ADAS自动驾驶入门实践

第4讲 人工神经网络与深度学习

李坤乾

工程学院 自动化及测控系

2021年11月

■ 机器学习与人工神经网络

图像分类的任务，就是从已有的固定分类标签集合中选择一个并分配给一张图像

{dog, cat, car, plane,…}

从图像到标签分值的参数化映射

Class scores
Cat: 0.9
Dog: 0.1
Car: 0.01

方法的两个组成部分：

评分函数(score function): 原始图像数据到类别分值的映射

损失函数(loss function): 量化预测分类标签的得分与真实标签之间的一致性

- 可转化为一个最优化问题，在最优化过程中，将通过更新评分函数的参数来最小化损失函数值。

■ 机器学习与人工神经网络

线性分类器：

X：原始像素数值被拉成一个[3072*1]的列向量

3072x1

b：大小为[10*1]的偏差向量

$$f(x,W) = Wx + b$$

10x1

10x1 10x3072

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(x,W)

10 numbers giving class scores

以CIFAR10为例总共有10类物体

W

parameters or weights

W：大小为[10*3072]的权重

输入x是给定的，但参数W和b是可控制改变的。机器学习的目标就是：

通过设置这些参数，使得计算出来的分类分值情况和真实类别标签相符。

■ 机器学习与人工神经网络

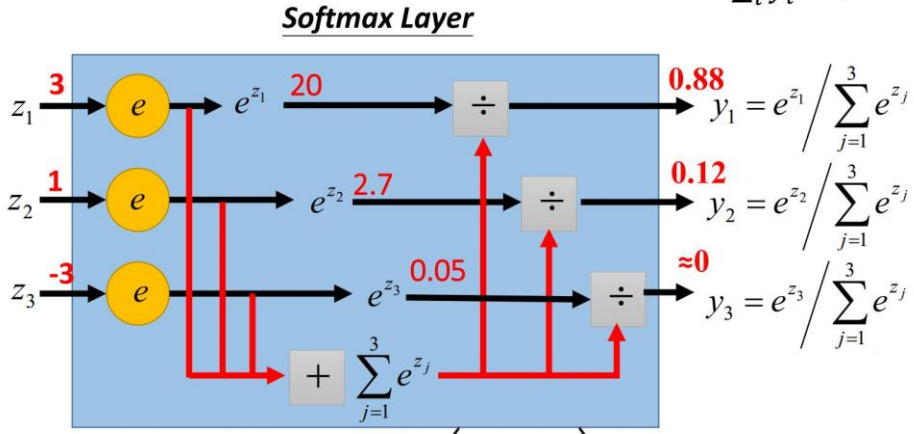
损失函数： 衡量我们对结果的不满意程度

Softmax函数： $P=(y=k|X=x_i)=\frac{e^{z_k}}{\sum_j e^{z_j}}$ 其中 $z=f(x_i,W)$

Softmax分类器将评分视为每个分类的未归一化的对数概率。我们想最大化对数概率相当于最小化负对数概率形式的损失函数。

- Softmax layer as the output layer

Probability:
- $1>y_i>0$
- $\sum_i y_i=1$



交叉熵损失(cross-entropy loss):

$$L_i=-\log\left(P=(y=k|X=x_i)\right)=-\log\left(\frac{e^{z_k}}{\sum_j e^{z_j}}\right)$$

$$L=\frac{1}{N}\sum_{i=1}^{N}L_i+\lambda R(W)$$

$L(\theta)=L(\theta)+\lambda\sum_i^n\theta_i^2$

R(W)为正则项，防止过拟合

■ 机器学习与人工神经网络

最优化(Optimization):

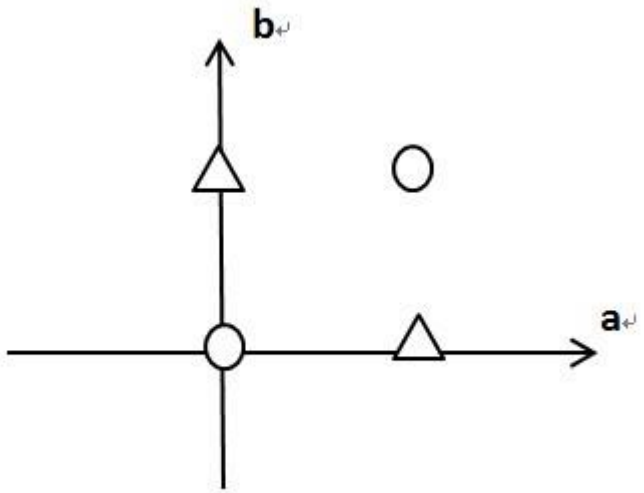
最优化是寻找能使得损失函数值最小化的参数的过程。

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

梯度下降算法：函数的梯度给出了该函数最陡峭的上升方向，负梯度方向即是下降最快的方向，沿着梯度方向不断迭代即可达到最小值

■ 机器学习与人工神经网络

$$Score = Wx+b$$



输入	运算符	输入	结果
1	⊕	0	1
1	⊕	1	0
0	⊕	0	0
0	⊕	1	1

线性分类器的局限性

■ 机器学习与人工神经网络

线性分类器：

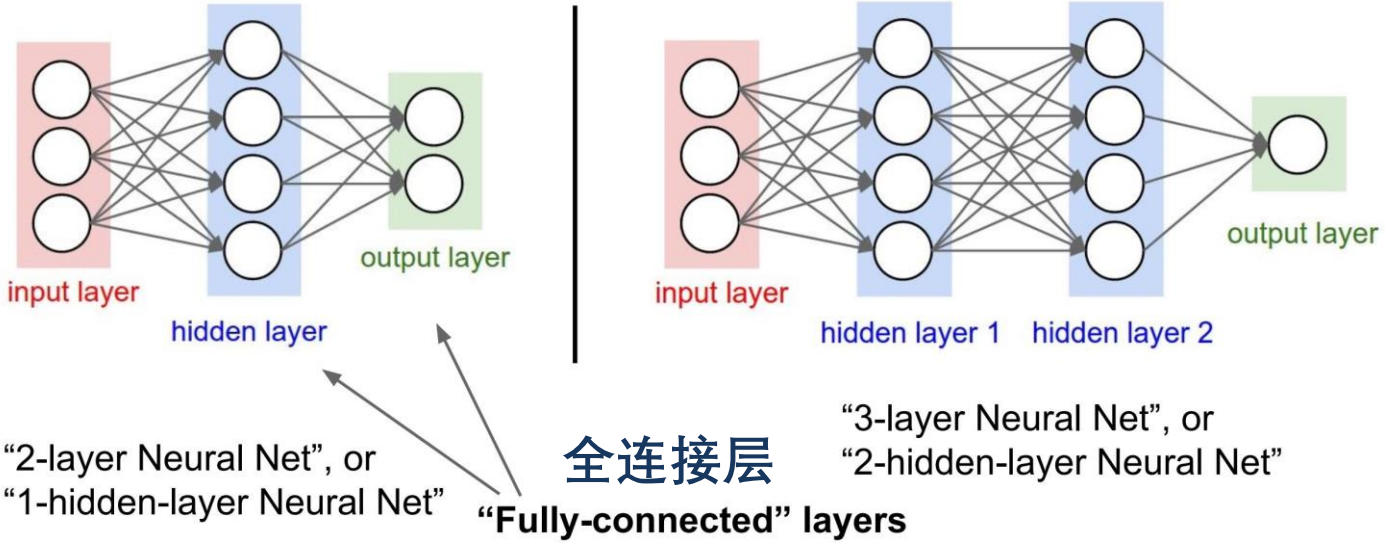
$Score = Wx+b$

由于非线性函数的引入以及多层的结构，使得神经网络具有更强的数据拟合能力

神经网络：

$$Score = f(W_3(f(W_2(f(W_1x+b)+b))+b))$$

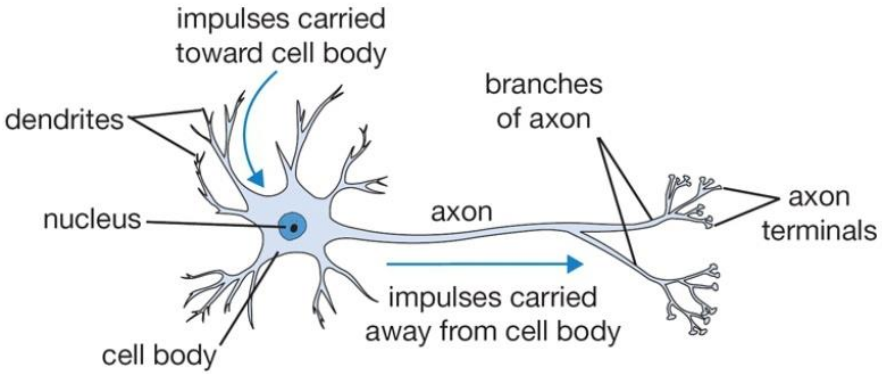
$f()$为非线性函数

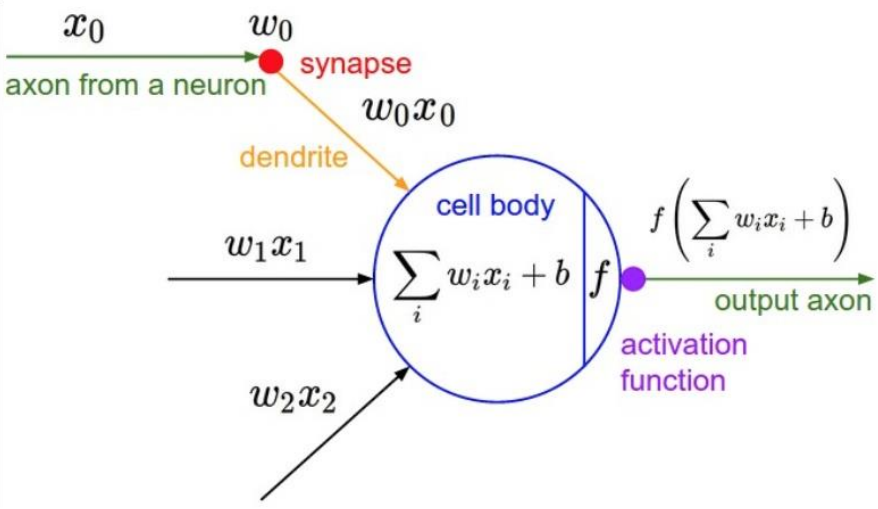


全连接层中的神经元与其前后两层的神经元是完全成对连接的，但是在同一个全连接层内的神经元之间没有连接

■ 机器学习与人工神经网络

人工神经网络的生物学启示：

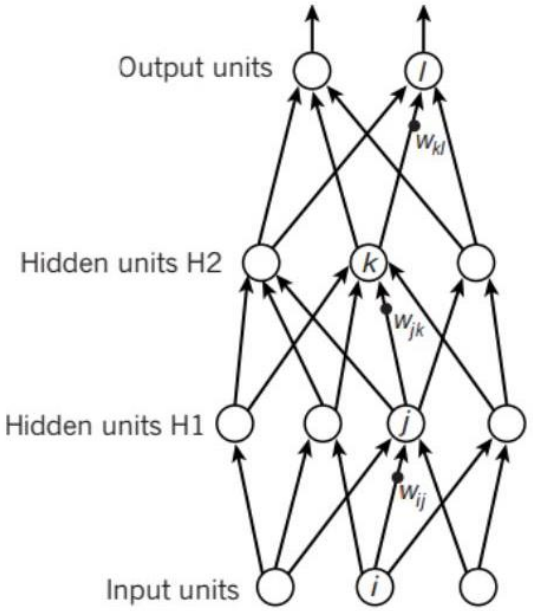




■ 机器学习与人工神经网络

人工神经网络的训练与参数优化：

网络前向计算



$$y_l = f(z_l)$$
$$z_l = \sum_{k \varepsilon \text{ H2}} w_{kl} y_k$$

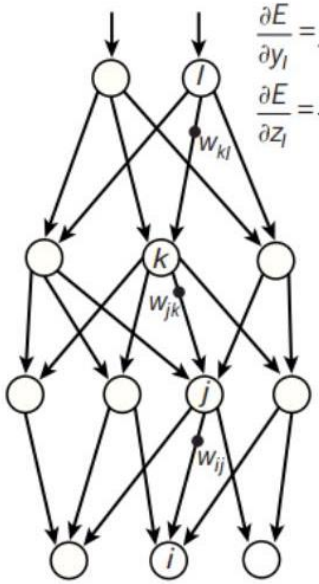
$$y_k = f(z_k)$$
$$z_k = \sum_{j \varepsilon \text{ H1}} w_{jk} y_j$$

$$y_j = f(z_j)$$
$$z_j = \sum_{i \varepsilon \text{ Input}} w_{ij} x_i$$

误差反向传播

$$\frac{\partial E}{\partial y_k} = \sum_{l \varepsilon \text{ out}} w_{kl} \frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$$

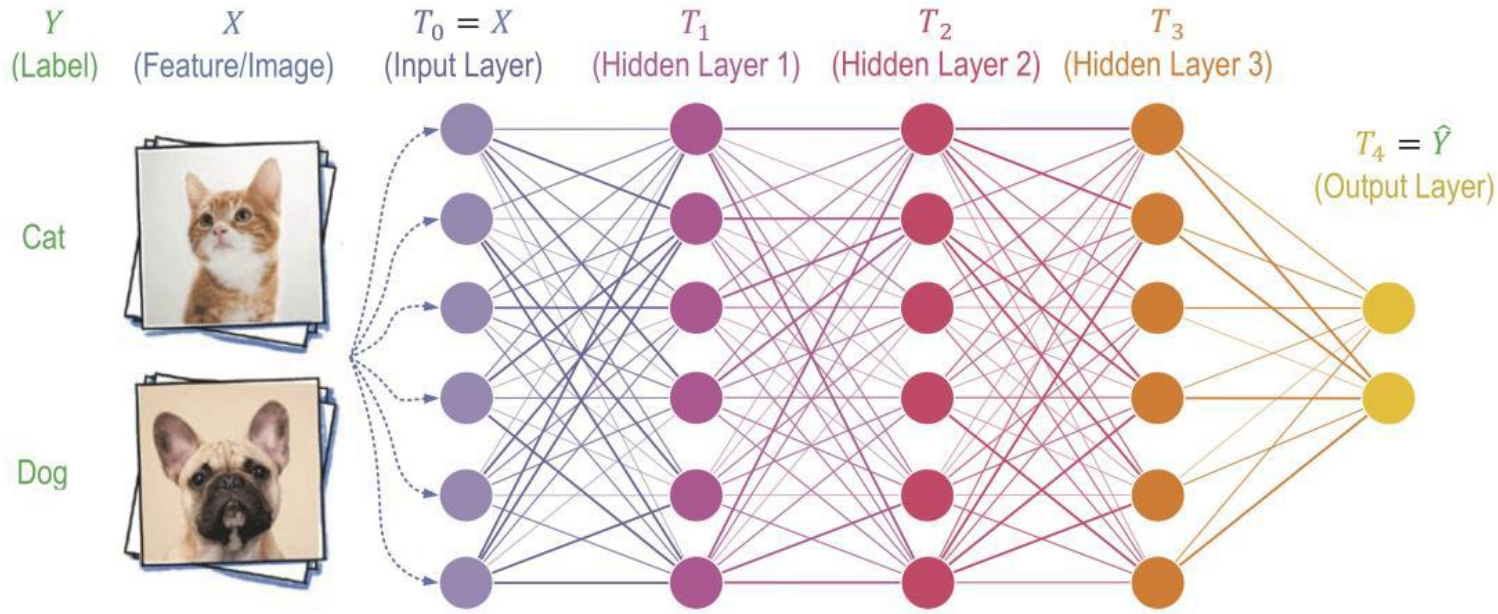
$$\frac{\partial E}{\partial y_j} = \sum_{k \varepsilon \text{ H2}} w_{jk} \frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$

Back Propagation算法 (反向传播)

■ 机器学习与人工神经网络

传统人工神经网络的局限性与深度学习：

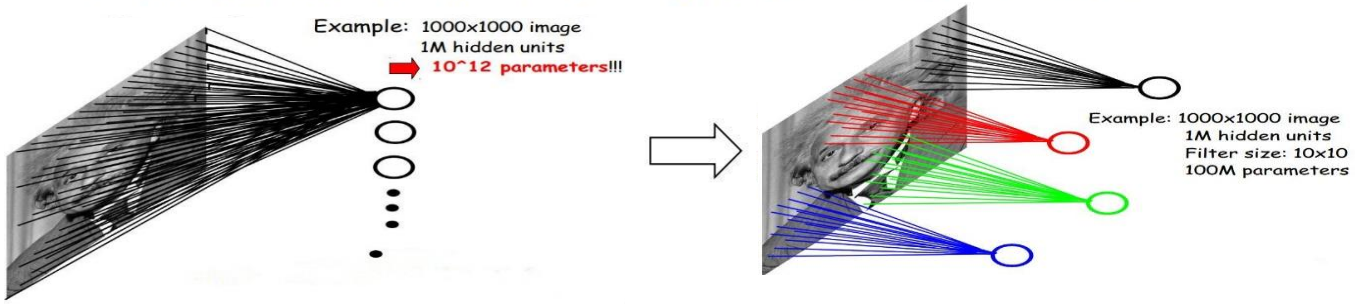


当隐藏层增多（模型变深），参数量呈指数增加，难以优化，模型训练代价极高。

■ 机器学习与人工神经网络

传统人工神经网络的局限性与深度学习：

- 稀疏连接: 输出层神经元只和部分输入层神经元相连 感受野

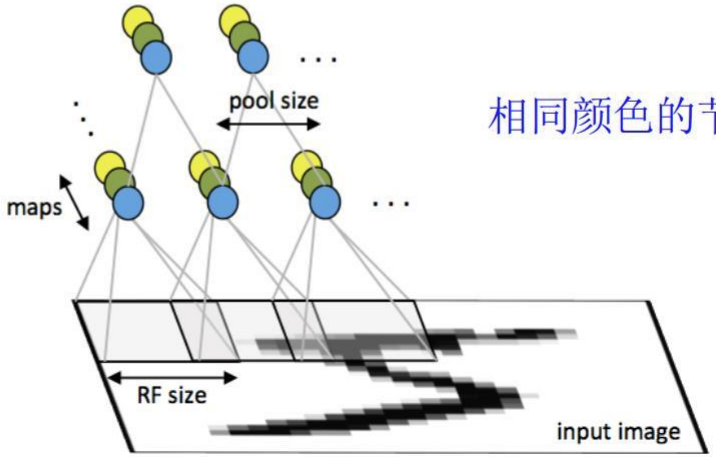


- 权值共享

Pooling层： 3x22x22

Conv层: 3x24x24
Kernel Size: 5x5
Stride: 1

Data层: 1x28x28



相同颜色的节点共享权值

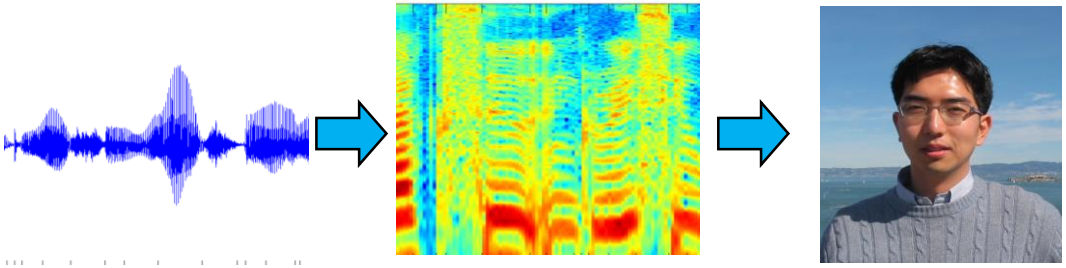
权值共享与感受野的引入，使得需要优化的参数量大大减少。

什么是深度学习?

Object Detection

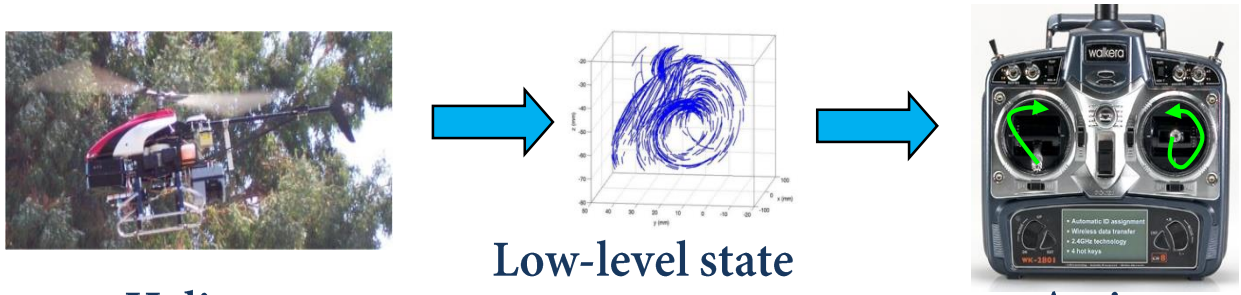
Image → Low-level vision features → Recognition

Audio Classification



Audio → Low-level audio features → Speaker identification

Helicopter control

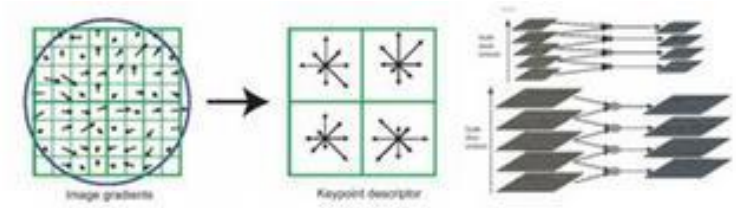


Helicopter → Low-level state features → Action

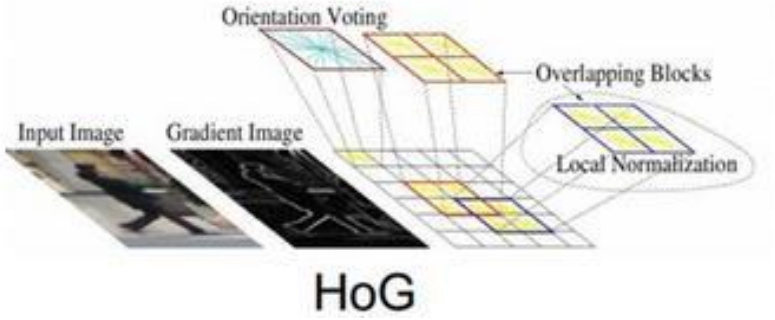
什么是深度学习?

传感器 → 特征表达 → 学习算法

- 深度学习是一种自动学习特征的方法



SIFT



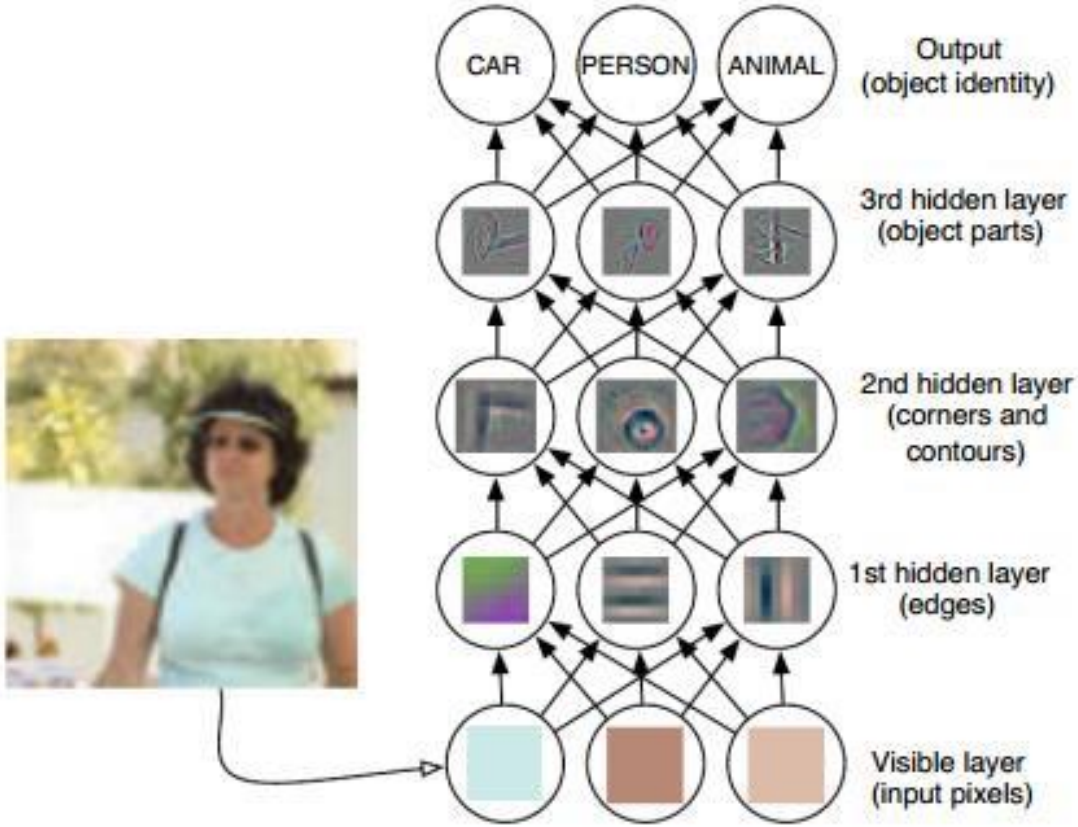
HoG

人工设计特征

Feature learning

深度学习自动学习特征!

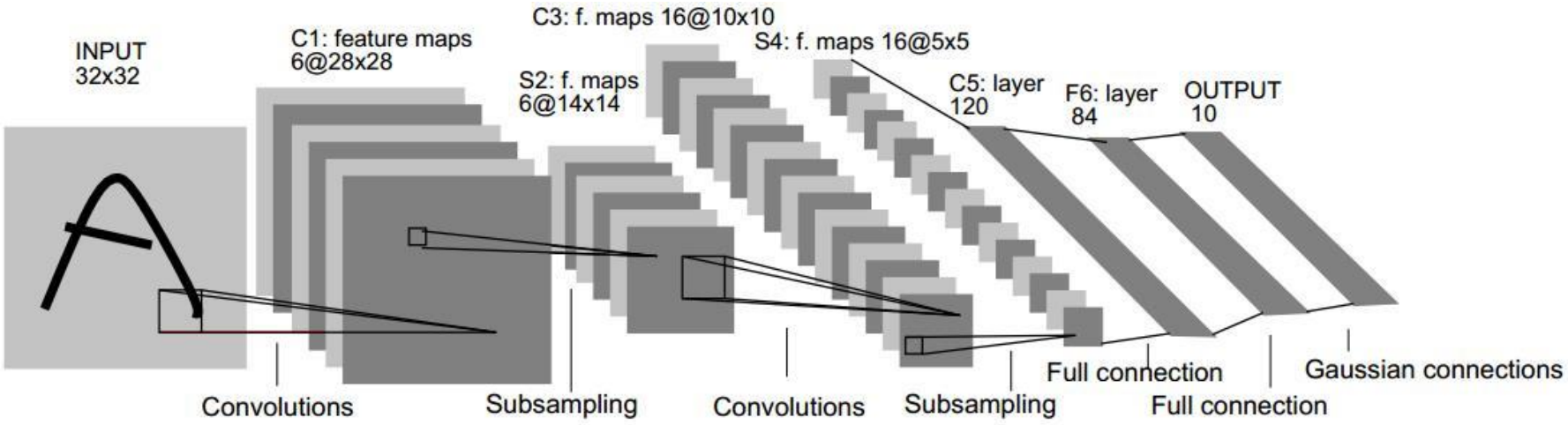
什么是深度学习?



“深度”

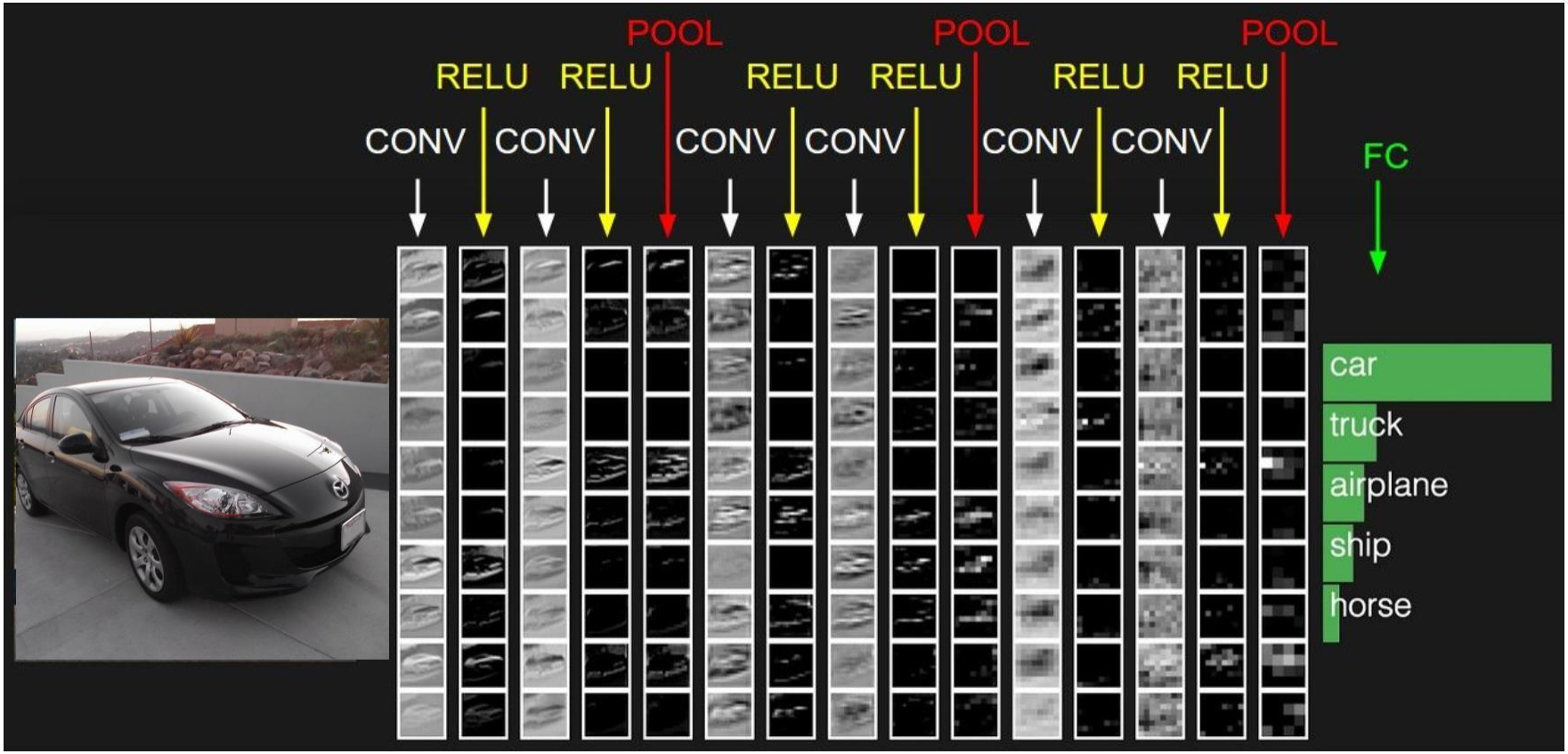
通过组合低层特征形成更加抽象的高层特征

CNN模型



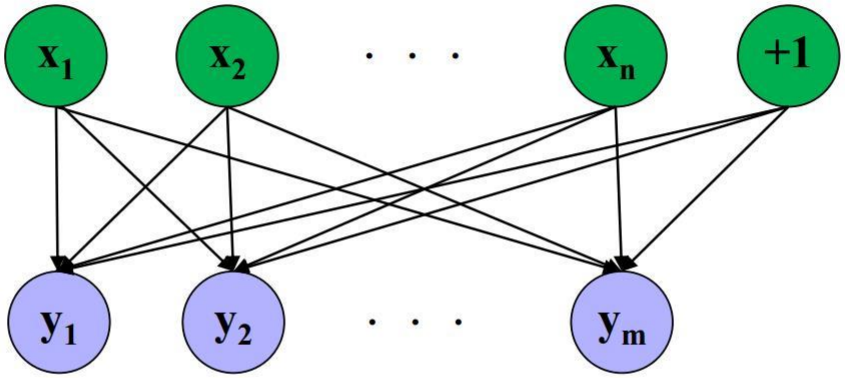
卷积层　　池化层　　全连接层

CNN基础结构


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

全连接层——原有神经网络中的层

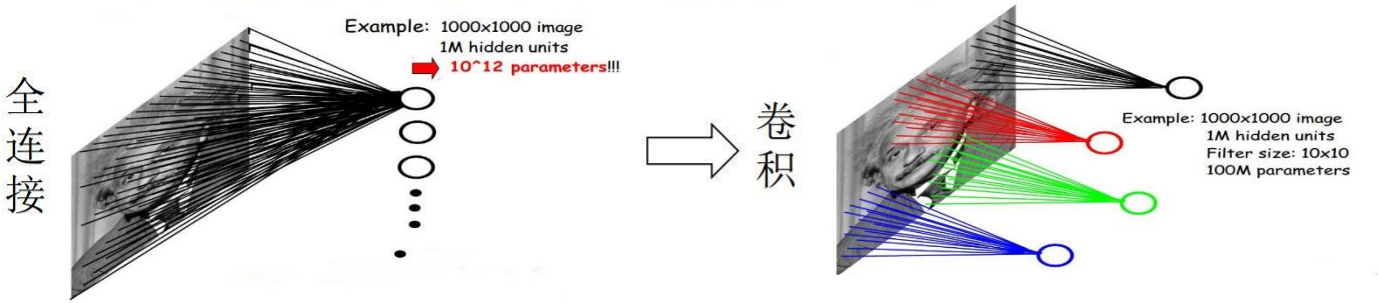
- 相当于内积运算，图中“+1”表示偏置项b
- 输出层的神经元和输入层的每个神经元都相连：得名“全”连接



- Forward运算：$y = W^T x + b$，其中$y \in R^{m\times 1}, x \in R^{n\times 1}, W \in R^{n\times m}$
- Backward运算：$\frac{\partial L}{\partial x} = W * \frac{\partial L}{\partial y}$，$\frac{\partial L}{\partial W} = x * \left(\frac{\partial L}{\partial y}\right)^T$

卷积层

- 稀疏连接: 输出层神经元只和部分输入层神经元相连 感受野

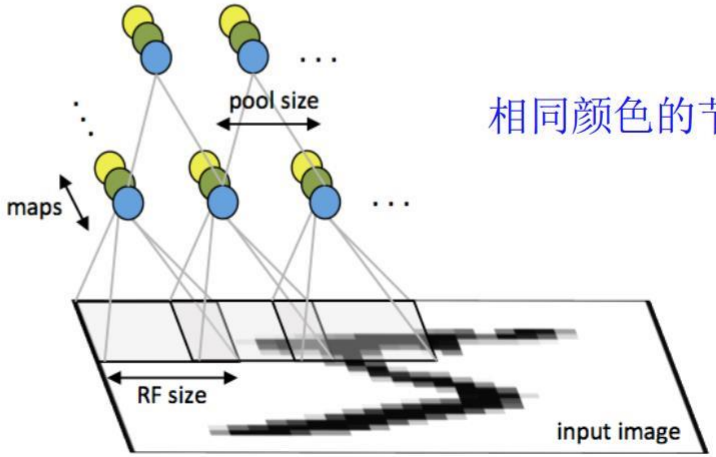


- 权值共享

Pooling层： 3x22x22

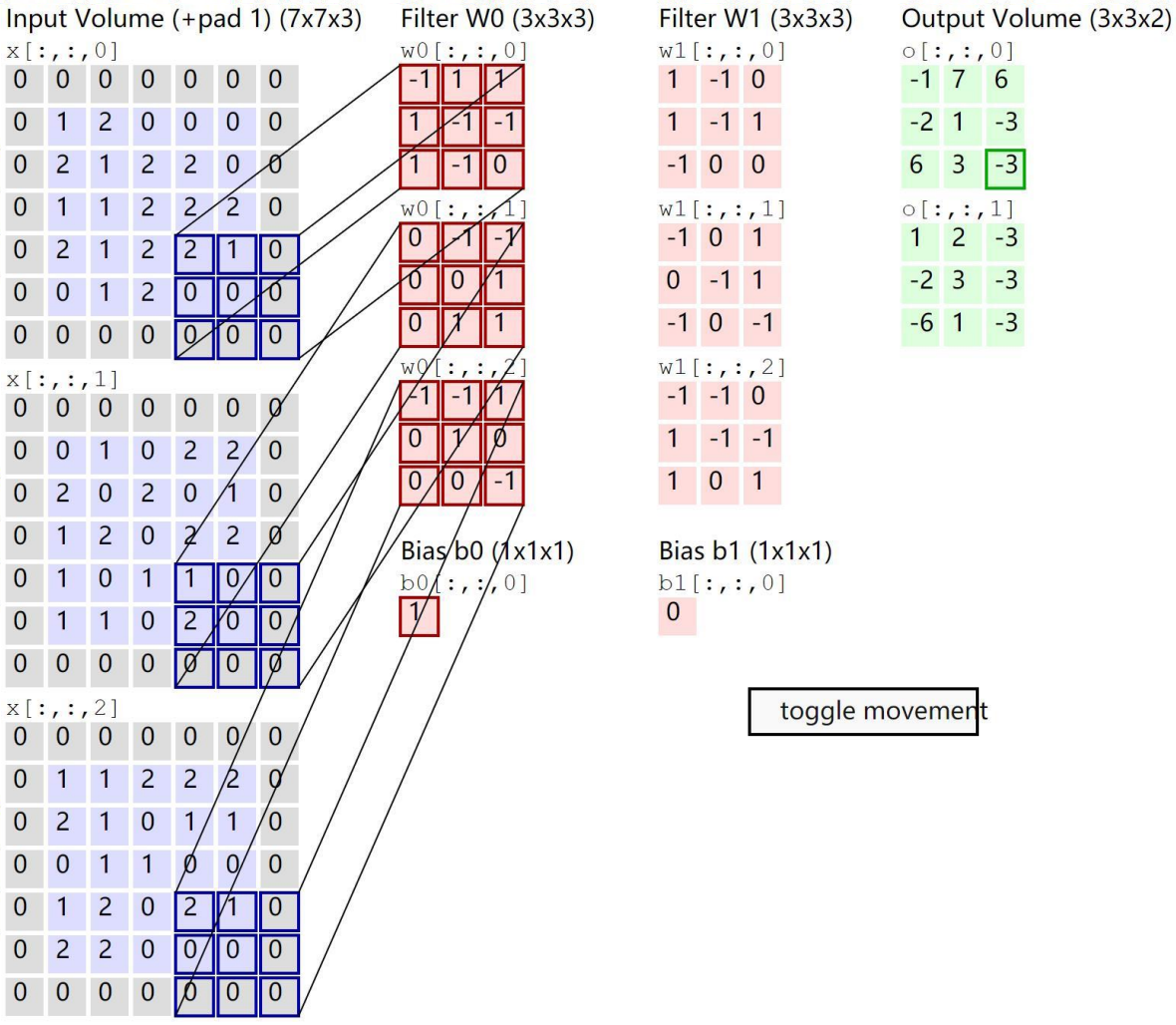
Conv层: 3x24x24
Kernel Size: 5x5
Stride: 1

Data层: 1x28x28



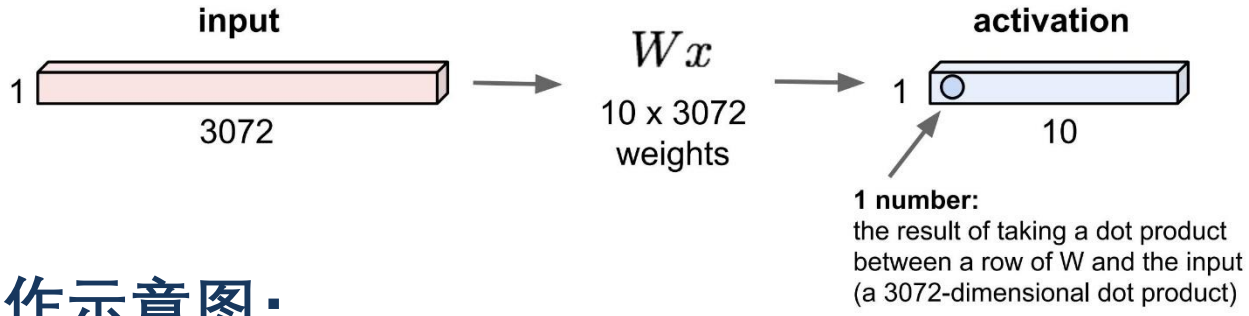
相同颜色的节点共享权值

卷积层

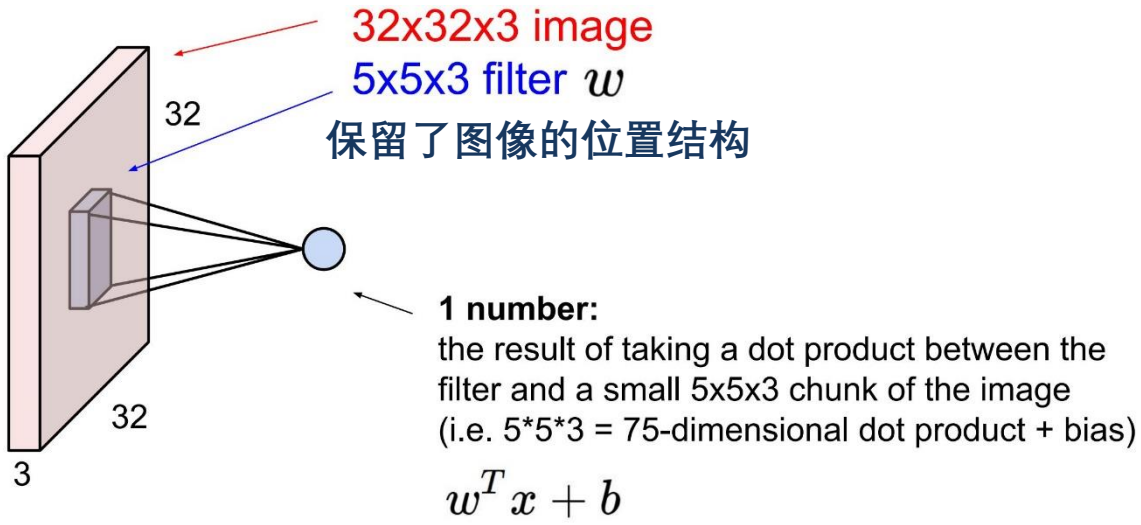

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 1 2 2 0 0
0 1 1 2 2 2 0
0 2 1 2 2 1 0
0 0 1 2 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 1 0 2 2 0
0 2 0 2 0 1 0
0 1 2 0 2 2 0
0 1 0 1 1 0 0
0 1 1 0 2 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 2 2 0
0 2 1 0 1 1 0
0 0 1 1 0 0 0
0 1 2 0 2 1 0
0 2 2 0 0 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 1 1
1 -1 -1
1 -1 0
w0[:,:,1]
0 -1 -1
0 0 1
0 1 1
w0[:,:,2]
-1 -1 1
0 1 0
0 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 -1 0
1 -1 1
-1 0 0
w1[:,:,1]
-1 0 1
0 -1 1
-1 0 -1
w1[:,:,2]
-1 -1 0
1 -1 -1
1 0 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 7 6
-2 1 -3
6 3 -3
o[:,:,1]
1 2 -3
-2 3 -3
-6 1 -3
toggle movement

全连接层示意图：

32x32x3 image -> stretch to 3072 x 1

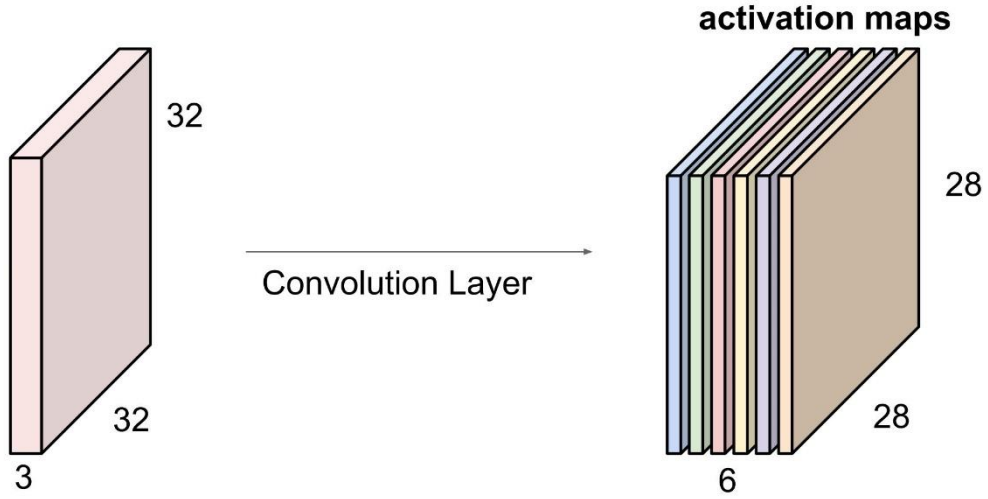


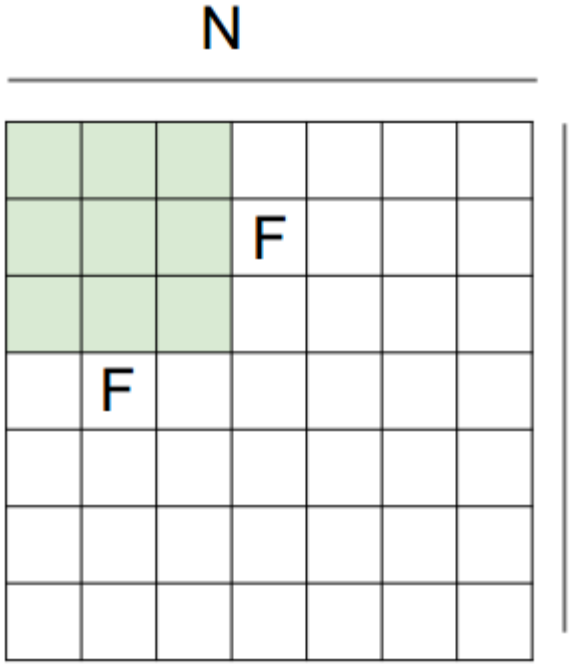
卷积操作示意图：



卷积操作示意图：

如果有6个5*5的卷积核，那么最终将得到6个分离的激活图





输入尺寸：N*N

卷积核尺寸：F*F

输出尺寸：

(N - F) / stride + 1

e.g. N = 7, F = 3:

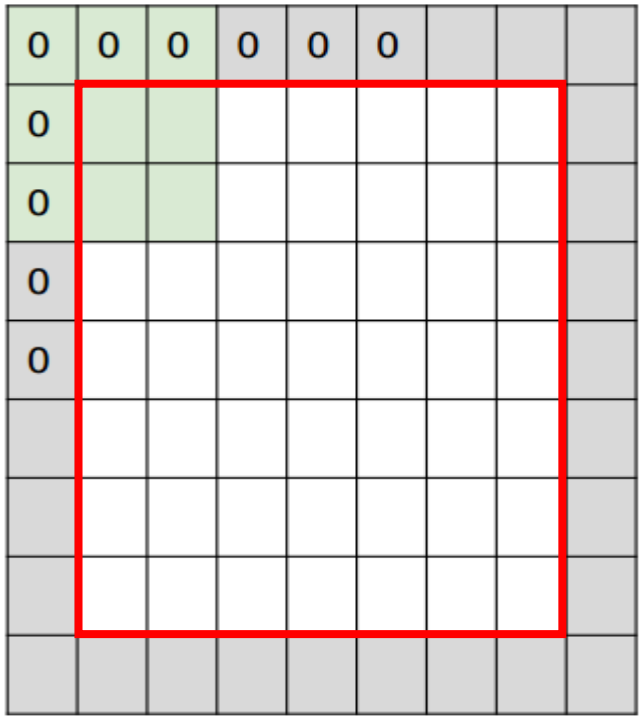
stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33

Stride：x, y方向移动的步长

实际应用中采用0填充边界

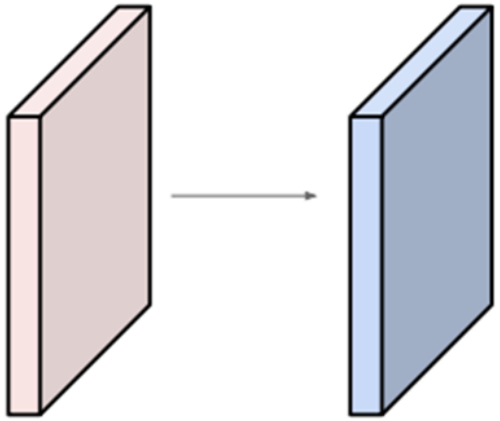


例如：输入7×7
卷积核是3×3，步长为1，填充1个像素的边界
输出大小任然是7×1
通常卷积层步长为1×1，卷积核是F×F，填充（F - 1）/ 2个像素的边界
例如：
F = 3 => 填充1个像素的边界
F = 5 => 填充2个像素的边界
F = 7 => 填充3个像素的边界

实例：

输入体积：32×32×3

10 个5×5的卷积核，步长为1，边界填充2个像素的0

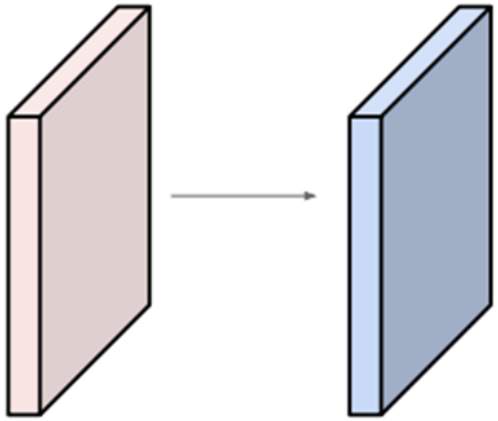


输出体积？

实例：

输入体积：32×32×3

10 个5×5的卷积核，步长为1，边界填充2个像素的0



输出体积：　(N-F)/Stride + 1

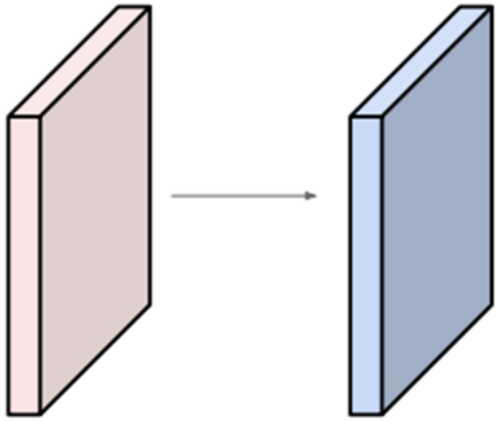
（32 + 2 * 2 - 5）/ 1 + 1 =32

因此输出体积：32×32×10

实例：

输入体积：32×32×3

10 个5×5的卷积核，步长为1，边界填充2个像素的0

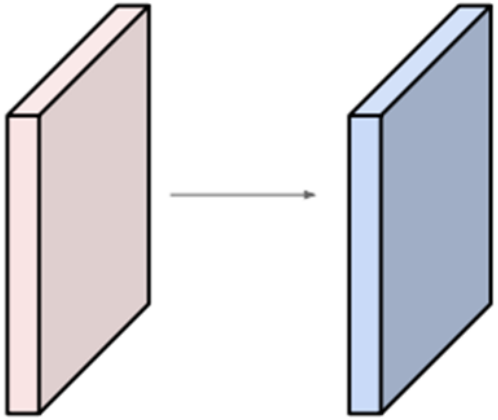


这层卷积层的参数量？

实例：

输入体积：32×32×3

10个 5×5的卷积核，步长为1，边界填充2个像素的0



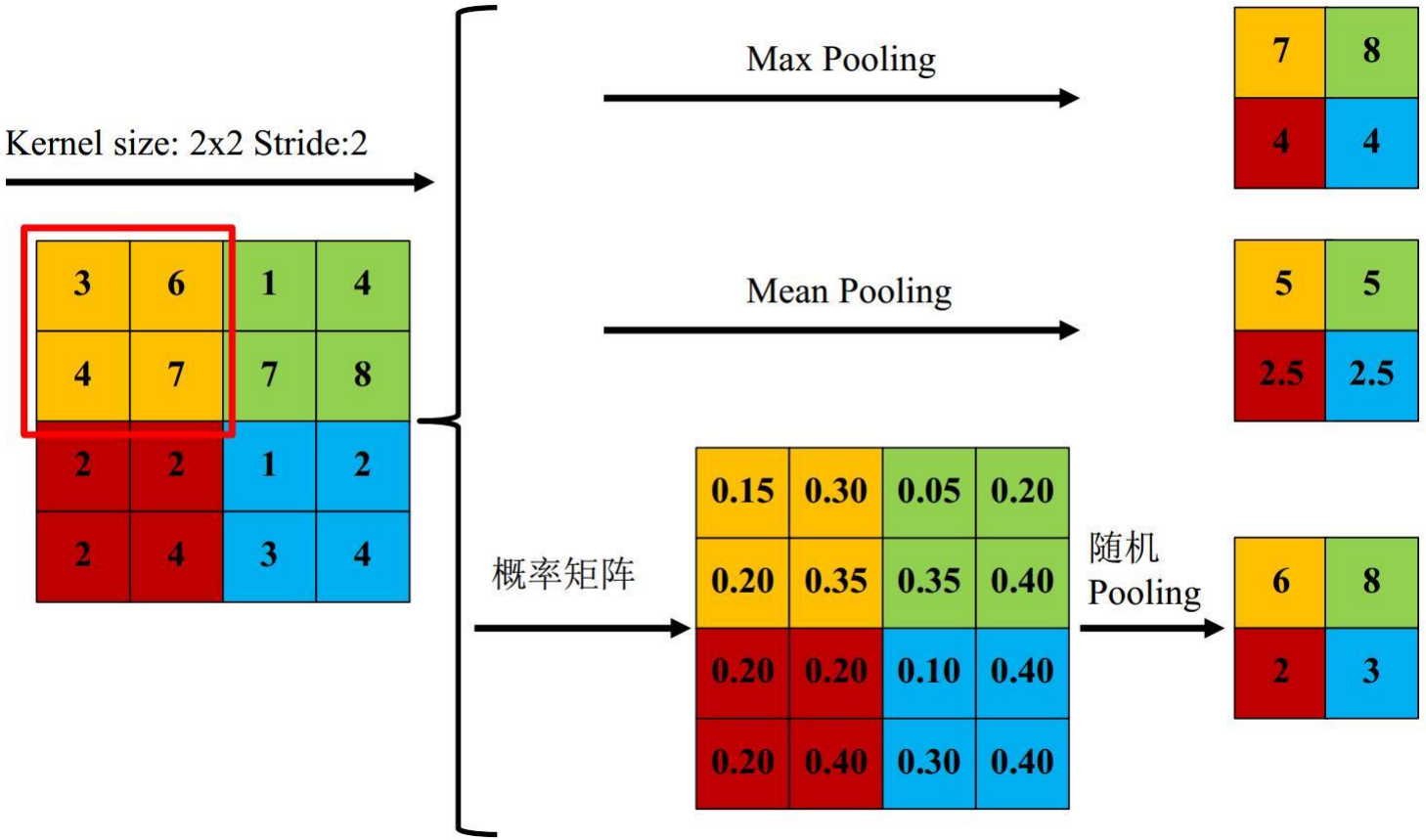
这层卷积层的参数量？

每个滤波器有：5 × 5 × 3 + 1 = 76 个参数=> 76 × 10 = 760

池化层

——作用：逐渐降低数据体的空间尺寸，减少网络中参数的数量，有效控制过拟合。

- 一般配合卷积层使用



激活函数种类：

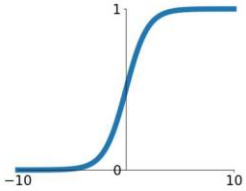
如果不用激励函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合。

激活函数给神经元引入了**非线性因素**，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。

Activation functions

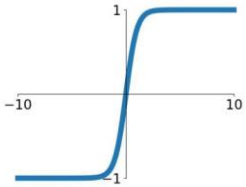
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



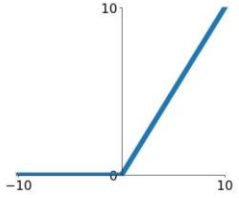
tanh

$$\tanh(x)$$



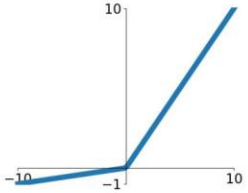
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

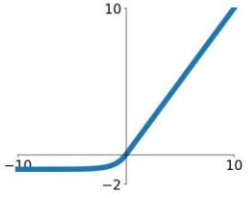


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



激活函数选择：

（1） Sigmoid

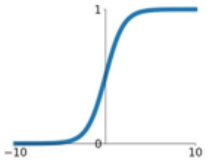
Sigmoid的优点：输出范围0~1；对神经元的激活频率有良好的解释

Sigmoid的缺点：函数饱和使梯度消失；输出不是零中心的

可能导致模型收敛速度慢

Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



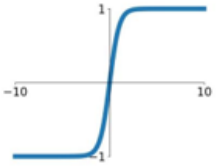
（2） Tanh的优点：输出范围-1~1，并且是零中心的

Tanh的缺点：函数饱和使梯度消失；

tanh

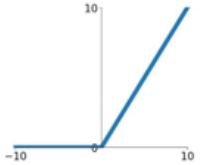
$\tanh(x)$

双曲正切



ReLU

$\max(0, x)$



（3） ReLU

ReLU的优点：对随机梯度下降的收敛有巨大的加速作用；节省计算资源

ReLU的缺点：梯度为0，神经元无法再次激活

激活函数选择：

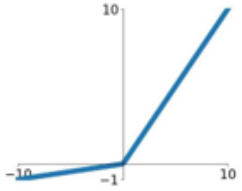
（4） Leaky ReLU

Leaky ReLU的优点：解决了ReLU的缺点

Leaky ReLU的缺点：有研究表明效果不稳定

Leaky ReLU

$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

（5） Maxout

Maxout的优点：Maxout是对ReLU和leaky ReLU的一般化归纳，没有ReLU的缺点

Maxout的缺点：和ReLU对比，每个神经元的参数量增加了一倍，整体参数的量激增

总结：优先用ReLU函数，注意设置学习率；可以试试Leaky ReLU、Maxout、tanh。

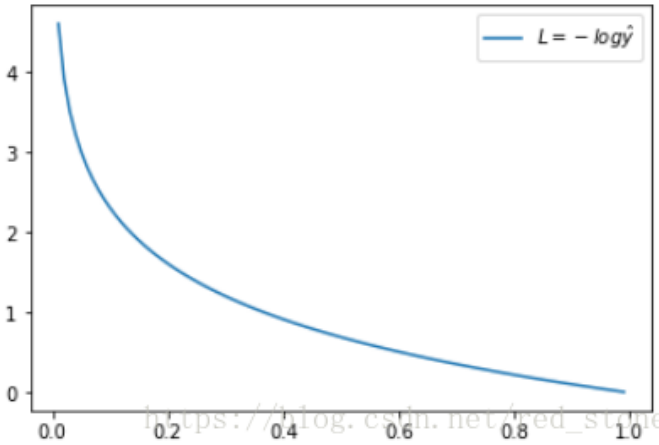
损失函数

- **Softmax Loss**

损失函数：$E=\frac{-1}{N}\sum_{n=1}^{N}\log(\hat{p}_{nl_n}), l_n \in [0,1,...,K-1]$

其中 $\hat{p}_{nl_n}$ 由softmax函数计算 $\hat{p}_{nk}=\frac{e^{x_{nk}}}{\sum_{l=0}^{K-1}e^{x_{nl}}}$

适用场景：单标签分类问题



- **Euclidean Loss**

损失函数：$E=\frac{-1}{N}\sum_{n=1}^{N}\|\hat{y}_n - y_n\|_2^2$

适用场景：实数值回归问题

- **Sigmoid Cross Entropy Loss**

损失函数：$E=\frac{-1}{N}\sum_{n=1}^{N}[p_n \log \hat{p}_n + (1-p_n)\log(1-\hat{p}_n)]$

适用场景：预测目标概率分布，可用于多标签学习

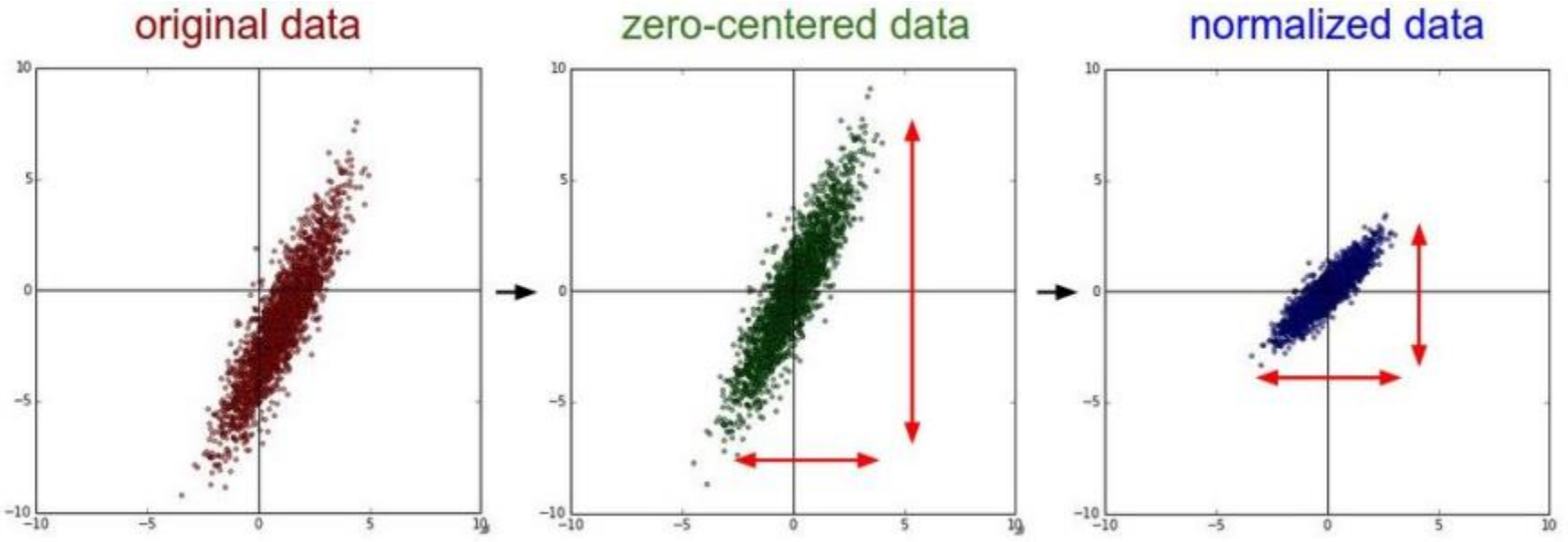
- **Contrastive Loss**

损失函数：$E=\frac{-1}{N}\sum_{n=1}^{N}[(y)d^2 + (1-y)\max(\text{margin}-d,0)^2, d=\|a_n - b_n\|_2$

适用场景：深度度量学习

数据预处理（Data Preprocessing）

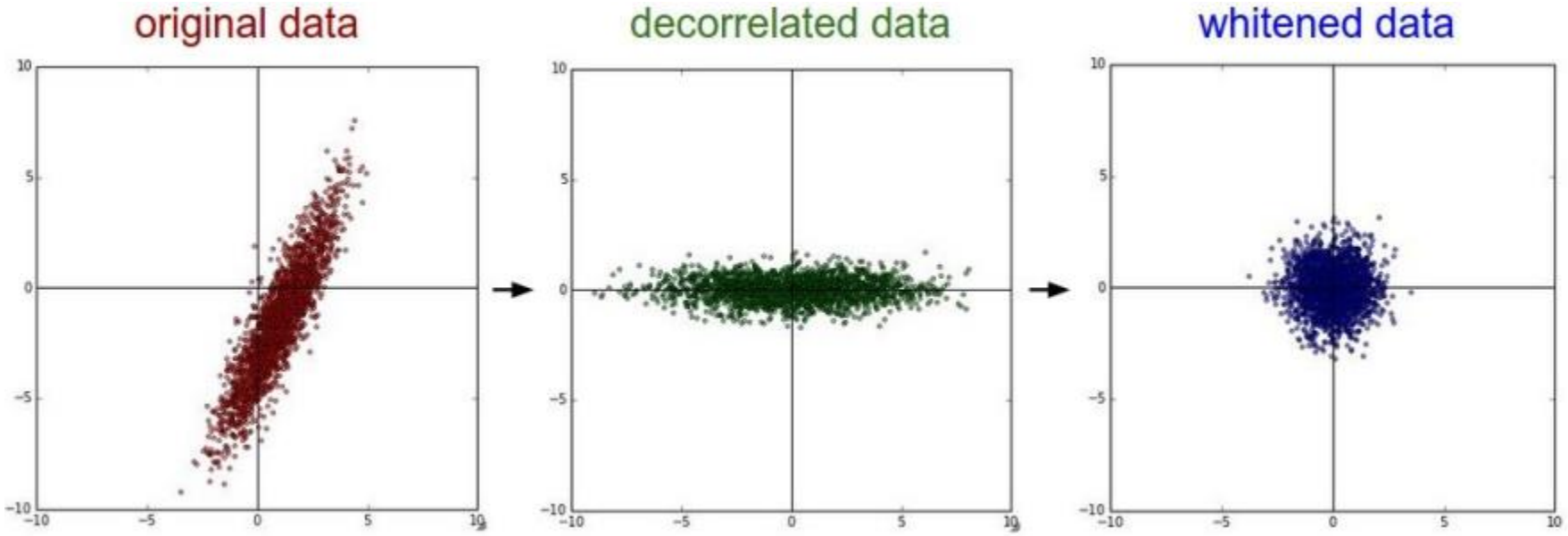
去均值和归一化：



一般数据预处理流程：**左边**：原始的2维输入数据。**中间**：在每个维度上都减去平均值后得到零中心化数据，现在数据云是以原点为中心的。**右边**：每个维度都除以其标准差来调整其数值范围。

数据预处理（Data Preprocessing）

PCA和白化：



协方差是对角矩阵

协方差是单位矩阵

左边是二维的原始数据。**中间：**经过PCA操作的数据。可以看出数据首先是零中心的，然后变换到了数据协方差矩阵的基准轴上。这样就对数据进行了去相关（协方差矩阵变成对角阵）。**右边：**每个维度都被特征值调整数值范围，将数据协方差矩阵变为单位矩阵。从几何上看，就是对数据在各个方向上拉伸压缩，使之变成服从高斯分布的一个数据点分布。

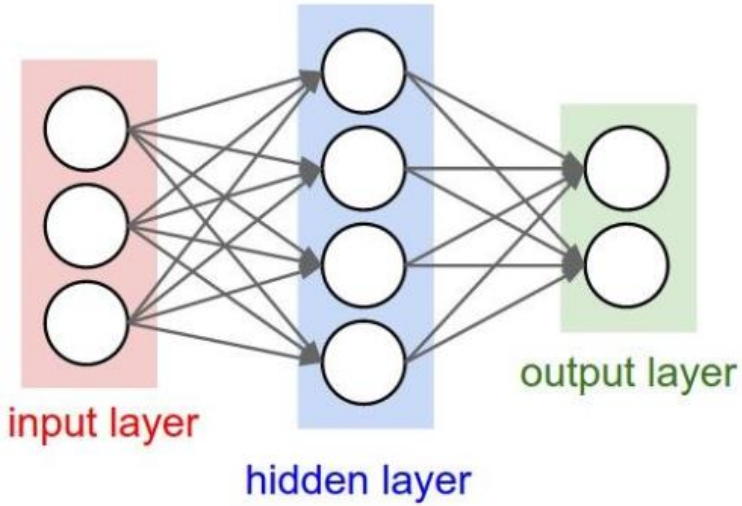
权重初始化

1、全零初始化（错误的初始化方式）

2、小随机数初始化（0均值，标准正态分布）
W = 0.01 * np.random.randn(D, H)

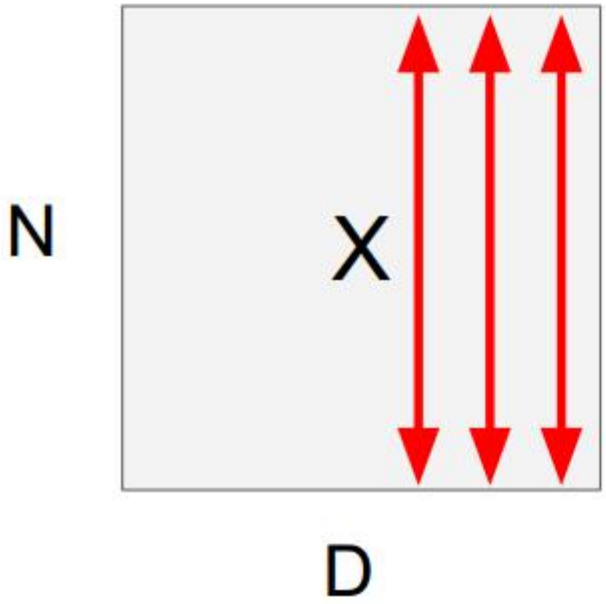
3、使用1/sqrt(n)校准方差（Xavier 初始化）
W = np.random.randn(D, H) / np.sqrt(D)

通常用sqrt(D / 2) 代替sqrt(D)



Batch normalization(2015)

随着训练过程中参数的改变，该层的输出数据的分布可能会改变；此时，对于下一层，相当于输入数据的分布改变了，这种输入数据分布的改变，可能会使 DNN 的难以学习到好的参数，从而影响 DNN 的效果。DNN 为了补偿输入数据分布改变带来的损失，需要更多的时间来调整参数，这可能会使训练速度下降



1、在每一个维度上计算样本均值和方差

2、归一化

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

通常加在卷积层或者全连接层后面，非线性变换前面

Batch normalization(2015)

- 逐层尺度归一，避免了梯度消失和梯度溢出
- 加速收敛5x~20x, 同时作为一种正则化技术也提高了泛化能力

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

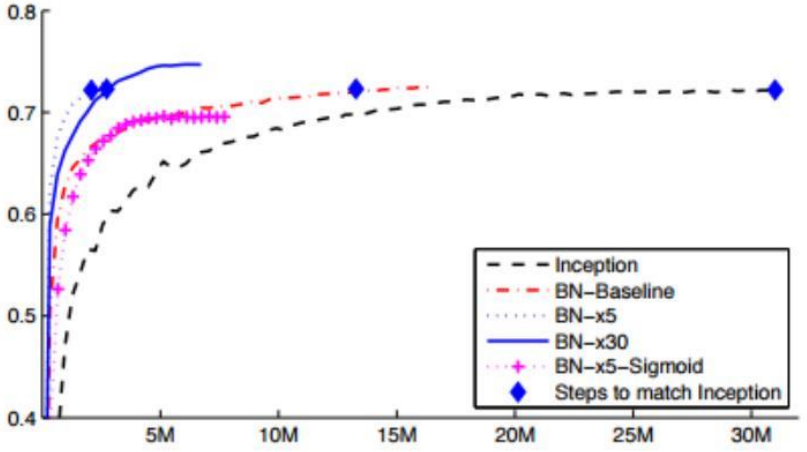
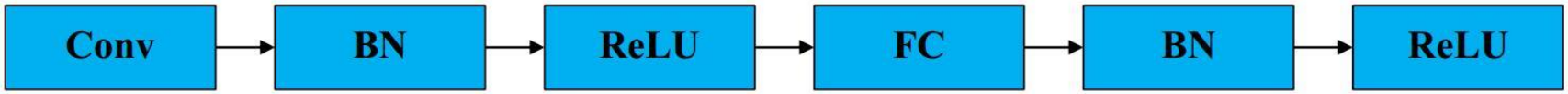


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*



L1正则化

对于每个w都向目标函数增加一个$\lambda|w|$。L1正则化会让权重向量在优化的过程中变得稀疏（非常接近0）对于噪音输入则几乎是不变的。

L2正则化

对于网络中的每个权重w，向目标函数中增加一个$\frac{1}{2}\lambda w^2$。L2正则化可以直观理解为它对于大数值的权重向量进行严厉惩罚。

L1+L2：$\lambda_1|w| + \frac{1}{2}\lambda_2 w^2$

Dropout(2012)

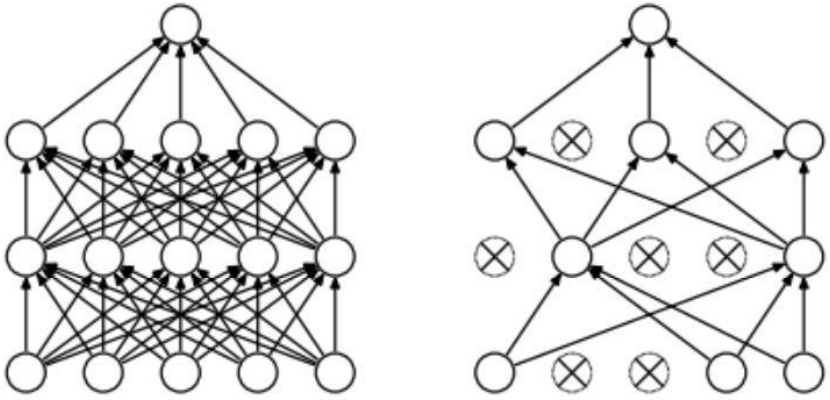
- 引入Bernoulli随机数u, p代表dropout ratio

$$y_{\text{train}} = \begin{cases} \frac{x}{1-p} & \text{if } u > p \\ 0 & \text{otherwise} \end{cases} \qquad \text{Where, } u \sim U(0,1)$$

$$E(y_{\text{train}}) = p \cdot 0 + (1-p)\frac{E(x)}{1-p} = E(x)$$

- 测试阶段：Do Nothing

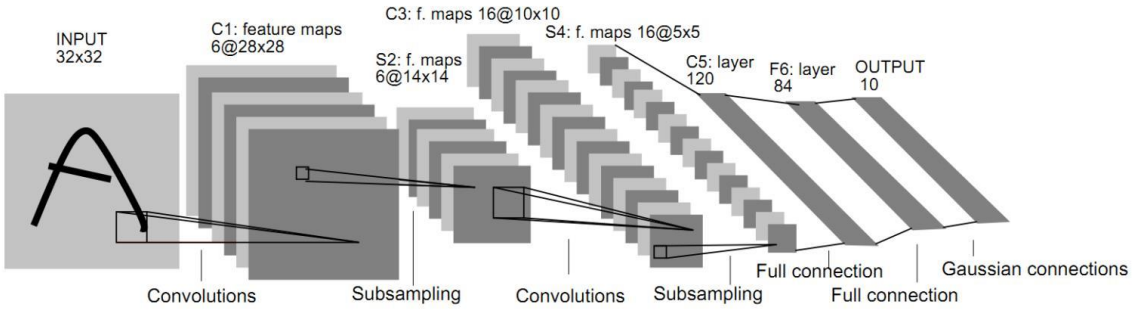
- 正则化手段，提高泛化能力



(a) Standard Neural Net (b) After applying dropout.

LeNet(1998)——早期尝试

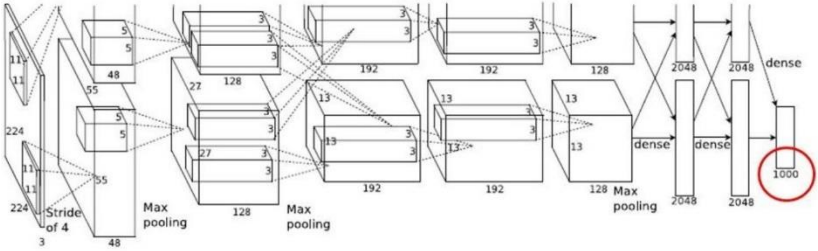
LeNet



- Convolution:
  - locally-connected
  - spatially weight-sharing
    - weight-sharing is a key in DL (e.g., RNN shares weights temporally)
- Subsampling
- Fully-connected outputs
- Train by BackProp

- All are still the basic components of modern ConvNets!

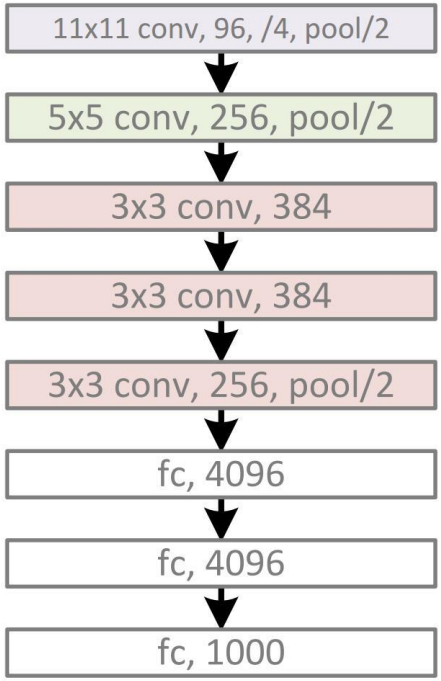
AlexNet(2012)——历史突破



AlexNet

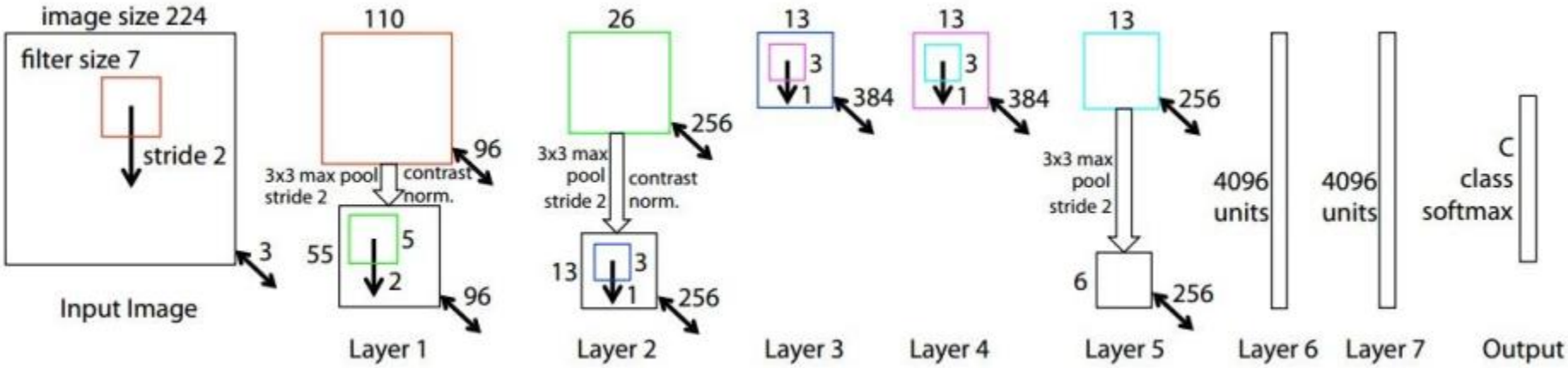
LeNet-style backbone, plus:

- ReLU [Nair & Hinton 2010] 非线性激活函数
  - "RevoLUtion of deep learning"*
  - Accelerate training; better grad prop (vs. tanh)
- Dropout [Hinton et al 2012] 防止过拟合
  - In-network ensembling
  - Reduce overfitting (might be instead done by BN)
- Data augmentation 数据增广
  - Label-preserving transformation
  - Reduce overfitting



实现了大数据训练：百万级ImageNet图像数据

ZFNet(2013)——ILSVRC-13



AlexNet改进:

CONV1: 将(11x11 stride 4) 改为 (7x7 stride 2)

CONV3,4,5: 将滤波器数量384, 384, 256 改为用512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

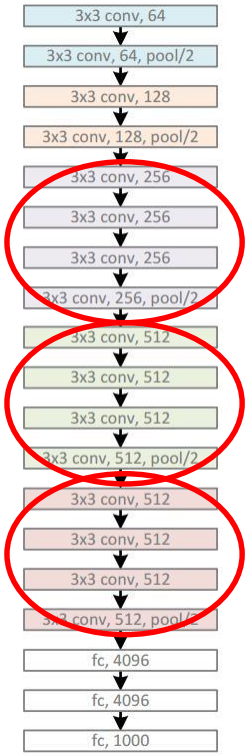
VGGNet(2014)——网络加深

Simply “Very Deep”!

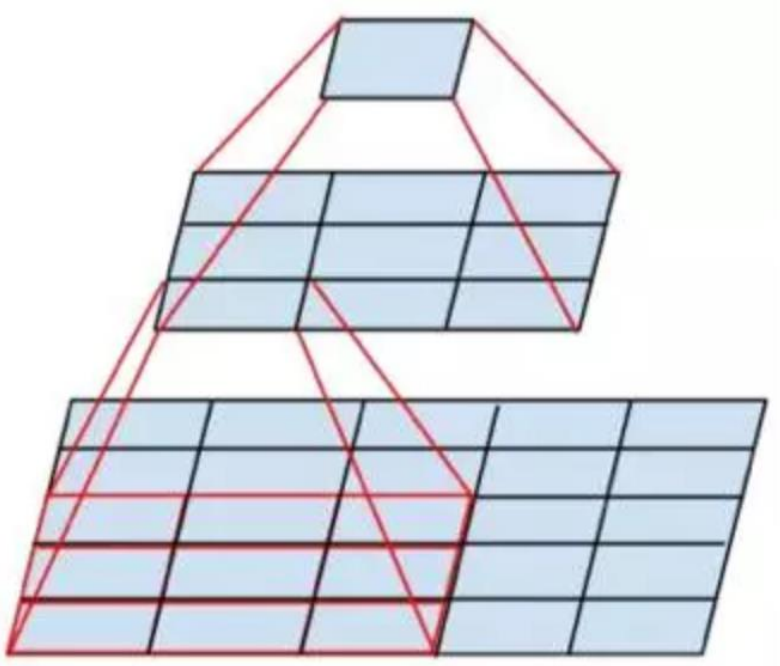
- Modularized design 模块化设计
  - 3x3 Conv as the module
  - Stack the same module
  - Same computation for each module (1/2 spatial size => 2x filters)

- Stage-wise training
  - VGG-11 => VGG-13 => VGG-16
  - We need a better initialization…

反复堆叠的
3*3小卷积核



VGGNet(2014)——网络加深



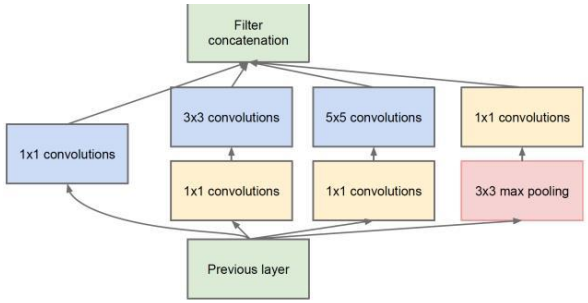
为什么用反复堆叠的3*3小卷积核代替大的(5*5、7*7)卷积核？

1. **感受野相同**——两个3*3的卷积层串联相当于1个5*5的卷积层，即一个像素会跟周围5*5的像素产生关联；
2. **更少的参数量**——3个的3*3的卷积层只有一个7*7卷积层参数量的55%；
3. **CNN对特征的学习能力更强**——3个3*3的卷积层拥有比1个7*7的卷积层更多的非线性变换

GoogLeNet(2014)——增加卷积模块功能

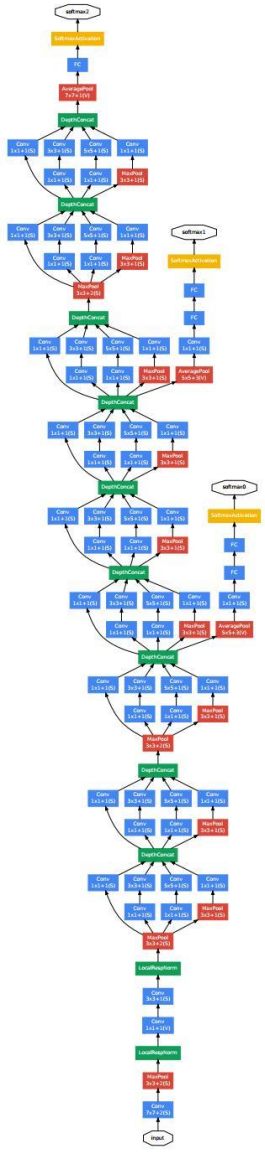
目的：控制计算量和参数量

1. 采用卷积模块——替代传统的卷积层进行卷积，高效的增加了网络的宽度和深度；
2. Batch Normalization——避免了梯度溢出和消失，提升了模型的泛化能力
3. Bottleneck设计——在3*3/5*5卷积之前先用1*1的卷积进行降维。这种设计可以减少网络参数及训练时间，使得训练更高效



Inception Module

在后续模型中得到了广泛的应用



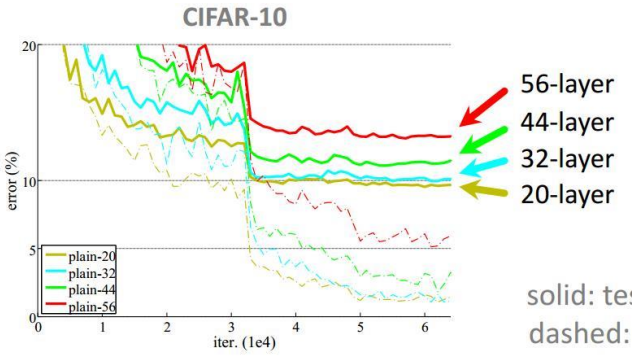
1*1卷积的作用：

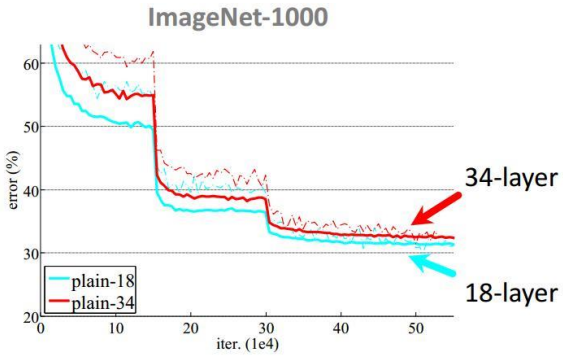
1. 降维　2. 增加非线性　3.跨通道组织信息

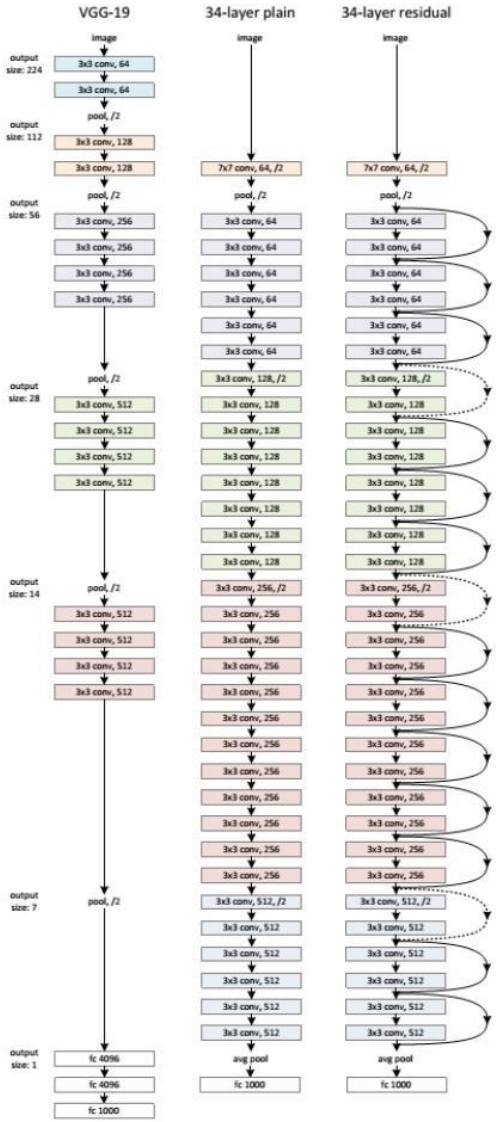
CNN结构演化

中国海洋大学 OCEAN UNIVERSITY OF CHINA

ResNet(2015)——解决网络深度问题







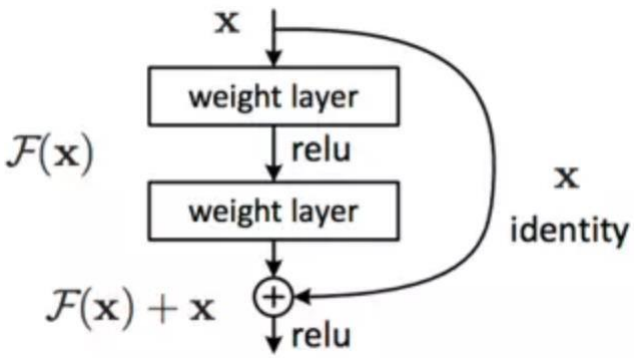
问题：

在不断加神经网络的深度时，会出现模型退化，即准确率会先上升然后达到饱和，再持续增加深度则会导致准确率下降

解决方法：

加入shortcut connection，提出残差学习

ResNet(2015)——解决网络深度问题



残差学习单元

假定某段神经网络的输入是x，期望输出是$H(x)$，如果直接把输入x传到输出作为初始结果，那么此时需要学习的目标就是$F(x)=H(x)-x$，即残差

ResNet在某种程度上解决了传递过程中的信息丢失问题，通过直接将输入信息绕道传到输出，保护信息的完整性，整个网络则只需要学习输入、输出差别的那一部分，简化学习目标和难度。

模型特点：

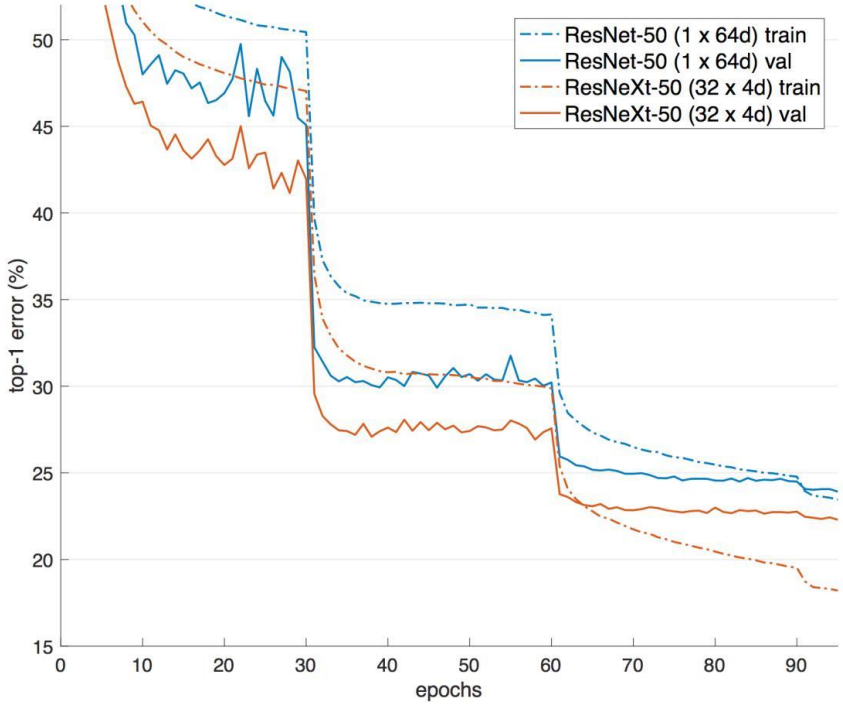
1.继承了VGGNet的重复模块堆叠风格，降低了模型设计的复杂度

2.沿用了GoogLeNet的Inception Module

3.提出残差学习，使得CNN网络的深度得到了极大的提升。成功训练了152层深的网络，一举拿下了当年ILSVRC比赛的冠军，top-5错误率降低至3.46%。

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i) \quad \rightarrow \quad \frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \left(1 + \frac{\partial}{\partial x_L} \sum_{i=l}^{L-1} F(x_i, W_i)\right)$$

ResNeXt(2016)——集成路线

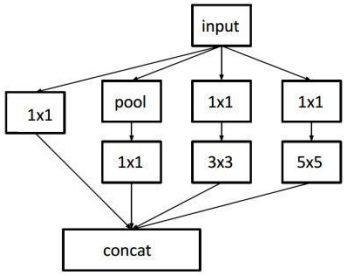


stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

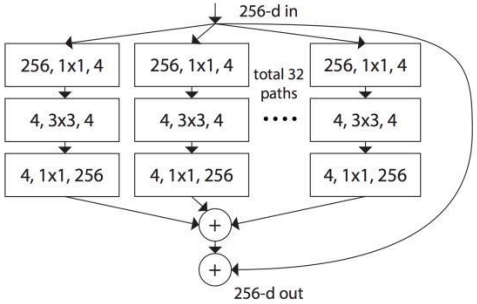
出发点：

随着网络越来越复杂，网络的超参数越来越多，这将影响网络的训练难度和普适性。而VGGNet和ResNet采取模块堆叠的方式减少了参数的数目，所以作者将这种设计和Inception module的精细结构进行了整合。

ResNeXt(2016)——集成路线



Inception:



ResNeXt:

特点：

1.类似于ResNet由用多个重复模块堆叠而成，并且有残差学习单元；

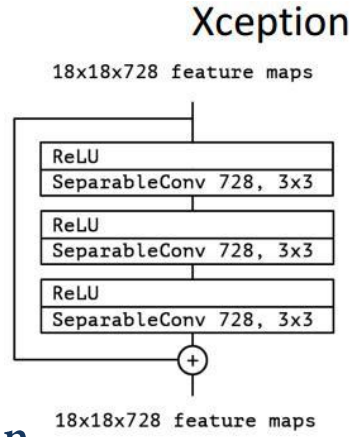
2.类似于inception module，每个模块有多条路径(path),但是ResNeXt中每个Path的结构是一样的；

3.针对path的数量，作者提出了一个新的维度cardinality，实验表明增加其维数比增加网络的深度和宽度更高效；

4.在超参数大大减少的情况下，网络的性能和复杂度可以与 ResNet 相当

轻量化网络——性能与效率的平衡

Xception(2017)，MobileNets(2017)



特点：

Xception=ResNet+改进后的Inception模块

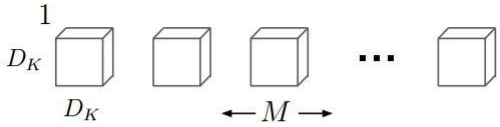
将Inception模块中的卷积操作变为depthwise separable convolution

什么是depthwise separable convolution：

=depthwise convolution + pointwise convolution

Depthwise convolution：

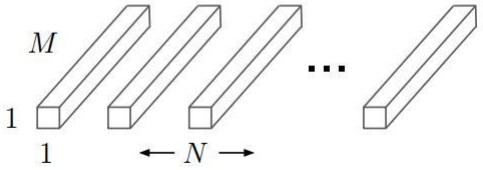
指的是输入的每个通道单独与卷积核做卷积，不同通道之间没有交流



(b) Depthwise Convolutional Filters

pointwise convolution：

用1*1的卷积进行跨通道交流

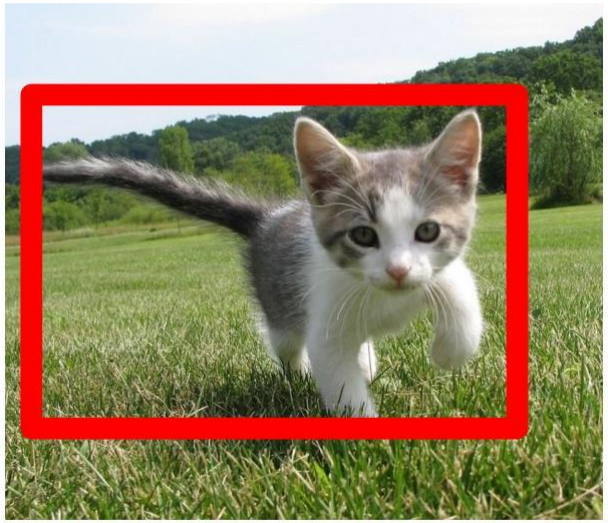


(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

计算量是标准卷积的 $= \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$

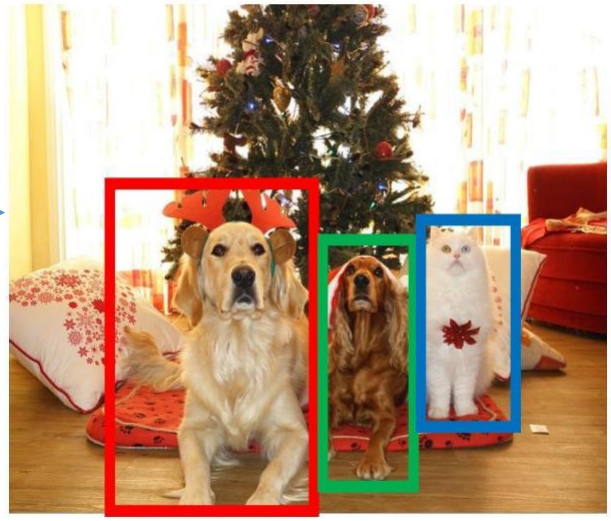
什么是目标检测?

Classification + Localization



CAT

Object Detection



DOG, DOG, CAT

传统目标检测方法

区域选择 → 特征提取 → 分类器分类

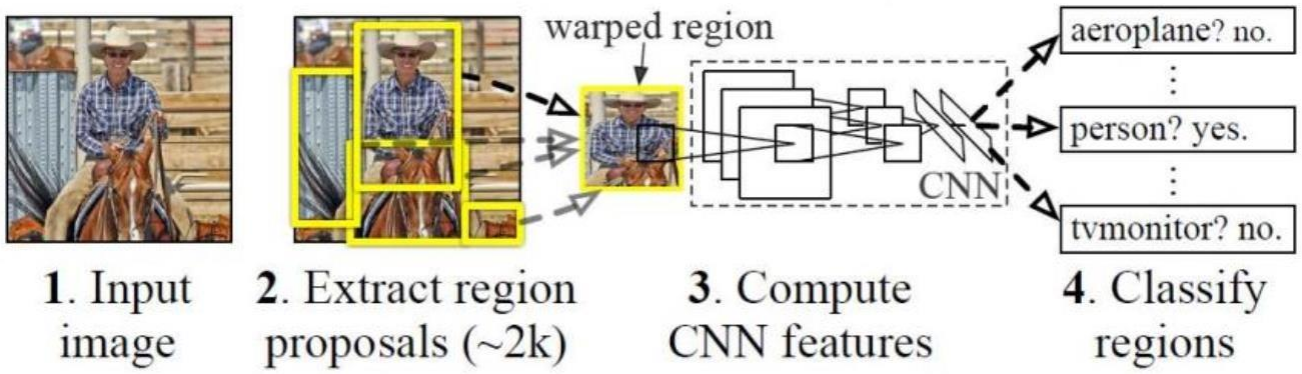
对目标的位置进行定位。由于不同大小的目标可能出现在图像的任何位置，所以最初采用滑动窗口的策略对整幅图像进行遍历，而且需要设置不同的尺度，不同的长宽比。

由于目标的形态多样性，光照变化多样性，背景多样性等因素使得我么需要设计一个鲁棒的特征(如SIFT、HOG等)

对目标所属类别进行分类。主要有SVM、Adaboost等

存在两个主要问题：一个是基于滑动窗口的区域选择策略没有针对性，时间复杂度高，窗口冗余；二是手工设计的特征对于多样性的变化并没有很好的鲁棒性。

R-CNN(2014, CVPR)——从分类到检测

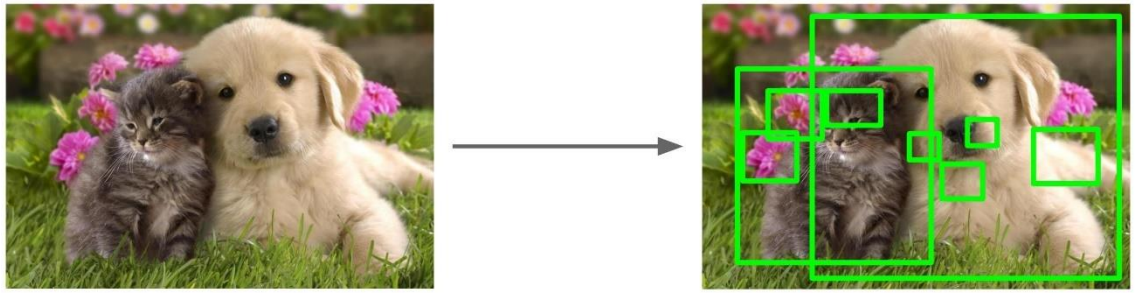


R-CNN用于目标检测的流程：

1. 提取2000个左右的region proposals，即感兴趣区域(ROI, region of interest)
2. 对于每一个region proposal将其缩放成227*227大小的图像输入到CNN中提取特征
3. 将提取到的每个region proposal的特征输入到SVM分类器中进行分类
4. 对检测框进行位置回归得到更为精确的位置

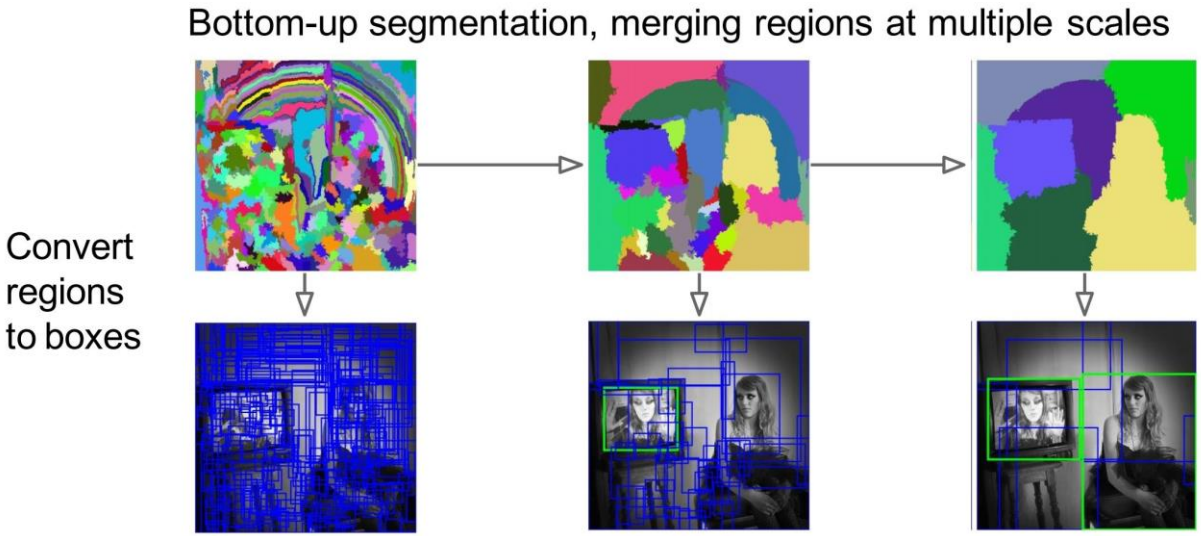
Object Detection

什么是region proposal?



就是可能包含目标的矩形区域。Region proposal减小了目标搜索的范围，使得我们不用对整张图像进行遍历。
R-CNN使用了selective search提取region proposal

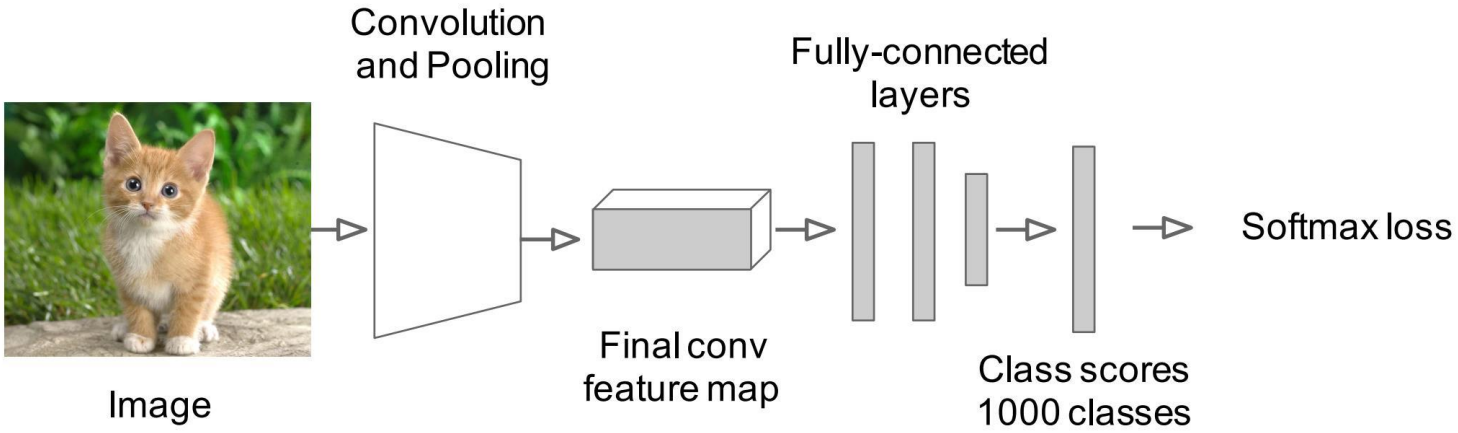
selective search主要思想：
利用过分割方法将图像分割成众多小区域，然后对按照合并规则合并可能性最高的相邻两个区域，最终得到候选区域



R-CNN——训练流程

R-CNN Training

Step 1: Train (or download) a classification model for ImageNet (AlexNet)

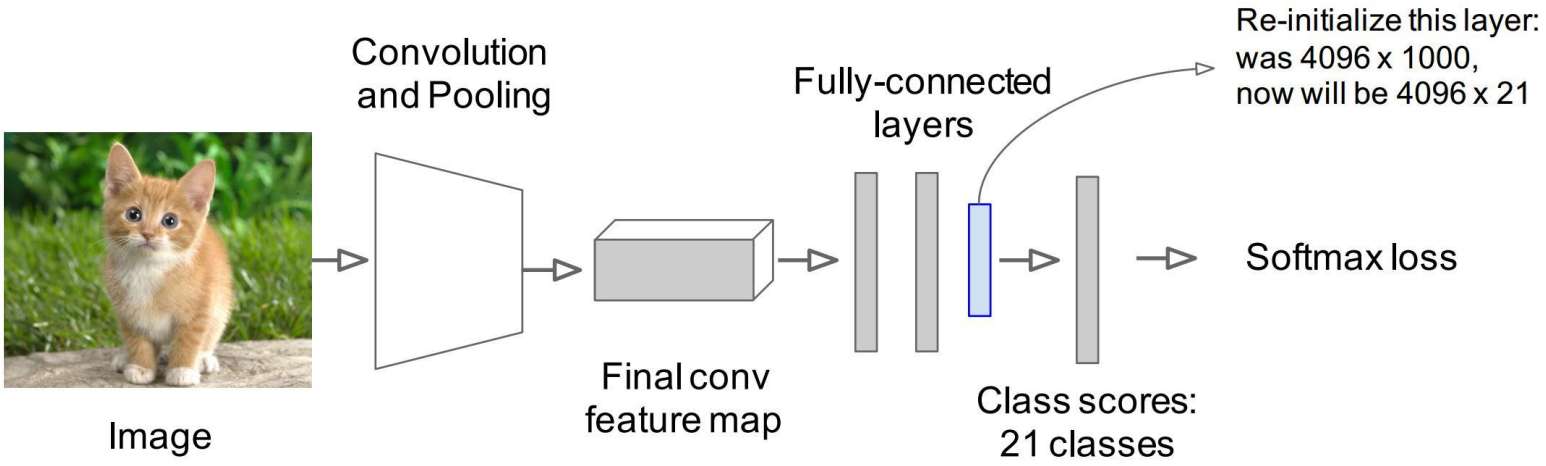


R-CNN——训练流程

R-CNN Training

Step 2: Fine-tune model for detection

- Instead of 1000 ImageNet classes, want 20 object classes + background
- Throw away final fully-connected layer, reinitialize from scratch
- Keep training model using positive / negative regions from detection images

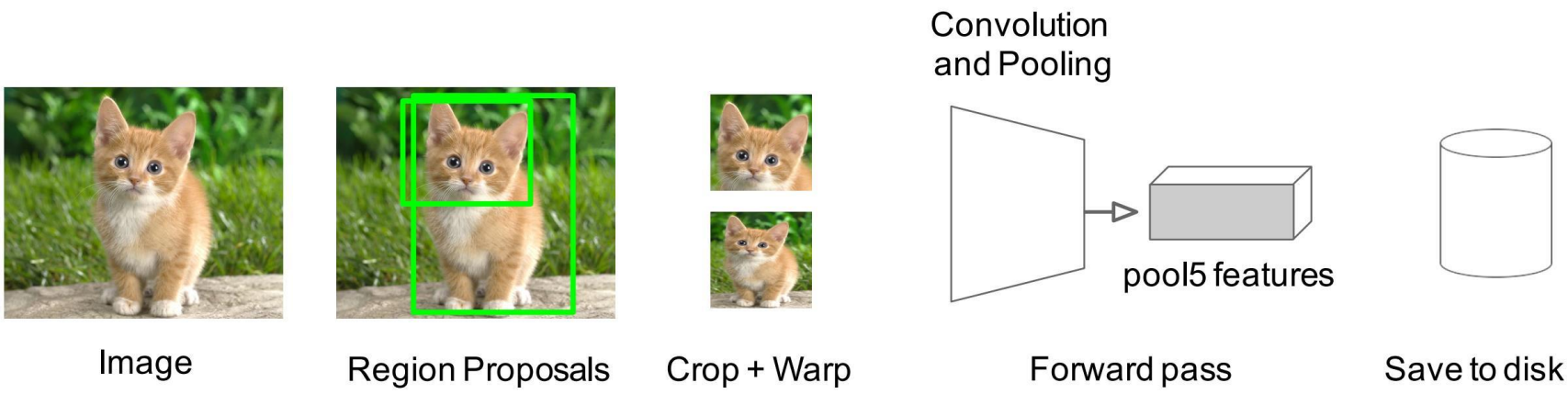


R-CNN——训练流程

R-CNN Training

Step 3: Extract features

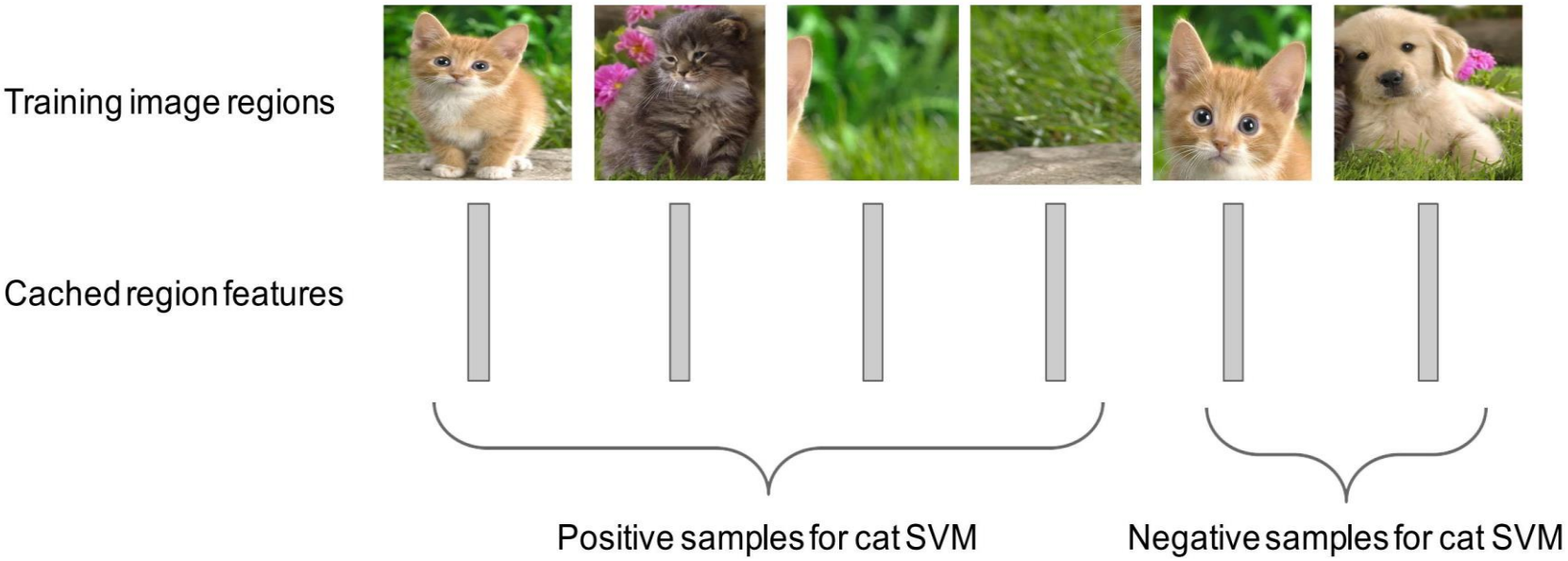
- Extract region proposals for all images
- For each region: warp to CNN input size, run forward through CNN, save pool5 features to disk
- Have a big hard drive: features are ~200GB for PASCAL dataset!



R-CNN——训练流程

R-CNN Training

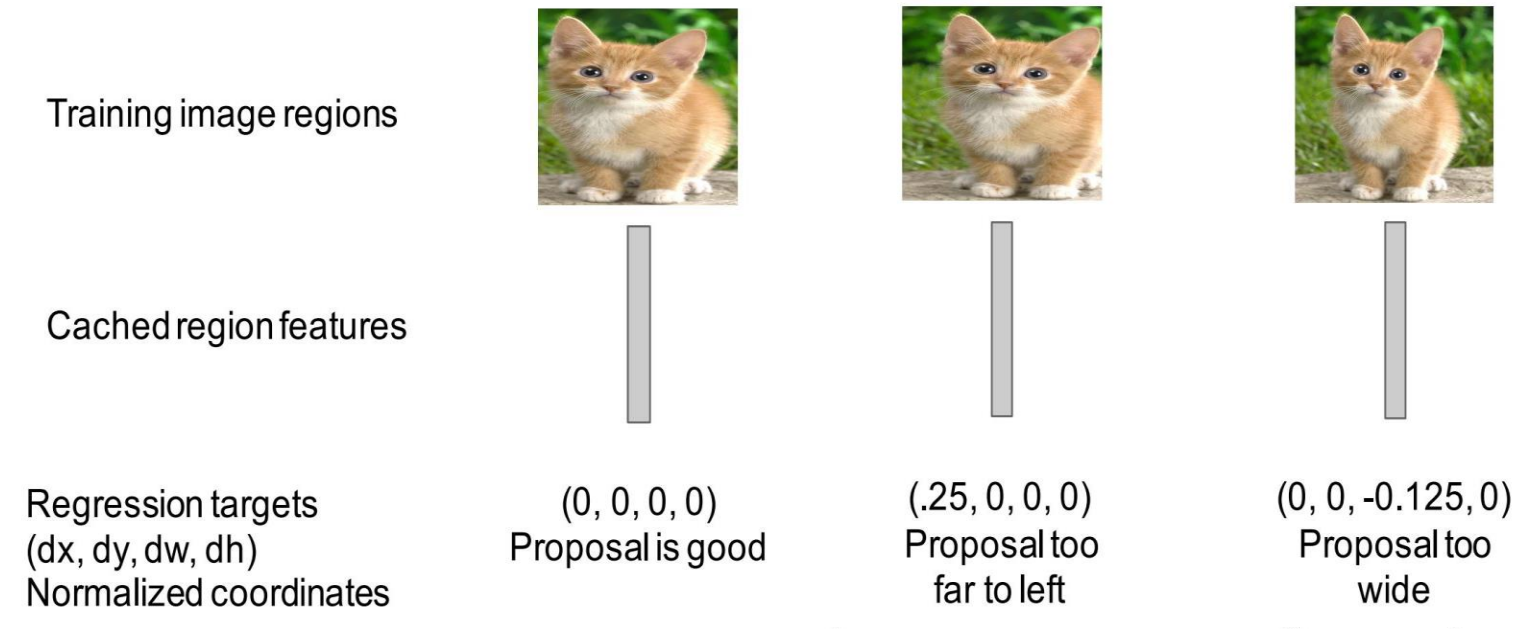
Step 4: Train one binary SVM per class to classify region features



R-CNN——训练流程

R-CNN Training

Step 5 (bbox regression): For each class, train a linear regression model to map from cached features to offsets to GT boxes to make up for "slightly wrong" proposals



R-CNN——缺点

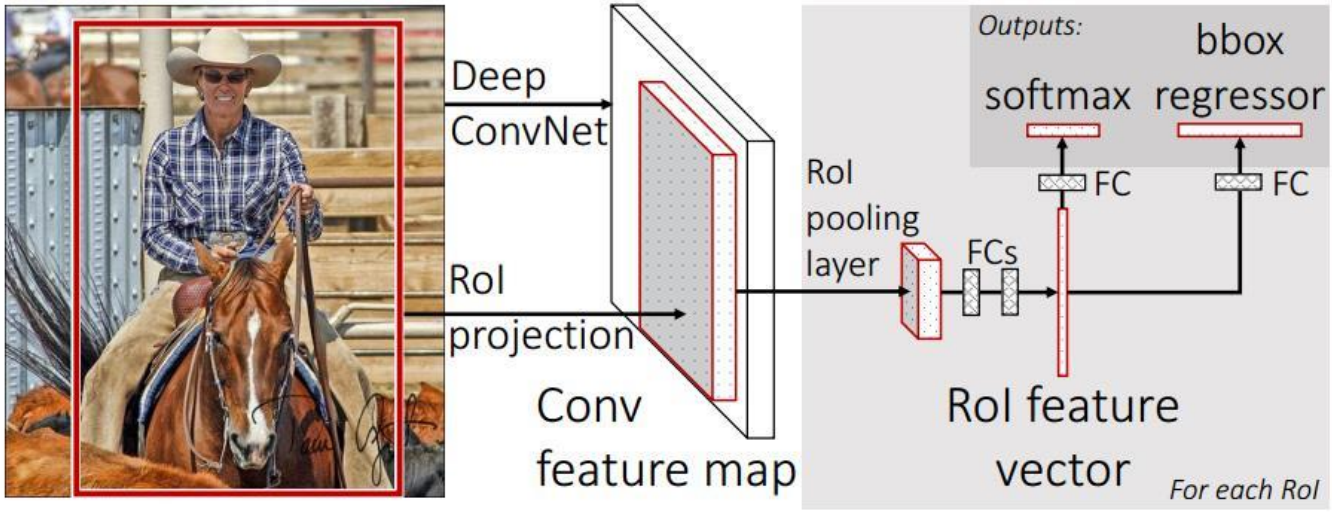
1. Training is multi-stage pipeline

特征提取、分类、位置回归等都是分开训练的。训练过程复杂(训练了87小时)，而且由于需要保留region proposals经特征提取后的特征，将占用大量磁盘空间

2. 速度慢

特征提取效率低，每个region proposal需要单独处理，导致速度慢，GPU上47s/张

Fast R-CNN(2015, CVPR)

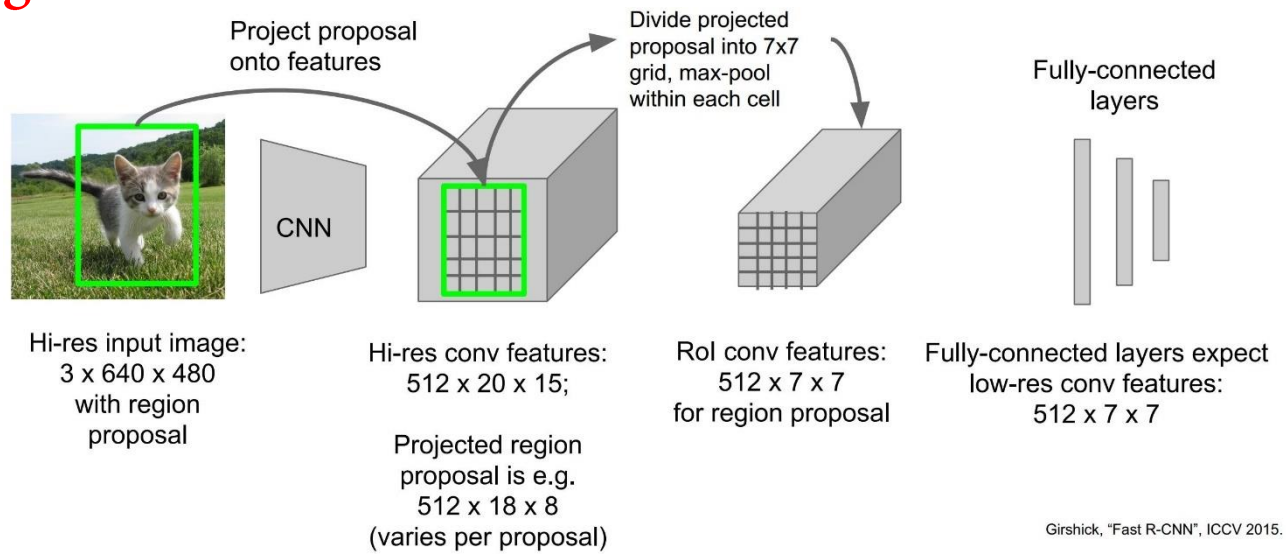


针对R-CNN存在的问题， Fast R-CNN提出了：

1. ROI pooling层：将整张图像输入到网络中就可以计算所有ROI的特征

2. single-stage training：对目标分类和位置回归同时进行优化

ROI pooling：



1. 将整张图像输入进CNN中得到feature map
2. 将原图中ROI的坐标映射到feature map上，得到对应区域
3. 对该区域进行pooling操作：由于各ROI的大小不同，所以将ROI分成固定大小的网格(如7*7)，在网格中再进行pooling操作

图像中的所有ROI共享特征计算的过程，然后通过ROI pooling操作便可得到各自的特征并输入到分类模块中，大大节省了特征提取的时间

Single-stage training：

将类别分类与位置回归联合起来组成multi-task loss进行联合训练，对于一个ROI，其损失函数定义为：

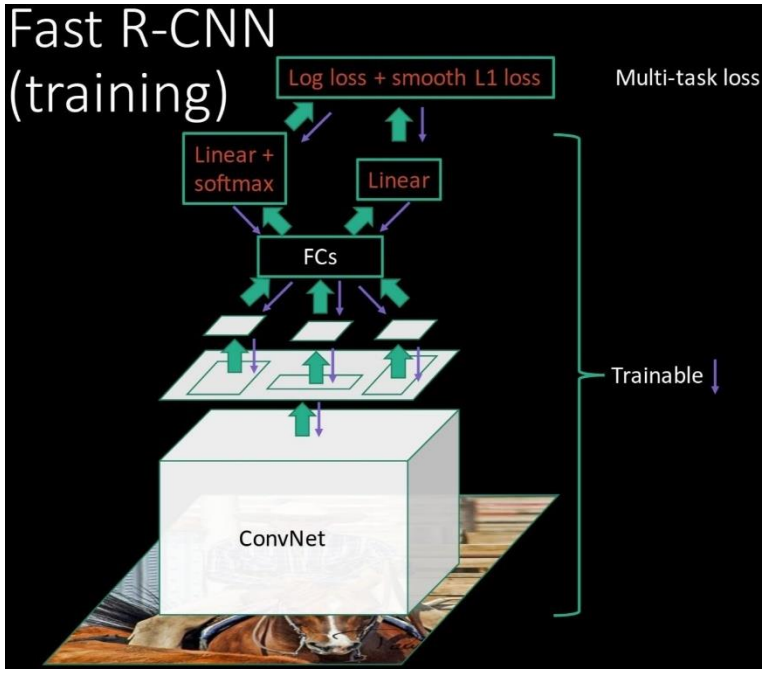
$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$$

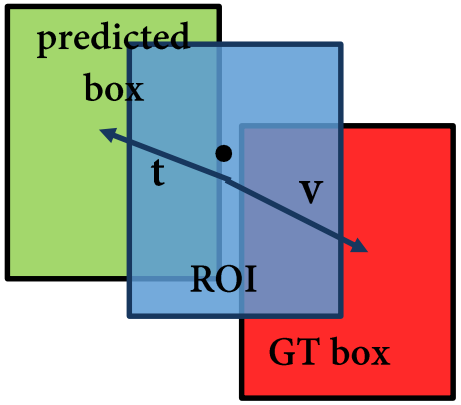
$L_{cls}(p, u)$为分类损失(cross-entropy loss)，p=$(p_0, \ldots, p_K)$，p为softmax的输出，对于每一类的分类得分，u为对应class label

$L_{loc}(t^u, v)$为位置回归损失，t和v分别为为回归后得到的predicted box和ground-truth box与ROI之间的偏移(offsets)，L_{loc}对两者的差异进行惩罚

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

指示函数$[u \geq 1]$，表示只有ROI的标签为前景目标(背景的标签为0)的才计算位置回归损失





Object Detection

Faster!

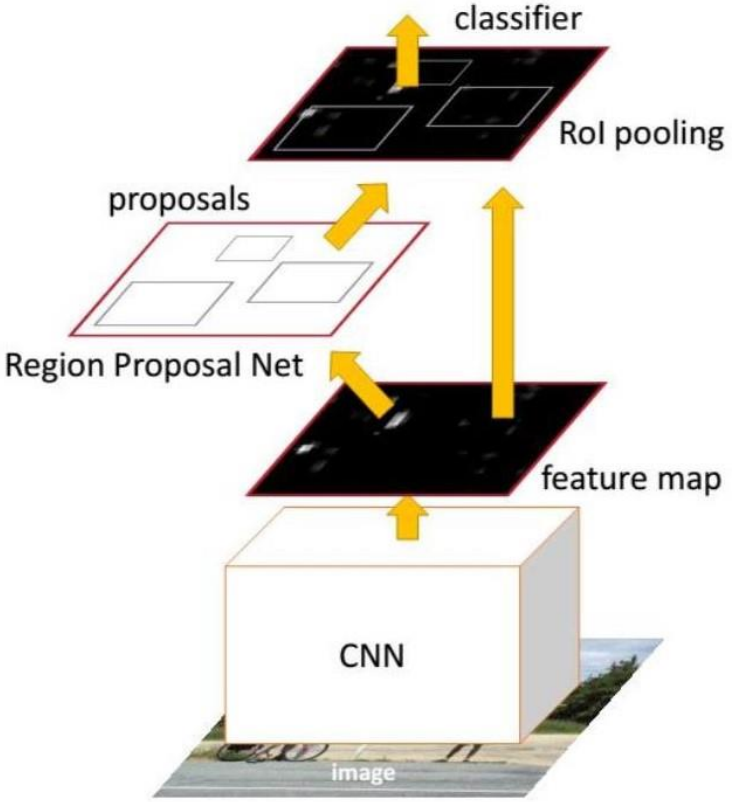
FASTER!

Better!

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
mAP (VOC 2007)	66.0	**66.9**

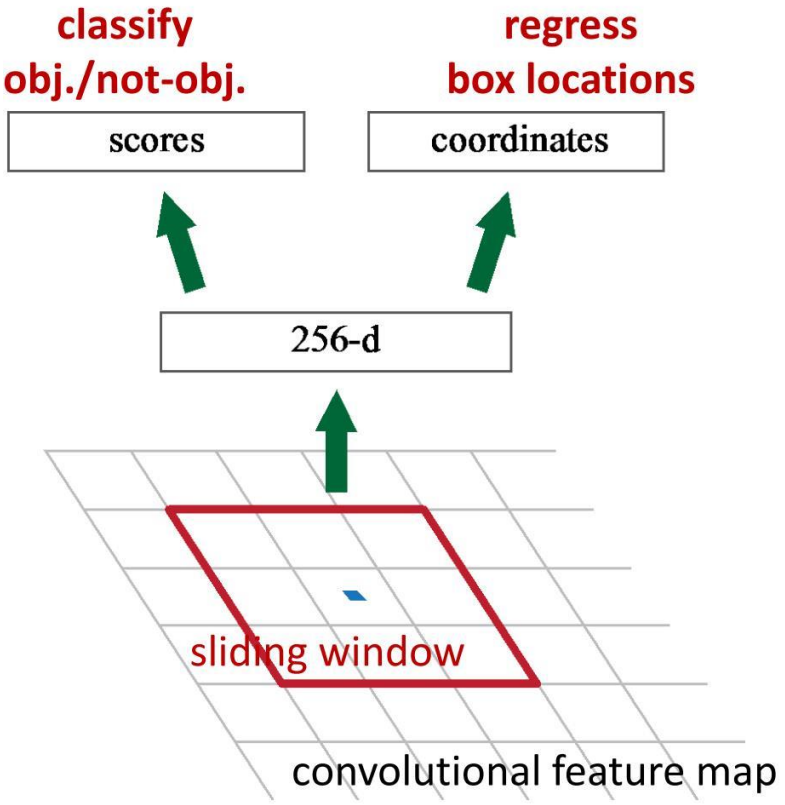
Faster R-CNN(2015, NIPS)

- Faster R-CNN=RPN + Fast R-CNN
- 在feature map后加入RPN网络代替先前速度较慢的Region Proposal方法，RPN和Fast R-CNN共享卷积层参数，并支持端到端的训练
- 速度显著提高，GPU 0.2s/张



什么是RPN网络？

Region Proposal Network，通过deep learning的方式，通过与原有的Fast R-CNN进行端到端的学习，从而提取region proposals

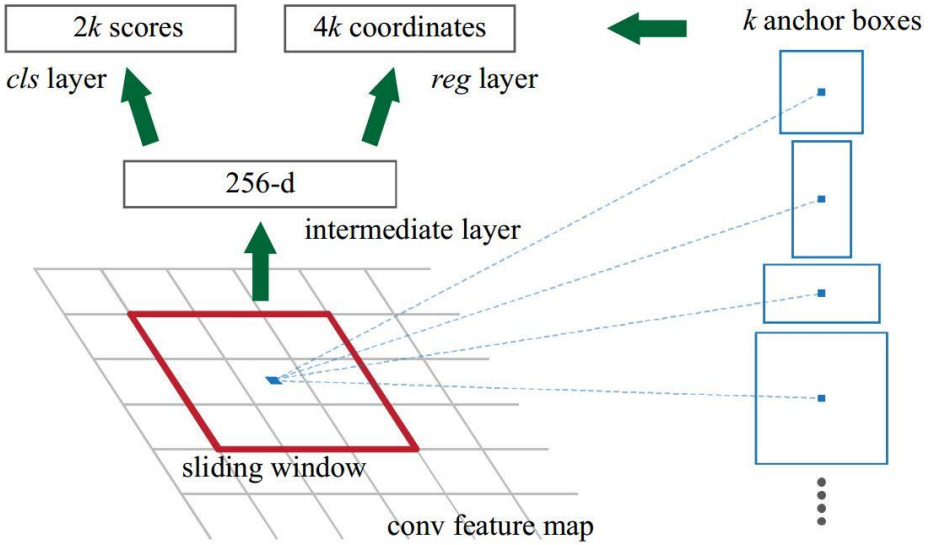


- RPN采用一个子网络遍历整个feature map
- 子网络对每一个位置：
  1. 对其是目标还是非目标进行分类
  2. 进行位置回归

anchors

- 每一位置上会有k个预先定义好的box，称为anchors
- Anchors的大小和长宽比都不同，用来模拟可能出现的不同大小的目标
- 子网络对anchors进行分类与位置回归：

  1. 每个anchors有两类标签，即目标或非目标，所以分类分支的输出的长度为2*k
  2. 对每个anchor进行位置回归，得到相对原始位置的偏移，所以回归分支的输出长度为4*k



总共有20000个anchors，选取score最高的前300个anchors作为Fast R-CNN网络的region proposals

训练：

RPN与Fast R-CNN联合训练，共有四个loss：

- RPN classification：

  二分类，判断目标或非目标

- RPN regression：

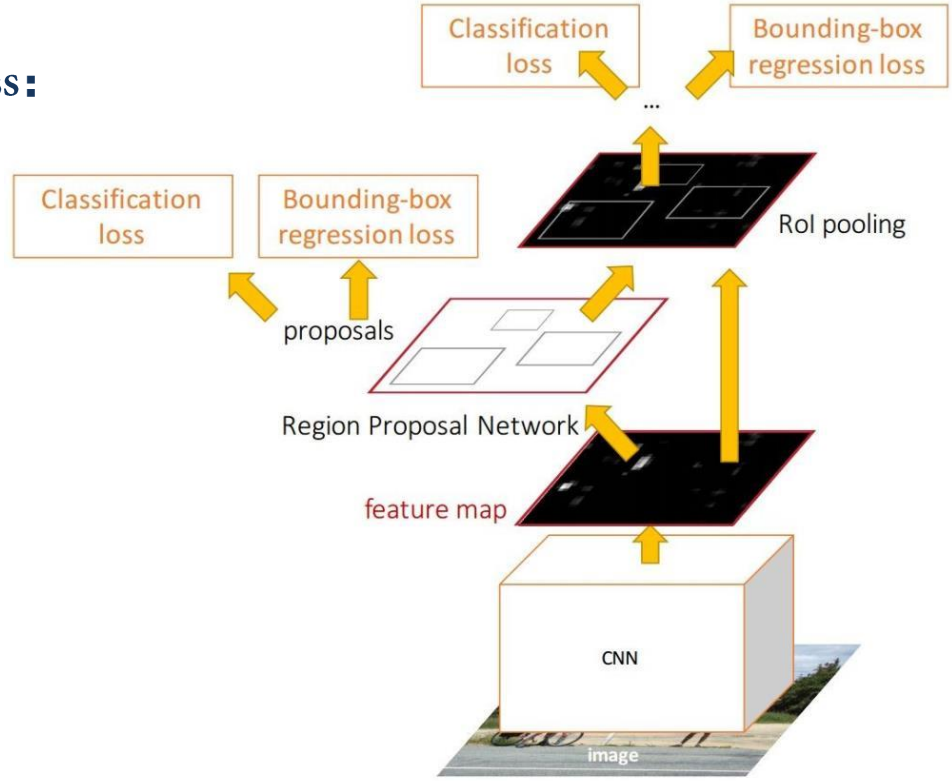
  anchor→proposal

- Fast R-CNN classification：

  多分类，判断目标类别

- Fast R-CNN regression：

  proposal→box



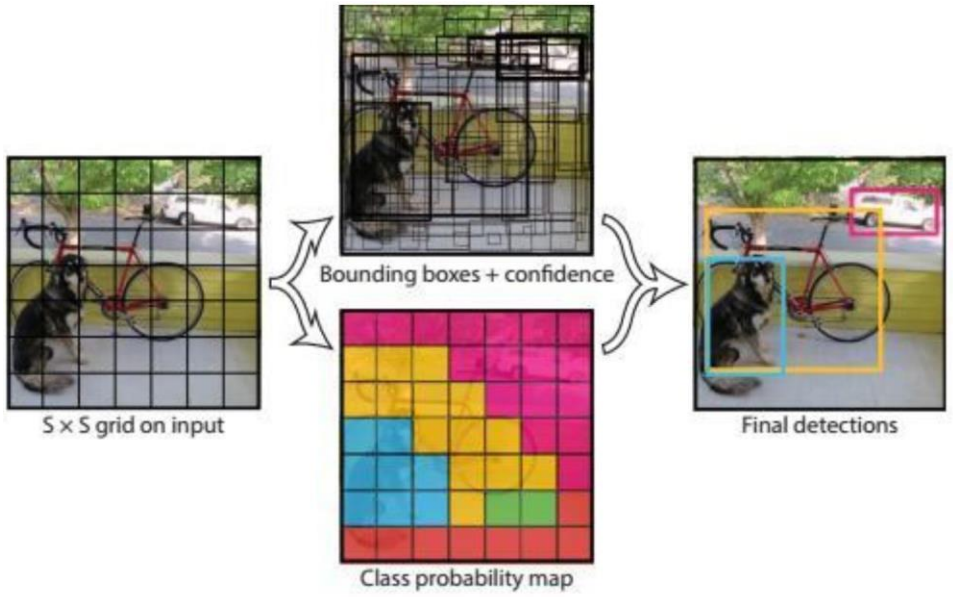
R-CNN、Fast R-CNN与Faster R-CNN的比较：

	Bounding box Selection	Feature Extraction	Classification	Test time per image	mAP(VOC2007)
R-CNN	Region proposal	Deep learning	SVM	50s	66.0%
Fast R-CNN	Region proposal	Deep learning		2s	66.9%
Faster R-CNN	Deep learning	Deep learning		0.2s	69.9%

R-CNN系列可以总结为一类基于proposals的方法，首先要产生proposal，然后在proposal的基础上预测得到目标框(bbox)。我们可以称其为two-stage method

Detection without Proposals：

- YOLO(2016, CVPR)
- SSD(2016, ECCV)



不需要通过RPN产生proposals，直接基于预先定义好的anchor对其进行分类及位置回归得到bbox，即由anchor→bbox。我们可以将此类方法称为one-stage methods

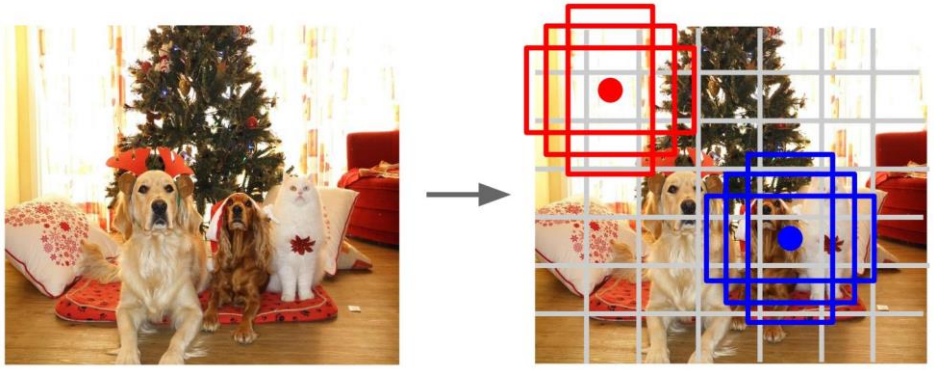
YOLO(2016, CVPR)， You Only Look Once

流程：

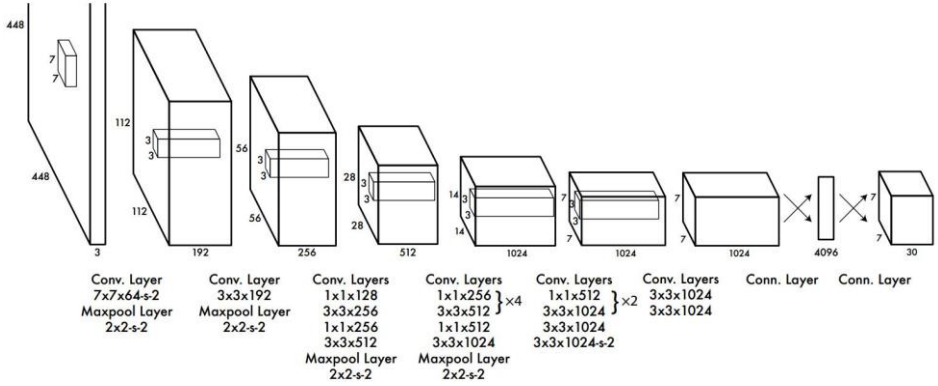
1. 将图像resize成448*448，并划分成7*7的网格

2. 网络前面的卷积部分用于提取图像特征，后接两个全连接层用于预测：
a) 每个网格包含两个anchors(总共98个)，对其位置(x,y,w,h)及其object confidence进行预测
b) 对该网格进行分类(20类)

3. 过滤bbox得到

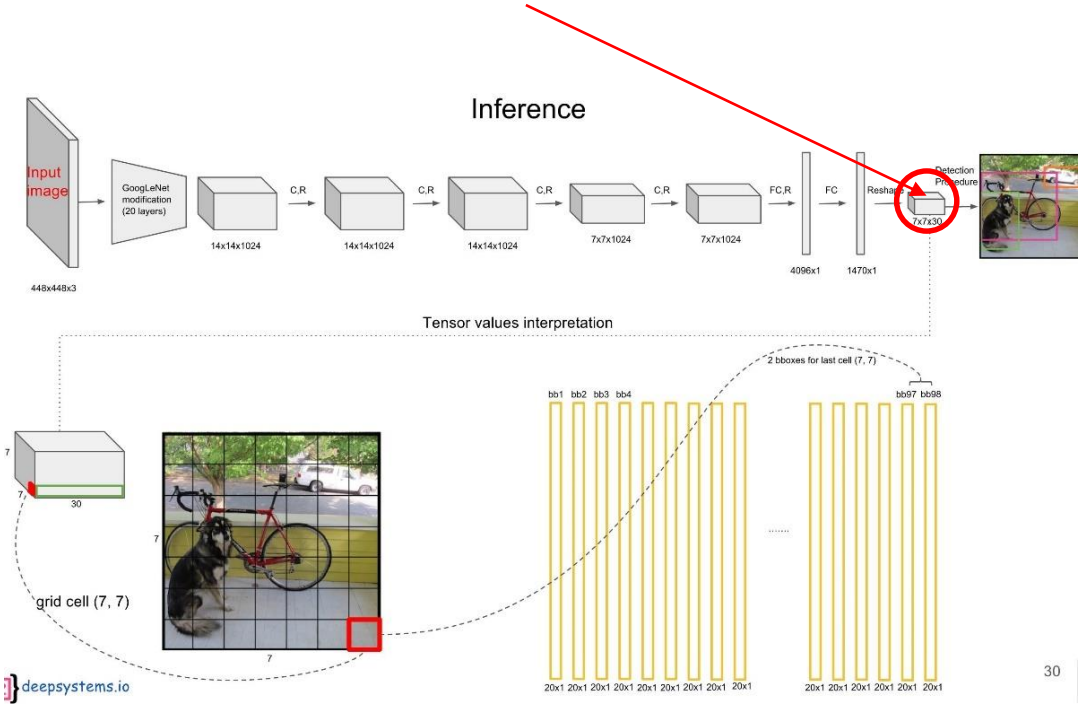


如果目标中心落在哪个网格内，哪个网格就负责检测该物体



YOLO(2016, CVPR)

每个anchors各有5个predictions(4个坐标+object confidence)，然后加上所在网格的20个class probabilities　2*5+20=30



优点：

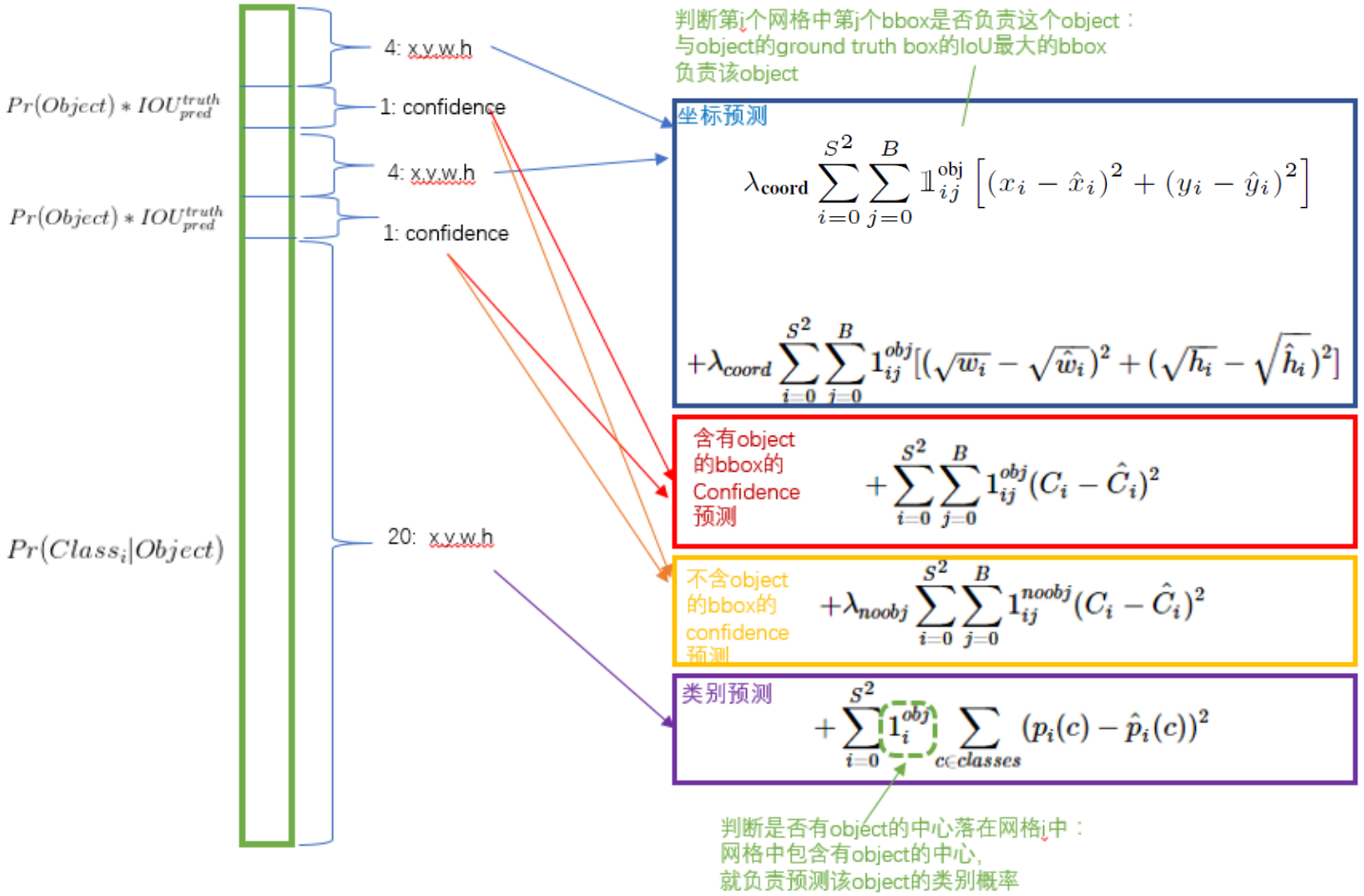
- 速度快 45帧/s

直接对最后一个卷积层生成的feature map通过全连接层进行预测，且anchors的数目大大减少(98 VS 20000)

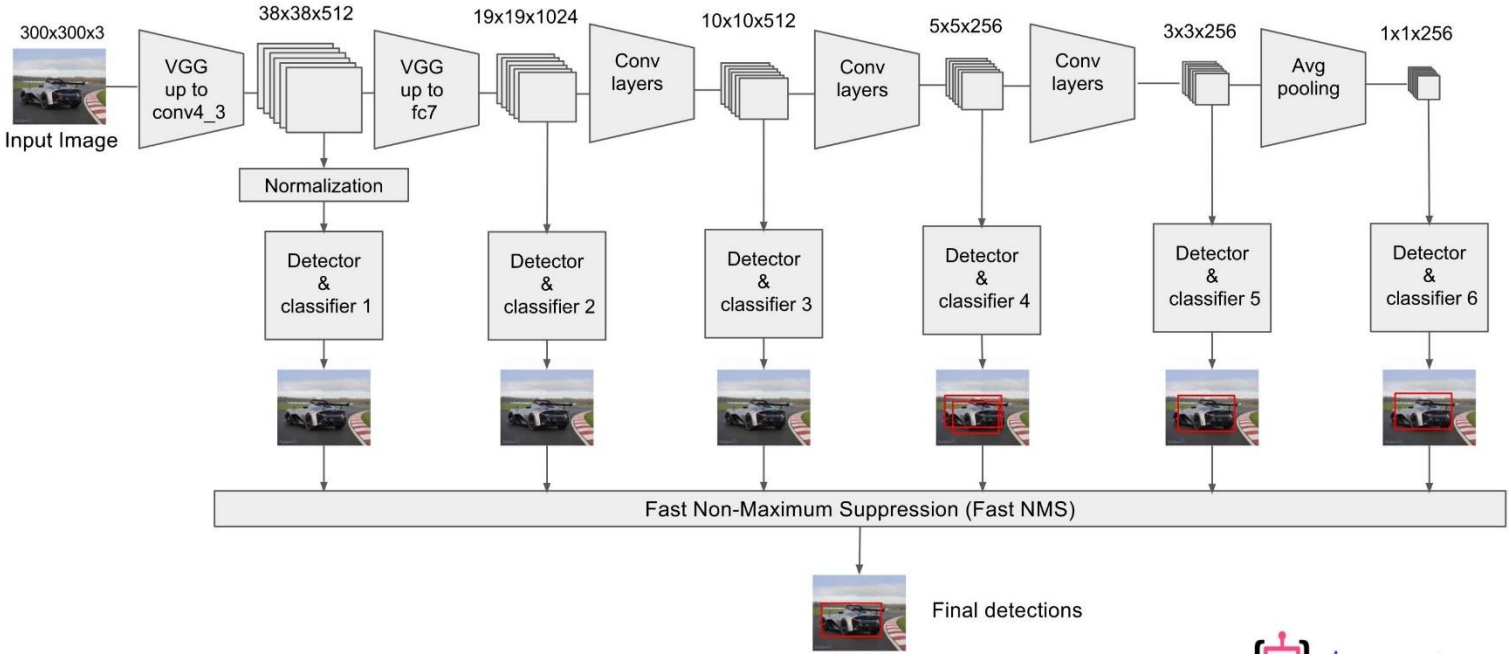
缺点：

- YOLO对相互靠的很近的物体(中心点都落在同一个格子上的情况)，这是因为一个网格中只预测了两个框，并且只属于一类。
- 测试图像中，当同一类物体出现的不常见的长宽比和其他情况时泛化能力偏弱。

损失函数设计：


4: x,y,w,h
1: confidence
4: x,y,w,h
1: confidence
20: x,y,w,h
$Pr(Object) * IOU_{pred}^{truth}$
$Pr(Object) * IOU_{pred}^{truth}$
$Pr(Class_i|Object)$
判断第i个网格中第j个bbox是否负责这个object：
与object的ground truth box的IoU最大的bbox
负责该object
坐标预测
$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$
$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$
含有object的bbox的Confidence预测
$+\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2$
不含object的bbox的confidence预测
$+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$
类别预测
$+\sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$
判断是否有object的中心落在网格i中：
网格中包含有object的中心，
就负责预测该object的类别概率

SSD(2016, ECCV), Single Shot MultiBox Detector

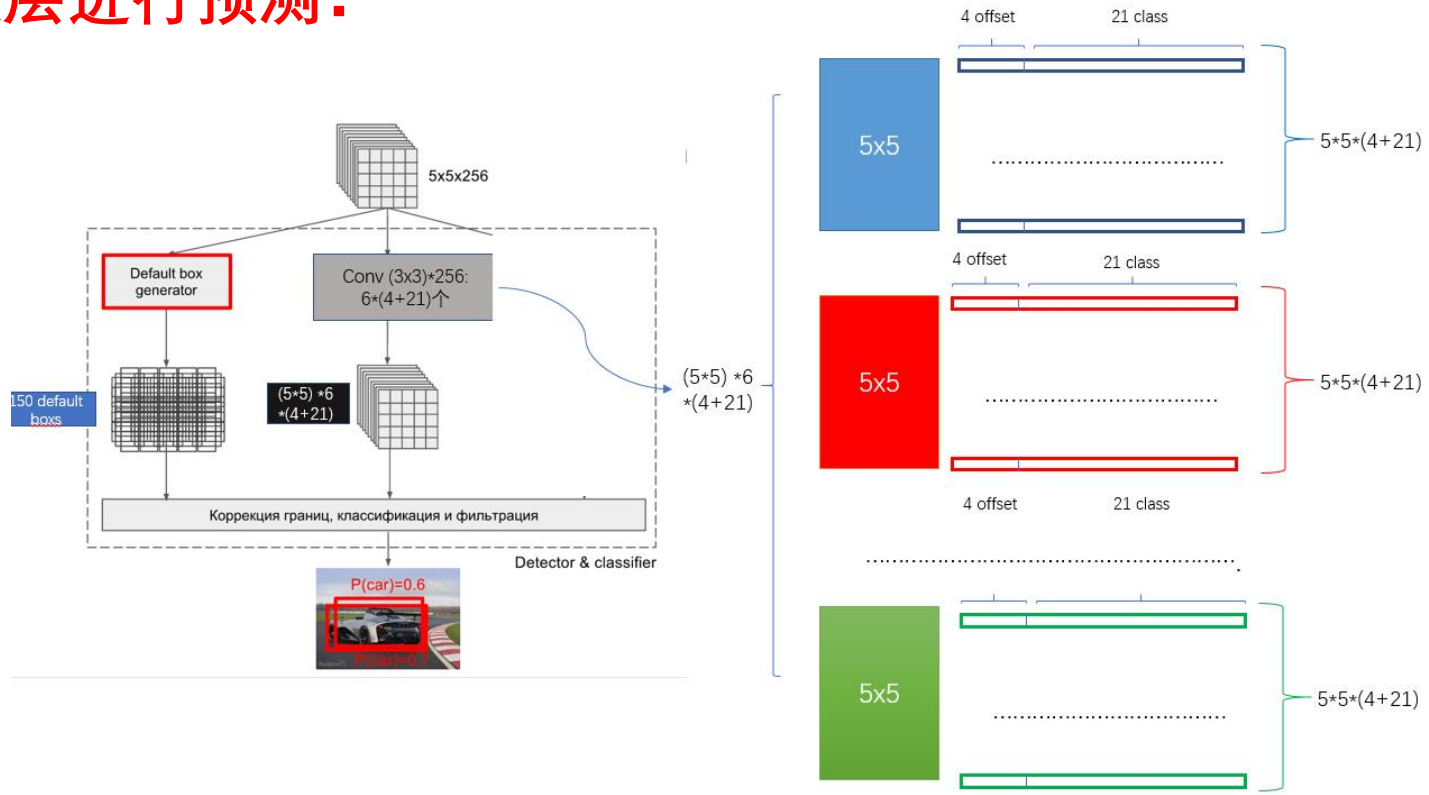


相对YOLO的改进：

- 选择在多尺度feature map产生多个 anchors，这样一来anchors就有不同大小的感受野，有利于检测出不同尺度的物体
- YOLO预测的时候使用全连接层，丢掉了相对位置信息，且计算量大。SSD用卷积层代替全连接层。

在保证性能的同时也实现了快速检测，达到了59帧/s

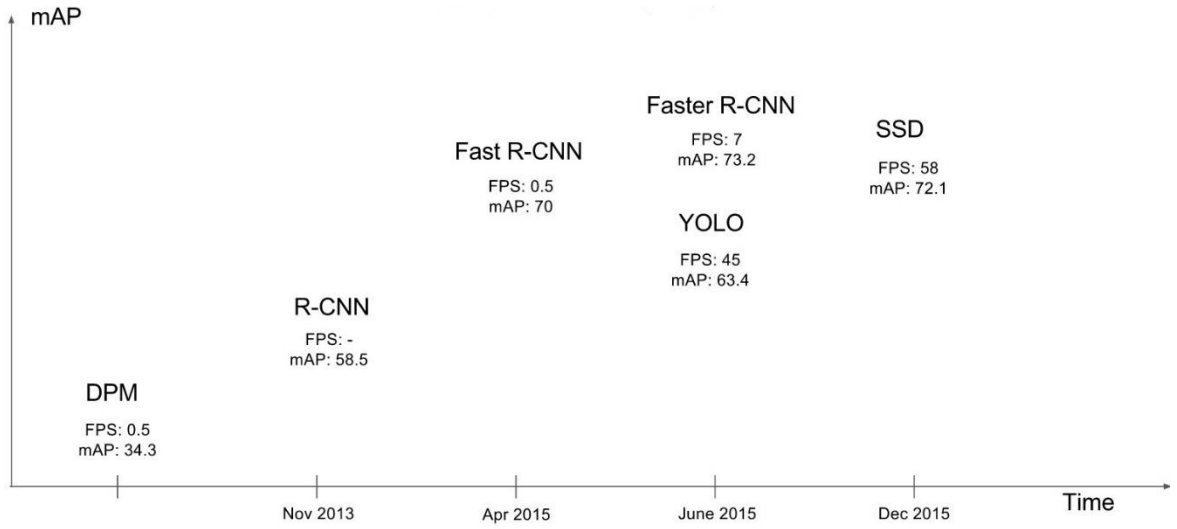
用卷积层进行预测：



这里用卷积核进行预测，以大小为5*5*256的feature map为例，每个网格有6个anchors，所以需要预测的有6*(4+21)，所以卷积核的大小为3*3*150

Object Detection

总结：



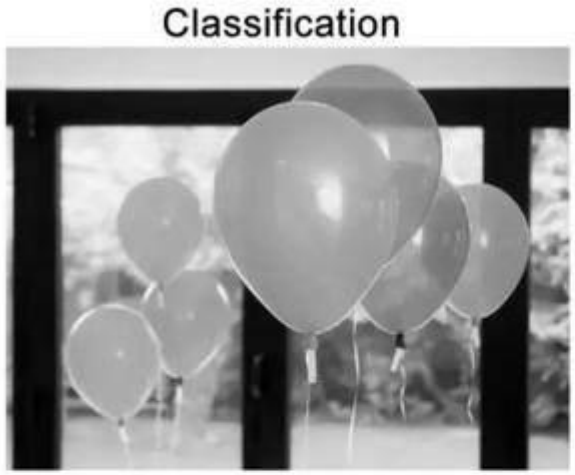
基于CNN的目标检测方法主要分为两类：

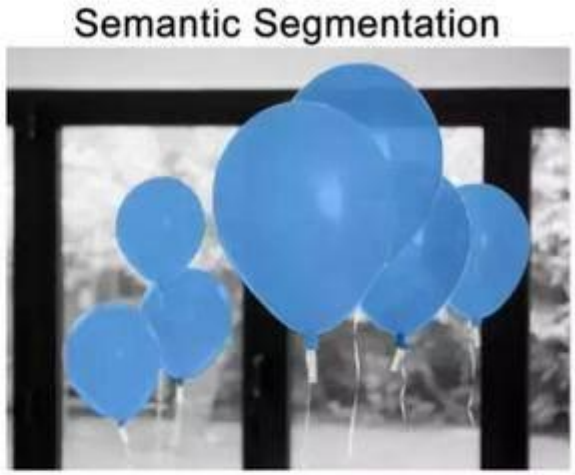
- 以Fast R-CNN、Faster R-CNN为代表的Two-stage method，由anchors→proposals→bbox;
- 以YOLO、SSD为代表的One-stage方法，其由anchor→bbox

由于two-stage方法产生proposals过滤了大量的负样本，并且从anchors到bbox经历了两次的位置回归，所以取得了比较好的性能；而one-stage方法直接从anchor到bbox，虽然在性能上有所欠缺但是可以取得较快的速度。

Instance Segmentation （Mask RCNN）

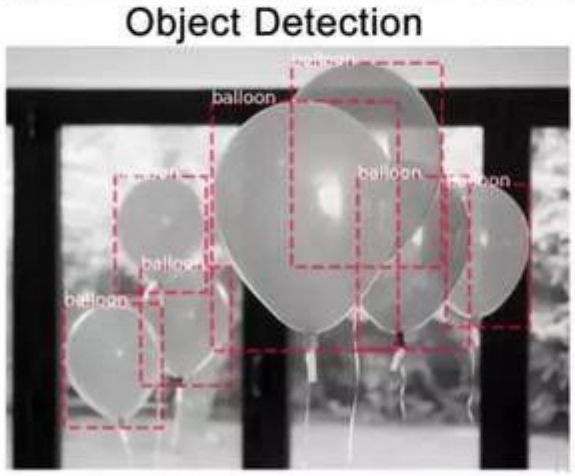
分类

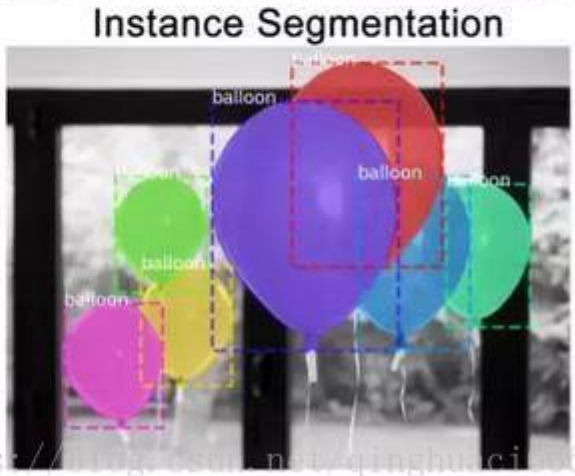




语义分割

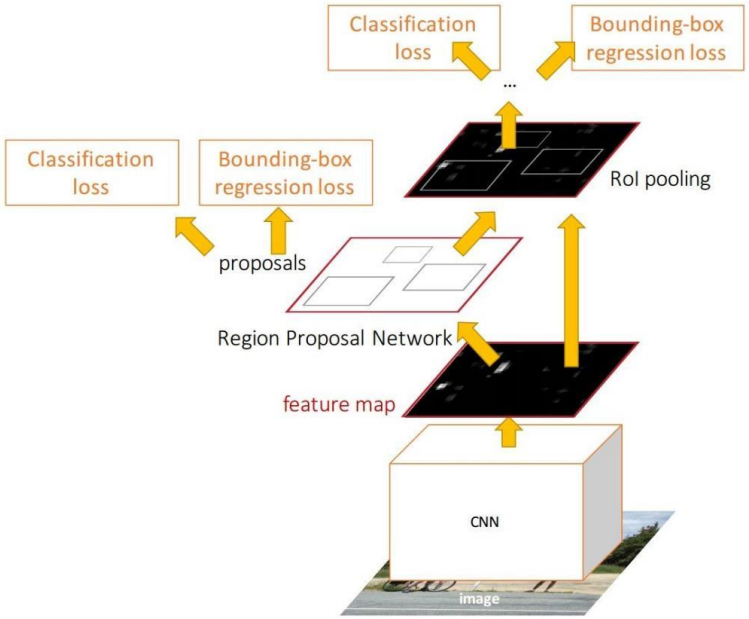
目标检测



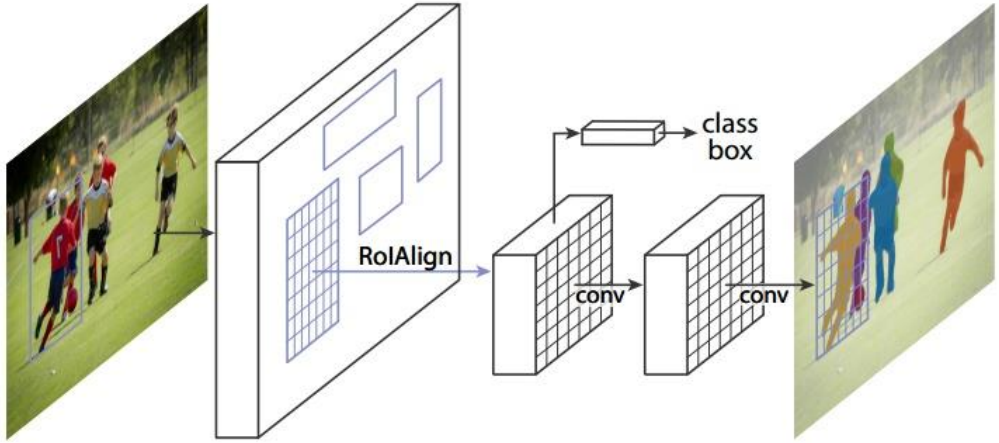


实例分割

Instance Segmentation （Mask RCNN）

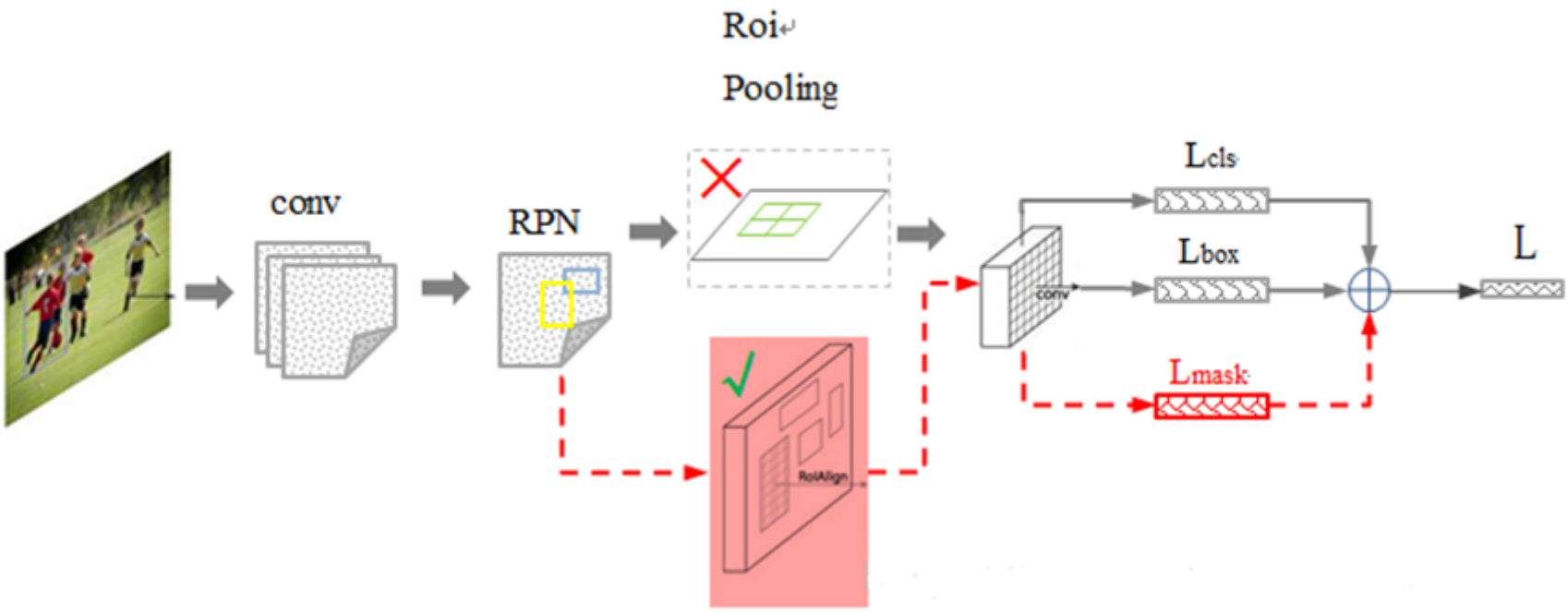


Faster-RCNN
（目标检测）



MASK RCNN
（实例分割）

Instance Segmentation （Mask RCNN）



Faster-RCNN与Mask RCNN结构对比图

黑色部分为 Faster-RCNN结构，**红色部分**为在 Faster-RCNN网络上的改进，流程如下：

1）输入图像；

2）将整张图片输入CNN，进行特征提取；

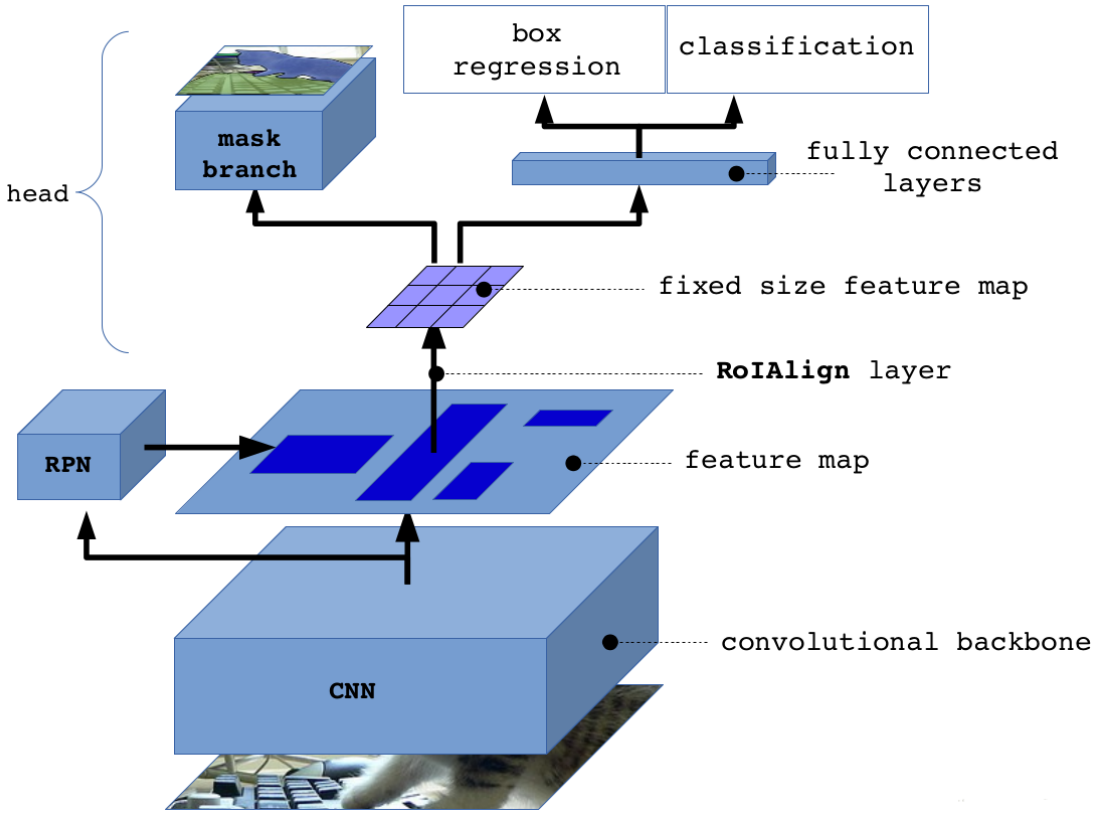
3）用FPN生成建议窗口(proposals)，每张图片生成N个建议窗口；

4）把建议窗口映射到CNN的最后一层卷积feature map上；

5）通过**RoI Align**层使每个RoI生成固定尺寸的feature map；

6）最后利用全连接分类，边框，mask进行回归。

Instance Segmentation （Mask RCNN）



Mask RCNN网络示意图

与faster RCNN的区别：

1）使用ResNet101网络

2）由RPN网络转变成FPN网络

3）将 Roi Pooling 层替换成了RoiAlign层；

4）添加并列的 Mask 分支；

Region Proposal Network

→

Feature Pyramid Networks

ROI Pooling 的局限性

ROI Pooling 的作用是根据预选框的位置坐标在特征图中将相应区域池化为固定尺寸的特征图，以便进行后续的分类和包围框回归操作。步骤为：

1. 将proposal映射到feature map对应位置
2. 将映射后的区域划分为相同大小的sections
3. 对每个sections进行max pooling/avg pooling操作

问题：

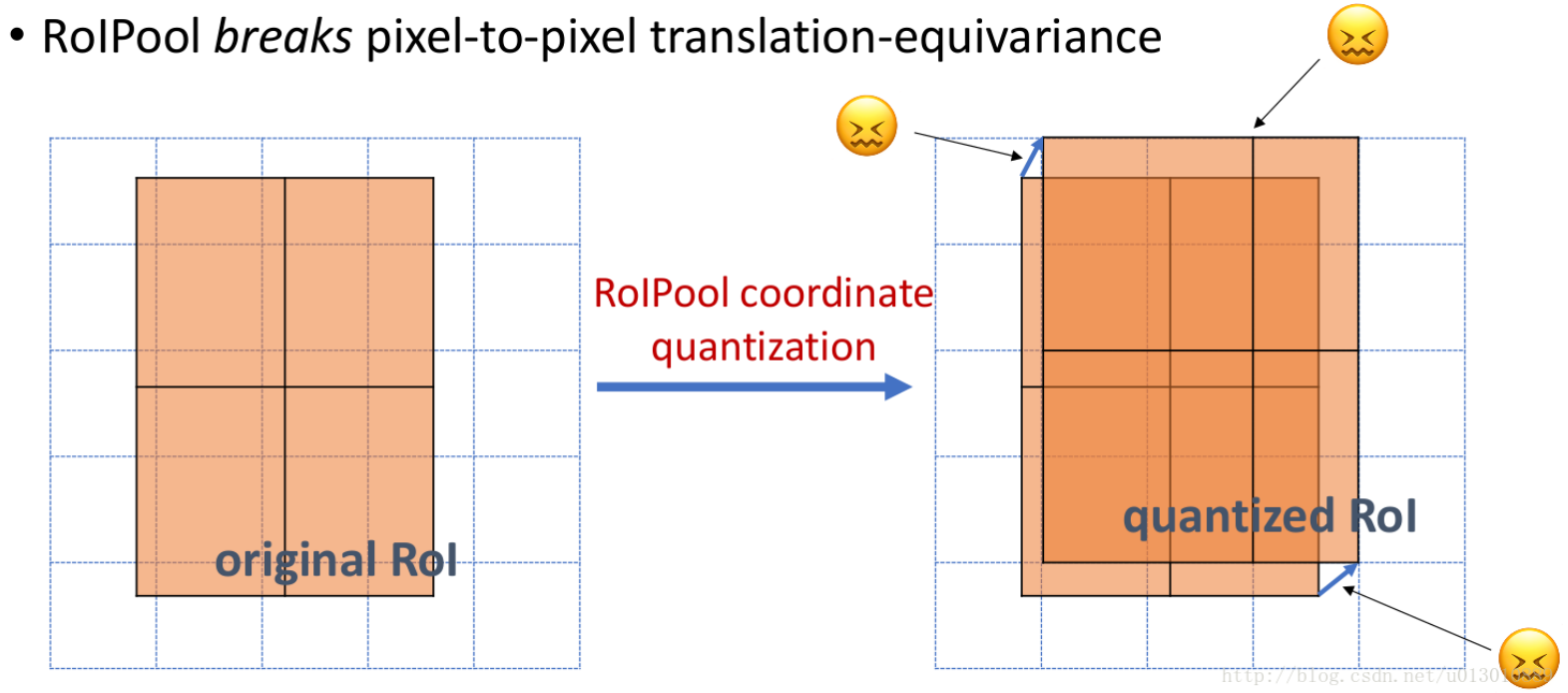
由于预选框的位置通常是由模型回归得到的，一般来讲是浮点数，而池化后的特征图要求尺寸固定。故ROI Pooling这一操作存在两次量化的过程。即：

1. 将候选框边界量化为整数点坐标值
2. 将量化后的边界区域平均分割成 $k \times k$ 个单元

经过上述两次量化，此时的候选框已经和最开始回归出来的位置有一定的偏差，这个偏差会影响检测或者分割的准确度

ROI Pooling 的局限性

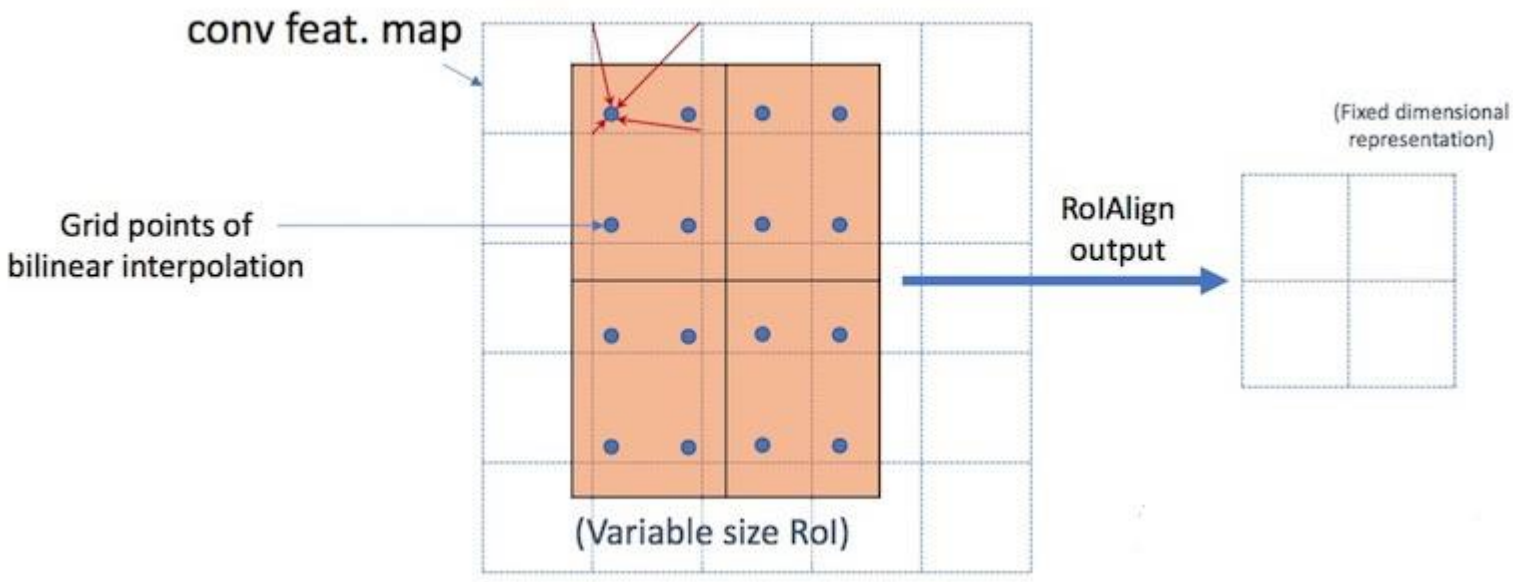
ROI Pooling 的上述问题被称为**不匹配问题（misalignment）**，也即segment是pixel-level的任务，但是ROI pooling有2次量化操作导致了没有对齐 。



ROI Align 的主要思想

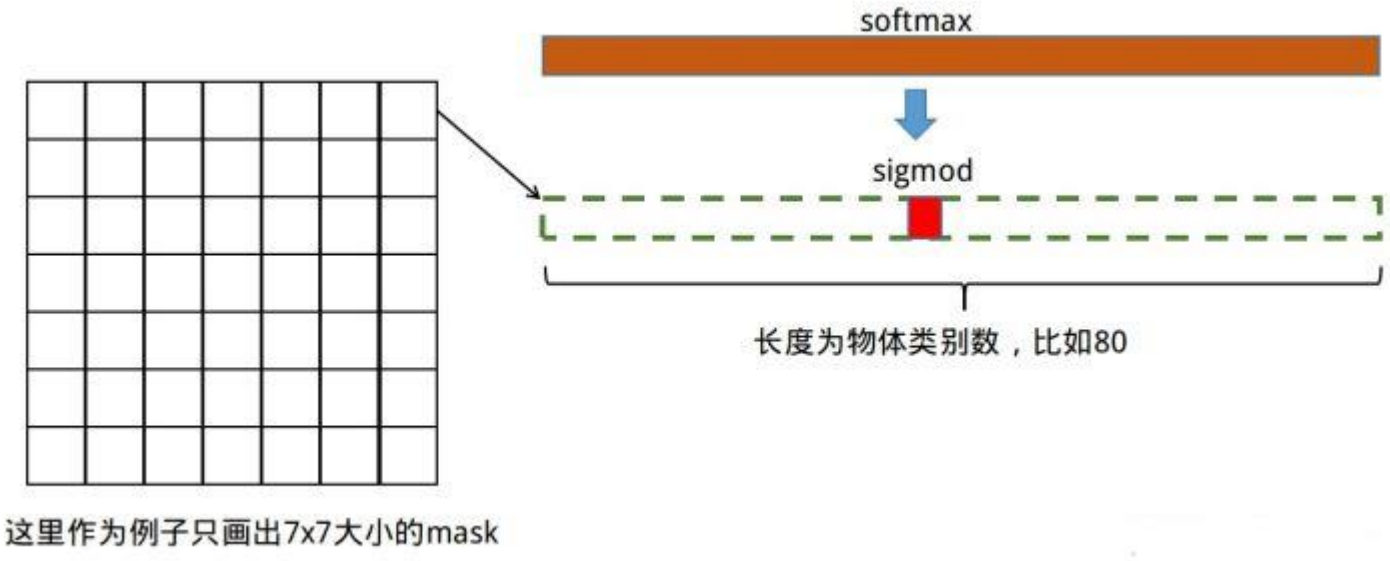
为了解决ROI Pooling的上述缺点，Mask RCNN提出了ROI Align策略。

ROI Align的主要思路：取消量化操作，使用双线性内插的方法获得坐标为浮点数的像素点上的图像数值,从而将整个特征聚集过程转化为一个连续的操作。



Mask的预测

Mask RCNN损失函数： $L = L_{cls} + L_{box} + L_{mask}$



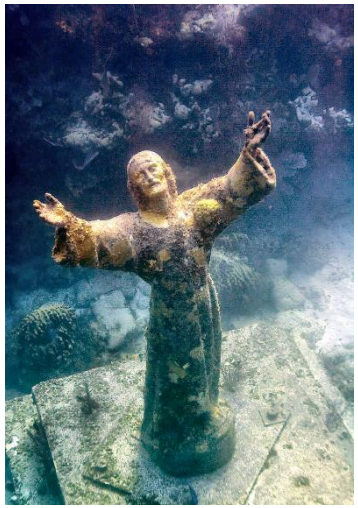
若共有K个类别，mask分割分支的输出维度是$K*M*N$， $M*N$ 为图像尺寸，也即每个像素都会输出K个二值Mask（每个类别使用sigmoid输出）

Underwater Image Enhancement

- 水下环境复杂多变，由于受波长依赖的吸收、散射、湍流干扰和悬浮泥沙等因素的影响，水下图像的退化是不可避免的。



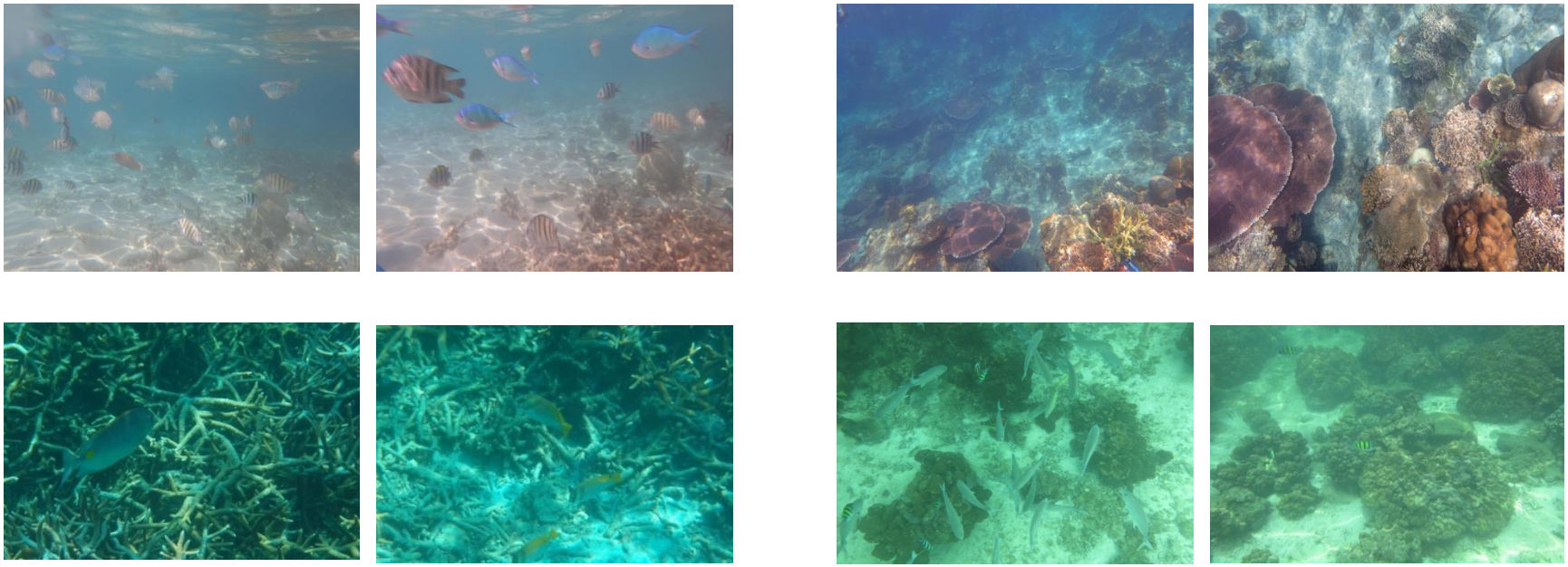


Underwater Image Co-Enhancement

（一）问题背景

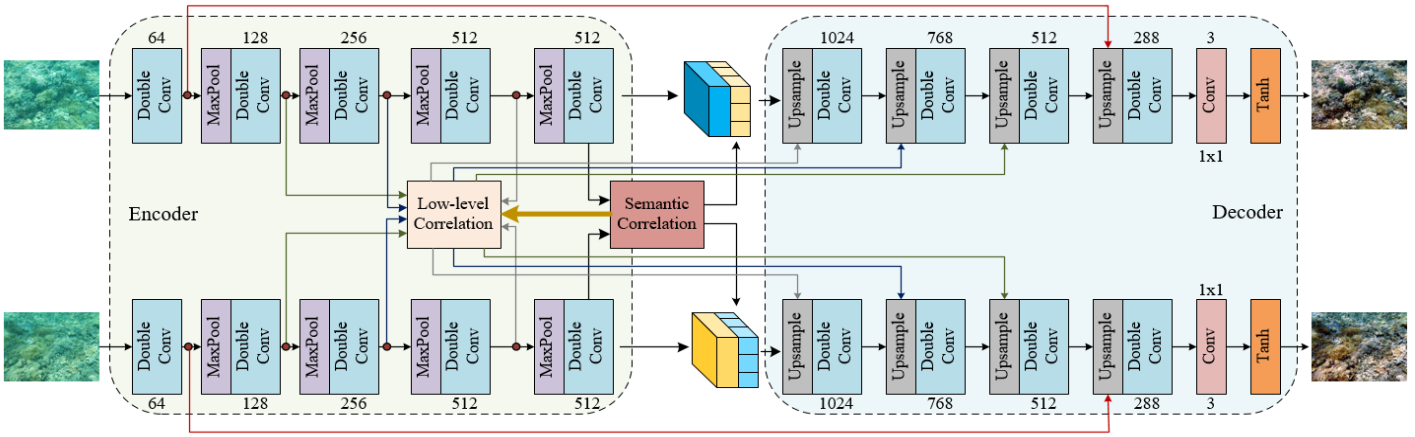
- 几乎所有现有的水下图像增强方法都是针对单幅独立图像的；
- 单幅图像所能提供的有效信息是相当有限的
- 在同一水下场景拍摄的图像集合往往具有类似的成像条件，包含大量的共享和互补信息，可为单一图像的增强提供额外信息，提高增强性能。



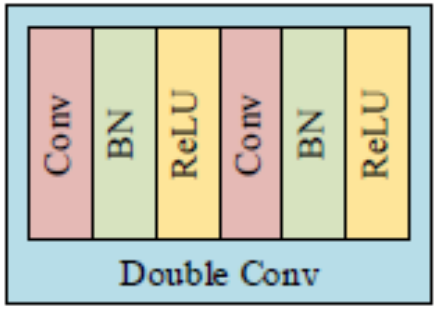
- 关联场景水下图像示例

Underwater Image Co-Enhancement

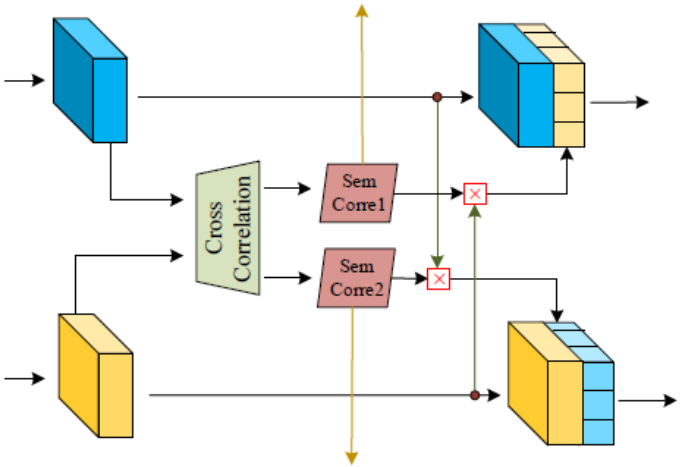
（二）水下图像协同增强网络：UICoE-Net



■ UICoE-Net网络结构



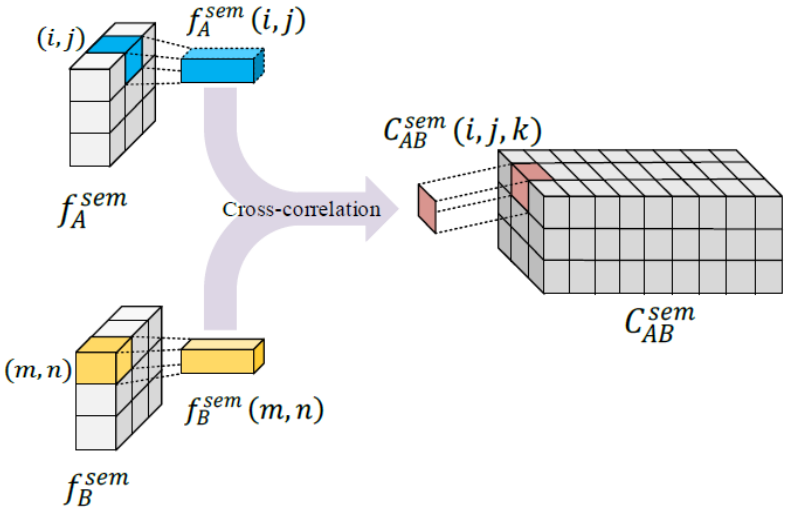
■ 孪生编码器结构单元（Conv-Units）



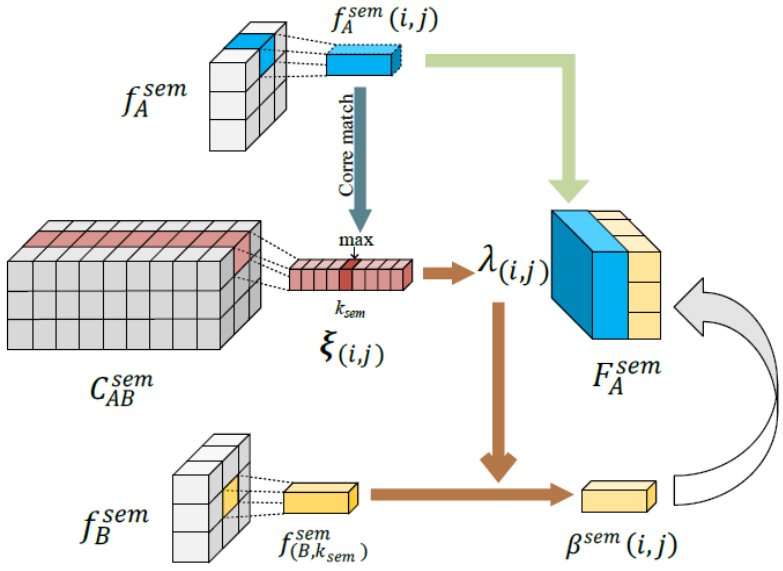
■ 语义关联特征匹配单元结构

Underwater Image Co-Enhancement

（二）水下图像协同增强网络：UICoE-Net



■ 关联系数谱计算示意图



■ 协同特征重组与拼接示意图

$$C_{AB}^{sem}(i,j,k) = \langle f_A^{sem}(i,j), f_B^{sem}(m,n) \rangle,$$

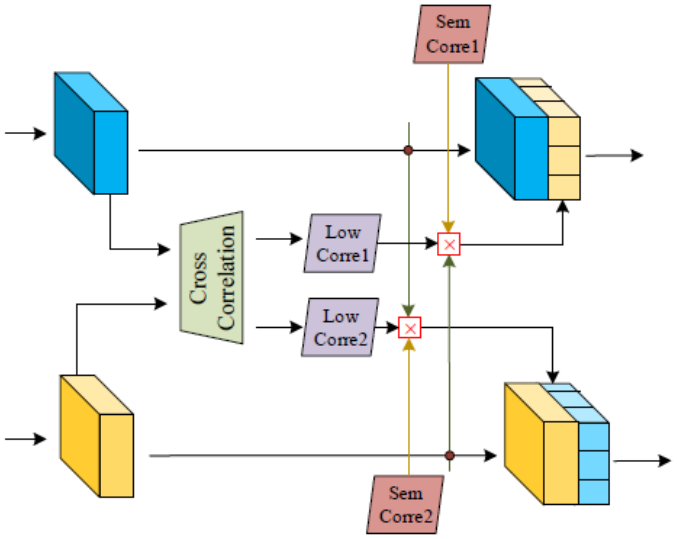
$$\beta_{(i,j)}^{sem} = \lambda_{(i,j)} f_{(B,k_{sem})}^{sem},$$

运算$\langle a,b \rangle$表示向量a和b的点乘积，(i,j)和(m,n)分别是特征图f_A^{sem}和f_B^{sem}中的位置索引，

$k = (m-1)\omega_{sem} + n$是相关矩阵$C_{AB}^{sem}$的通道索引，并与位置$(m,n)$保持一一对应关系，$\omega_{sem}$是特征图的宽。

Underwater Image Co-Enhancement

（二）水下图像协同增强网络：UICoE-Net



$$C_{AB}^{low}(i,j,k) = \langle f_A^{sem}(\hat{i},\hat{j}), f_B^{sem}(\hat{m},\hat{n})\rangle \times \langle f_A^{low}(i,j), f_B^{low}(m,n)\rangle,$$

低层关联特征的匹配需借助高层语义特征的引导，同时可提高计算效率

$$\beta_{(i,j)}^{low} = \mu_{(i,j)} f_{(B,k_{low})}^{low},$$

■ 低层关联特征匹配单元结构

（三）损失函数

■ l_2-loss：预测增强图与参考增强图间的差异。

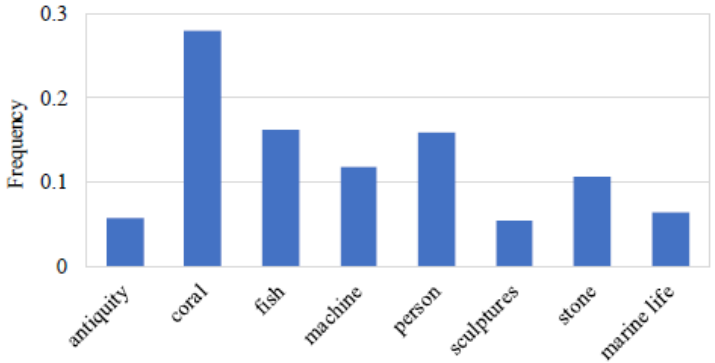
$$\mathcal{L}_{AB} = \frac{1}{N_A}\sum_{i}^{N_A}\left(E_A(i) - R_A(i)\right)^2 + \frac{1}{N_B}\sum_{i}^{N_B}\left(E_B(i) - R_B(i)\right)^2$$

其中E_A和E_B分别是I_A和I_B用UICoE-Net生成的增强图像，R_A和R_B适用于训练的参考增强图，N_A和N_B是I_A和I_B中包含的像素点个数。

（四）实验验证与分析

➢ 实验数据集

（1）分类的水下增强数据集UIEB



■ 水下增强数据集UIEB中包含场景/目标的统计分布

（2）水下协同增强数据集UICoD （新建）

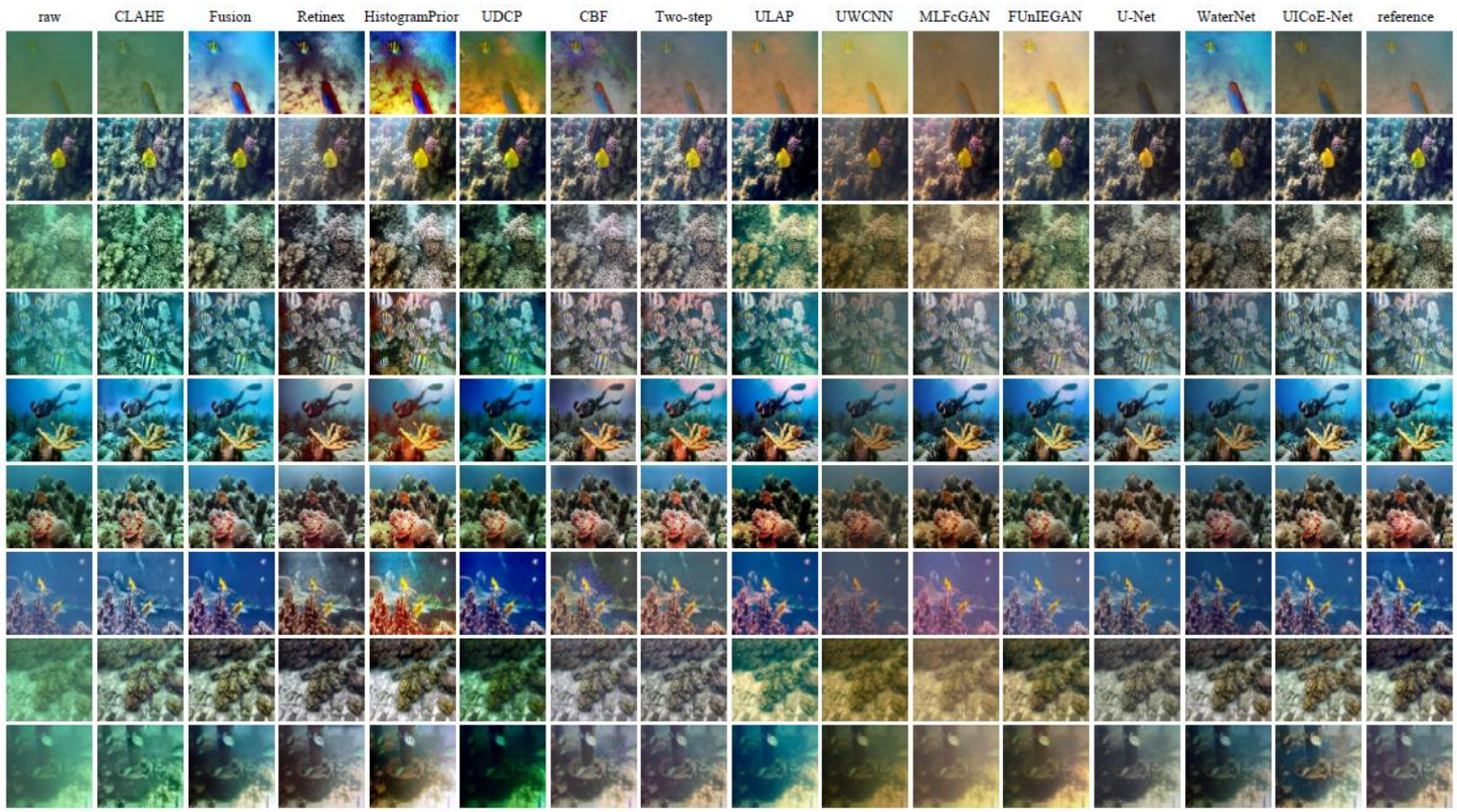
■ 水下协同增强数据集UICoD中的图像及参考增强图像示例

Underwater Image Co-Enhancement

（四）实验验证与分析

本方法结果

➢ **UIEB实验结果**



■ 本文方法与对比方法在部分UIEB数据集图像上的增强结果对比示例

Underwater Image Co-Enhancement

（四）实验验证与分析

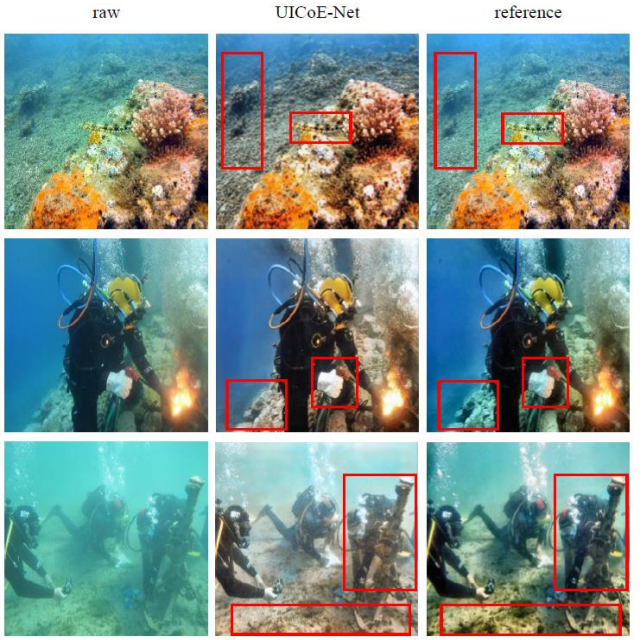
➢ UIEB实验结果

■ 在分类UIEB数据集上与代表性水下图像增强方法的对比结果

	Method	PSNR (dB)↑	MSE(×10^3) ↓	SSIM ↑
传统方法	CLAHE	16.8448	1.5619	0.7957
	Fusion	20.1696	0.8541	0.8649
	Retinex	17.4437	1.4200	0.7999
	Histogram Prior	16.9184	1.5922	0.7910
	UDCP	13.7478	3.3368	0.6521
	CBF	17.9034	1.2740	0.8374
	Two-Step	20.5817	1.0373	0.8667
	ULAP	16.8836	1.8617	0.7584
基于GAN网络	UWCNN	14.1921	2.7882	0.6801
	MLFcGAN	16.5759	1.7473	0.6407
	FUnIE-GAN	18.8872	1.0637	0.7945
带参考CNN学习	U-Net	20.2393	0.7607	0.8578
	WaterNet	19.5732	0.9744	0.8532
	UICoE-Net	**21.7456**	**0.4711**	**0.8944**

本方法结果

参考增强图



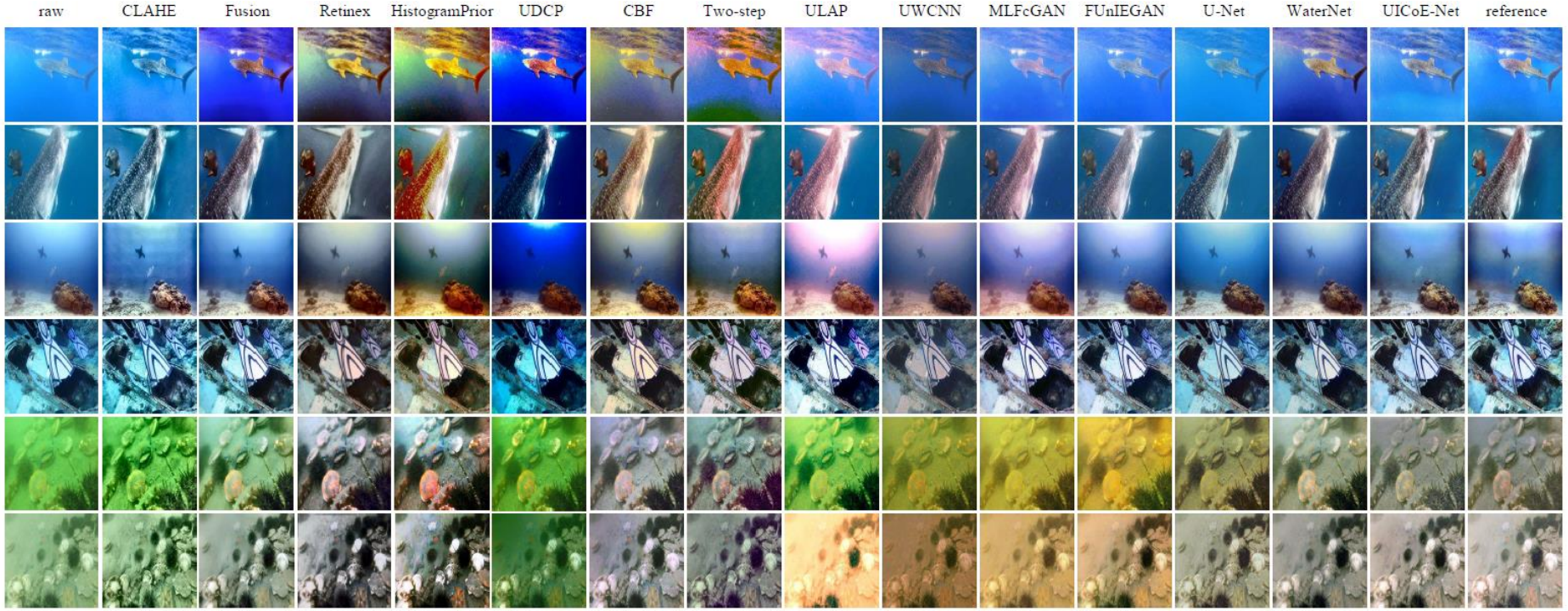
■ UICoE-Net的部分增强结果优于数据集给定的参考增强图

Underwater Image Co-Enhancement

（四）实验验证与分析

本方法结果

➢ **UICoD实验结果**



■ 本文方法与对比方法在部分UICoD数据集图像上的增强结果对比示例

Underwater Image Co-Enhancement

（四）实验验证与分析

➢ UICoD实验结果

■ 在分类UICoD数据集上与代表性水下图像增强方法的对比结果

	Method	PSNR (dB)↑	MSE($\times 10^3$) ↓	SSIM ↑
传统方法	CLAHE	0.7515	16.7253	1.6143
	Fusion	0.7915	18.7363	1.3235
	Retinex	0.7143	15.5938	2.2678
	Histogram Prior	0.7231	15.6859	2.2629
	UDCP	0.5740	12.3389	4.4362
	CBF	0.7862	17.1231	1.8320
	Two-Step	0.7953	17.8514	1.4649
	ULAP	0.7873	16.5439	1.7492
基于GAN网络	UWCNN	0.6674	14.2166	2.7788
	MLFcGAN	0.7066	16.7418	1.5977
	FUnIE-GAN	0.7702	17.8983	1.4846
带参考CNN学习	U-Net	0.8379	20.6821	0.7779
	WaterNet	0.7742	19.3246	0.9747
	UICoE-Net	**0.9124**	**23.1643**	**0.3756**

■ 总体上，相比在UIEB数据集上的性能，增强算法在UICoD数据集上的性能大多略有下降，这说明**UICoD是一个相对更具挑战性的数据集**。

■ 然而，本文提出的UICoE-Net方法的性能不但没有下降，反而略有上升，在一定程度上证明了协同增强策略对于**采集自相似场景的水下图像**具有**更优的增强性能**。

Underwater Image Co-Enhancement

（四）实验验证与分析

➢ 消融实验

■ 消融实验量化对比结果

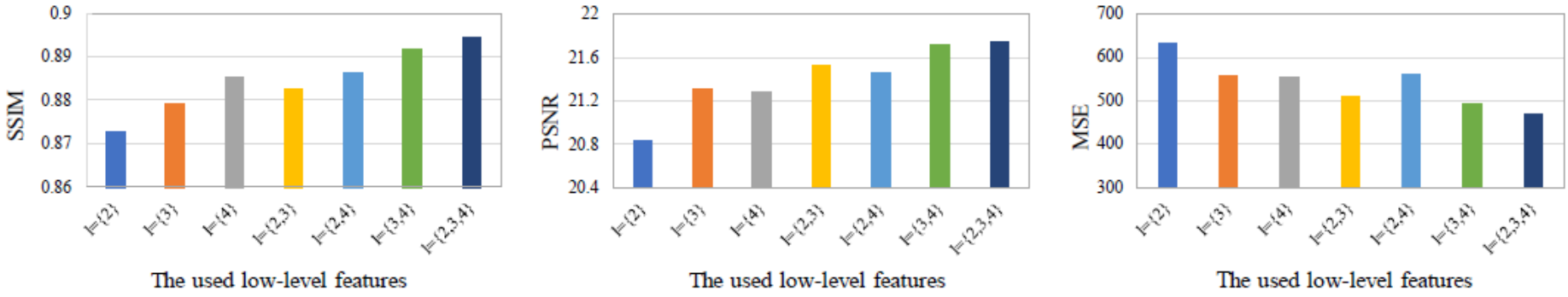
量化指标	数据集	UICoE-Net-w/o-LC	UICoE-Net-w/o-SC	UICoE-Net-w/o-S	UICoE-Net
SSIM↑	UIEB	0.8803	0.8837	0.8578	**0.8944**
	UICoD	0.8830	0.8877	0.8379	**0.9124**
PSNR(dB)↑	UIEB	20.3718	20.4537	20.2393	**21.7456**
	UICoD	21.1751	20.6574	20.6821	**23.1643**
MSE($\times 10^3$)↓	UIEB	0.6701	0.6304	0.7607	**0.4711**
	UICoD	0.6019	0.6062	0.7779	**0.3756**

■ 消融实验增强结果示例

Underwater Image Co-Enhancement

（四）实验验证与分析

➢ 消融实验



■ 采用不同低层协同重建特征组合的水下图像增强性能量化评价

■ 相比去除部分层的低层协同重建特征，当采用全部低层协同重建特征时的增强性能最优。这也证明，UICoE-Net中所设计的**多层特征匹配策略的有效性**。

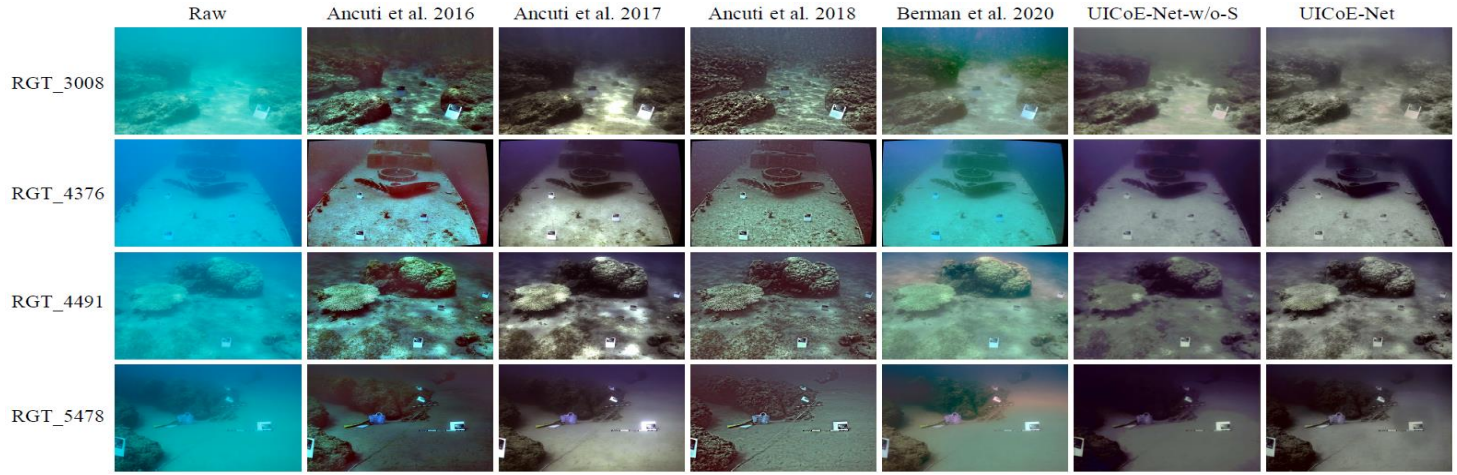
■ 此外，值得注意的是，当单独采用较高层次的低层协同重建特征时，例如$l = 4$，其增强性能略优于采用较低层次的低层协同重建特征。

■ 然而，对比特征组合$l = \{3,4\}$和特征组合$l = \{2,3,4\}$的增强结果可以发现，**更低层的低层协同重建特征对最终良好的UICoE-Net性能同样贡献颇大**。

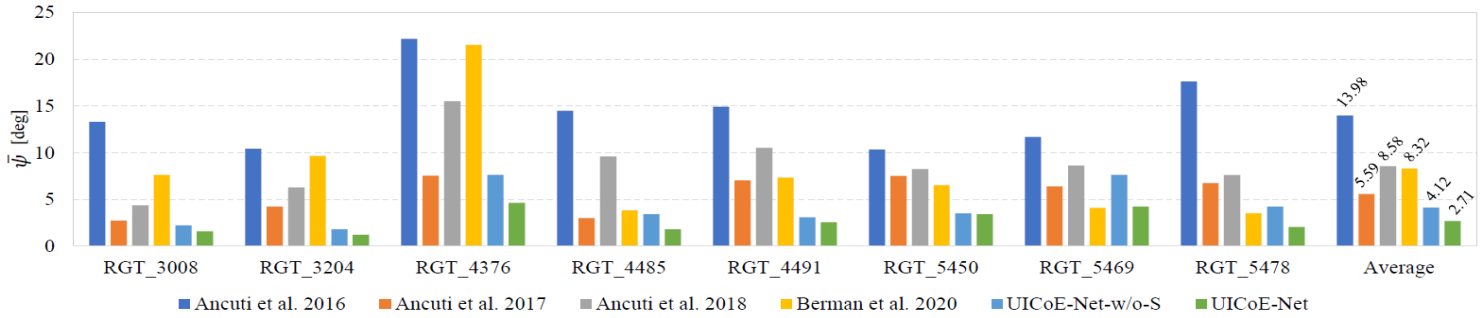
Underwater Image Co-Enhancement

（四）实验验证与分析

➢ 水下图像复原数据集（SQUID）实验



■ 不同方法在SQUID水下图像复原数据集上的结果对比

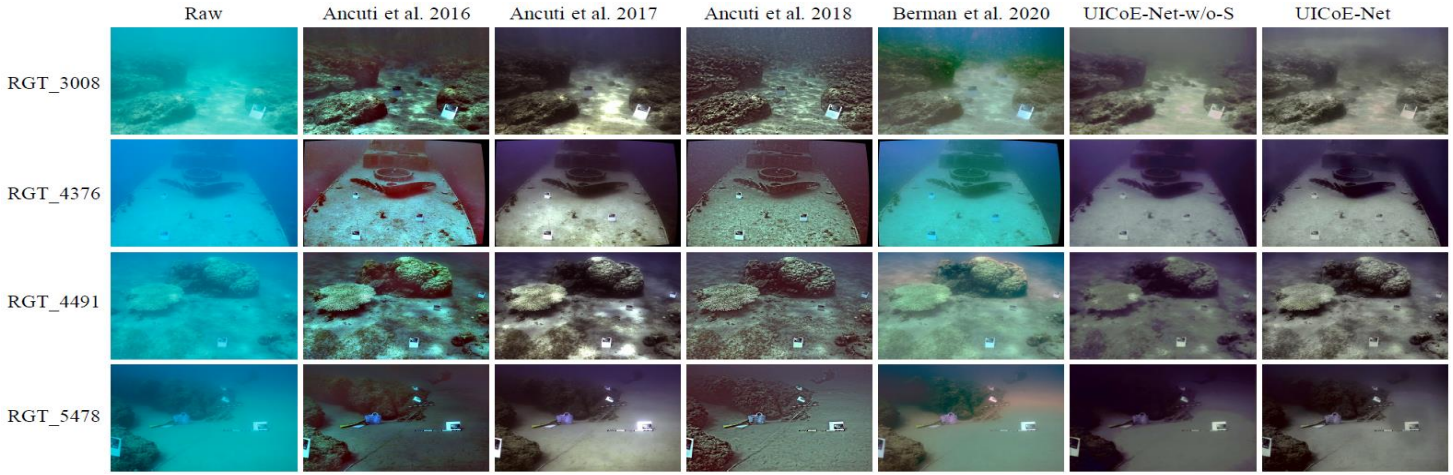


■ 各方法在灰度色卡区域上的平均再现角误差$\bar{\psi}$

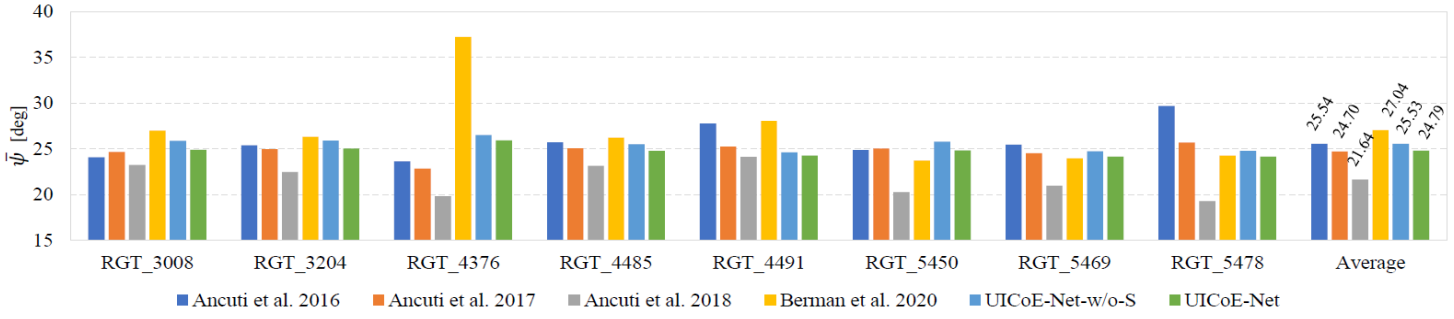
Underwater Image Co-Enhancement

（四）实验验证与分析

➢ 水下图像复原数据集（SQUID）实验



■ 不同方法在SQUID水下图像复原数据集上的结果对比



■ 各方法在在除灰度以外颜色区域上的平均再现角误差$\bar{\psi}$

Underwater Image Co-Enhancement

（四）实验验证与分析

➢ 讨论

■ 算法运行时间分解

采用模型		图像编码	关联特征匹配		图像解码	平均测试时间
			语义特征	低层特征		
UICoE-Net-w/o-LC	128×128	0.002	0.013	—	0.003	0.018
	256×256	0.003	0.053	—	0.004	0.060
UICoE-Net-w/o-SC	128×128	0.002	—	0.668	0.028	0.698
	256×256	0.003	—	2.788	0.131	2.922
UICoE-Net-w/o-S	128×128	0.001	—	—	0.001	0.002
	256×256	0.002	—	—	0.002	0.004
UICoE-Net	128×128	0.002	0.010	0.334	0.021	0.367
	256×256	0.003	0.038	1.289	0.070	1.400

- UICoE-Net可在0.4s内处理一幅128×128的水下图像，对256×256大小的图像平均需要1.4s，其中低层特征匹配单元耗时占总耗时的90%以上。
- 相比之下，消融模型UICoE-Net-w/o-LC，即去除了低层特征匹配单元的UICoE-Net模型，对于128×128（256×256）的处理效率达到了55FPS（16FPS）。
- 对于对水下图像增强算法有较高效率要求的应用，UICoE-Net-w/o-LC是一种理想的替代选择；
- 对增强质量要求较高、但效率要求较低的应用，完整的UICoE-Net模型是更为理想的选择。

Any questions?