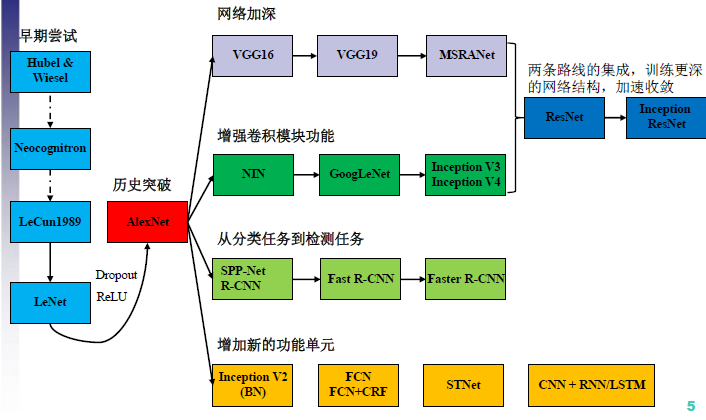
# 一 inception系列演进

## 1.1 inceptionv1

Inceptionv1的核心点

Inceptionv1的创新点

inceptionv1, v2, v3, v4的演进.

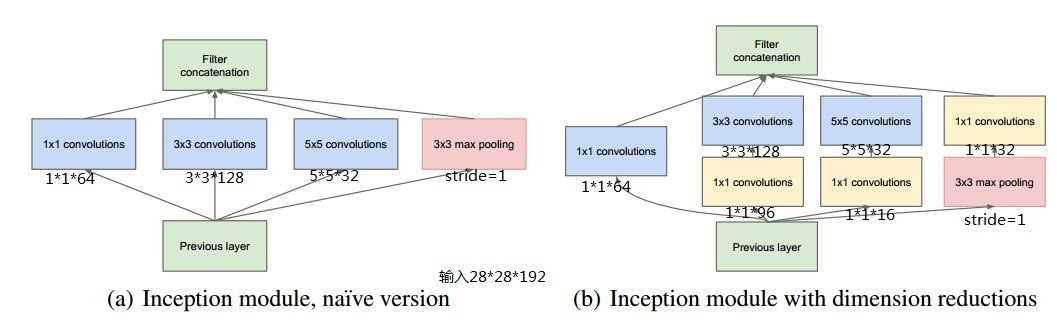


### 1.1.1 inception V1 特点

a) 从NIN中引入1x1卷积.

b) 代表作, googleLeNet.

c) 1x1的卷积,相当于对所有通道(chn)的像素线性组合输出.类似全连接.



假设previous layer的大小为28\*28\*192，则，

a的weights大小，1\*1\*192\*64+3\*3\*192\*128+5\*5\*192\*32=387072

a的输出featuremap大小，28\*28\*64+28\*28\*128+28\*28\*32+28\*28\*192=28\*28\*416

b的weights大小，1\*1\*192\*64+(1\*1\*192\*96+3\*3\*96\*128)+(1\*1\*192\*16+5\*5\*16\*32)+1\*1\*192\*32=163328

b的输出feature map大小，28\*28\*64+28\*28\*128+28\*28\*32+28\*28\*32=28\*28\*256

1\*1 conv，从上面的数据可以**看出一方面减少了weights(?)，另一方面降低了dimension**。

Inception v1的亮点总结如下：

(1)卷积层共有的一个功能，可以实现通道方向的降维和增维，至于是降还是增，取决于卷积层的通道数（滤波器个数），在Inception v1中1\*1卷积用于降维，减少weights大小和feature map维度。◇◇◇

(2)1\*1卷积特有的功能，由于1\*1卷积只有一个参数，相当于对原始feature map做了一个scale，并且这个scale还是训练学出来的，无疑会对识别精度有提升。◇◇

(3)增加了网络的深度

(4)增加了网络的宽度

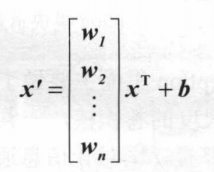
(5)同时使用了1\*1，3\*3，5\*5的卷积，增加了网络对尺度的适应性

### 1.1.2 1x1卷积用处

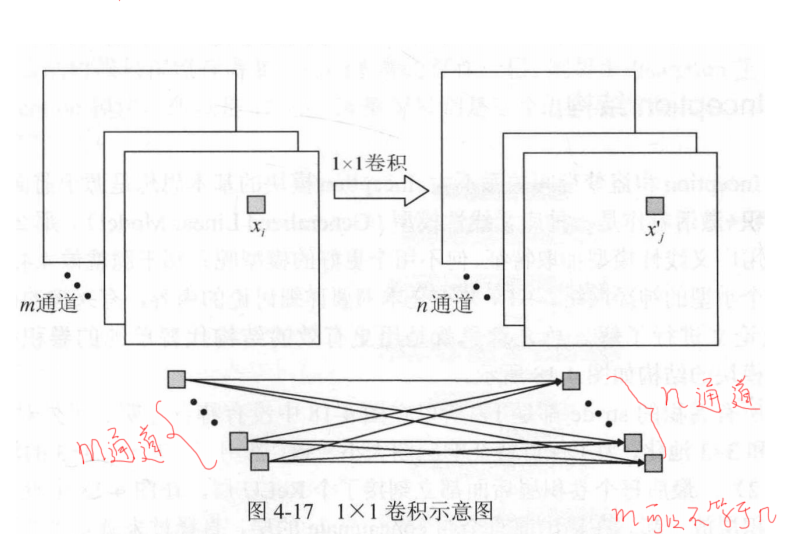
1. (直观上)给特征响应图都乘以一个系数. 这个只是单通道是每个Fmap值都乘以一个系数.但是多通道就不是了.
2. M通道上同一位置的像素, 经1x1卷积后, 会在对应位置产生一组新的n个通道像素. ◇
3. 如果只看”同一位置”像素的话, 就是一个全连接层. ◇◇

Xi是第i个输入通道上的像素值, Xj’是第j个输出通道上的像素值.





1. 类似同一位置上像素做的一个线性组合. ◇◇
2. 如果m>n, 则会有降维效果.(通道数降低) ◇◇◇



## 1.2 Inception V2

Inception v2的网络，代表作为加入了BN（Batch Normalization）层，并且使用2个3\*3替代1个5\*5卷积的改进版GoogleNet。

Inception v2的亮点总结如下：

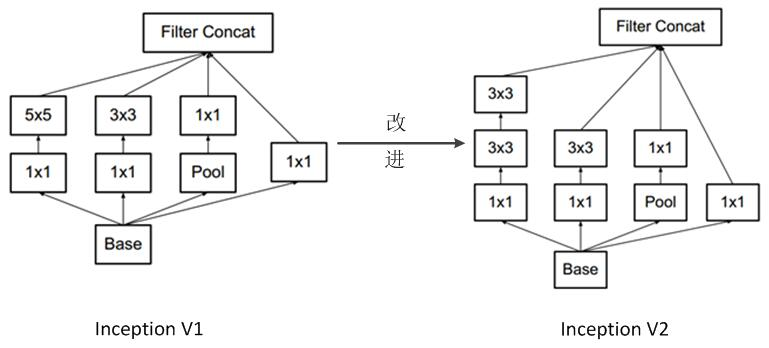
(1)加入了BN层，减少了InternalCovariate Shift（内部neuron的数据分布发生变化），使每一层的输出都规范化到一个N(0, 1)的高斯，从而增加了模型的鲁棒性，可以以更大的学习速率训练，收敛更快，初始化操作更加随意，同时作为一种正则化技术，可以减少dropout层的使用。

(2)用2个连续的3\*3 conv替代inception模块中的5\*5，从而实现网络深度的增加，

**网络参数减少:**

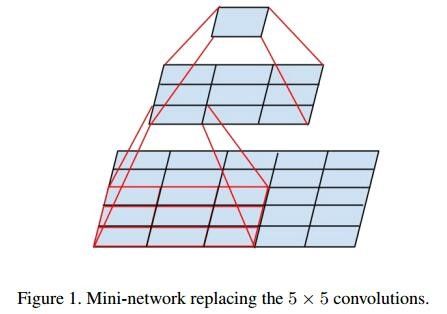
1. 5x5xchn个参数.
2. 2x(3x3xchn) 个参数

减少了(1 – 18/25) = 28%



### 5x5替换成2个级联3x3在参数量和计算量上的影响

大尺寸的卷积核可以带来更大的感受野，但也意味着更多的参数，比如5x5卷积核参数(仅考虑w不考虑b)是3x3卷积核的（5\*5+1）/（3\*3+1）=2.6倍。为此，《Rethinking the Inception Architecture for Computer Vision》的作者提出可以用2个连续的3x3卷积层(stride=1)组成的小网络来代替单个的5x5卷积层，(保持感受野范围的同时又减少了参数量)如下图：



#### 参数对比

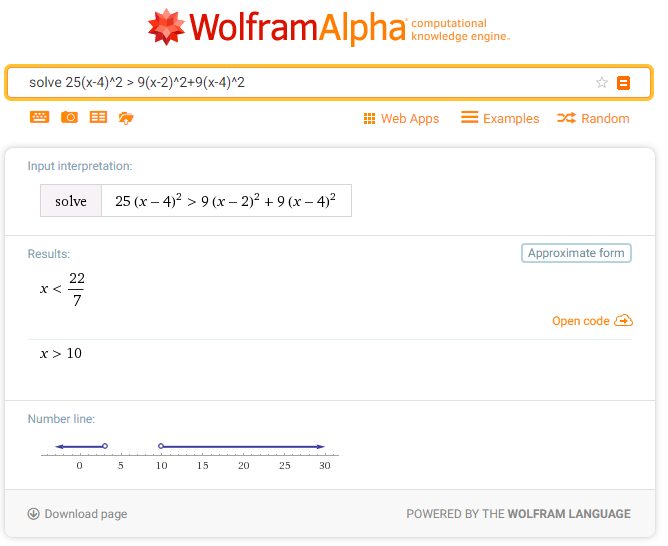
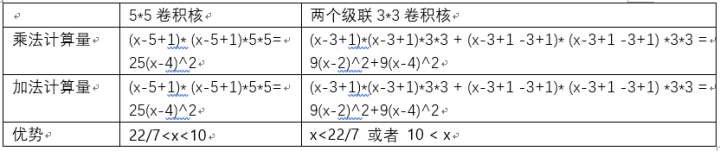


由于参数个数仅与卷积核大小有关，所以3\*3级联卷积核占优势。

#### 计算量对比

输入记为x，为了方便讨论假设padding=0,stride=1。此时卷积计算公式 output =( input – kernel + 2padding) / stride + 1简化为output = input – kernel + 1。

* 5\*5卷积：有(x-5+1)\* (x-5+1)个输出点，每个输出点对应5\*5次乘法和5\*5次加法（5\*5次乘法的结果求和再加上b，一共5\*5+1个数相加，所以需要5\*5次加法）
* 3\*3卷积：第一个3\*3卷积有(x-3+1)\*(x-3+1)个输出点，每个输出点对应3\*3次乘法和3\*3次加法，第二个3\*3卷积的输入是(x-3+1)\*(x-3+1)，在其上做卷积有(x-3+1 -3+1)\* (x-3+1 -3+1)个输出点，每个输出点对应3\*3次乘法和3\*3次加分。



总的来说当**x<22/7 或者10<x** ，两个3\*3的卷积核在参数个数和计算量上都占优势。

## 1.3 inceptionv3

v3:Rethinking the InceptionArchitecture for Computer Vision

Inception v3网络，主要在v2的基础上，提出了卷积分解（Factorization），代表作是Inceptionv3版本的GoogleNet。

### Inception v3的亮点

(1) 将7\*7分解成两个一维的卷积（1\*7,7\*1），3\*3也是一样（1\*3,3\*1），这样的好处，既可以加速计算（多余的计算能力可以用来加深网络），又可以将1个conv拆成2个conv，使得网络深度进一步增加，增加了网络的非线性，更加精细设计了35\*35/17\*17/8\*8的模块。

(2)增加网络宽度，网络输入从224\*224变为了299\*299。

### 为什么非对称卷积（Asymmetric Convolution）减少了运算量?

a) 可以引入类似前面的思考.求出两层的输出尺寸, 每个尺寸对应乘法加法运算. 计算运算量(乘法运算量).

输入是x\*x的图. 经过1x3和3x1的级联卷积.

考虑, stride=1, padding=0.  输出尺寸: out=(input-kernel+2\*pad)/stride+1



第一层, 经1x3之后的输出是:

           i)   "3"那个维度对应尺寸是: (x-3+1), 每个像素计算量是3. 则为(x-3+1)\*3

           ii)  "1"那个维度对应的尺寸是: (x-1+1), 每个像素计算量是1, 则为 (x-1+1)

           iii) 第一层后的尺寸是: (x-3+1) \* (x-1+1)

第二层, 运算3x1,

          i)  要对(x-3+1)小尺寸上运算"1"维度. 大尺寸上运算"3"维度.

          ii) "3"维度上, (x-1+1  -3 +1 ) \* 3

          iii) "1"维度上, (x-3+1 -1 +1) \* 1

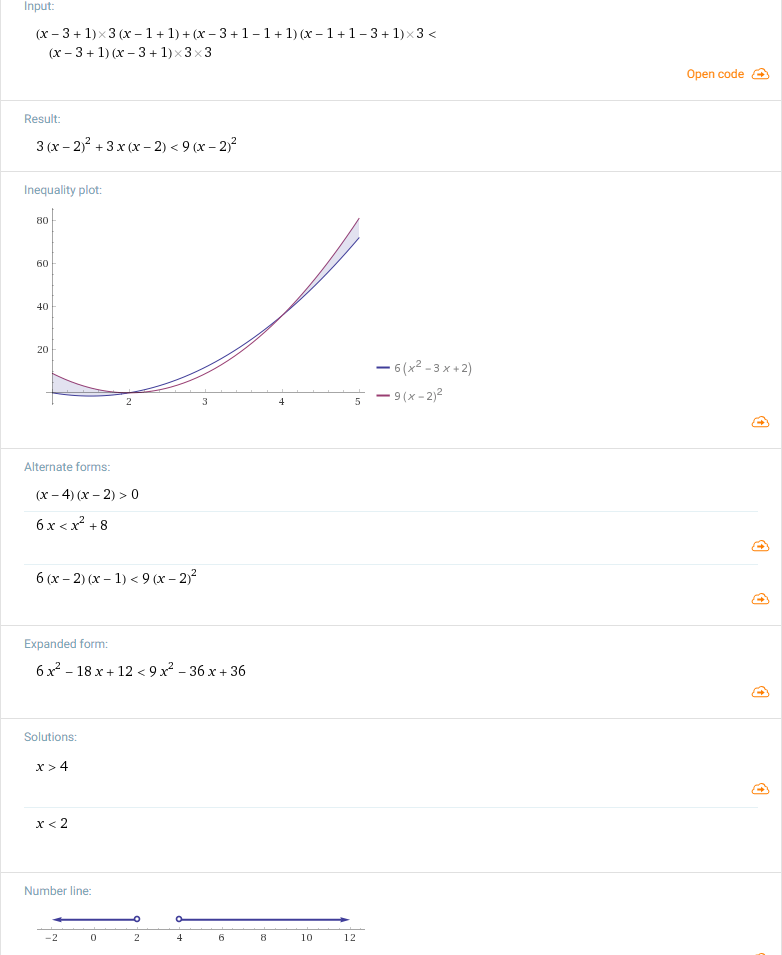
对比3x3的计算量:

       (x-3+1)\*(x-3+1)\*3\*3.

计算:

(x-3+1)\*3(x-1+1)+(x-3+1-1+1)\*(x-1+1-3+1)\*3 < (x-3+1)\*(x-3+1)\*3\*3

得 x<2,或x>4时, 非对称卷积有优势.



## **1.4 inceptionv4**

### 为了解决什么问题?

a) 以前的inceptionv3有许多负责不高效的内容, 为了能够在DistBelief上做分布式训练,  现在用tensorflow重写(精简, 优化)inception

b) resnet结构表现很好, 和inception结构做一个融合, 引出:

i) Inception-ResNet v1：Inception加ResNet，计算量和Inception v3相当，较小的模型

ii) Inception-ResNet v2：Inception加ResNet，计算量和Inception v4相当，较大的模型，当然准确率也更高

### 主要特点

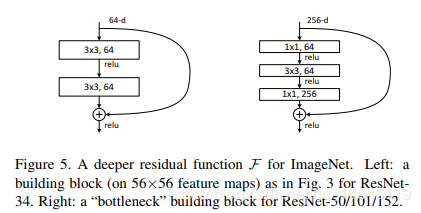
Inception v4**主要利用残差连接（Residual Connection）来改进v3结构**，代表作为，Inception-ResNet-v1，Inception-ResNet-v2，Inception-v4

resnet中的残差结构如下，这个结构设计的就很巧妙，简直神来之笔，使用原始层和经过2个卷基层或者3个卷积层的feature map做Eltwise。

首先介绍几个概念:

1. 左边的3\*3+3\*3(ResNet18，ResNet34)和1\*1+3\*3+1\*1（ResNet50，ResNet101，ResNet152）称为瓶颈单元（**bootlenect，因为输入为256，中间为64，输出为256，宽窄宽的结构，像瓶子的颈部**）。
2. 右面的直线，有的实现是直线中有1\*1卷积，称为shortcut。整个bootlenect+shortcut称为Residual uint。几个Residual uint的叠加称为Residual block。Resnet结构都是由4个Residual block组成的。
3. Eltwise区别于concat
4. Eltwise和concat都能实现”拼接”含义.
5. Concat是在某个维度上拼接(追加), 需要保证其他维度时相等的.
6. Eltwise是两个tensor做product(点乘), sum(求和), max(最大值)的方法”拼接”.

Inception-ResNet的改进就是使用上文的Inception module来替换resnet shortcut中的bootlenect。



Inception v4的亮点总结如下：

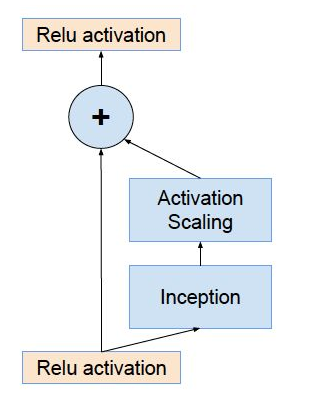
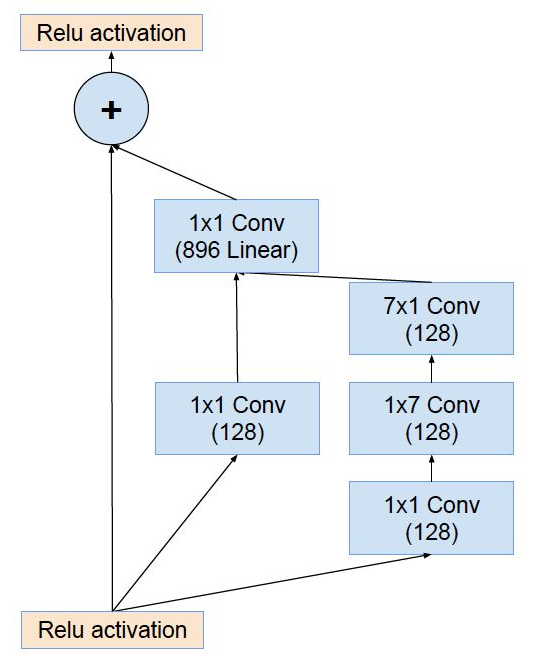
(1)将Inception模块和ResidualConnection结合，提出了Inception-ResNet-v1，Inception-ResNet-v2，使得训练加速收敛更快，精度更高。

ILSVRC-2012测试结果如下（single crop），

(2)设计了更深的Inception-v4版本，效果和Inception-ResNet-v2相当。

(3)网络输入大小和V3一样，还是299\*299

Inception模块结构



1. Residual部分并不能提升模型的精度,但是能够加快训练收敛.

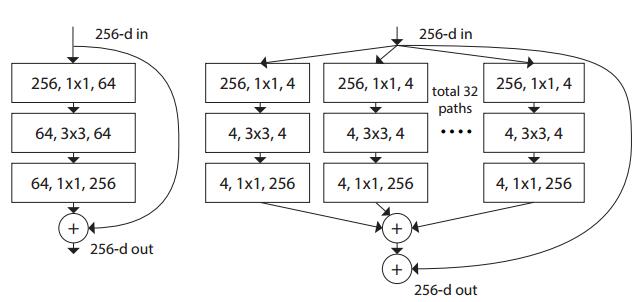
|  |
| --- |
| ***Although the residual version converges faster, the final accuracy seems to mainly depend on the model size.*** |

1. 如果网络太深, 引入residual connection之后网络会不好训练.到后面(?, 后面层)输出是0.
2. 为了解决这个零值问题, 引入scale系数(缩小系数, 通常0.1).
3. Scale如上图右图, 在激活函数之前添加一个scale.
4. 引入scale后, 没有降低精度, 同时使网络更稳定.

|  |
| --- |
| **def** forward**(**self**,** x**):**  out **=** self**.**conv2d**(**x**)** # 这里可以是卷积层、可以是Inception模块等任意sub-network  out **=** out **\*** self**.**scale **+** x # 乘以一个比例系数再相加  out **=** self**.**relu**(**out**)**  **return** out |

# 二 resnet的升级版ResNeXt

1. The next dimension.
2. 引入新的维度, cardinality, 基数.
3. 描述resnet模块个数.
4. 评价网络的其他的”维度”有, width和depth.
5. 增加cardinality(基数,resnet模块数)的效果比增大网络的width和depth更有效.
6. 相比resnet, resNeXt含有更少的参数.
7. 采用” split-transform-merge”
8. 左图是resnet结构. 含有bottlenect和shortcut.
9. 右图是resNeXt结构, 除了含有bottlenect和shortcut外, 还有(体现出)split-transform-merge的结构.

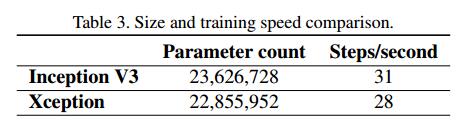


# 三 Xception网络

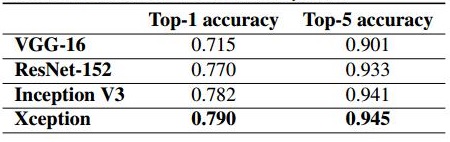
1. 基于inceptionV3
2. 引入通道可分离卷积(depthwise seperable convolution).

效果上,

1. 参数量稍微减少



1. 精度上稍微增强

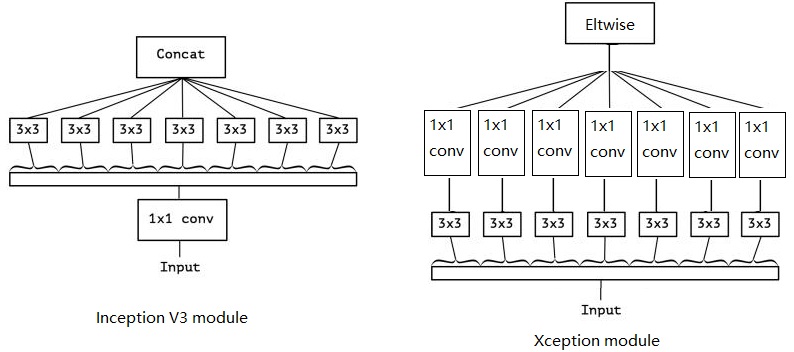


与inceptionv3的区别.

1. 下图从下往上看是数据流.(input 在下).
2. 左图inceptionv3先做1\*1卷积, 在通道对应位置像素上线性组合.

然后在3\*3卷积.

1. Xception(右图)先做3\*3卷积, 然后在做1\*1卷积.



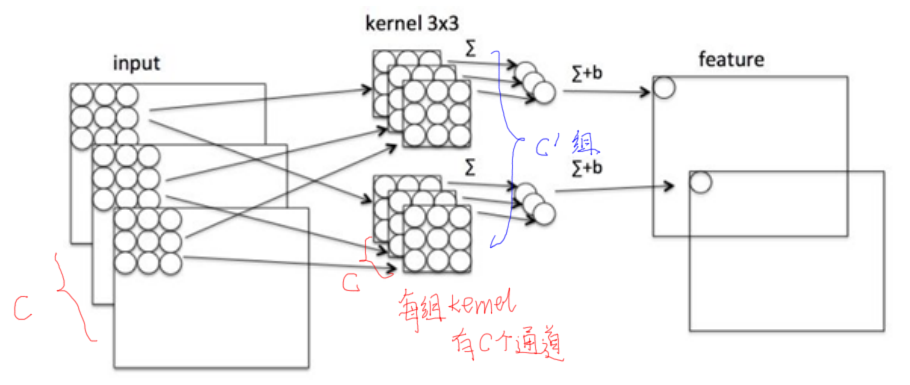
## 深度可分离卷积

1. 分成两步走. Depthwise和pointwise.



### 3.1.1 普通卷积过程





1. 设输入[B, H, W, C], stride=1, padding=1. 卷积kernel为3\*3, 则输出[B,H,W,c’]



1. 共有c’个kernel.
2. **乘法计算量为(不考虑batch): H\*W\*3\*3\*C\*c’**
3. **参数量: 3\*3\*C\*c’**

### 3.1.2 depthwise过程

a) 普通过程的C个通道的kernel替换成总共C个kernel.

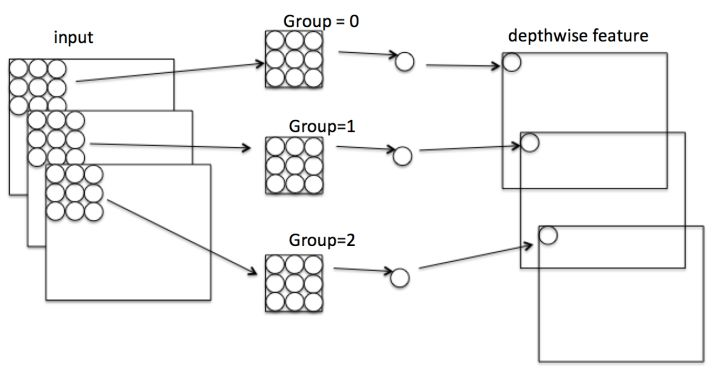
b) 效果是对每一个depth上做卷积.

c) 相当于收集每个chn上的空间特征.

d) 输出的chn还是和之前的Chn一样

**e) 乘法计算量(不考虑Batch): H\*W\*3\*3\*C**

**f) 参数量: 3\*3\*C**



### 3.1.3 pointwise过程

a) 接着上面depthwise的输出feature图.

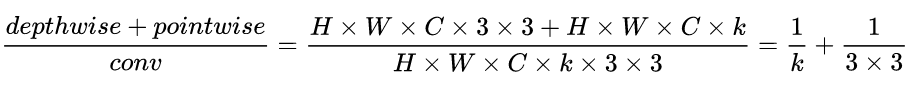
b) 引入1\*1的卷积. 类似对应chn上做全连接(线性组合)的效果.

c) 效果上是收集每个点的信息, point的wise特征.

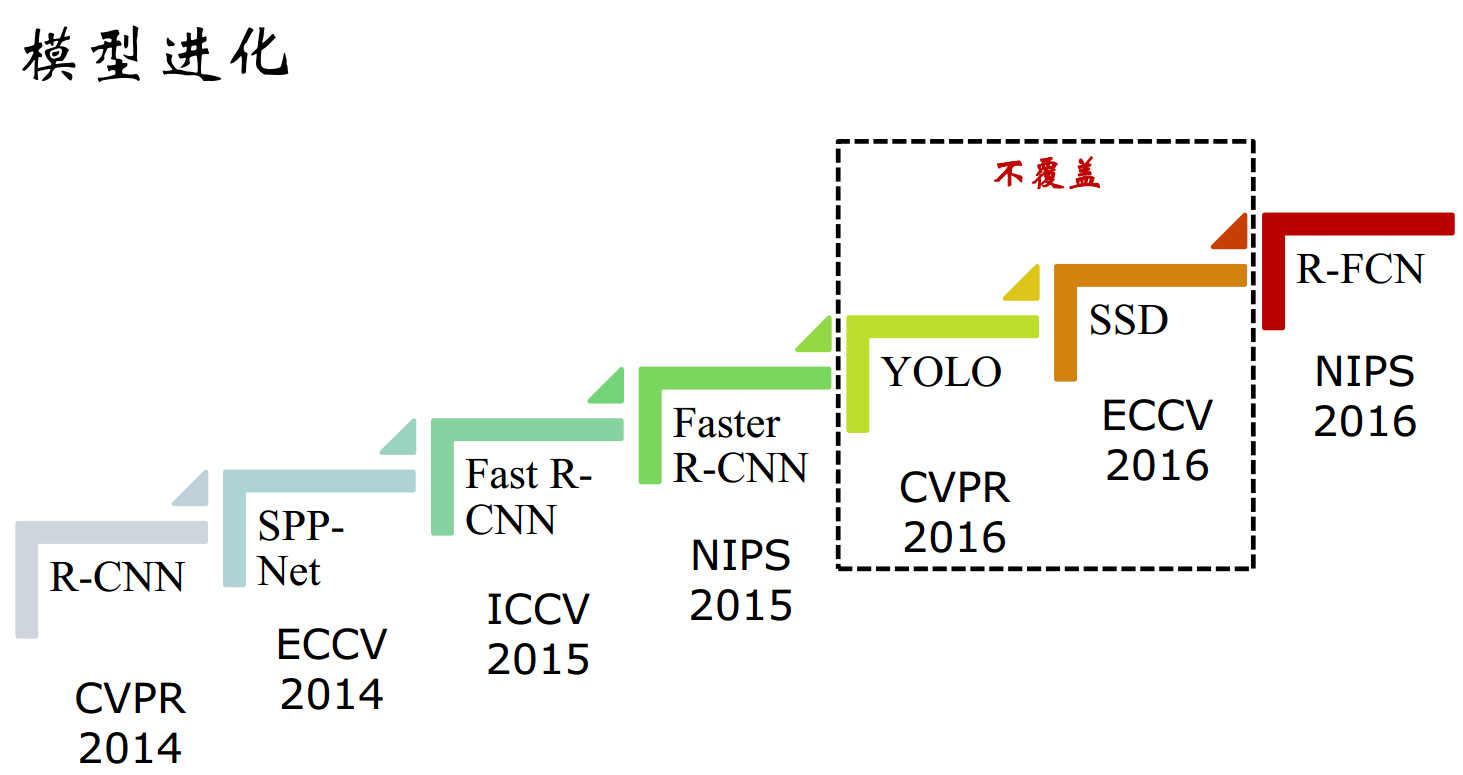
**d) 乘法计算量(不算batch): H\*W\*C\*1\*1\*c’**

**e) 参数量: 1\*1\*c’**

### 3.1.4 计算量和参数量降低效果

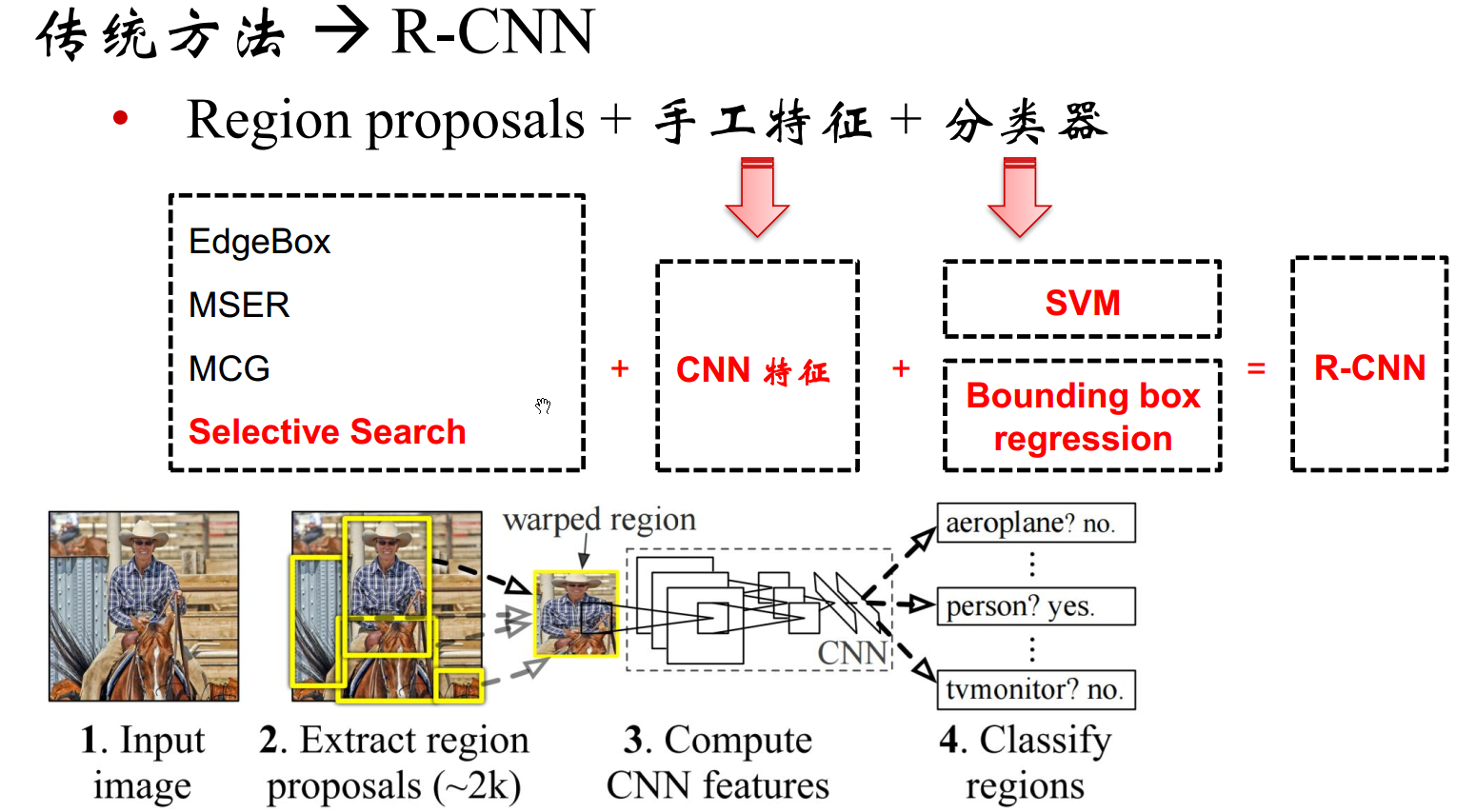


# 四 检测网络

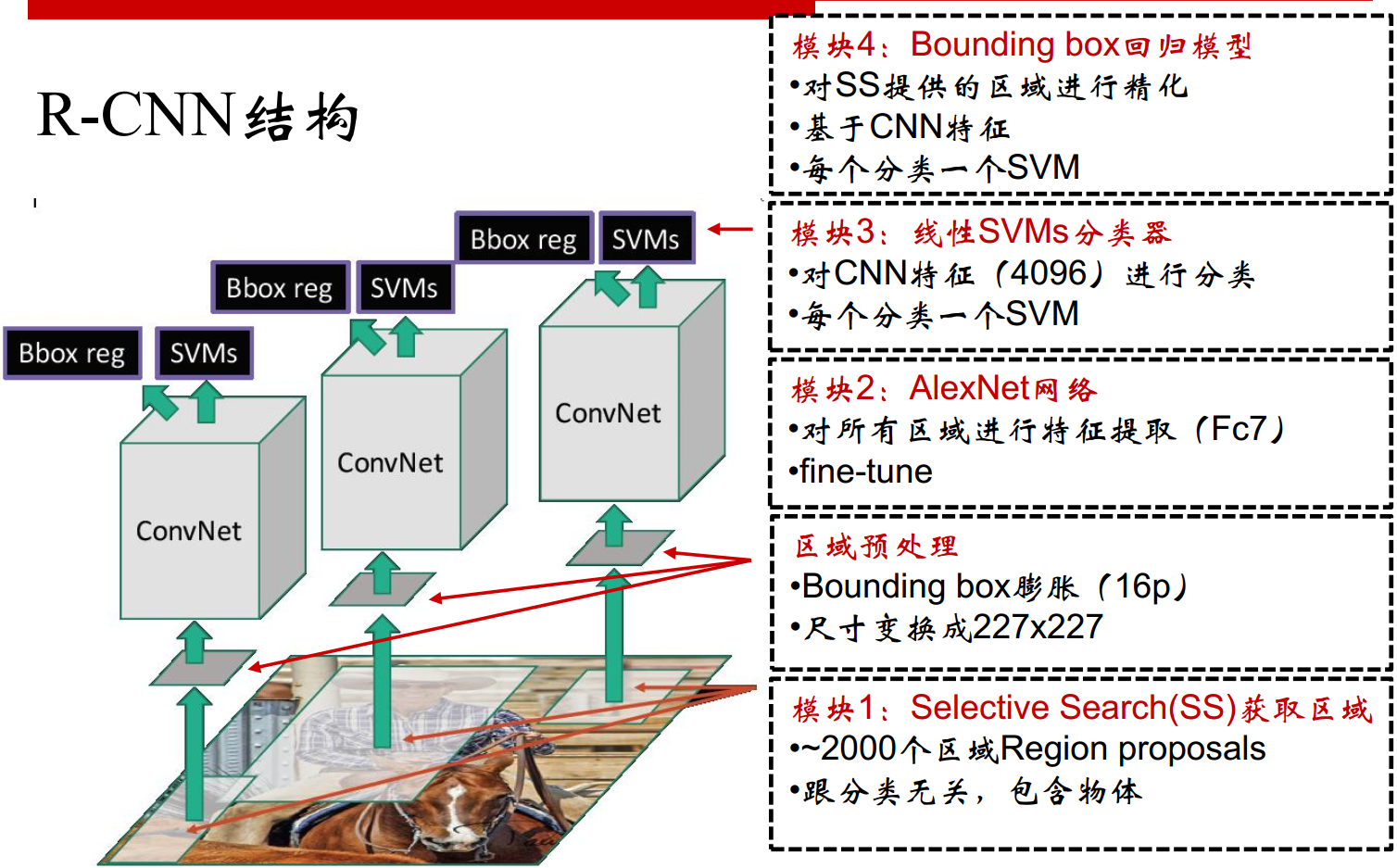


## 4.1 R-CNN

### R-CNN解决问题



1. 传统方法
2. 在整个图像上穷举出候选框
3. 然后提取特征,进入svm分类
4. 最后经nms处理.
5. R-CNN代替了传统的SIFT,HOG特征. 利用cnn自助提取特征.
6. 对于原图, 使用selective search方法搜索(出候选框)
7. 在候选框上做cnn提特征.(输入时固定大小的,因此需要对候选框缩放到统一大小)
8. 将特征送给svm做分类.
9. 通过nsm优化输出结果
10. R-CNN 的mAP为58.5%.
11. 缺点计算量大.
12. SS的候选框可能有2k个.
13. 对每个候选框都要做CNN推断提特征.很耗时.
14. 如下图, 每个候选框都有一个convNet.

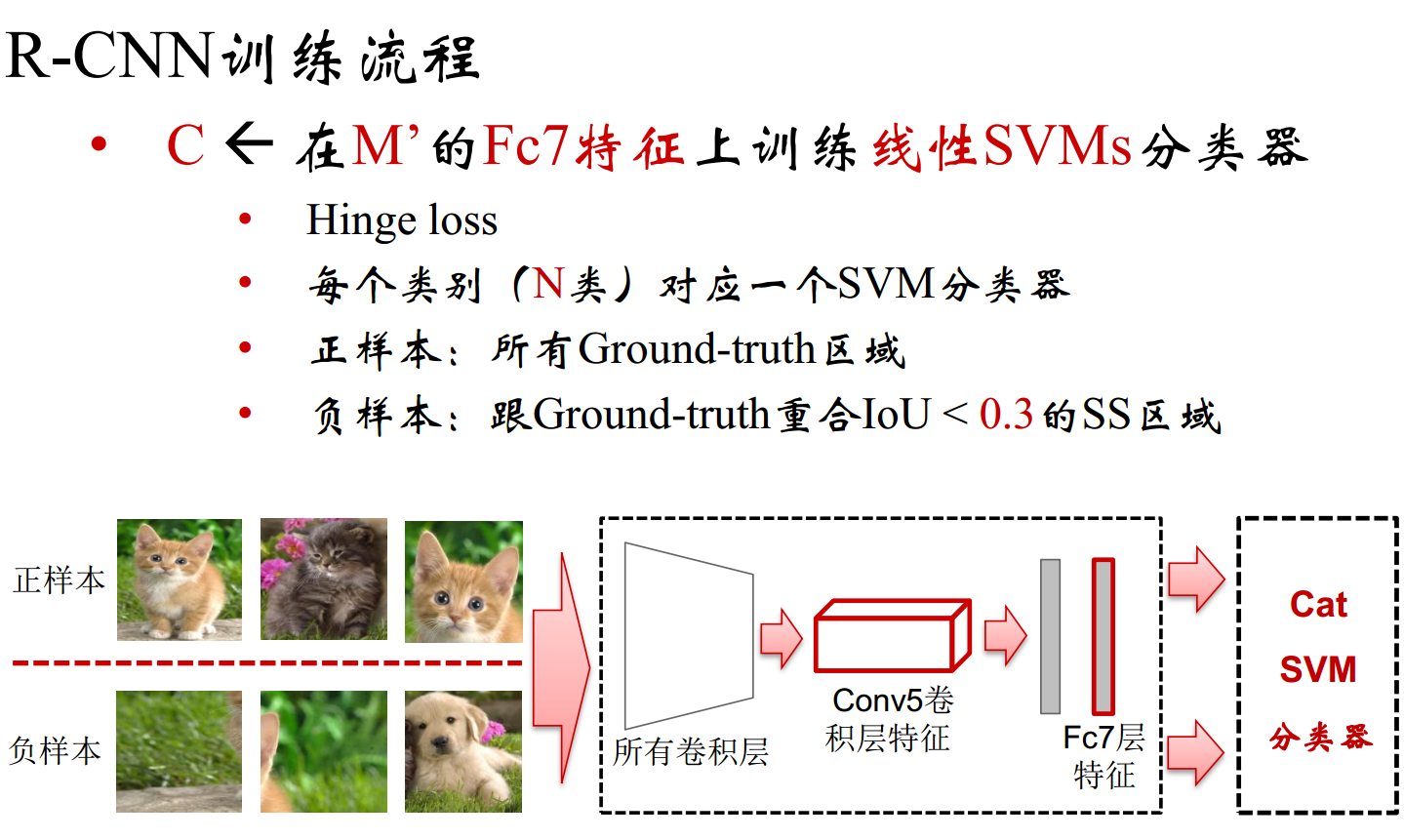


### R-CNN训练流程

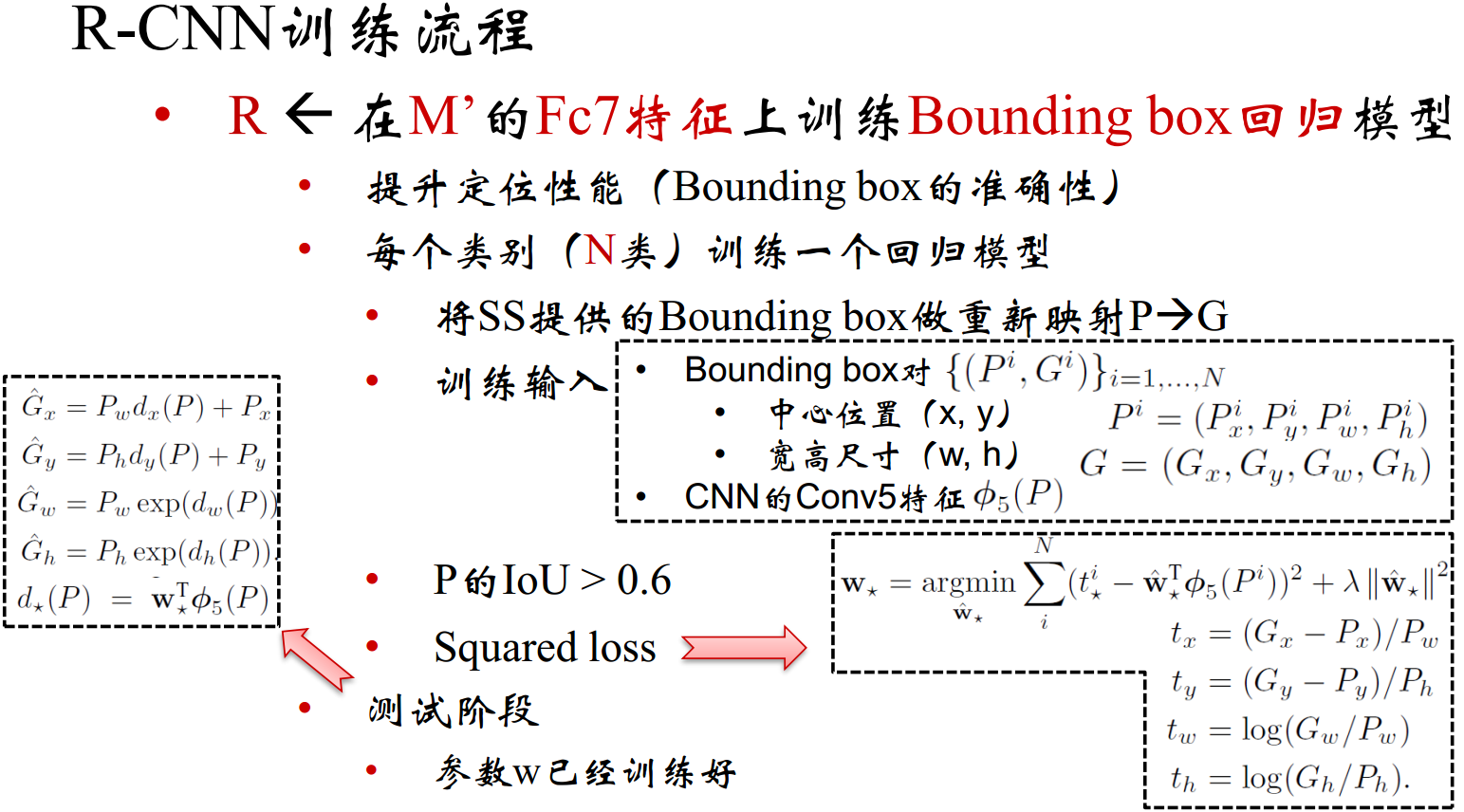
1. 在数据集上训练cnn体特征
2. 要想训练, 需要引入一个loss. 这里用softmax loss
3. 要想训练, 需要一个数据集, 这里用ImageNet.
4. 要想训练, 最好有一个基础的backbone, 这里用AlexNet.
5. 下图的M和M’都表示的网络.
6. M是ImageNet上预处理网络(类似步骤a) )
7. M’是对M做的finetune.
8. SS会生成一组候选框. 把每个候选框也可以看成ImageNet类似的”整幅”图片.
9. 用这些SS的候选框”图片”作为输入, 以图片的Gt为label做finetune.
10. 其中使用Log loss



1. 用SS生成候选框, 然后经M’的网络提取候选框的特征. 保存这些特征.
2. 这些特征用来train svm
3. 使用svm loss, hinge loss.



1. 分类器训练好了, 回归器训练.
2. 在M’输出的特征上训练回归网络.
3. SS的候选框有一组boundingbox. {Pi}
4. Gt的框也是一组boundingbox {Gi}
5. Pi和Gi肯定是有偏差的.
6. 构造这个基于(描述,说明,体现)偏差的损失.
7. 最小化这个损失.最终会得到一个较为合理的回归网络.



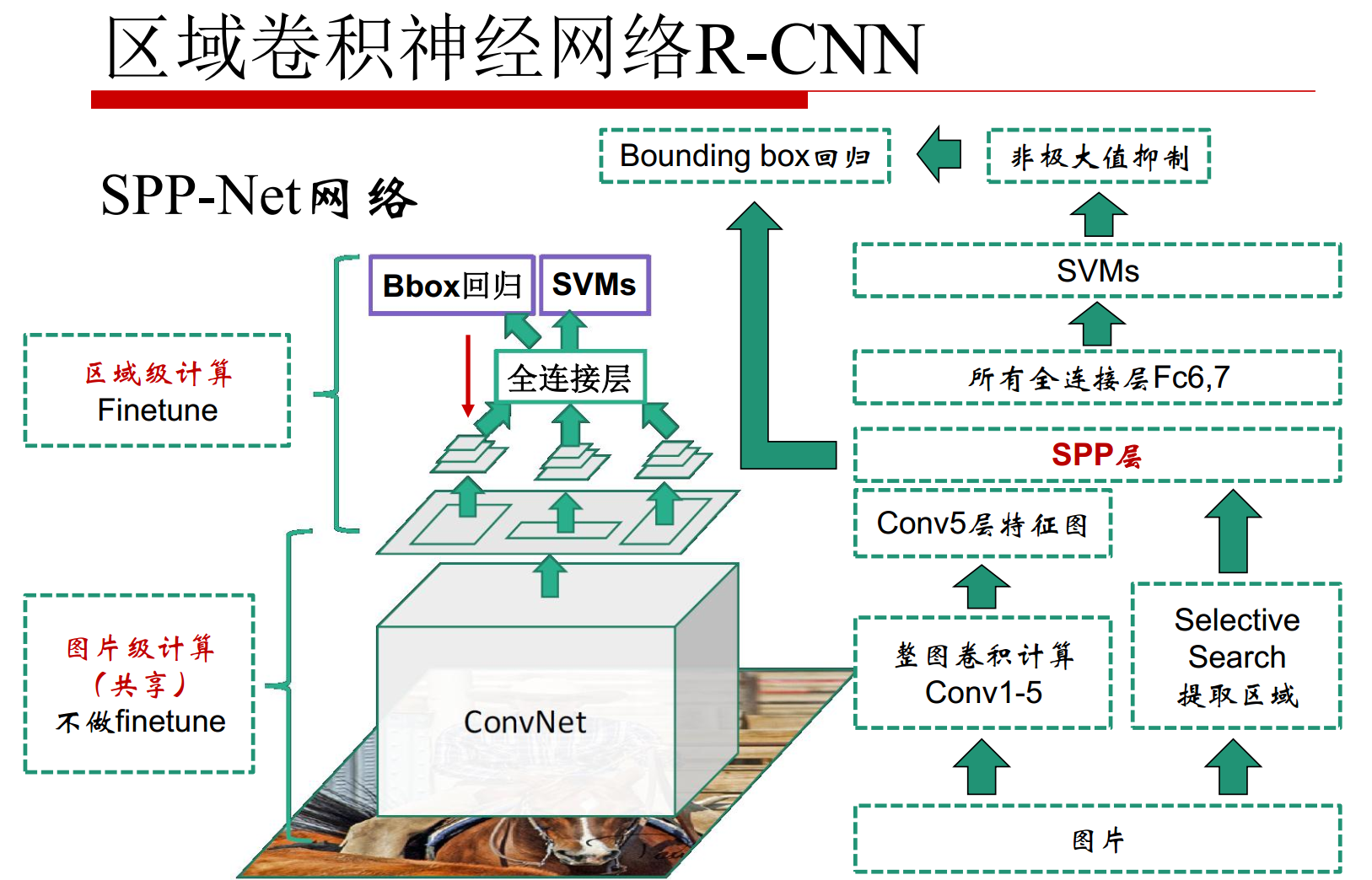
## 4.2 SPPNet

### 为了解决什么问题?

1. R-CNN都需要在cnn前有缩放到同一尺寸.
2. R-CNN需要依赖SS提取的候选框.
3. SS提取候选框一般有2k个.
4. 每个候选框都要经过一个cnn提特征,这是耗时点.

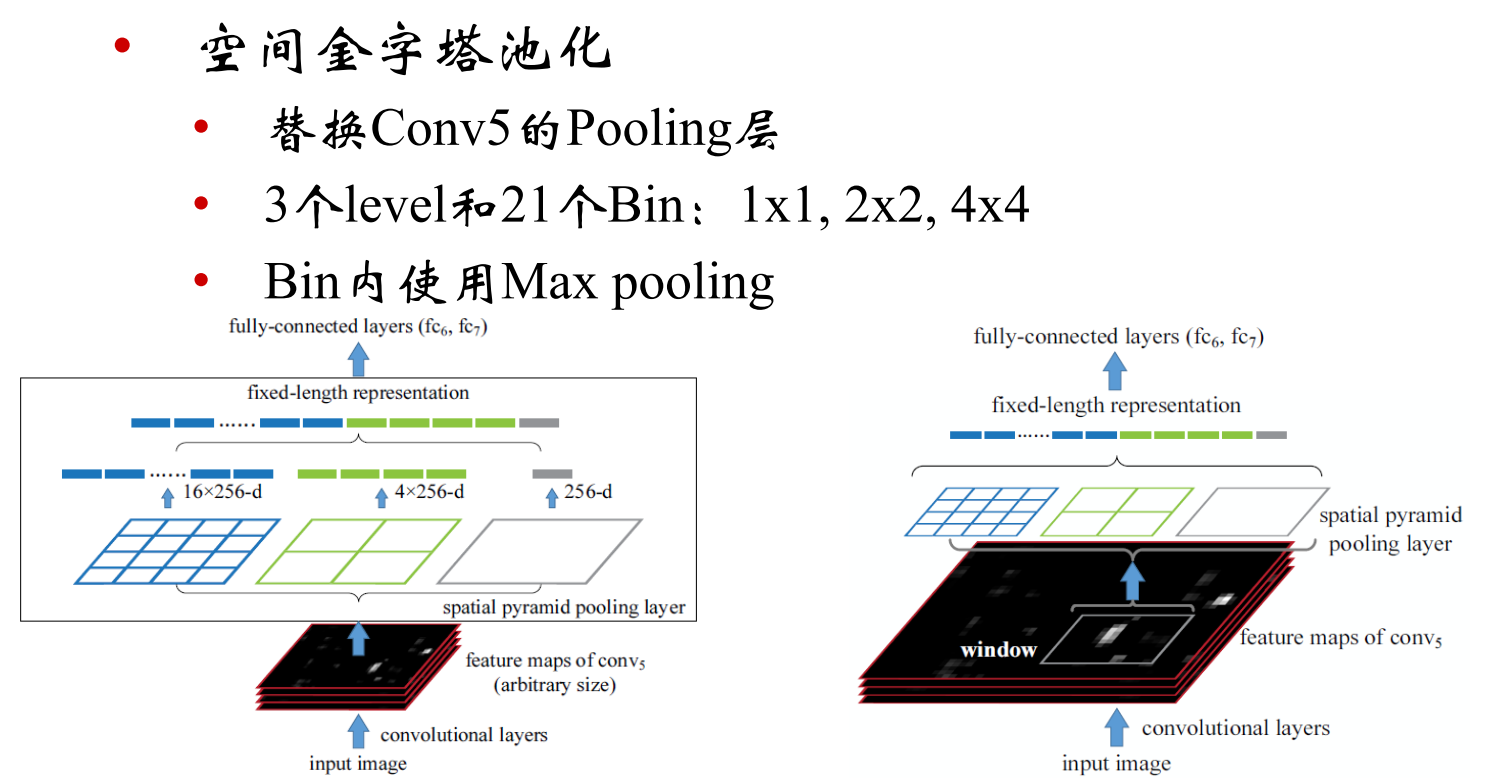
引入SPPNet.

1. 对一张输入图做一次cnn提取特征.(在conv5输出上提特征)
2. 利用SPPNet, 为不同尺寸的区域在conv5特征图上提特征.
3. SPPNet输出固定尺寸,映射到固定尺寸的全连接上.



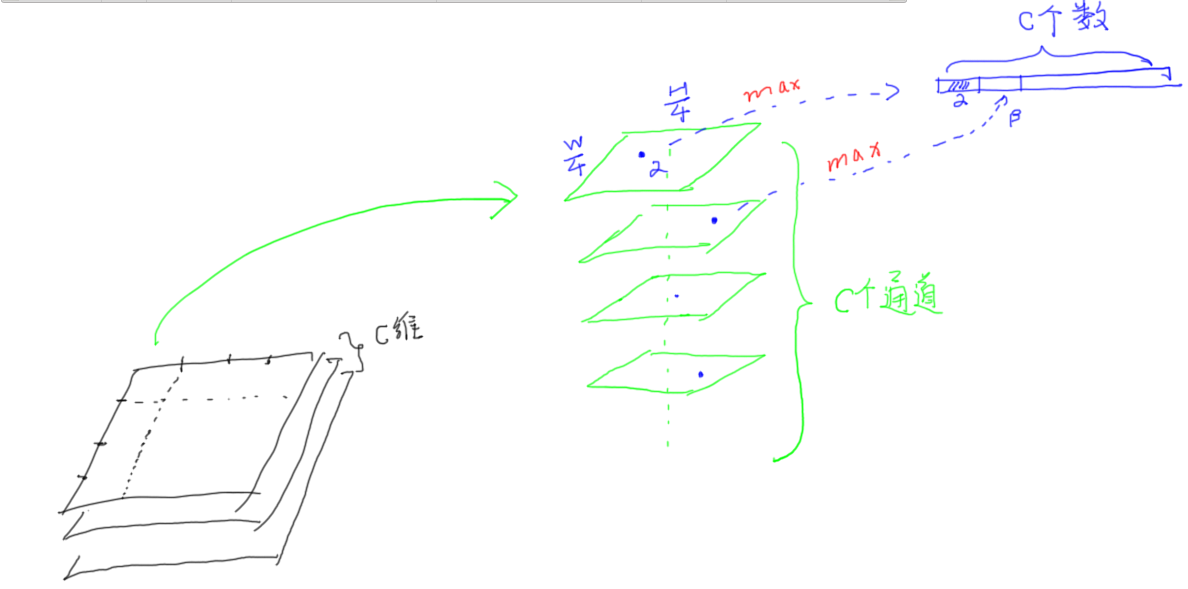
1. SPPNet接受的不固定的尺寸来自SS给出的候选框
2. SPPNet输出的固定尺寸是给后面的全连接层使用.

### SPP Pooling生效

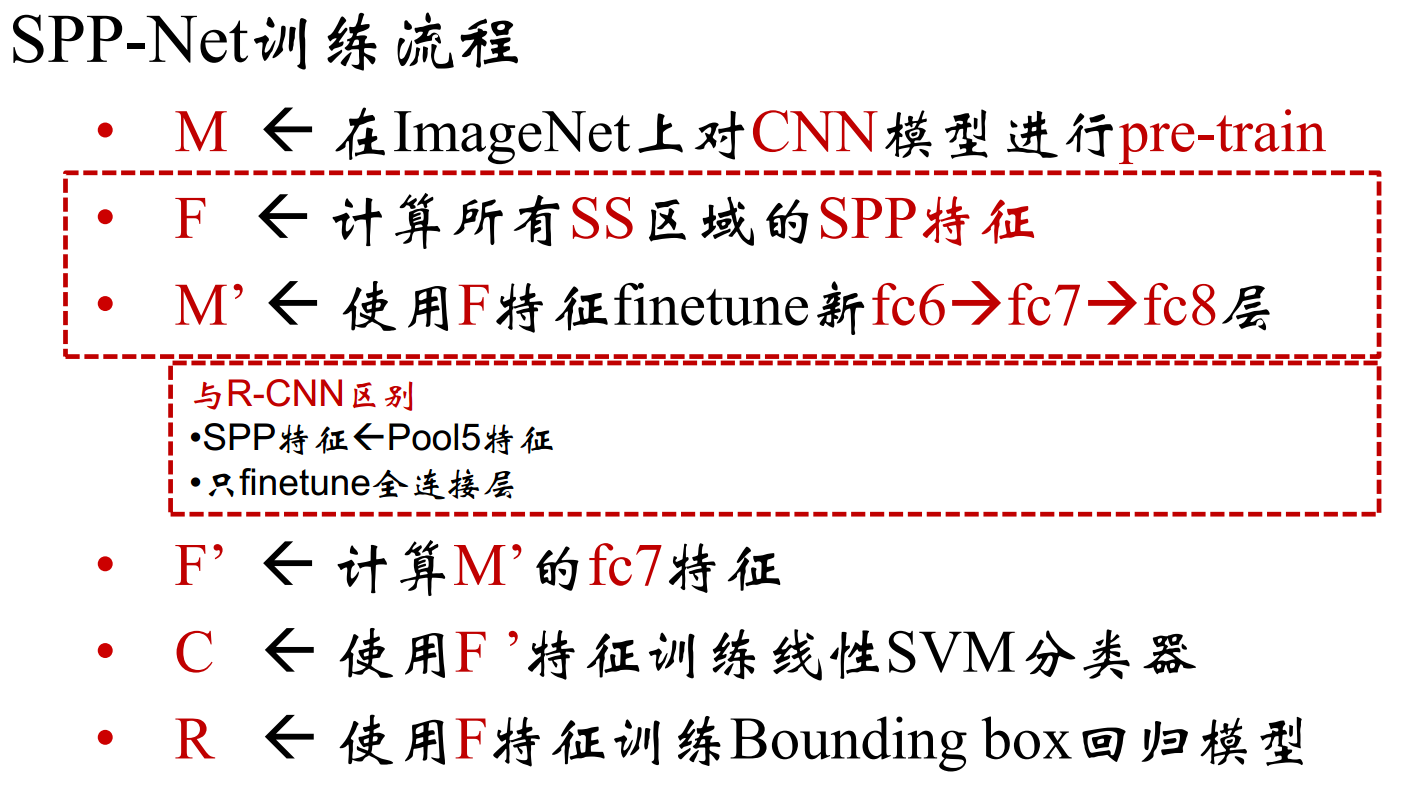


SPP Pooling操作(原理,生效原因)

1. SPP Pooling层是Pooling作用,不是卷积作用, 没有超参数, 不需要训练这一层, 直接使用这一层就可以了.
2. 输入特征图(feature maps)尺寸设为[H,W,C]
3. 整个SPP pooling过程C维度不变. SPP Pooling层要体现不随(依赖)H,W变化
4. 四等分
5. 同一份feature map, 先按照H,W等分4下, 共16分. 每一份是[H/4, W/4, C]的维度.
6. 每一份([H/4, W/4, C] )的每一个通道上就是 H/4\*W/4的图, 从中取出最大值(是个scaler标量).
7. 一共C个通道可以取出C个最大值. 组合成向量是C个.
8. 一共有16份这样的, 共有16C个向量.
9. 二等分
10. H,W各二等分, 共4C个向量.
11. H,W不均分
12. 共1C个向量
13. 总共是 16C+4C+1C = 21C
14. 这里是和H,W无关的向量了.



### SPPNet训练



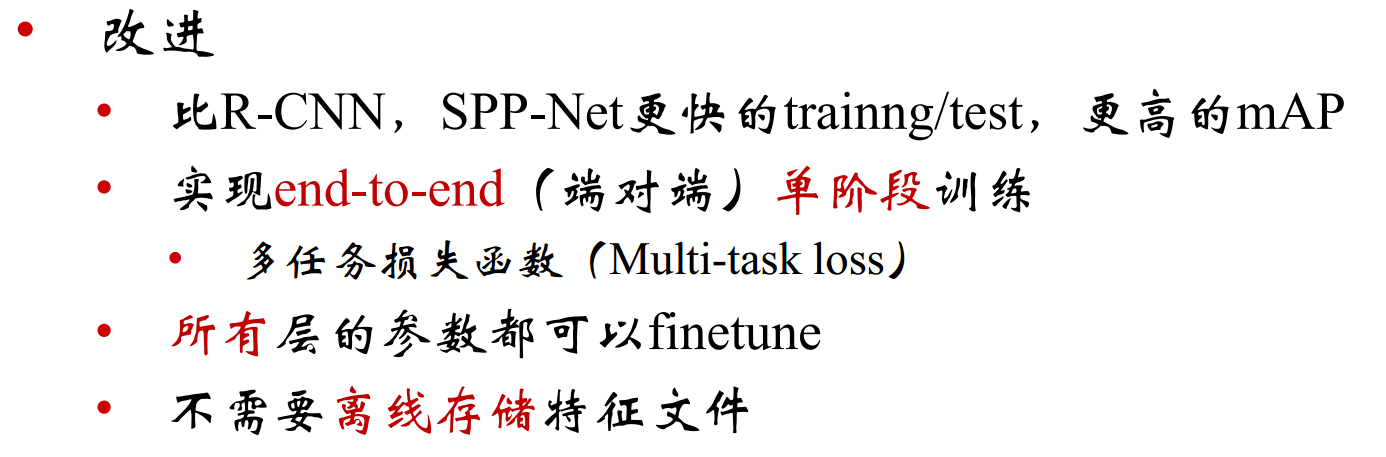
1. 在M上算SS提供的候选框的SPP特征.
2. SPPNet不是卷积. 是pooling作用, 不需要训练(没有超参数).

SPPNet问题

1. 继承R-CNN的问题
   1. 存储大量特征.
   2. **多阶段训练**
   3. 训练时间长.
2. 自身问题
   1. **SPP层之前的卷积不能fine-tune(?)**

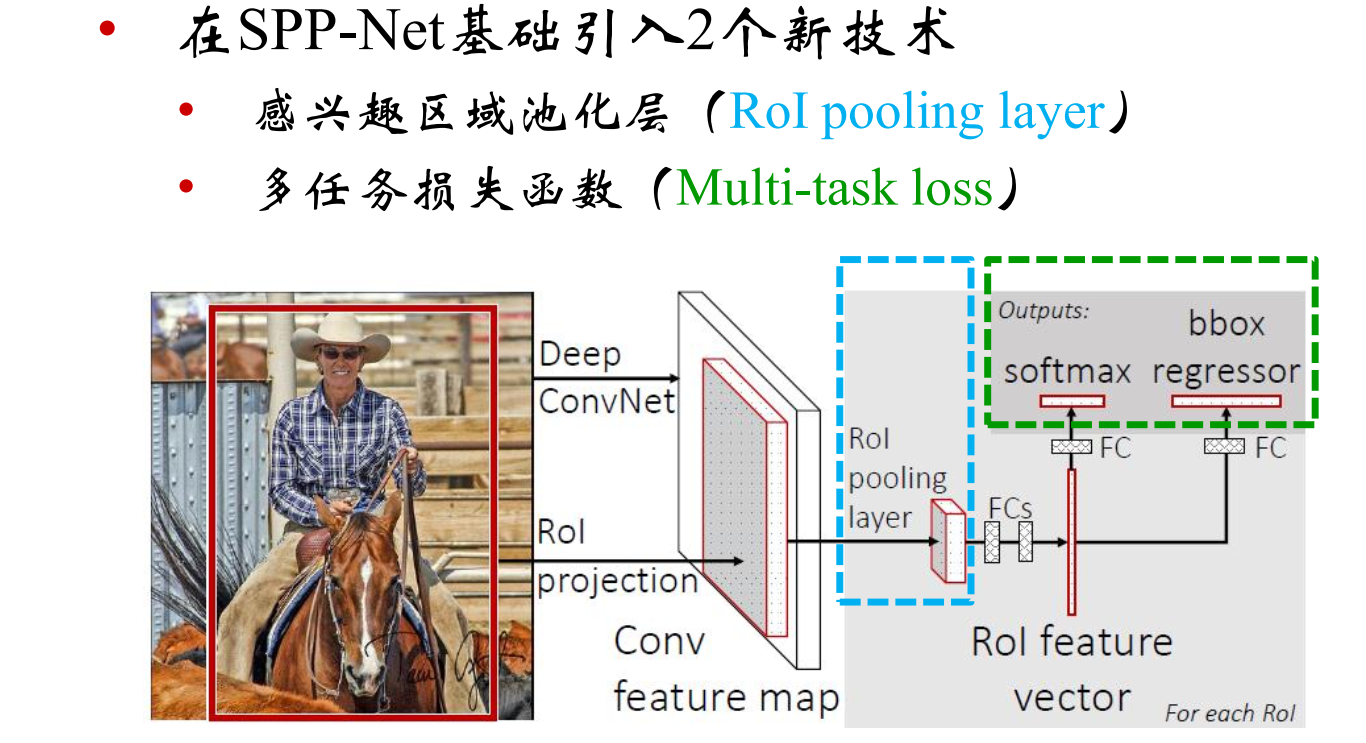
## 4.3 Fast R-CNN

### 改进

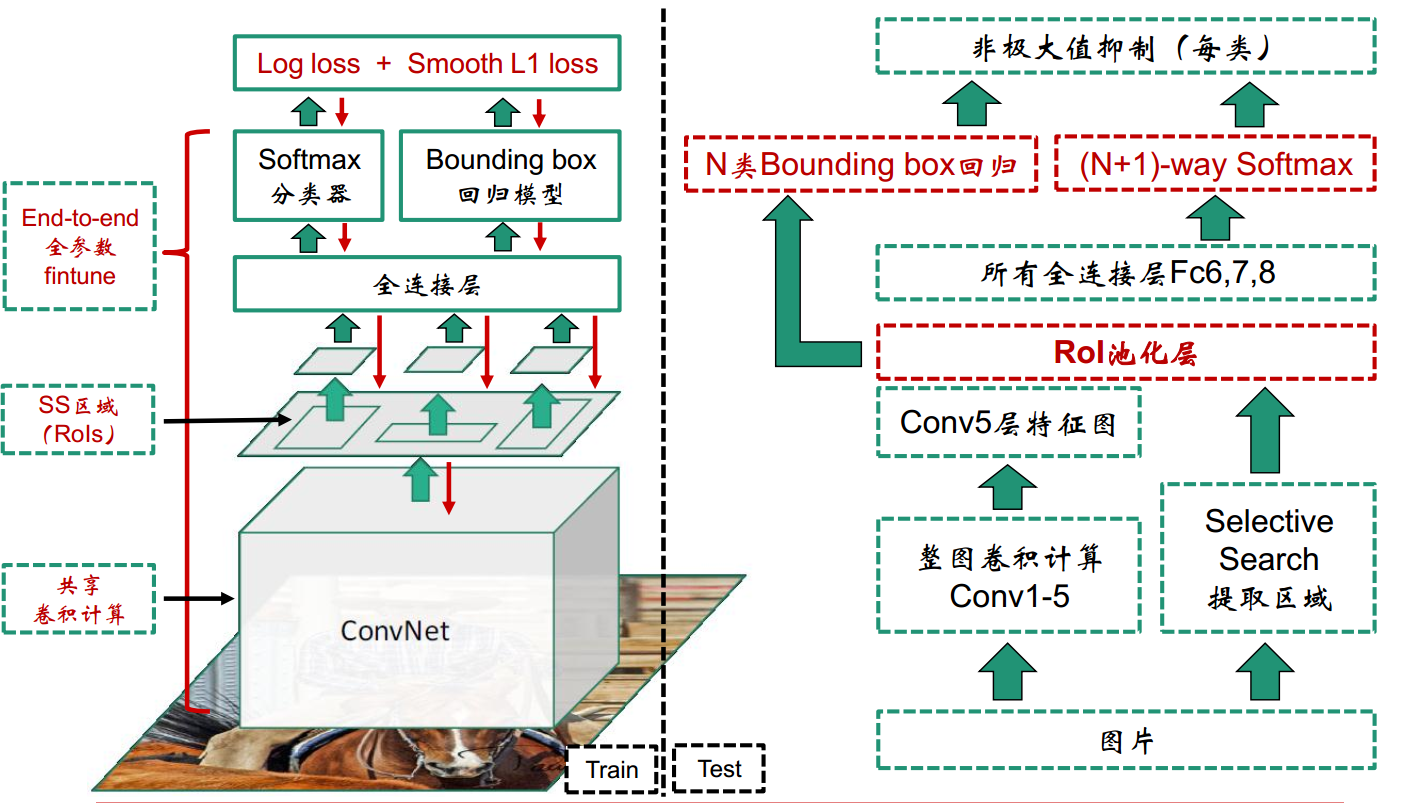


1. 去掉了SVM做分类器
2. 添加Softmax 做分类
3. 添加regressor做回归

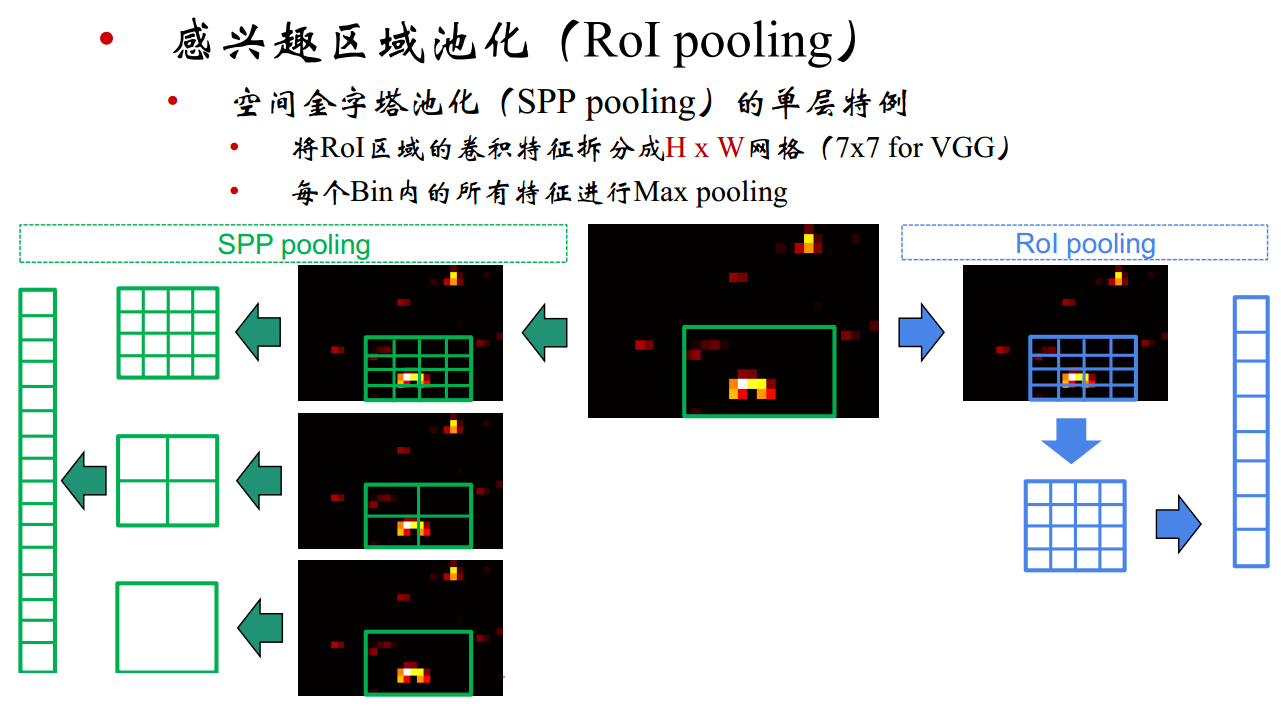
### 新功能



1. 用多任务代替了SVM

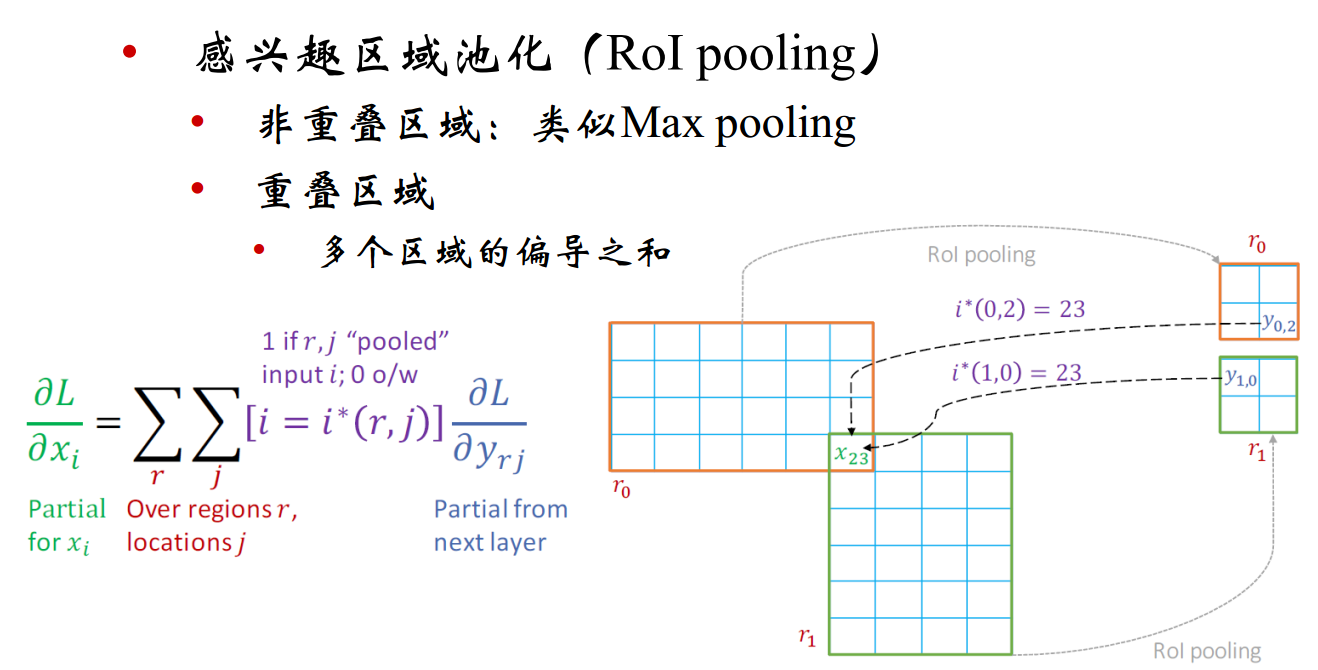


### SPP Pooling和ROI Pooling区别

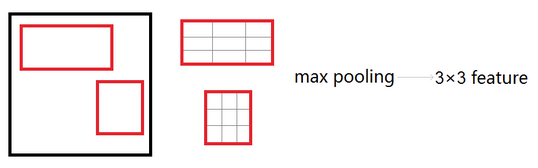


1. 同一张特征图, 左边是SPP pooling, 类似多尺度.
2. 右边是RoI pooling, 只会分成小bin操作. (参考RFCN的roi处理)

### ROI Pooling处理重叠区域



### RoI pooling层前项推断



1. 候选框大小不一. 输出是统一的.
2. 所以划分的bin大小依赖候选框大小和输出个数.

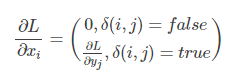
网格尺寸计算

1. 设Roi坐标为[x1,y1,x2,y2]
2. 对应要输出pooledHeight\*pooledWidth网格的输出.
3. 对应的每个小格子尺寸



### RoI Pooling的BP

非重叠区域



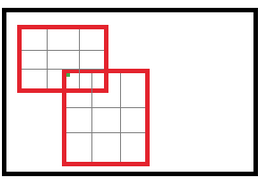
δ(i,j)表示节点i是否被区域(节点)j选为最大值.

1. 如果不被选(false值), 可能是xi不在j(节点)区域范围内.
2. 也可能是xi在j区域内,但xi不是j(节点)区域的最大值.

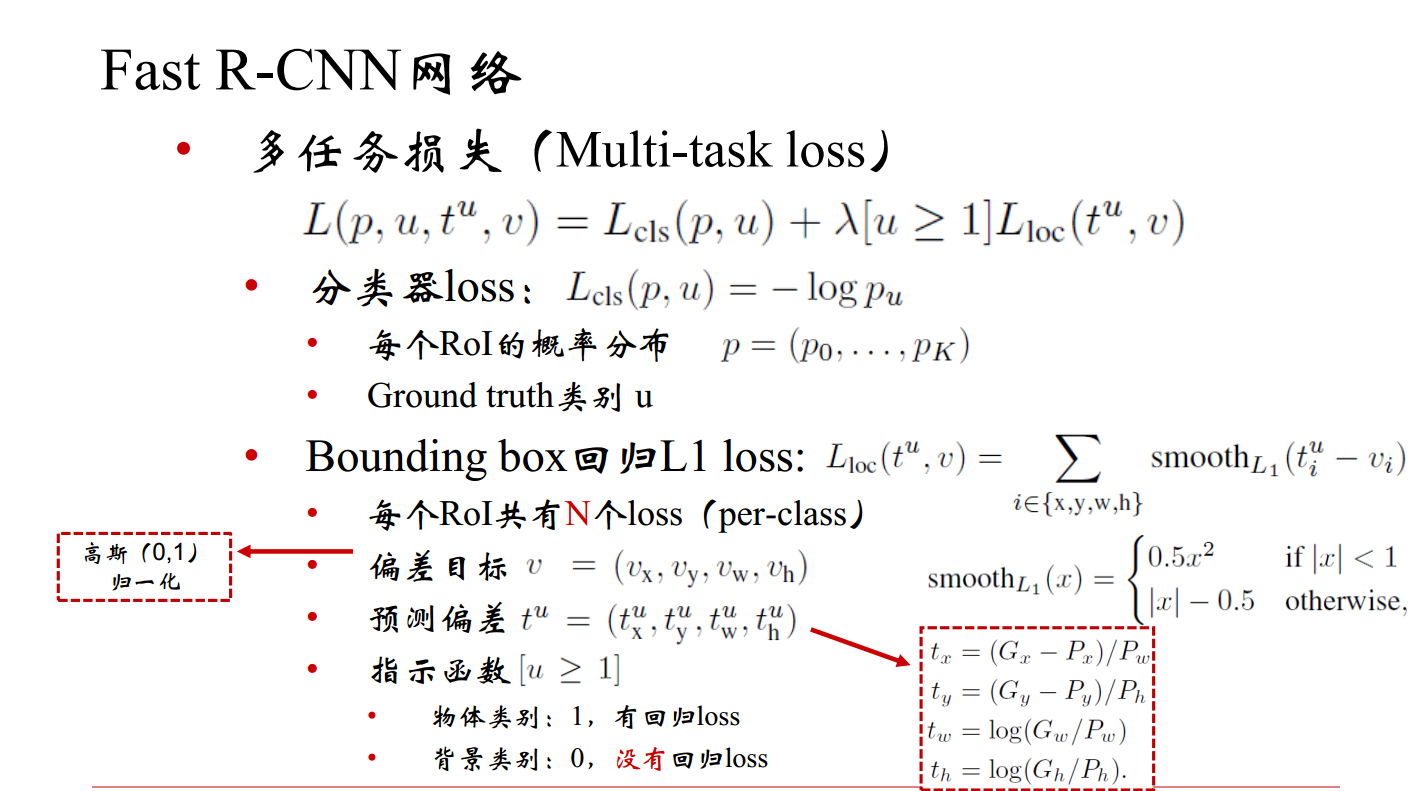
重叠区域



δ(i,r,j) 是说节点i是候选框r的j(节点)选为最大输出



### 多任务损失



### Fast R-CNN训练

