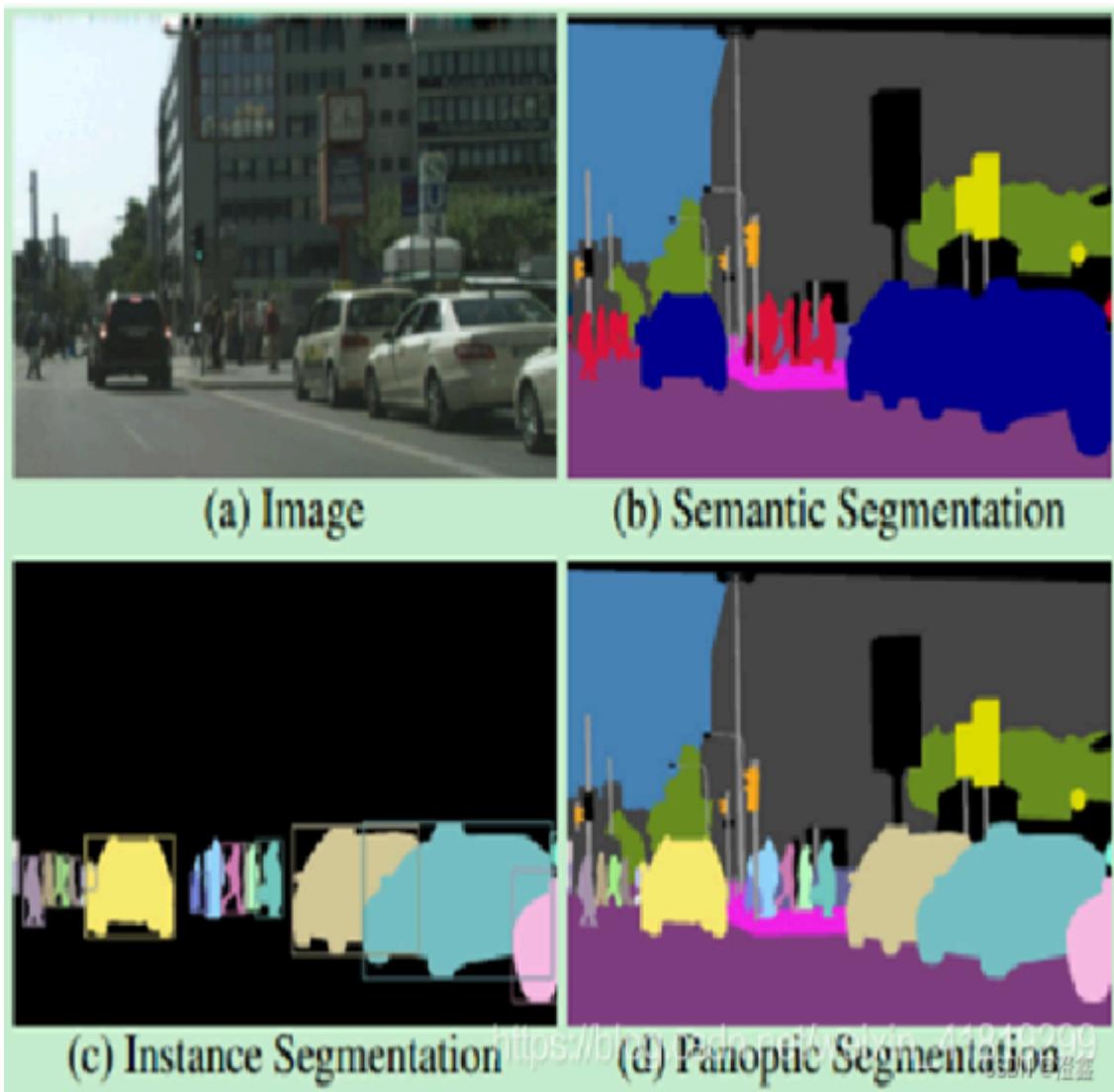
3. 对算力要求高，难以处理大分辨率图像，因为要把图像分割成很多个patch，还要进行全局注意力计算，对算力要求高



- 在语义分割的基础上，区分同一类别的不同实例（如区分图中不同的个体）

### 全景分割（Panoptic Segmentation）

- 统一语义分割和实例分割，要求对图像中所有像素进行分类，并区分可数对象（如车辆、人）和不可数区域（如天空、道路）



## FPN (Feature Pyramid Network)

还tm是何kaiming提出的，卧槽

为何要提出这个网络？

- 传统卷积神经网络仅在最后一个特征图上进行后续操作（如预测），而这类图像的分辨率又较小，造成小物体在此图中的信息较少，模型无法精确检测到小物体，造成性能缺失

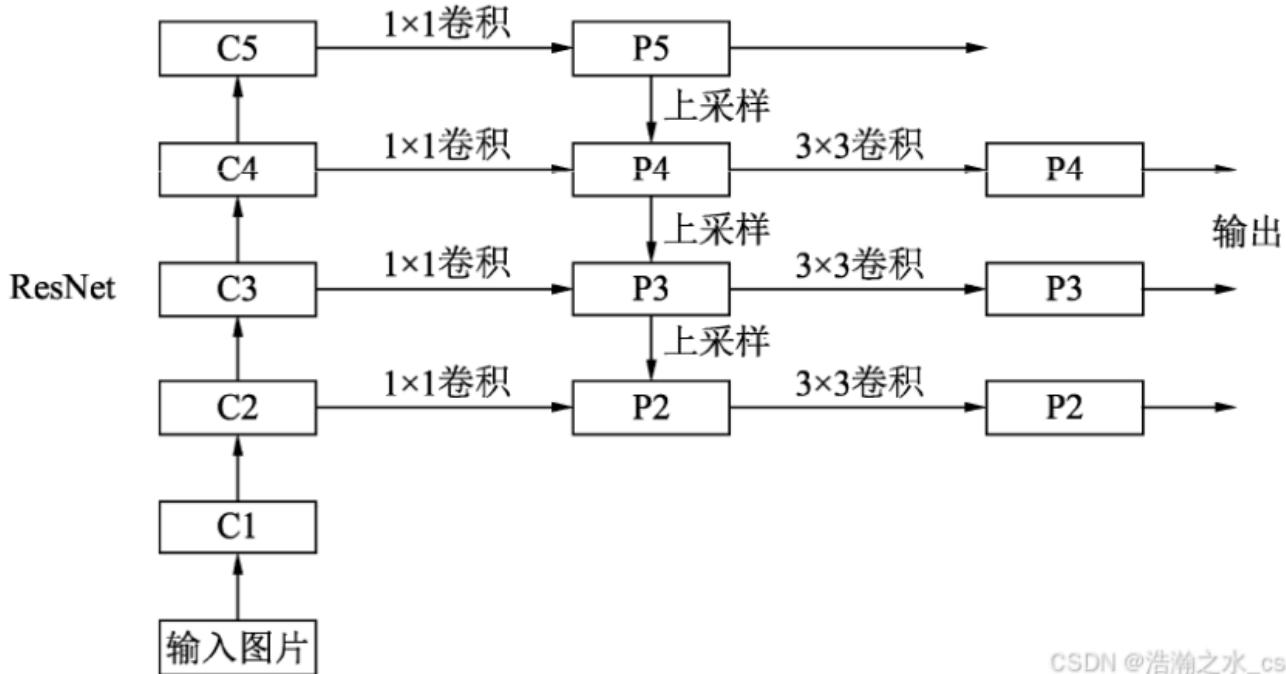
主要目的：通过特征融合的方式，在不显著增加计算量的情况下，提升多尺度目标检测性能

具体有什么贡献？

- 提出一种新的特征金字塔结构，构建自下向上和自上向下的特征融合路径，融合不同尺度特征图，使得大小物体的检测性能都提高了
- 最重要的还是不增加计算量

网络结构如下

- 自下而上逐步下采样采用的Backbone是ResNet，这个过程中，图像分辨率不断降低，语义信息不断丰富
- 自上而下则是将低分辨率图片进行转置卷积（上采样），并与对应的低层特征图融合（横向连接）
- 在特征金字塔的每一层上都可以独立的进行对应任务的预测



CSDN @浩瀚之水\_csdn

## FCN (Fully Convolutional Network)

为什么要提出这个网络？

- 传统卷积神经网络会对图片进行不断的下采样，即分辨率逐渐减小，这样子可以提取更加抽象的语义特征，但是对于语义分割这种像素级的预测任务来说，下采样过多会带来严重的信息丢失，导致预测边界不清晰。
- 为了弥补下采样带来的空间信息缺失，FCN提出了转置卷积进行上采样，同时引入跳跃连接将浅层特征和深层特征进行结合

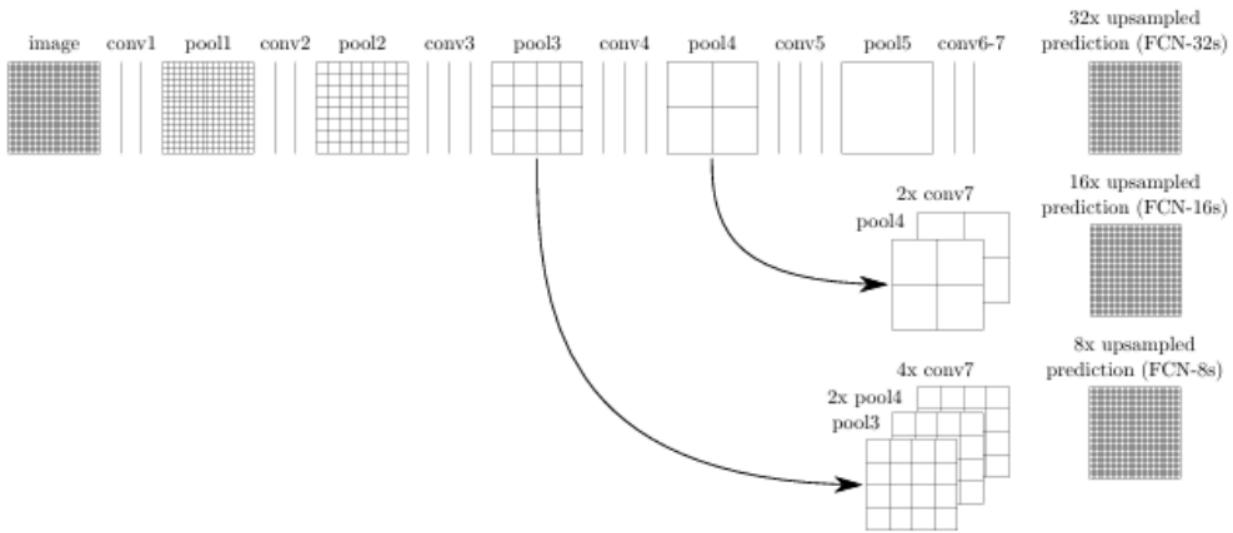
特点：全卷积网络，不含全连接层，常用于语义分割

- 不含全连接层，把最后的全连接层换成 $1 \times 1$ 的卷积核，可以适应任意尺寸的输入
  - $1 \times 1$ 卷积核的输出通道数是要预测的类别数，最后在每个像素的通道数上作softmax即可预测这个像素属于什么类别
- 转置卷积进行上采样
- 引入跳跃连接

FCN主要分为两部分

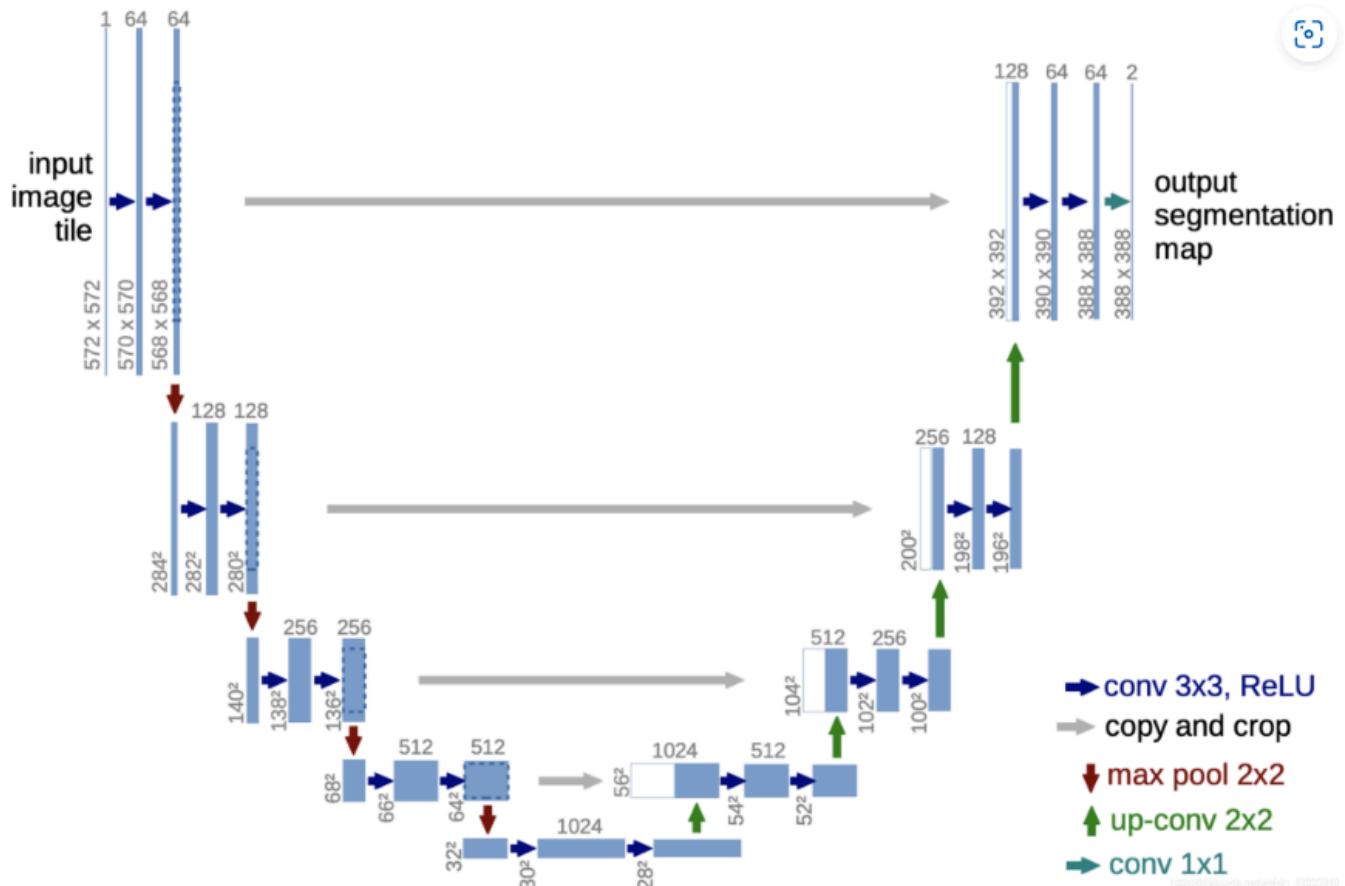
- 全卷积网络：以VGG、ResNet为主的Backbone
- 上采样部分：使用转置卷积还原回原图尺寸
  - FCN-32s：直接上采样32倍
  - FCN-16s：先上采样2倍，然后融合pool4，再上采样16倍

- FCN-8s：先上采样2倍，然后融合pool4，再上采样2倍，然后融合pool3，再上采样8倍



## U-Net

- Uunet采用了编码器-解码器架构，并加入了跳跃连接，将浅层的高分辨率特征直接传递到解码器中，有助于弥补上采样过程中的细节丢失
- 适合应用于小型数据集，常用于医学影像分割
- 常用于语义分割



## 工作流程

- 输入图像为 $1 \times 572 \times 572$ 大小

- 蓝色箭头卷积参数为 `in_channels=1, out_channels=64, kernel_size=3, stride=1, padding=0`, 每进行一次卷积和ReLU, 图像尺寸 -2, 通道数逐层增加至1024
- 最大池化的卷积核和步长都设置为2, 每进行一次池化, 输出尺寸减半, 通道数不变
- 每进行一次上采样 (转置卷积), 通道数减半, 图像尺寸加倍, 同时在解码器模块中也会有卷积参与
- 在进行跳跃连接时, 需要对下采样的图像进行裁剪, 之后与上采样图像进行拼接

## 深监督

原始Unet并未使用这个方法, 但这个方法在后来对Unet的改进中被使用

我个人认为其作用和GoogleNet的辅助分类器大差不差, 都是加入了辅助损失函数防止梯度消失

深监督指的是在网络的中间层添加辅助损失函数, 简单来说就是加网络的中间层加一个输出层, 有利于

- 缓解梯度消失: 通过多路径反向传播加速训练
- 改善特征学习
- 早期收敛: 浅层网络也能输出初步结果
- 防止过拟合

## IoU (交互比)

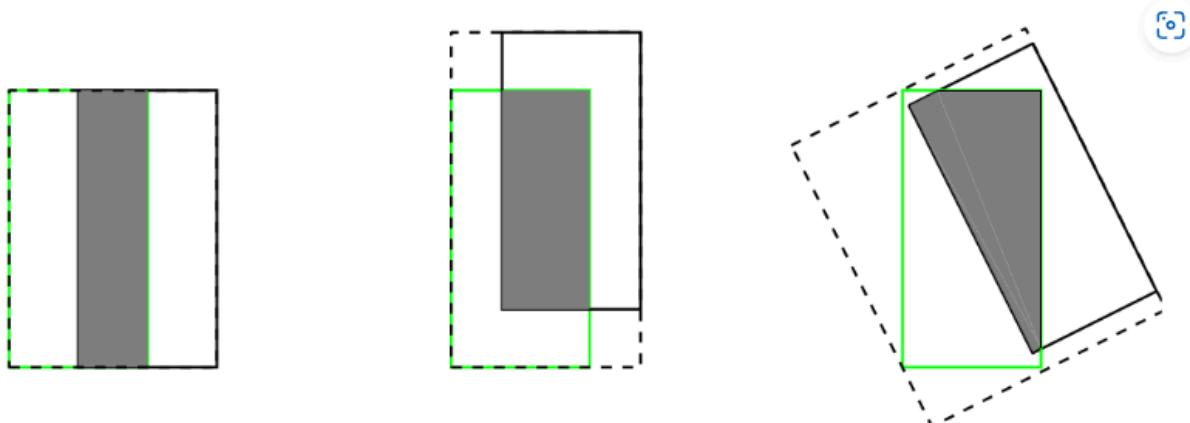
用于衡量预测区域与真实区域重叠程度的指标, 核心思想是计算两者的交集与并集的比值, 如下, 用预测框 (A) 和真实框 (B) 的交集除二者并集

$$IoU = \frac{A \cap B}{A \cup B}$$

常用于目标检测, 图像分割等场景, 实际应用时分子分母都要加上平滑项

但IoU也有如下缺点:

- 当预测框和真实框完全不重合时, 梯度为0, 无法优化
- 无法精确反映两者重合度大小, 相同的IoU可能有不同的重合度情况。如下所示, 三种情况IoU都相等, 但他们的重合度情况不一样, 左边最好, 右边最差。



## Dice Loss

基于Dice系数, 用于衡量预测分割掩码和真实掩码的重叠程度, 主要用于二分类任务

$$DiceLoss = 1 - \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

其中：

- $A$ : 预测的分割掩码（通常为概率图，值在0~1之间）。
- $B$ : 真实的分割掩码（二值图，0或1）。
- $|A \cap B|$ : 预测与真实掩码的交集（逐像素相乘后求和）。
- $|A| + |B|$ : 预测和真实掩码的像素值之和。

不依赖绝对像素数量，只关注重叠比例，对像素少的目标进行分割仍可以有效优化

实际应用时分子分母都要加上平滑项

## BCE Loss

---

二分类交叉熵损失函数，衡量预测概率分布与真实分布的差异，在图像分割领域常与**DiceLoss**结合，

$$BCE = - \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

其中：

- $y_i$ : 真实标签
- $p_i$ : 预测概率
- $N$ : 像素总数

其梯度稳定，且对小目标友好（因为其对像素级分类更敏感）

但是在类别不平衡时，会被多数类（背景）主导，导致模型忽视少数类（肿瘤）

## Deeplab

---

用于语义分割

v2、v3的backbone（骨干网络）为ResNet

问题：

1. 图像多次下采样分辨率降低
2. 要对目标进行多尺度特征提取

## LargeFov

---

空洞卷积

Deeplab中引入空洞卷积扩大了卷积核的感受野而不增加参数量或者计算量！！！，有效捕捉多尺度上下文信息

- 无需下采样即可获得大感受野
- 高膨胀率下，卷积核采样点过于稀疏，可能丢失局部连续性信息

相较于VGG的改进

1. 将VGG的全连接层（FC6、FC7）转换为 $7 \times 7$ 卷积以保留空间信息。

2. 进一步用 $3 \times 3$ 空洞卷积( $r=12$ )替换 $7 \times 7$ 卷积,以极低计算量实现全局感受野

3. 最后VGG的全连接分类层用 $1 \times 1$ 卷积核进行分类(卷积核也可以进行分类)

## ASPP model (V3)

为了水论文而水。。。

核心: 多尺度空洞卷积并行处理

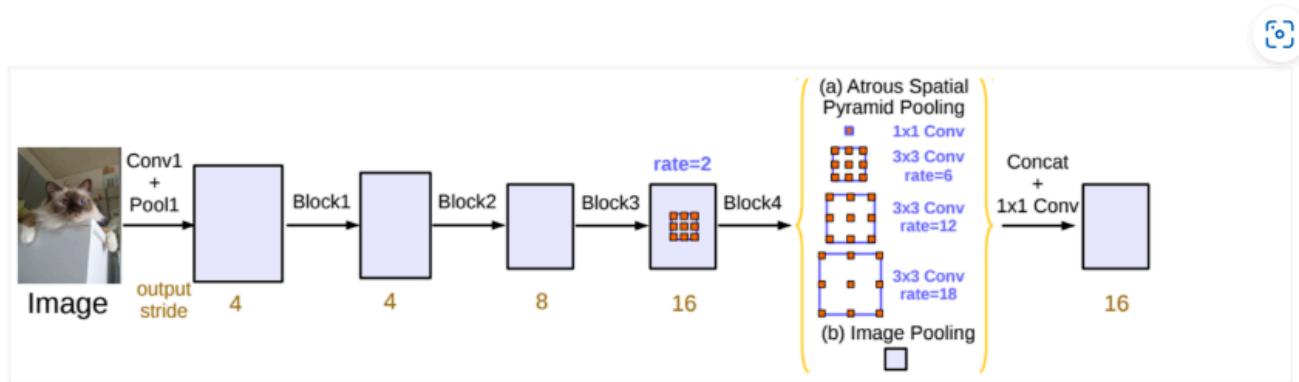


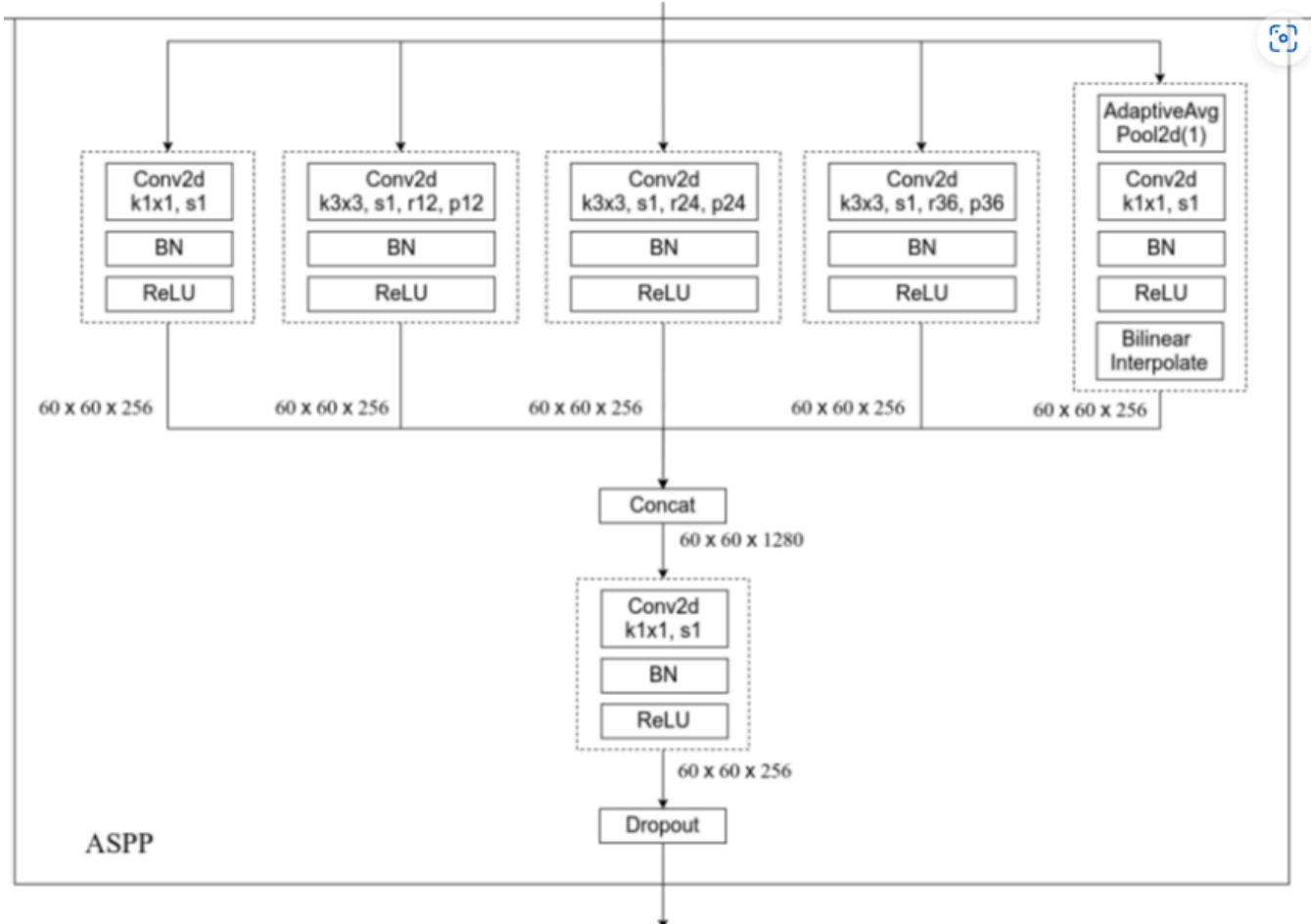
Figure 5. Parallel modules with atrous convolution (ASPP), augmented with image-level features.

输入为**backbone**的输出, 也就是特征图

五个尺度:

- $1 \times 1$ 卷积
- 膨胀系数为12的 $3 \times 3$ 卷积
- 膨胀系数为24的 $3 \times 3$ 卷积
- 膨胀系数为36的 $3 \times 3$ 卷积
- 全局池化

最后进行**channel**上的拼接, 再通过 $1 \times 1$ 的卷积层进行进一步融合



需要注意的是：

- 当下采样率设置为**16**的时候，上图的 $3 \times 3$ 膨胀卷积的膨胀系数分别为**6、12、18**
- 当下采样率设置为**8**的时候，上图的 $3 \times 3$ 膨胀卷积的膨胀系数分别为**12、24、36**（作者说要翻倍，我也不知道为什么）

## poly

V2采用的学习率策略（炼丹炼上瘾了）

$$poly = lr \times \left(1 - \frac{iter}{max\_iter}\right)^{power}$$

其中：

- $power$ : 衰减强度系数，通常取0.9-2.0
- $iter$ : 当前步数
- $max\_iter$ : 总步数

## RCNN

用于目标检测呀！

Backbone使用的是VGG16

总体来说分为4步（非常慢，很耗时，因为不是端对端网络，要分别训练多个网络）：

1. 一张图像生成1k~2k个候选区域

2. 对每个候选区域，使用深度网络提取特征（前面说的分类网络本质上都可以）

3. 深度网络输出的特征向量送入SVM进行分类

4. 使用回归器精细修正候选位置

### 非极大值抑制（NMS）剔除重叠建议框

1. 寻找得分最高的目标，也就是经过SVM分类器之后概率最高的

2. 计算其他目标与得分最高的目标的IoU值

3. 删除所有IoU值大于给定阈值的目标

## Fast R-CNN

Backbone使用的是VGG16

### 工作步骤

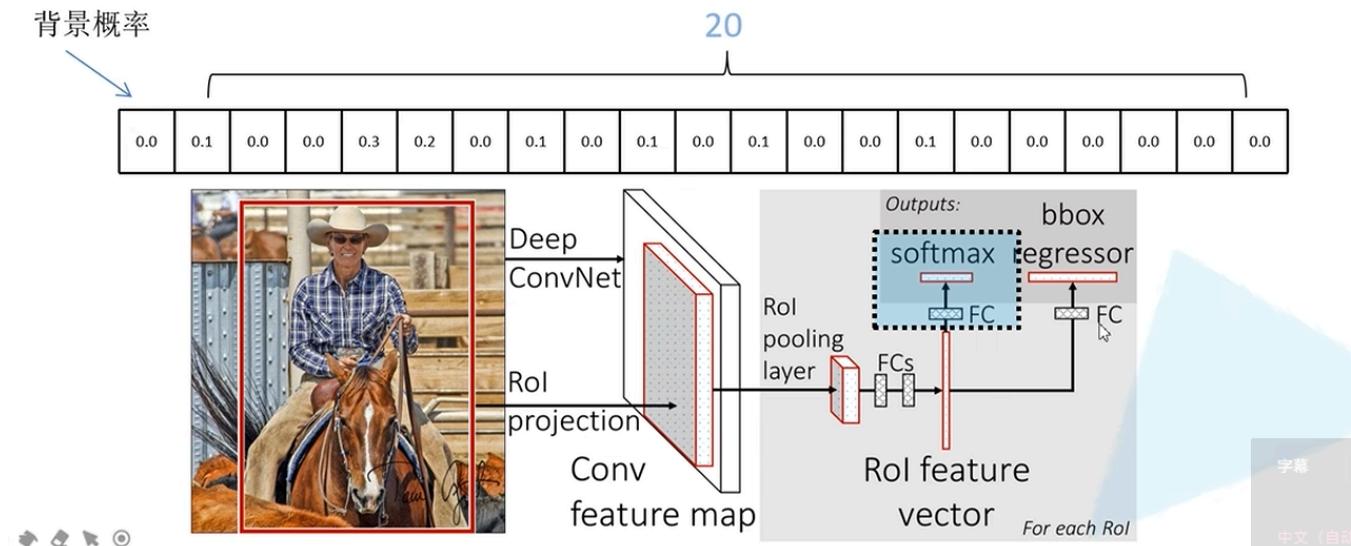
1. 一张图像生成1k~2k个候选区域

2. 将图像输入网络得到特征图，通过SS算法将生成的候选框直接投影到特征图上获得相应的特征矩阵

3. 将每个特征矩阵通过ROI pooling层统一缩放到 $7 \times 7$ 大小的特征图，将特征图展平通过一系列全连接层得到预测结果

- 有两个输出通道，并联两个全连接层
- 其中一个全连接层用于边界框回归参数的预测，另一个用于目标概率的预测（会输出 $N+1$ 个类别的概率， $N$ 为检测目标的种类，1为背景）

从SS算法提供的区域进行正负样本采样解决数据不平衡的问题，原论文中提出只要候选框和真实的目标框的IoU大于0.5就是正样本

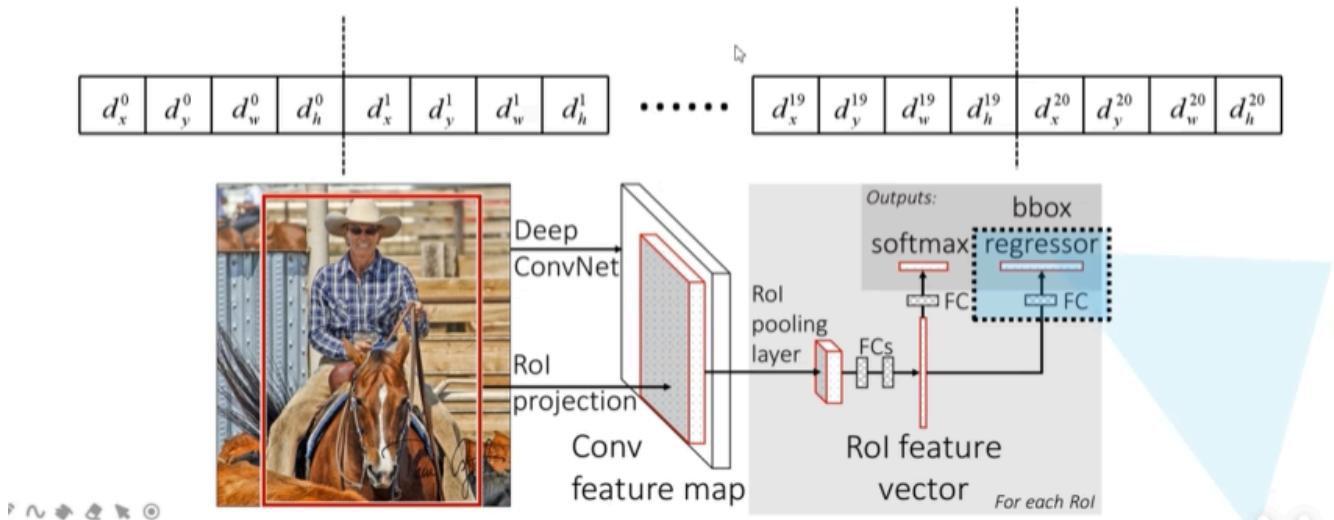


## 边界框回归参数的预测

全连接层会输出 $(N + 1) \times 4$ 个回归参数

## 边界框回归器

输出对应N+1个类别的候选边界框回归参数( $d_x, d_y, d_w, d_h$ ), 共(N+1)x4个节点



然后候选框进行参数更新

$$\hat{G}_x = P_w d_x(P) + P_x$$

$$\hat{G}_y = P_h d_y(P) + P_y$$

$$\hat{G}_w = P_w \exp(d_x(P))$$

$$\hat{G}_h = P_h \exp(d_x(P))$$

其中：

- $P_x, P_y, P_w, P_h$  分别是候选框的中心  $x, y$  坐标，以及宽高
- $\hat{G}_x, \hat{G}_y, \hat{G}_w, \hat{G}_h$  为最终预测的边界框的中心  $x, y$  坐标，以及宽高

## Multi-task loss

### 损失函数

$$L(p, \mu, t^\mu, v) = \underbrace{L_{cls}(p, \mu)}_{\text{分类损失}} + \underbrace{\lambda[\mu \geq 1] L_{loc}(t^\mu, v)}_{\text{边界框回归损失}}$$

其中：

- $p$  是分类器预测的 softmax 概率分布  $p = (p_0, \dots, p_k)$
- $\mu$  对应目标真实类别标签
- $t^\mu$  对应边界框回归器预测的对应类别  $u$  的回归参数  $(t_x^\mu, t_y^\mu, t_w^\mu, t_h^\mu)$
- $v$  对应真实目标的边界框回归参数  $(v_x, v_y, v_w, v_h)$
- $[\mu \geq 1]$  是艾佛森括号
  - 为 1 才有边界框回归损失
  - 为 0 说明预测的是背景，无需移动边界框

### 分类损失

$$L_{cls}(p, \mu) = -\log p_\mu$$

这是因为真实类别目标标签是一个单热编码，最后只有真实标签不为0

### 边界框回归损失

$$L_{loc}(t^\mu, v) = \sum_{i \in \{x, y, w, h\}} smooth_{L_1}(t_i^\mu - v_i)$$

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

### 真实目标框回归参数的计算

$$v_x = \frac{G_x - P_x}{P_w}$$

$$v_y = \frac{G_y - P_y}{P_h}$$

$$v_w = \ln \frac{G_w}{P_w}$$

$$v_h = \ln \frac{G_h}{P_h}$$

其中：

- $G_x, G_y, G_w, G_h$  分别为真实目标框的中心  $x, y$  坐标，以及宽高
- 计算这个参数是为了计算损失

## ROI pooling

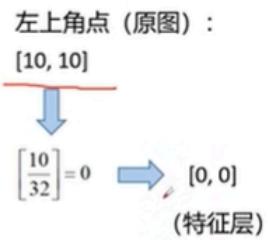
---

1. 输入图像后经过**特征提取**，得到**特征图**
2. **ROI区域映射到特征图上**（映射：与ROI在原图上的位置相对应）
3. 将映射后的区域划分成多个**部分**，**部分**的数量与目的输出的维度有关，**保证输出的特征维度统一**
4. 对每个部分进行**max pooling**操作

## RoIPooling

特征层相对原图步距为32

-1.5256	-0.7502	-0.6540	-1.6095	-0.1002	-0.6092
-0.9798	-1.6091	-0.7121	0.3037	-0.7773	-0.2515
-0.2223	1.6871	0.2284	0.4676	-0.6970	-1.1608
0.6995	0.1991	0.1991	0.0457	0.1530	-0.4757
-1.8821	-0.7765	2.0242	-0.0865	2.3571	-1.0373
1.5748	-0.6298	2.4070	0.2786	0.2468	1.1843



-1.5256	-0.7502	-0.6540	-1.6095	-0.1002
-0.9798	-1.6091	-0.7121	0.3037	-0.7773
-0.2223	1.6871	0.2284	0.4676	-0.6970
0.6995	0.1991	0.1991	0.0457	0.1530
-1.8821	-0.7765	2.0242	-0.0865	2.3571



# Faster R-CNN

还tm是何kaiming

核心: RPN + Fast R-CNN

1. 一张图像生成1k~2k个候选区域
2. 将图像输入网络得到特征图, 通过RPN网络将生成的候选框直接投影到特征图上获得相应的特征矩阵
3. 将每个特征矩阵通过ROI pooling层统一缩放到 $7 \times 7$ 大小的特征图, 将特征图展平通过一系列全连接层得到预测结果

## RPN (Region Proposal Network)

核心任务: 生成高质量的候选框

1. RPN网络的输入是Backbone提供的特征图
2. 然后进行锚点生成和滑动窗口, 再把卷积核的中心坐标映射回原来的特征图
  - 锚点生成: 在每个映射得到的原图中心坐标上, 预设k个不同尺度和长宽比的锚点框 (默认k=9)
    - 例如: 尺度为 [128, 256, 512], 长宽比为 [1:1, 1:2, 2:1]
  - 滑动窗口: 通过 $3 \times 3$ 卷积对每个位置进行滑动, 输出该位置的区域特征, 通常是一维向量, 这取决于Backbone输出的通道数 (256-d或者其他维度)
  - 映射: 用原图的尺寸除以特征图的尺寸并取整得到步距, 用特征图的坐标乘以这个步距得到原图坐标
3. 对于每个锚点, RPN通过两个 $1 \times 1$ 的卷积层并行输出
  1. 分类分支
    - 预测锚点是前景或者是背景的概率 (2k个输出), 用sigmoid或者softmax
  2. 回归分支
    - 输出4k个参数, 表示锚点需要的偏移量, 帮助锚点移动至高质量的候选框

## 正负样本采样

RPN网络会在对所有图像生成的**非常多个anchor**进行**正负样本采样**，总数一般为250，正负样本比为1:1

- 正样本：与任意**真实框（最终目标框）**的IoU大于0.7或者是与某个真实框有最大IoU（小于0.7）的anchor
- 负样本：与所有真实框的IoU小于0.3

RPN网络会对一张图像生成的非常多的anchor（如 $2w$ 个）保留下高质量的候选框（如 $2k$ 个）**作为Fast-RCNN的输入**，由其输出准确的目标框

## RPN Multi-task loss

$$L(\{p_i\}, \{t_i\}) = \underbrace{\frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*)}_{\text{分类损失}} + \lambda \underbrace{\frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)}_{\text{边界框回归损失}}$$

其中：

- $p_i$ 表示第*i*个anchor预测为真实标签的概率
- $p_i^*$ 当正样本时为1，负样本时为0
- $t_i$ 表示预测第*i*个anchor的**边界框回归参数**
- $t_i^*$ 表示第*i*个anchor对应的真实**目标框回归参数**
- $N_{cls}$ 表示一个mini-batch的所有样本数，也就是**一张图像采样的正负样本的总数**
- $N_{reg}$ 表示anchor位置的个数，即特征图的长和宽相乘
- $L_{reg}$ 是前文提到的**smooth**函数
- $L_{cls}$ 是交叉熵损失

## Mask R-CNN

能同时用于**目标检测**和**实例分割**，相对于Faster R-CNN加入了**mask分支**，改进**ROI Pooling**为**ROI Align**

**Mask分支**和**Faster R-CNN分支**并不共用一个**ROI Align**

**损失函数**： $L_{mask}$ 其实就是BCE-Loss

$$L_{mask_{rcnn}} = L_{RPN} + L_{Faster_{rcnn}} + L_{mask}$$

## ROI Align

是**ROI Pooling**的改进

为何要改进？

- ROI pooling涉及两次取整操作，精度会有所损失，预测性能不佳
  - 第一次取整在原图下采样至特征图这一过程
  - 第二次取整在将特征图划分区域，maxpooling为统一尺寸这一过程

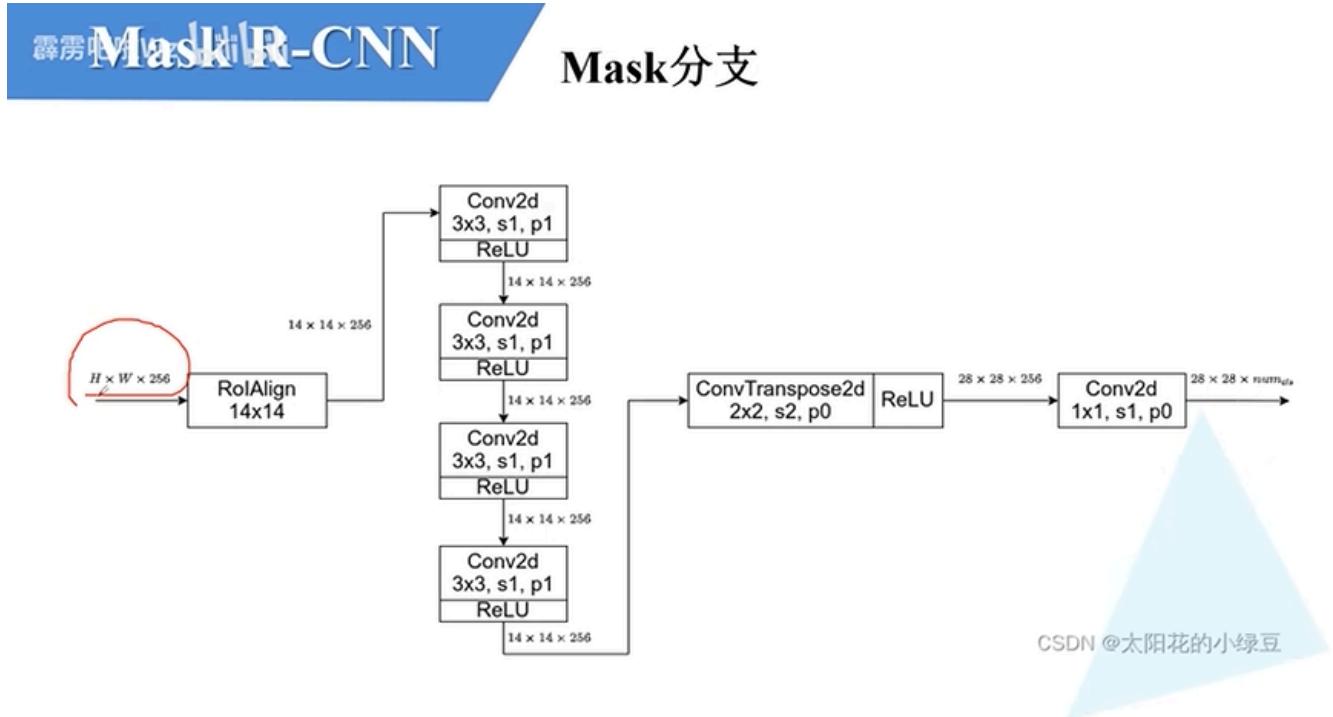
## ROI Align工作流程

1. 在原图下采样至特征图这一过程不进行取整，得到一个锚框

2. 将得到的锚框进行均分，也不涉及取整
3. ROI pooling是对每个子区域进行最大池化，ROI Align通过在每个区域进行采样得到每个区域的输出（当采用多个采样点时，每个子区域的输出取所有采样点的均值）
4. 找出每个区域的中心点，找离它最近的四个像素点，使用双线性插值法对采样点数值进行计算

## Mask分支

具有FPN结构的Mask分支



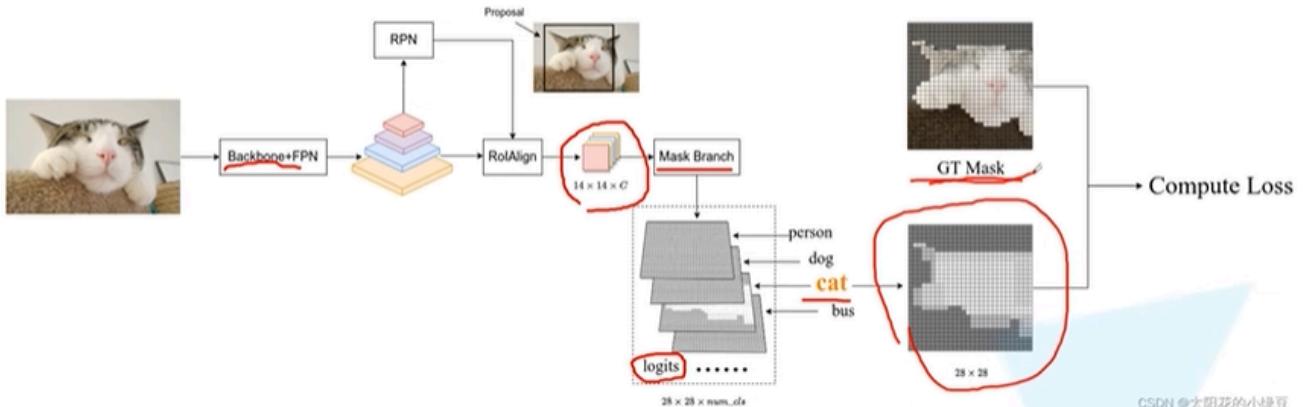
注意：

- 训练的时候输入Mask分支的目标是由RPN提供的，即Proposals（高质量候选框），且全是正样本
  - 输入mask分支进行预测，用sigmoid把像素值进行归一化，前景像素值为1，背景像素值为0，然后使用BCE计算损失
  - 输出的蒙版是小尺寸的，计算损失时要将原图进行裁剪成与输出蒙版相同尺寸

# Mask R-CNN

## Mask分支损失

BCELoss (BinaryCrossEntropyLoss)

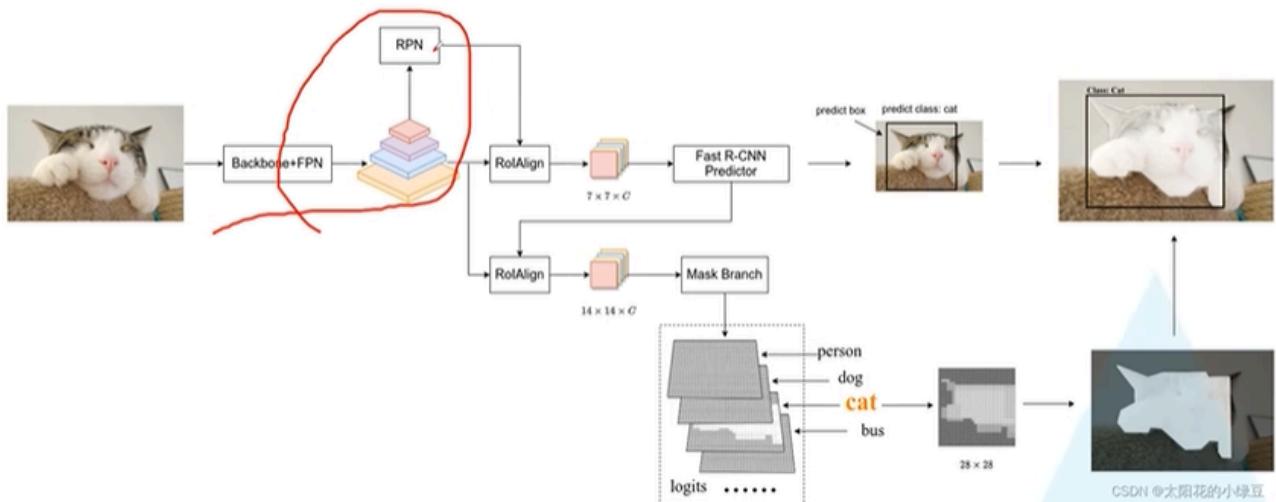


CSDN @太阳花的小绿豆

训练网络的时候输入Mask分支的目标是由RPN提供的，即Proposals，  
但在预测的时候输入Mask分支的目标是由Fast R-CNN提供

- 预测的时候输入Mask分支的目标是由Faster R-CNN提供的
  - mask分支输出的预测结果要经过上采样还原回原图大小

## Mask分支预测使用



CSDN @太阳花的小绿豆

训练网络的时候输入Mask分支的目标是由RPN提供的，即Proposals，  
但在预测的时候输入Mask分支的目标是由Fast R-CNN提供

通过这个分支可以为每个目标预测一个二值掩码，在像素级别上精确跟踪目标

在Mask R-CNN中，对预测mask以及class进行解耦

- 不在通道维度上进行softmax，否则一个类别分数高，另一个类别分数低，会存在竞争关系
- 对每个类别预测一个蒙版，对预测mask以及class进行解耦
  - 用sigmoid，对每个像素预测属于前景和背景的概率，输出蒙版

## **YoloV5**

---

Backbone采用了DarkNet

## **YoloV7**

---

## **YoloV8**

---

## **Clip**

---

## **Blip**

---

## **其它知识点补充**

---

- 上下采样指的是图像分辨率的变化，而不是通道数的变化
- 消融实验是逐步移除或修改模型的某个组件（如模块、层、技术等），观察性能变化，从而量化该组件对模型整体的贡献

## **COCO格式数据集**

---

COCO格式的数据集可以通过 `pycocotools.coco` 的 `coco` 类进行加载，需放入 `annotations` 的 json 文件进行加载。

初始化后的实例有以下功能

方法	作用	示例
<code>.getImgIds()</code>	获取所有图像ID列表	<code>train_coco.getImgIds()</code>
<code>.loadImgs(ids)</code>	加载指定ID的图像元信息	<code>train_coco.loadImgs([1,2])</code>
<code>.getAnnIds(imgIds=[])</code>	获取指定图像的标注ID	<code>train_coco.getAnnIds(imgIds=[1])</code>
<code>.loadAnns(ids)</code>	加载指定ID的标注详情	<code>train_coco.loadAnns([1,2])</code>
<code>.annToMask(annotation)</code>	将多边形标注转为二值掩码	
<code>.showAnns(annotations)</code>	可视化标注（需配合matplotlib）	