Python人工智能

讲师：覃秉丰

多任务学习

验证码识别方法一

把标签转为向量，向量长度为40。

比如有一个验证码为0782， 

它的标签可以转为长度为40的向量：1000000000 0000000100 0000000010 0010000000

训练方法跟0-9手写数字识别类似。

验证码识别方法二

拆分为4个标签

比如有一个验证码为0782， 

Label0：1000000000

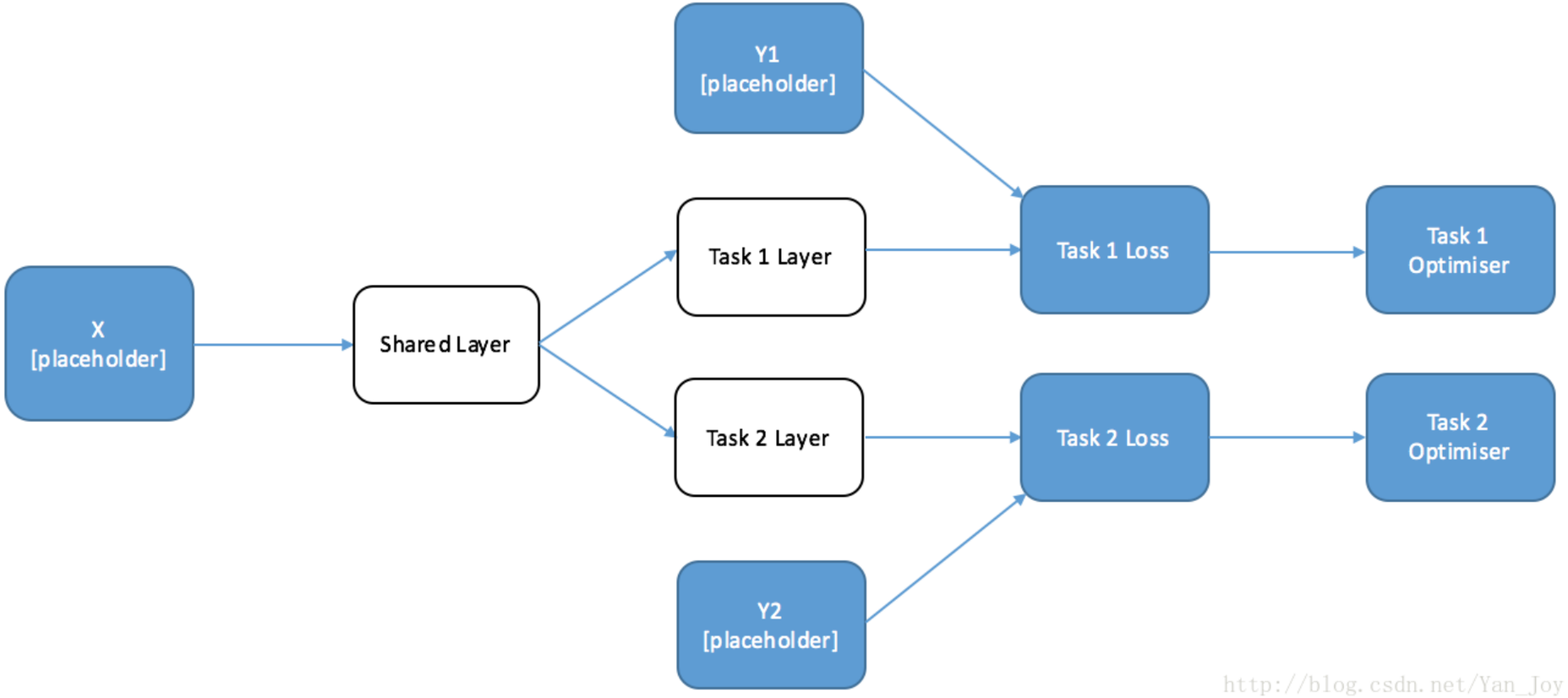
Label1：0000000100

Label2：0000000010

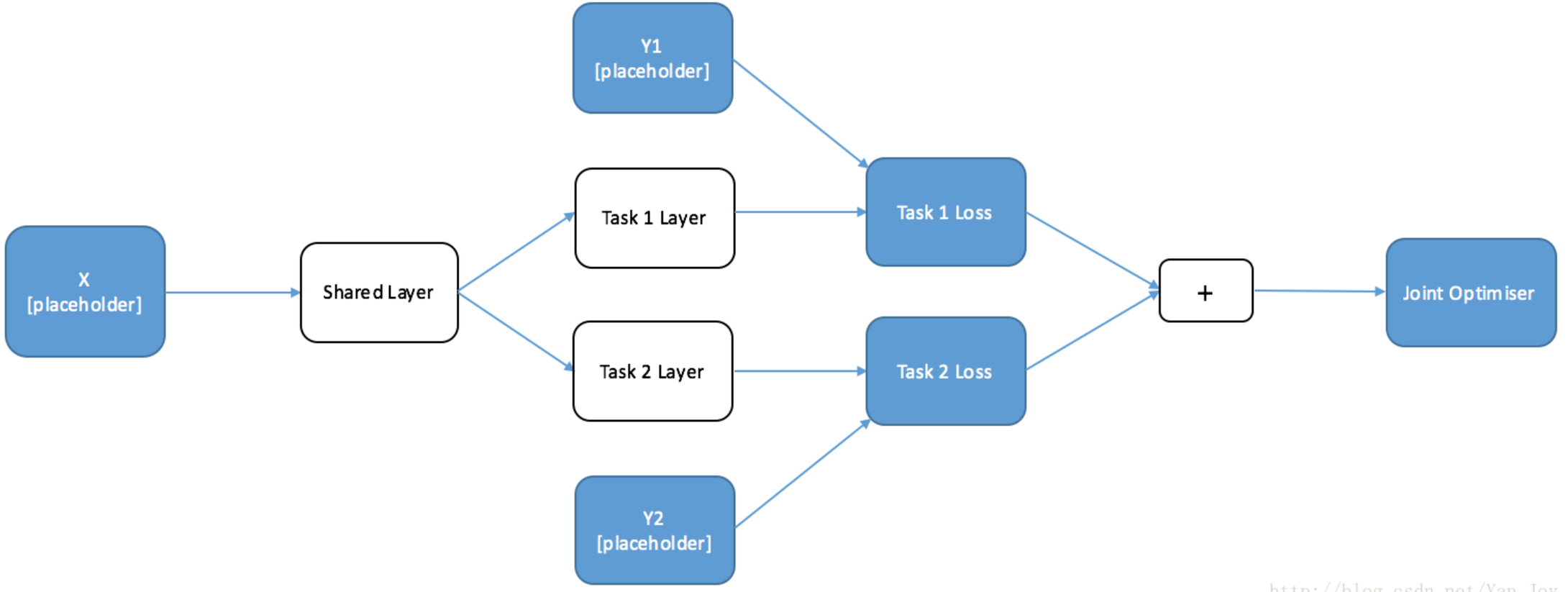
Label3：0010000000

可以使用多任务学习

Multi-task Learning - 交替训练



Multi-task Learning – 联合训练



Word2vec

word2vec

当我们分析图片或者语音的时候，我们通常都是在分析密集的，高纬度的数据集。我们所需的全部信息都储存在原始数据中。

```
[[ 0.65337904  0.96147407  0.89736144  0.97613636  0.53563182  0.65046753]
 [ 0.22471787  0.67623082  0.29457548  0.54820279  0.25811241  0.10811792]
 [ 0.15491558  0.4922566   0.94136616  0.18930393  0.43129747  0.0312585 ]
 [ 0.32249593  0.13105882  0.55929974  0.60043924  0.09488365  0.93599279]
 [ 0.18468721  0.80349133  0.77069437  0.34970681  0.04205231  0.07288426]
 [ 0.9713573   0.31079413  0.60528272  0.24704021  0.82908679  0.78950803]
 [ 0.92664684  0.77715744  0.55786552  0.85356888  0.19111345  0.20953576]
 [ 0.02344845  0.57778919  0.65908075  0.4059088   0.0907254   0.06996104]
 [ 0.72560051  0.91087261  0.66252184  0.06852047  0.56545598  0.40305866]
 [ 0.80040794  0.60398618  0.07660456  0.22238826  0.65349584  0.53116871]
 [ 0.41366496  0.30961498  0.78078967  0.21373827  0.11872793  0.13299166]
 [ 0.73777544  0.13902513  0.48004225  0.683896    0.20811546  0.30064903]
 [ 0.76508436  0.85263635  0.16590127  0.18754474  0.86105624  0.41046465]
 [ 0.37545851  0.02911257  0.27524078  0.00883495  0.53383195  0.72747815]
 [ 0.27355726  0.85399793  0.70522708  0.86964774  0.31380896  0.14360617]
 [ 0.92621366  0.81976771  0.34924696  0.11268561  0.15834104  0.25493069]
 [ 0.86482453  0.66849716  0.81176577  0.73482012  0.72419957  0.61101592]
 [ 0.65271702  0.22533039  0.25093796  0.90895525  0.56729463  0.15508486]
 [ 0.3398385   0.43496739  0.0772549   0.38408786  0.06412806  0.8306255 ]
 [ 0.63361307  0.20169828  0.36050179  0.38680661  0.63106815  0.03255401]]
```

word2vec

当我们处理自然语言问题的时候，我们通常会做分词，然后给每一个词一个编号，比如猫的编号是120，狗的编号是343。比如女生的编号是1232，女王的编号是2329。这些编号是没有规律，没有联系的，我们从编号中不能得到词与词之间的相关性。

例如：How are you？

How : 234

Are : 7

you : 987

000…1000000…

00000001000…

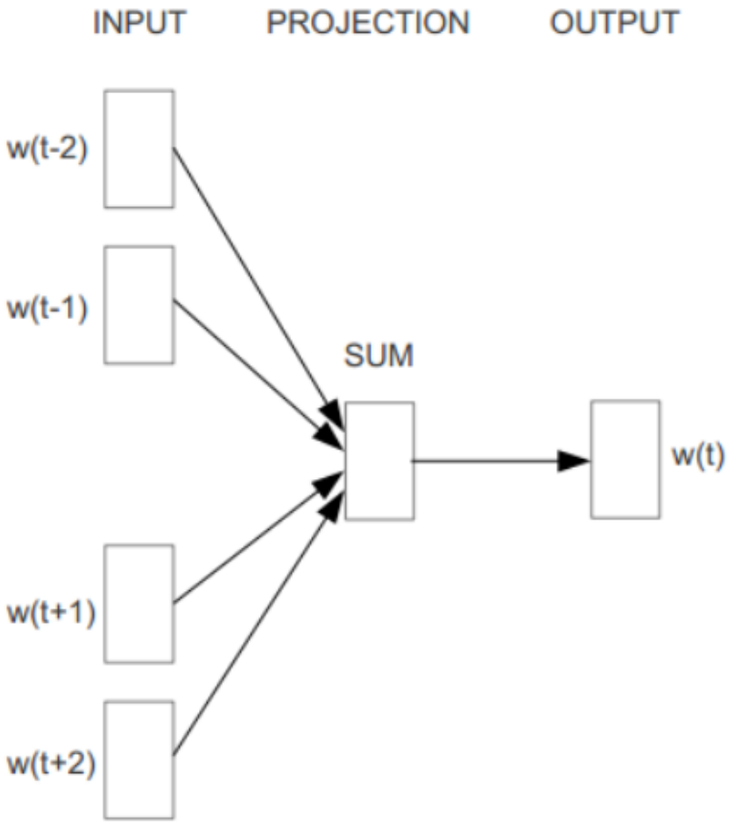
000…0000010

CBOW和Skip-Gram：

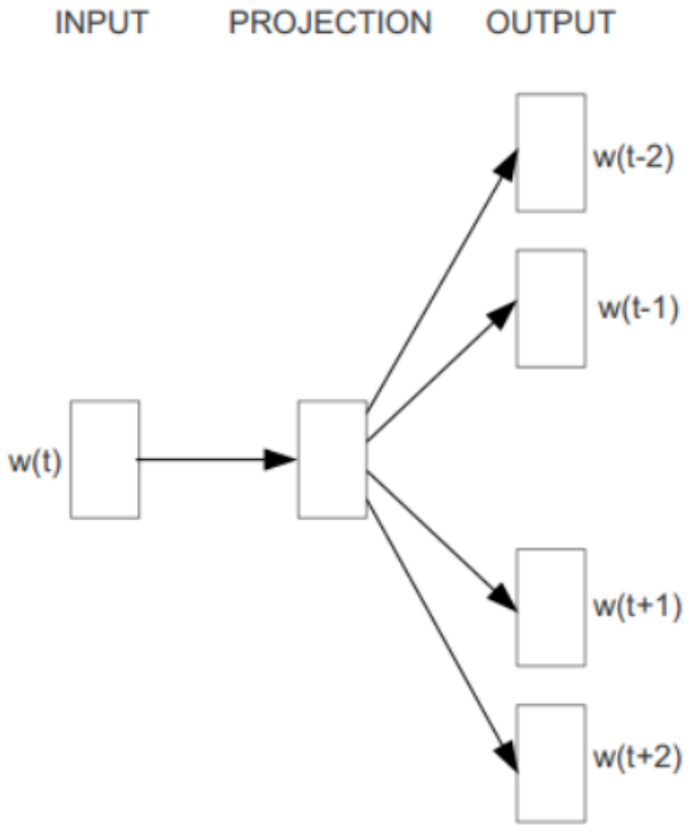
连续词袋模型（CBOW）：
根据词的上下文词汇来预测目标词汇，例如上下文词汇是“今天早餐吃__”，要预测的目标词汇可能是“面包”。

Skip-Gram模型：
Skip-Gram模型刚好和CBOW相反，它是通过目标词汇来预测上下文词汇。例如目标词汇是“早餐”
，上下文词汇可能是“今天”和“吃面包”。

CBOW和Skip-Gram：



CBOW



Skip-gram

CBOW

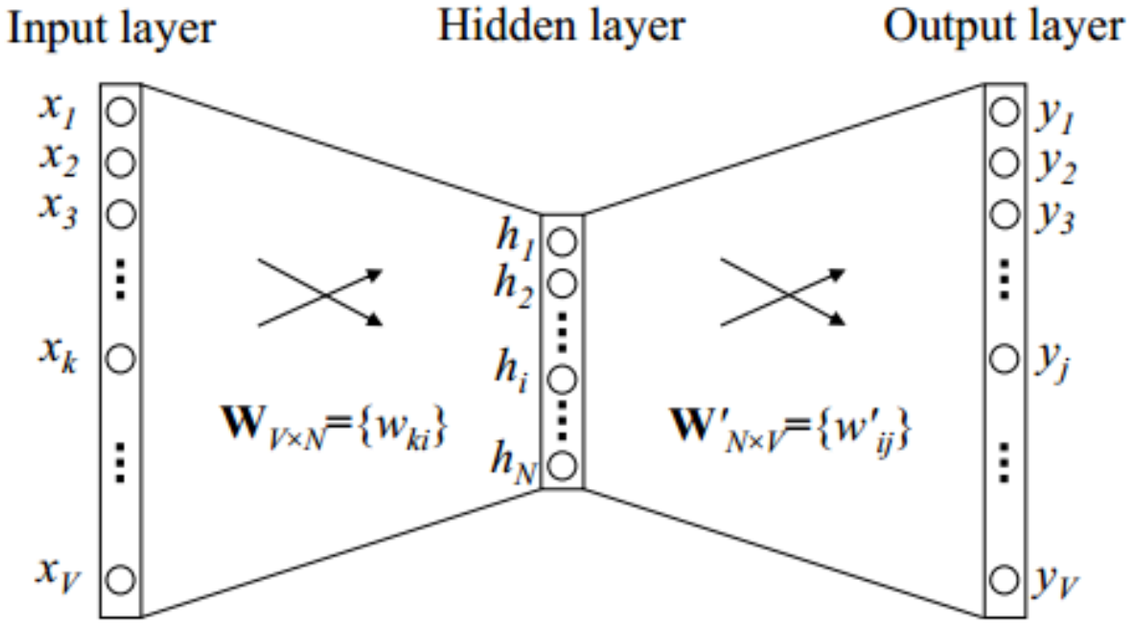
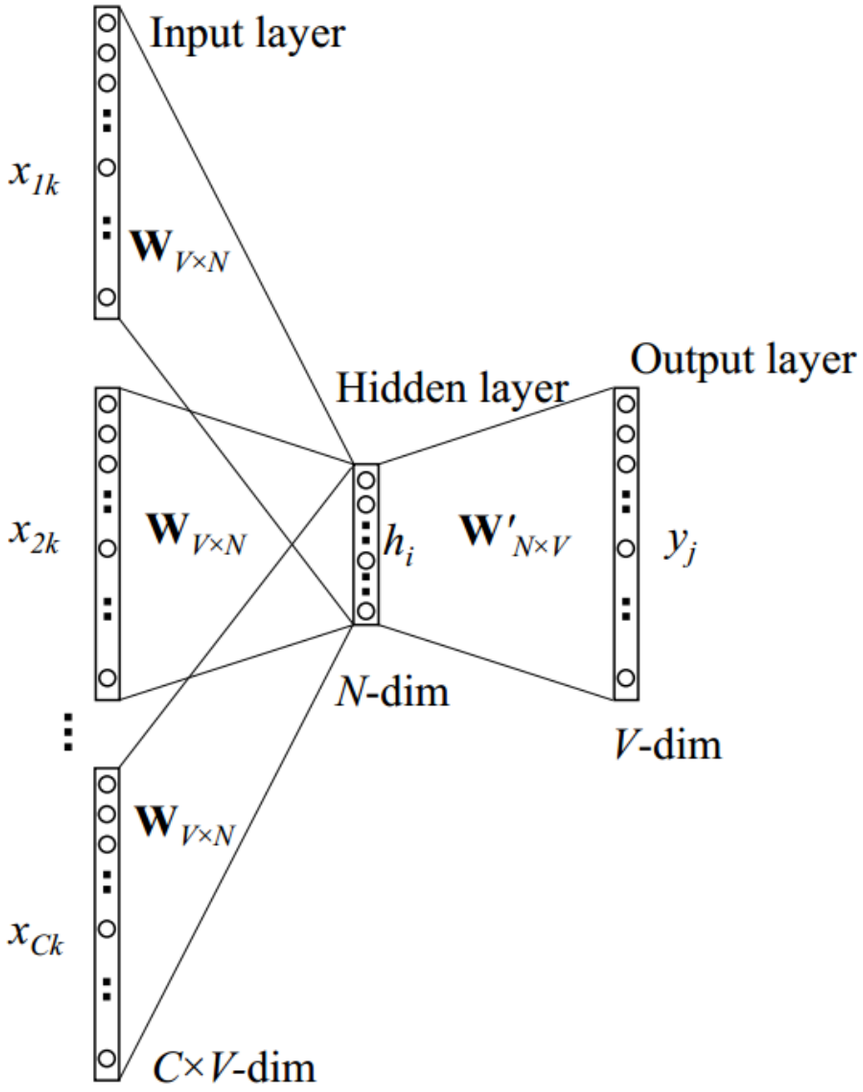
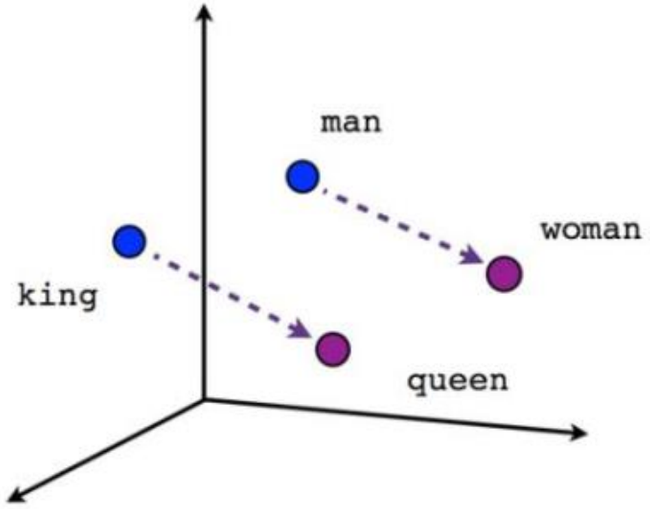


Figure 1: A simple CBOW model with only one word in the context

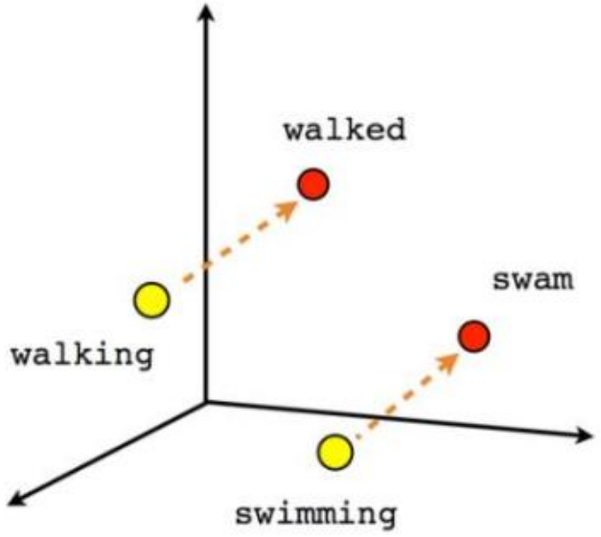
CBOW


Input layer
Hidden layer
Output layer
x_{1k}
x_{2k}
x_{Ck}
$\mathbf{W}_{V\times N}$
$\mathbf{W}_{V\times N}$
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
y_j
N-dim
V-dim
$C\times V$-dim

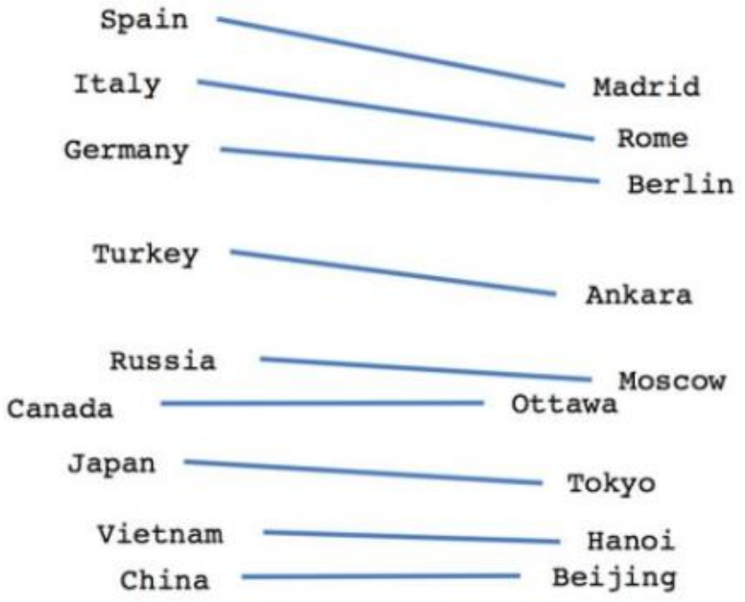
Word2vec结果可视化



Male-Female



Verb tense



Country-Capital

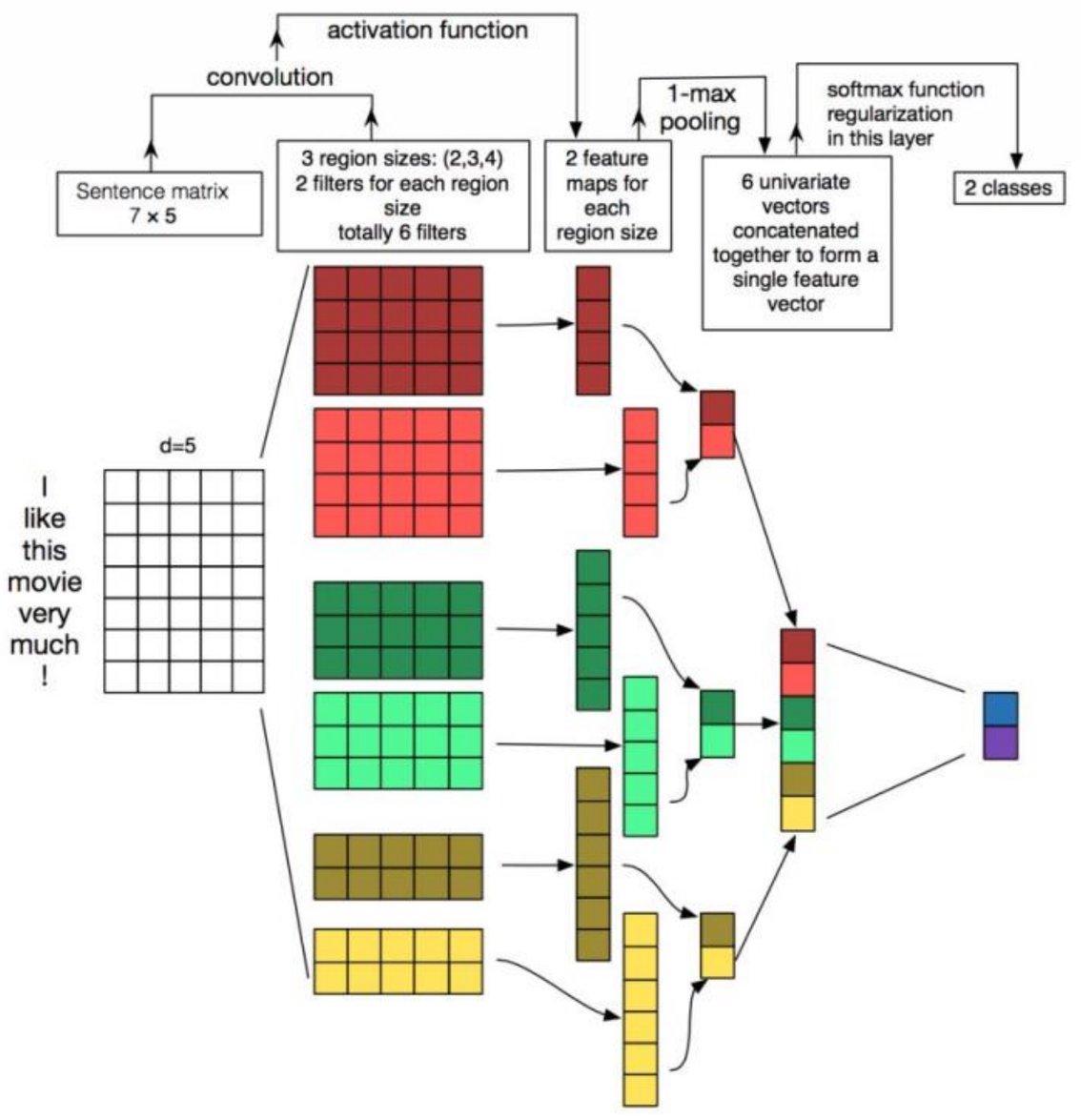
CNN在自然语言处理中的应用：

CNN在自然语言处理中的应用：

说到CNN我们首先可能会想到CNN在计算机视觉中的应用。近几年CNN也开始应用于自然语言处理，并取得了一些引人注目的成绩。

CNN应用于NLP的任务，处理的往往是以矩阵形式表达的句子或文本。矩阵中的每一行对应于一个分词元素，一般是一个单词，也可以是一个字符。也就是说每一行都是一个词或者字符的向量（比如前面说到的word2vec）。假设我们一共有10个词，每个词都用128维的向量来表示，那么我们就可以得到一个10×128维的矩阵。这个矩阵就相当于是一副“图像”。

CNN在自然语言处理中的应用：



目标检测
Object Detection

目标检测

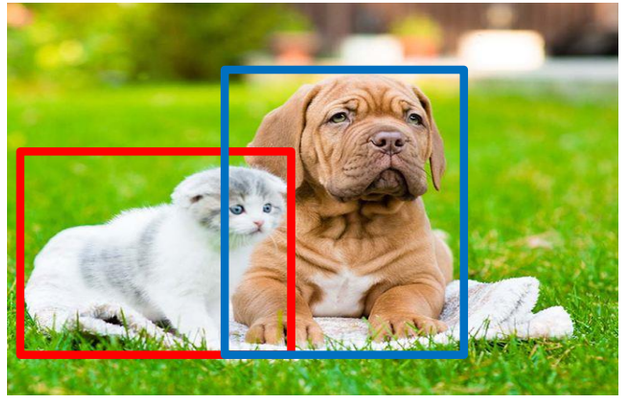
图像识别

猫

目标检测

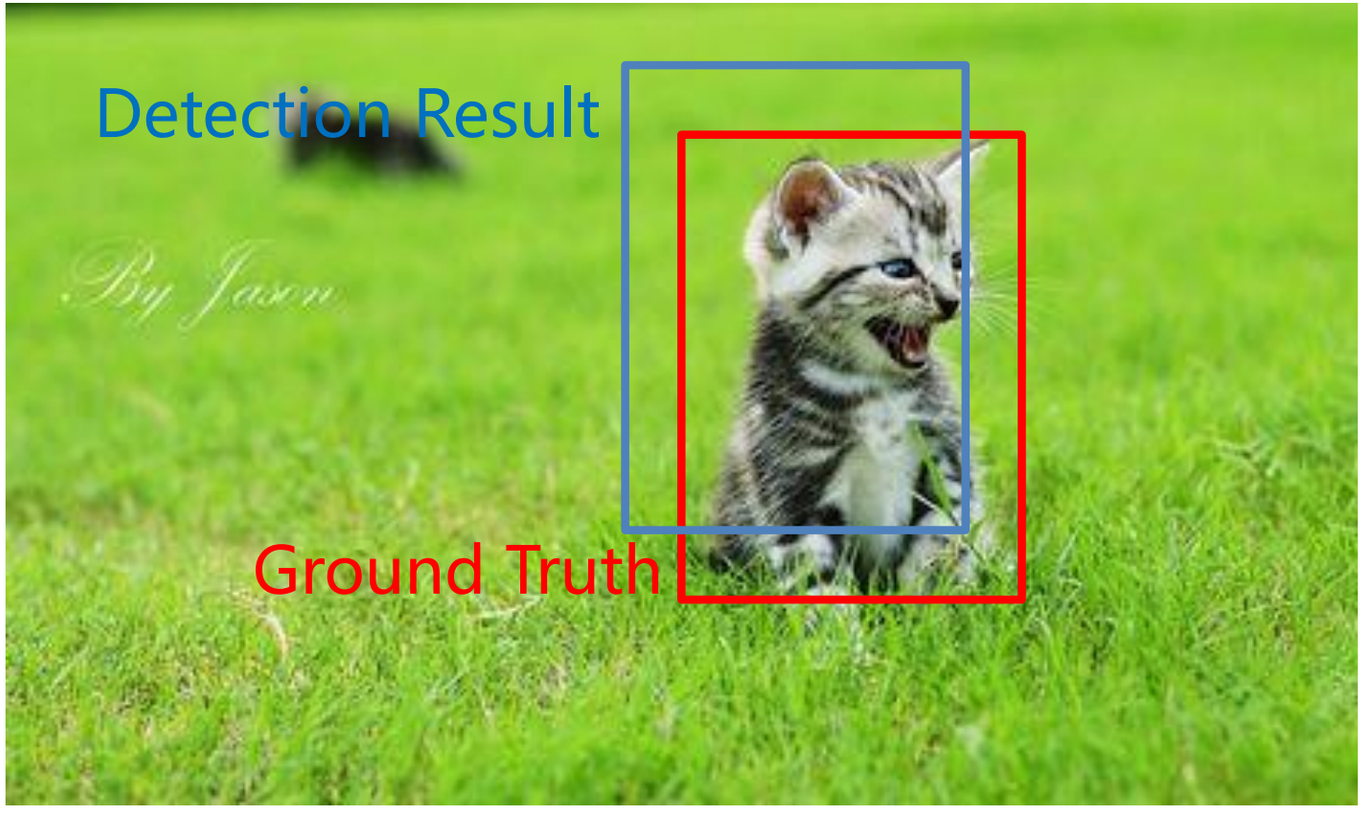
猫

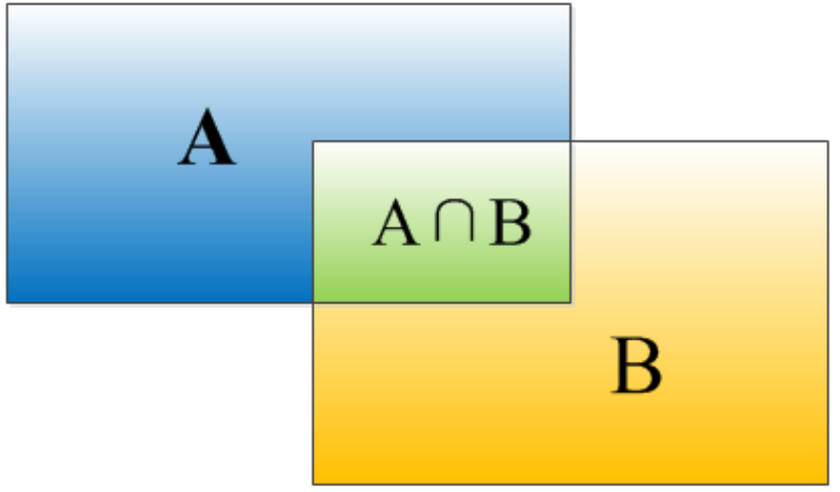
多目标检测



猫，狗

交并比IoU(intersection-over-union)

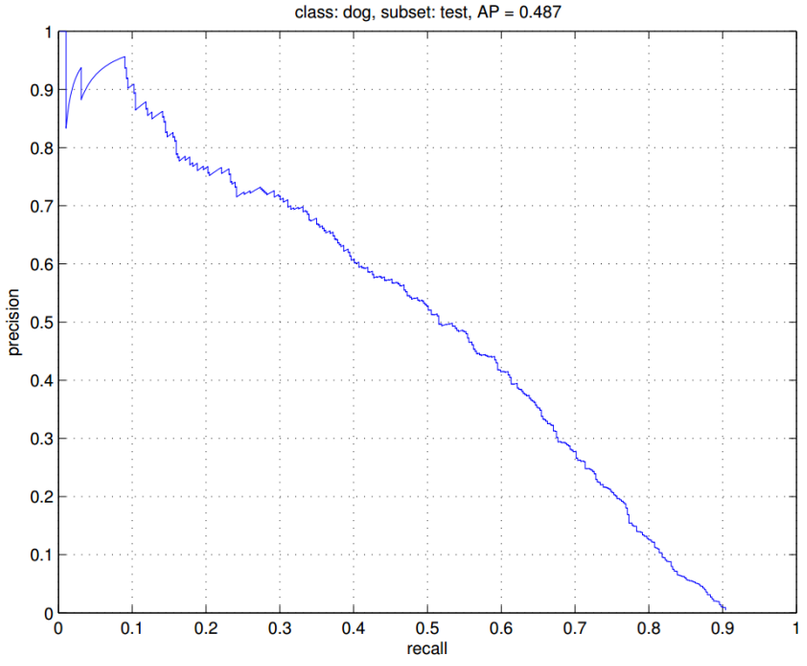


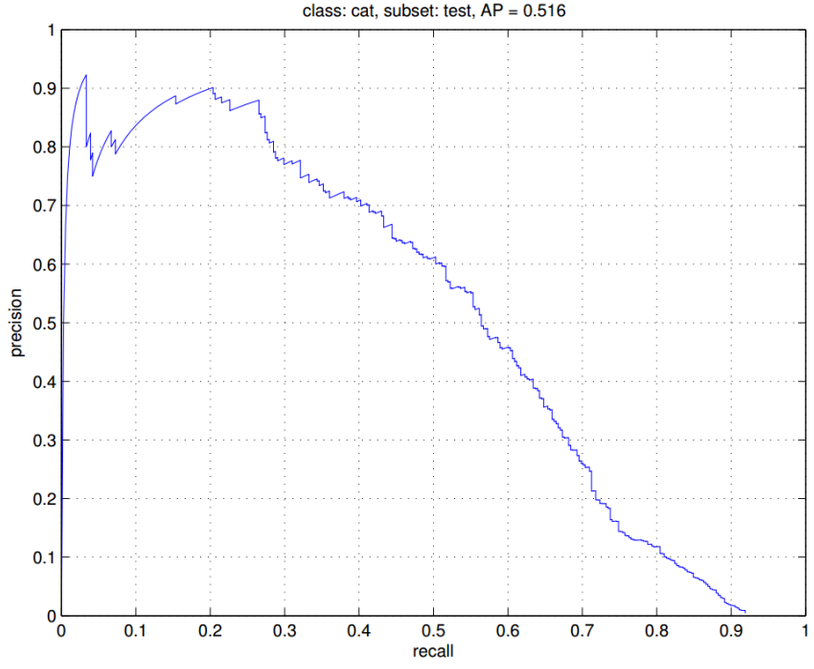


$$IOU = \frac{DetectionResult \bigcap GroundTruth}{DetectionResult \bigcup GroundTruth}$$

一般可以设置IoU的阈值大于等于0.5
比如把IoU阈值设置为0.5，则可以认为与Ground Truth的IoU大于等于0.5的Detection Result是正样本

mAP(mean average precision)平均准确率均值





每一个类别都可以根据recall和precision绘制一条曲线，那么AP就是该曲线下的面积，而mAP是多个类别AP的平均值，这个值介于0到1之间，且越大越好。这个指标是目标检测算法最为重要的一个。

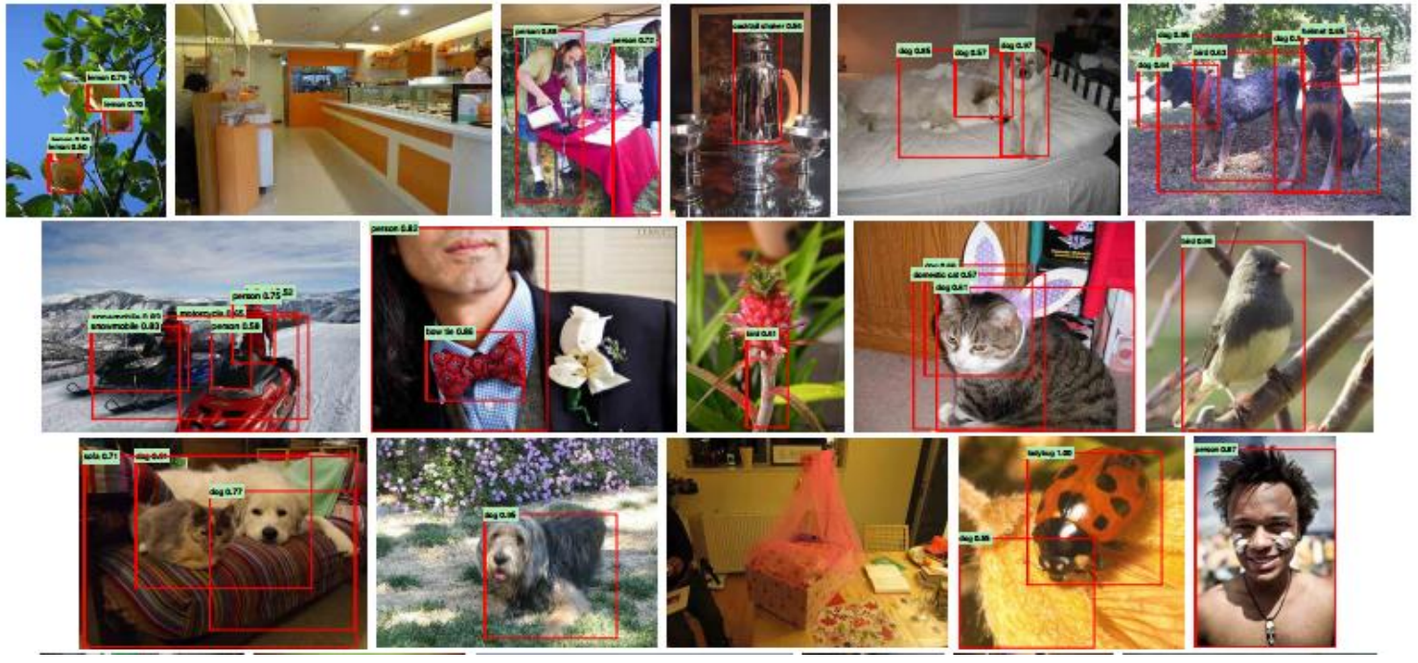
Ross B. Girshick

这一些系列的工作都是Ross B. Girshick(RBG)完成的，他博士后毕业于加州大学伯克利分校，在微软研究院做了一年研究员，现在是一名Facebook AI实验室的研究员。他不仅学术能力强，工程能力也是一流。

R-CNN

RCNN

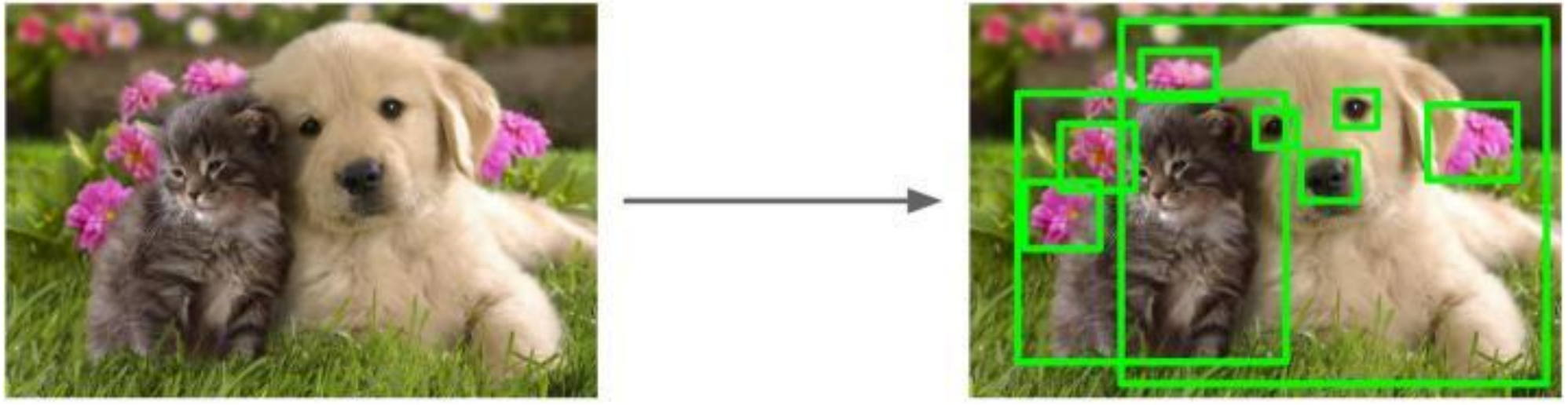
RCNN（Regions with CNN features）是RBG在2014年提出的一种目标检测算法，RCNN是将CNN方法应用到目标检测问题上的一个里程碑，借助CNN良好的特征提取和分类性能，通过RegionProposal方法实现目标检测。
前面我们提到的滑动窗口法可以得到目标所在的区域，但是会产生大量的计算。除了滑动窗口法之外还有另外一类基于区域(Region Proposal)的方法，selective search就是其中之一。



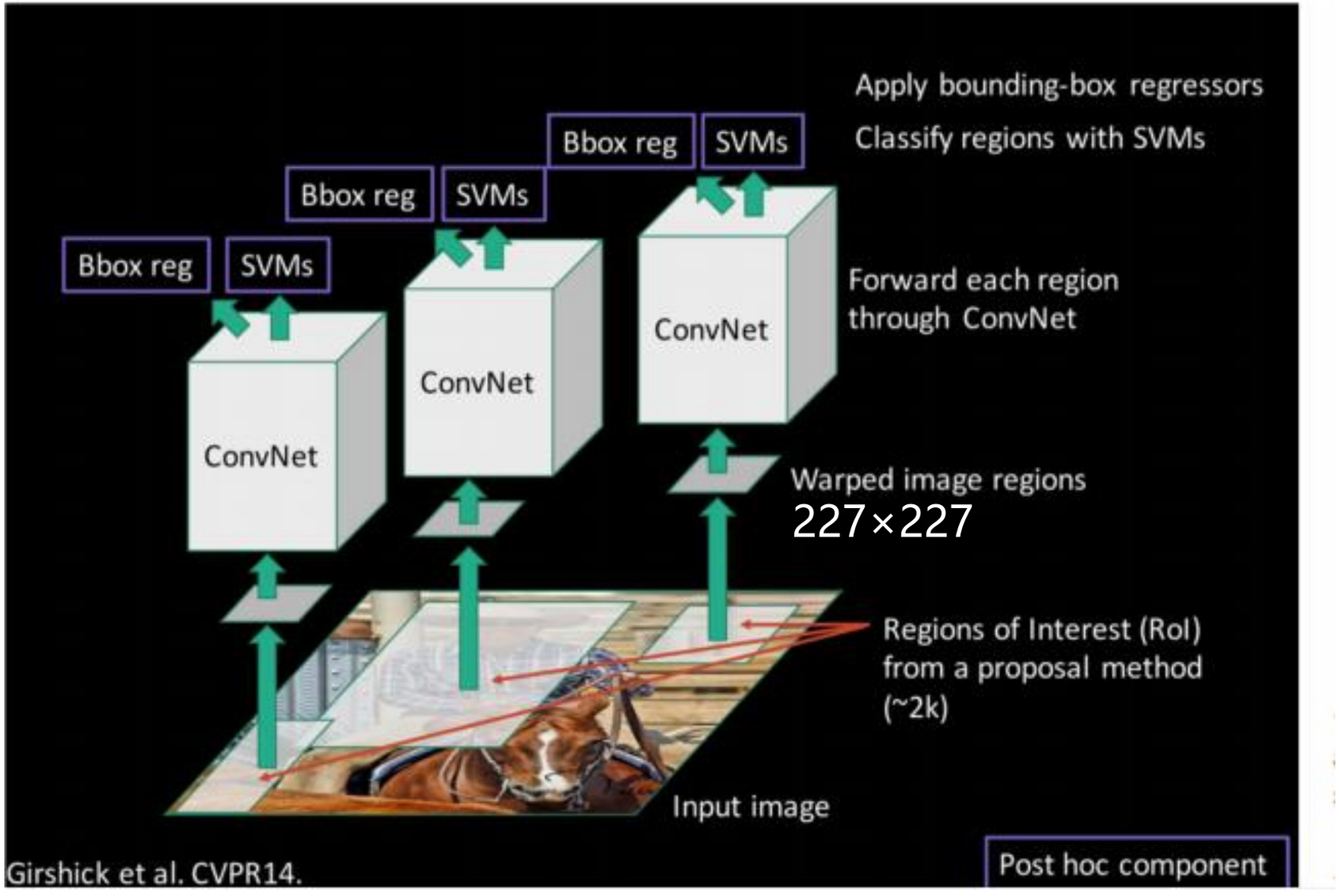
selective search

step1:计算区域集R里每个相邻区域的相似度S={s1,s2,...}
While S ≠∅：
step2:找出相似度最高的两个区域，将其合并为新集Rt，添加进R
step3:从S中移除所有与step2中有关的子集
step4:重新计算新集Rt与所有子集的相似度

这里的相似度主要是考虑颜色，纹理，尺寸，交叠四个方面

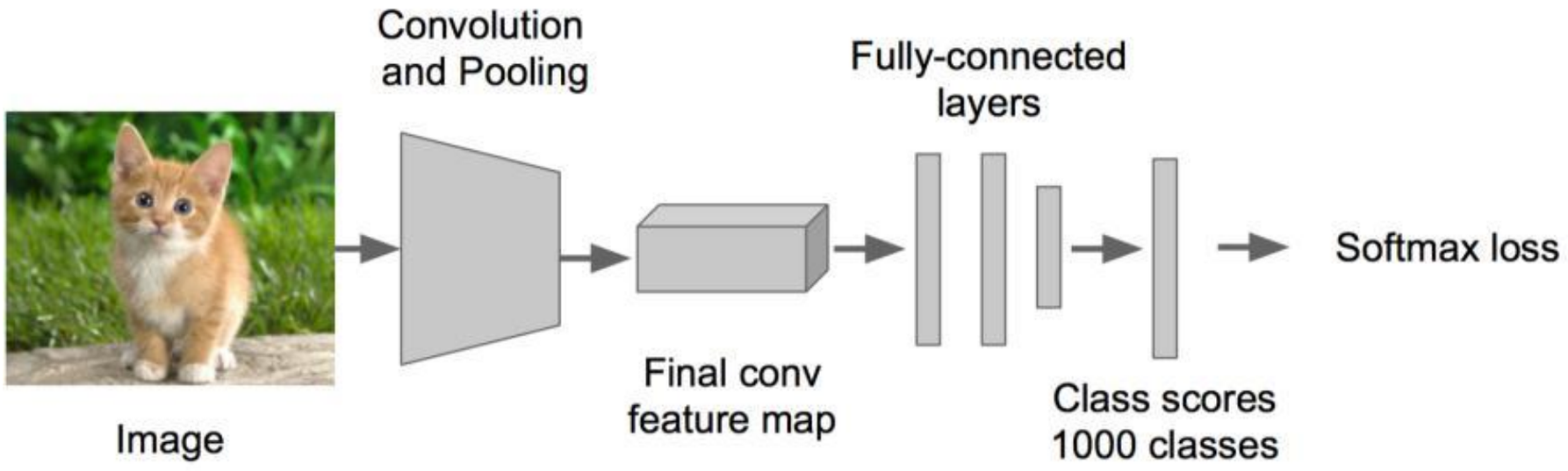


算法整体流程


Apply bounding-box regressors
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Forward each region through ConvNet
ConvNet
ConvNet
ConvNet
Warped image regions
227×227
Regions of Interest (RoI) from a proposal method (~2k)
Input image
Girshick et al. CVPR14.
Post hoc component

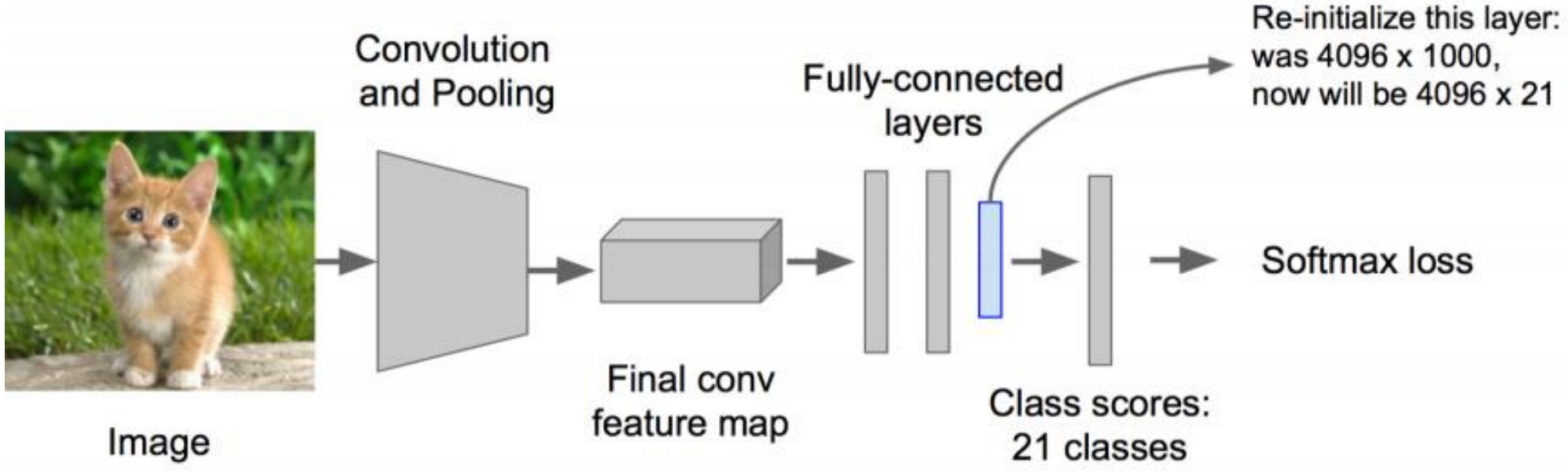
算法流程1

选择一个分类模型（比如AlexNet,VGGNet等）



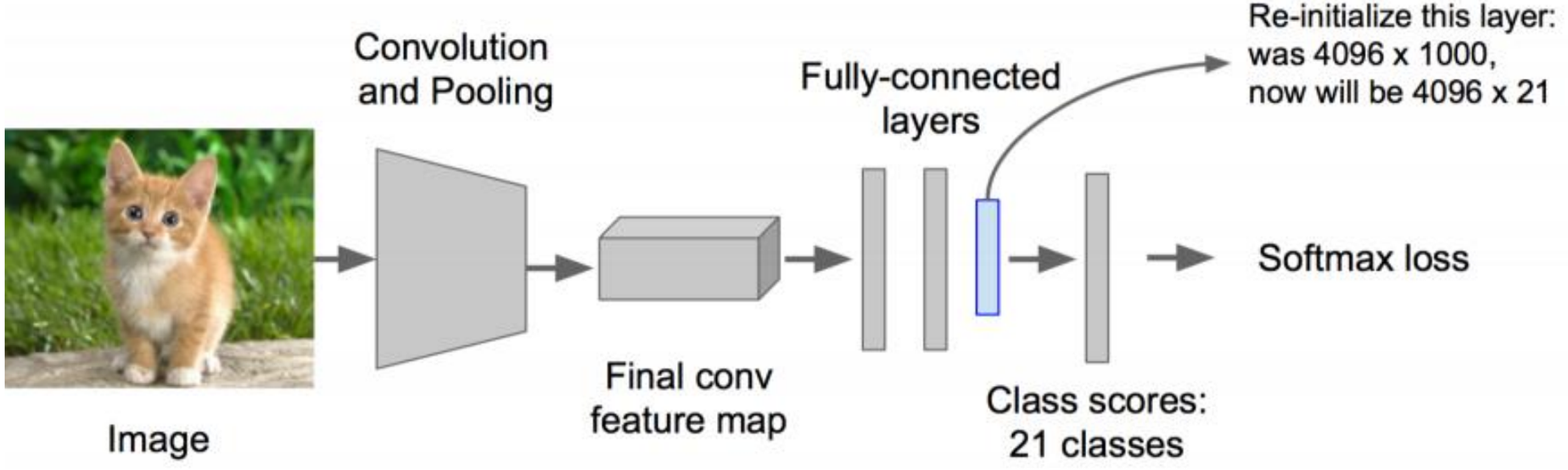
算法流程2

1.去掉最后一个全连接层。
2.将分类数从1000改为(N+1)。对于VOC，N=20；对于ILSVRC2013，N=200。
3.对该模型做fine-tuning。(主要目的是优化卷积层和池化层的参数)



算法流程2

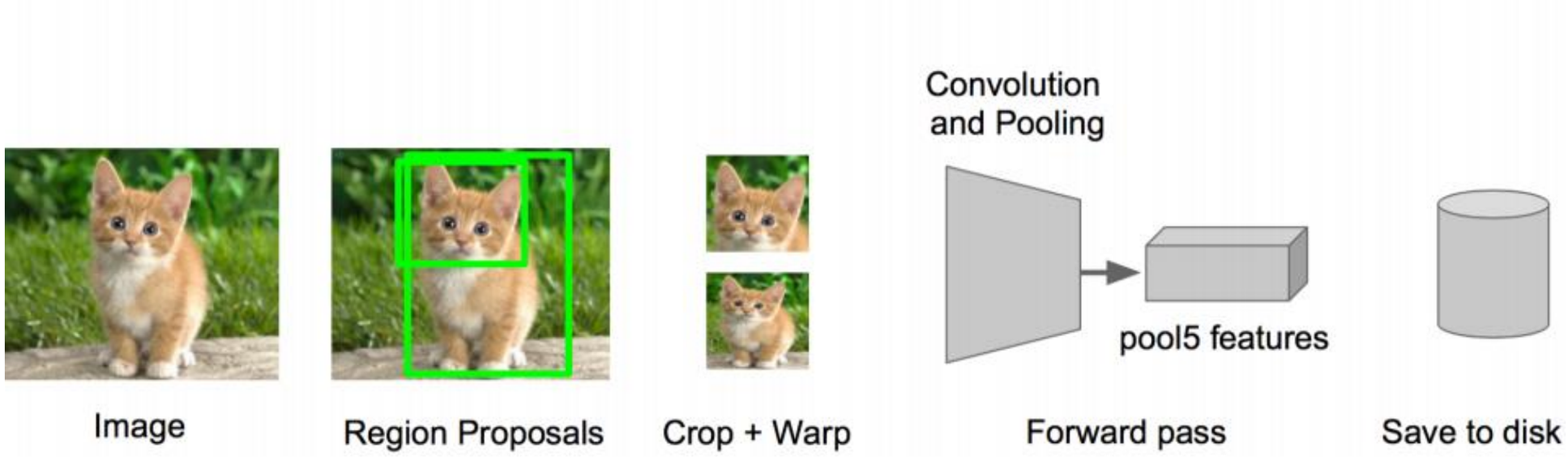
算法细节：
1.学习率是0.001(是初始化预训练时的1/10)
2.对所有的候选区域，如果其和真实标注的IoU≥0.5就认为是正例，否则就是负例(背景)。
3.训练一个batch大小是128，其中包含32个正例(可能包含所有种类)和96个背景窗口。



算法流程3

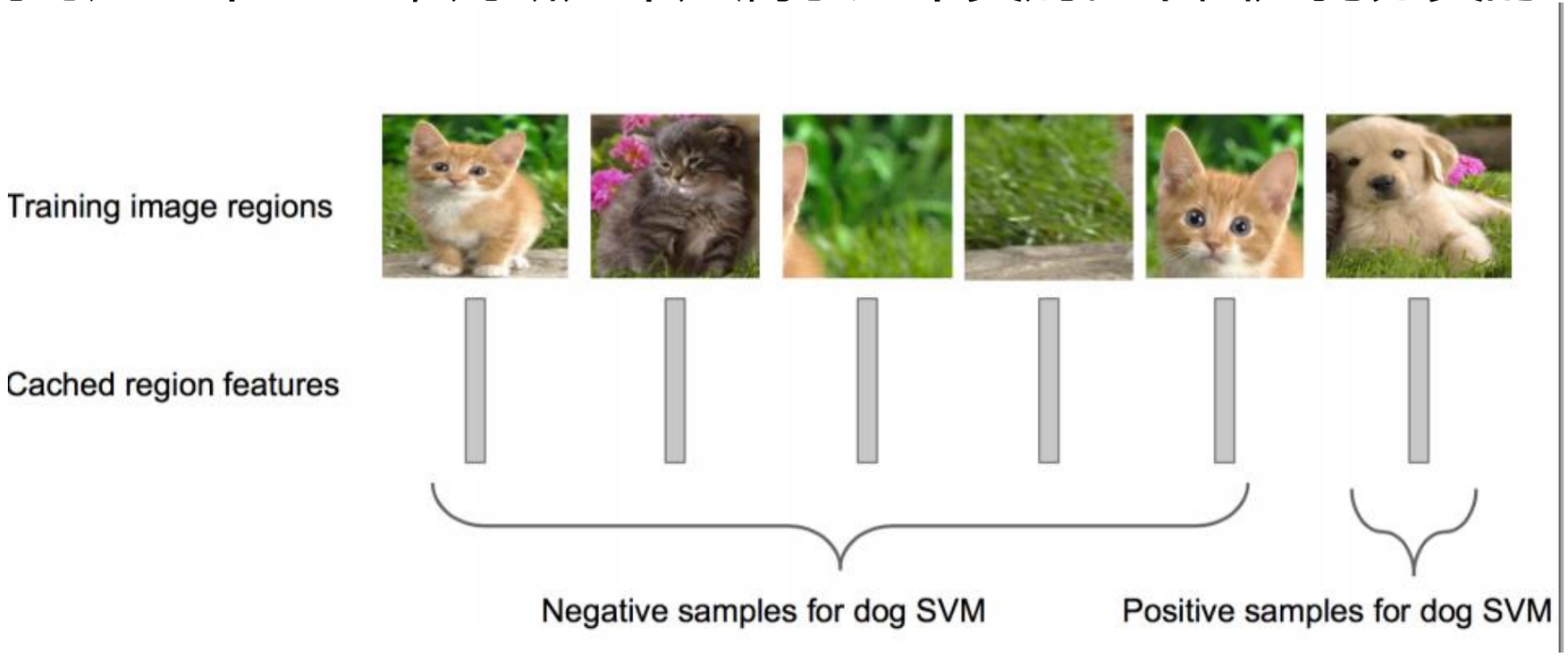
论文中用到的网络pool5后面的一层是fc6全连接层。

对每一个候选区域进行特征提取：
Resize区域大小，然后做一次前向运算，将第五个池化层的输出(也就是候选框提取到的特征)保存到硬盘。



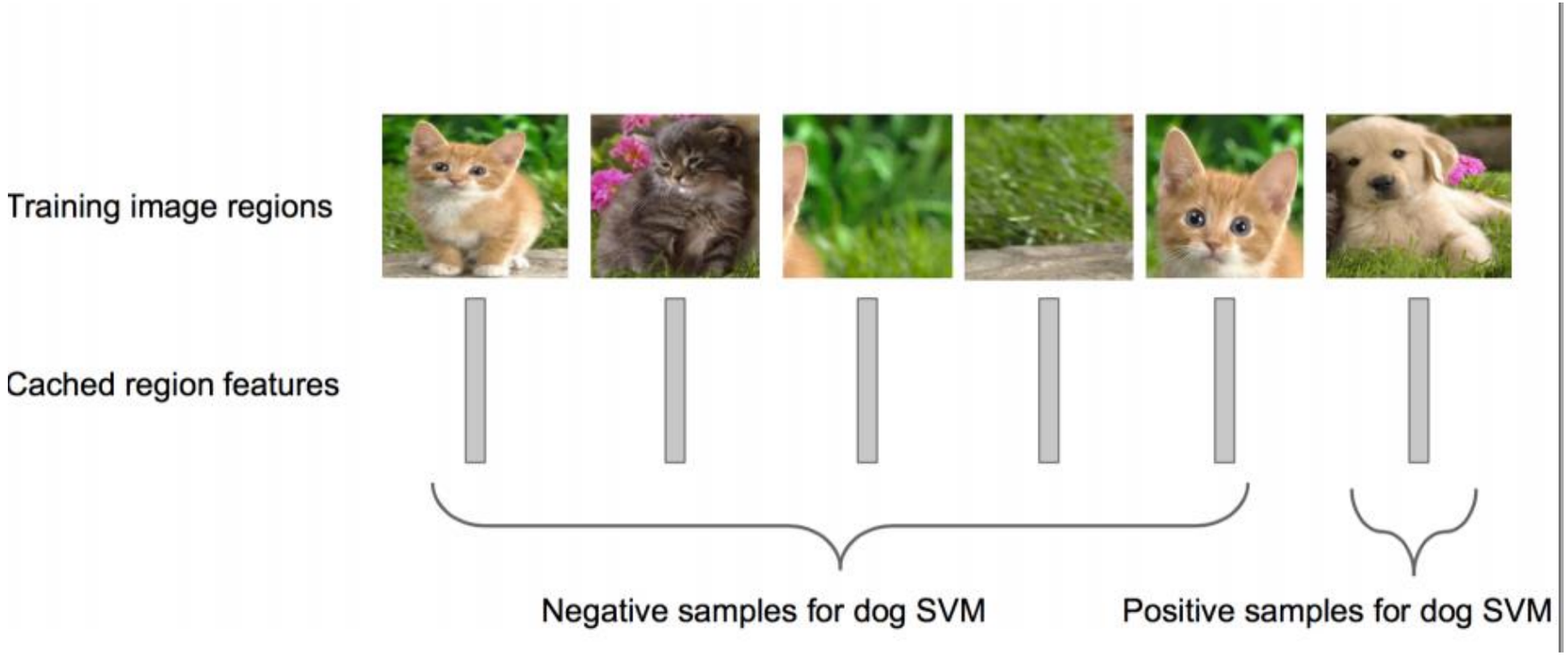
算法流程4

训练阶段：
使用pool5输出的图像特征训练SVM分类器(二分类)来判断这个候选框里面的物体类别。
测试阶段：
每个类别对应一个SVM，判断是不是属于这个类别。下图是狗分类的SVM：



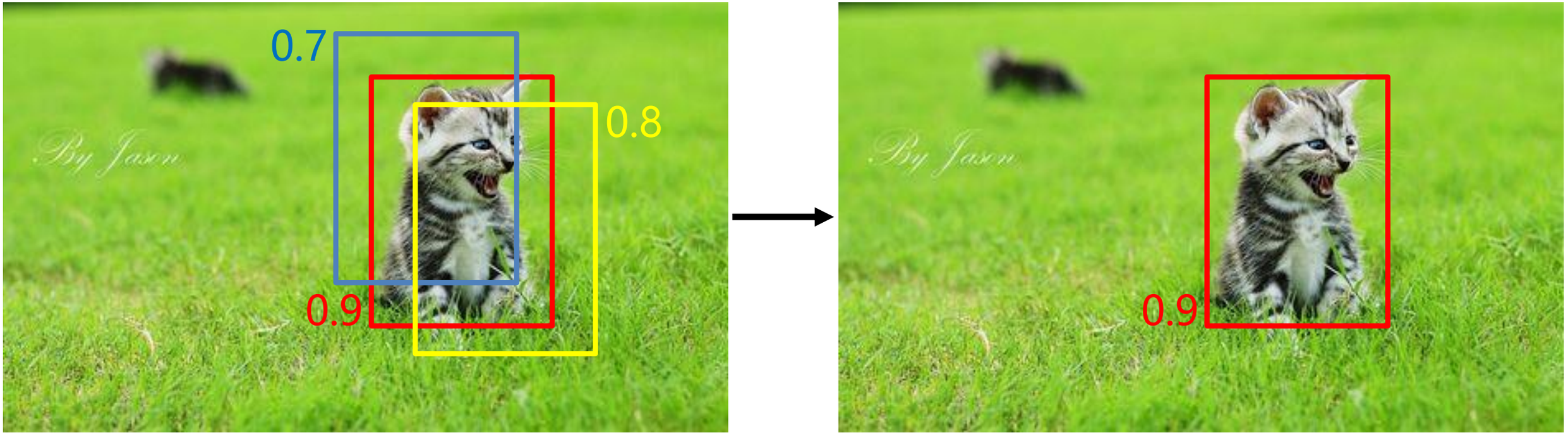
算法流程4

算法细节：
候选区域与真实框选区域的IoU大于0.7时认为是正样本，小于0.3时认为是负样本，鉴于0.3-0.7之间的丢掉不用来训练。(作者尝试了各种IoU的阈值，最后测试出这个值得到的mAP比较高)



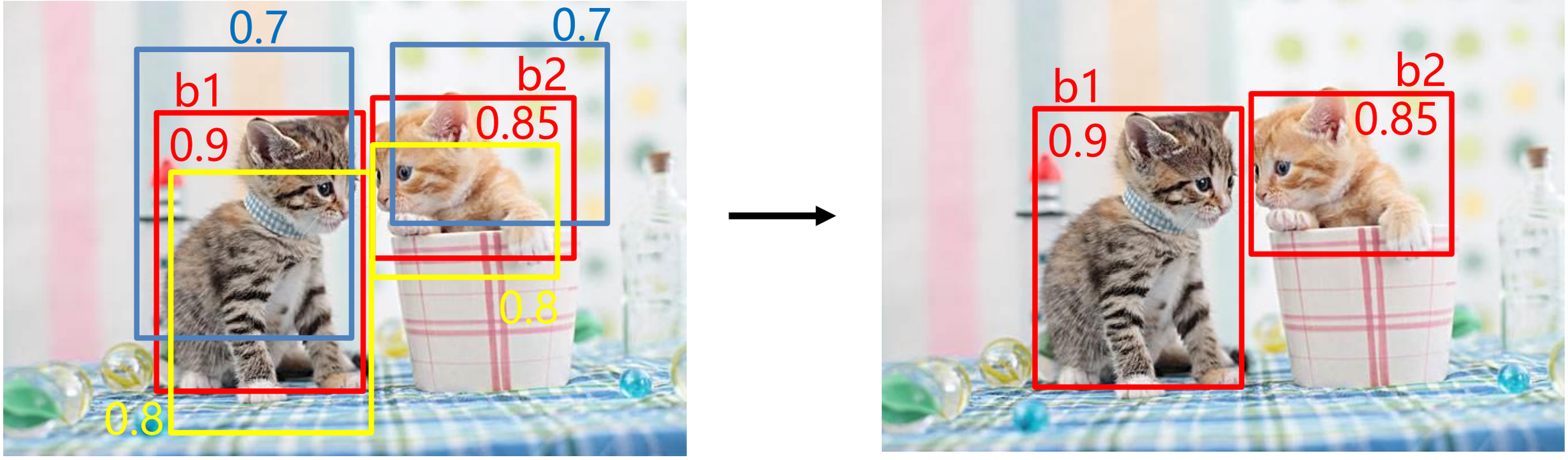
算法流程4-非极大值抑制(Non-max suppression)

选出置信度最高的候选框，如果和当前最高分的候选框重叠面积IoU大于一定阈值，就将其删除。



算法流程4-非极大值抑制(Non-max suppression)

当存在多预测目标时，先选取置信度最大的候选框b1，然后根据IoU阈值来去除b1候选框周围的候选框。然后再选取置信度第二大的候选框b2，然后去除b2候选框周围的候选框。



算法流程5

训练阶段：
使用pool5输出的图像特征训练一个回归器(dx,dy,dw,dh)。
dx代表水平平移，dy代表竖直平移，dw代表宽度缩放，dh代表高度缩放
测试阶段：
使用回归器调整候选框位置。

RCNN-mAP

R-CNN和OverFeat检测系统在ILSVRC2013的200分类检测数据集上对比。R-CNN表现更加卓越，mAP达到31.4%，大大超过了OverFeat的24.3%的表现。

test set	val_2	val_2	val_2	val_2	val_2	val_2	test	test
SVM training set	val_1	$val_1+train_{.5k}$	$val_1+train_{1k}$	$val_1+train_{1k}$	$val_1+train_{1k}$	$val_1+train_{1k}$	$val+train_{1k}$	$val+train_{1k}$
CNN fine-tuning set	n/a	n/a	n/a	val_1	$val_1+train_{1k}$	$val_1+train_{1k}$	$val_1+train_{1k}$	$val_1+train_{1k}$
bbox reg set	n/a	n/a	n/a	n/a	n/a	val_1	n/a	val
CNN feature layer	fc_6	fc_6	fc_6	fc_7	fc_7	fc_7	fc_7	fc_7
mAP	20.9	24.1	24.1	26.5	29.7	**31.0**	30.2	**31.4**
median AP	17.7	21.0	21.4	24.8	29.2	**29.6**	29.0	**30.3**

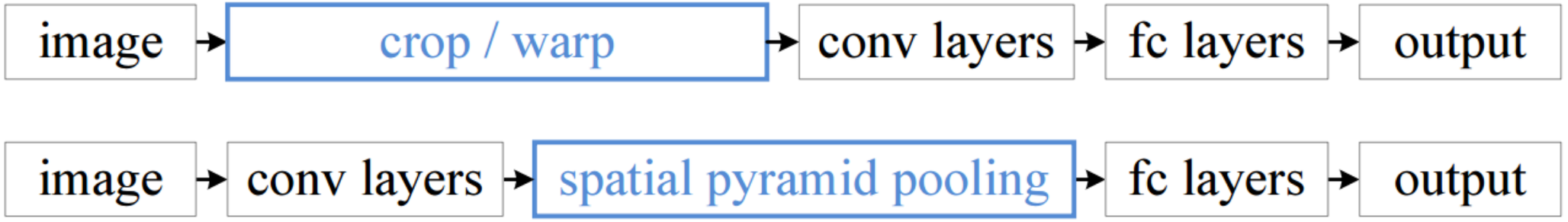
Table 4: ILSVRC2013 ablation study of data usage choices, fine-tuning, and bounding-box regression.

SPP-Net
Spatial Pyramid Pooling Net

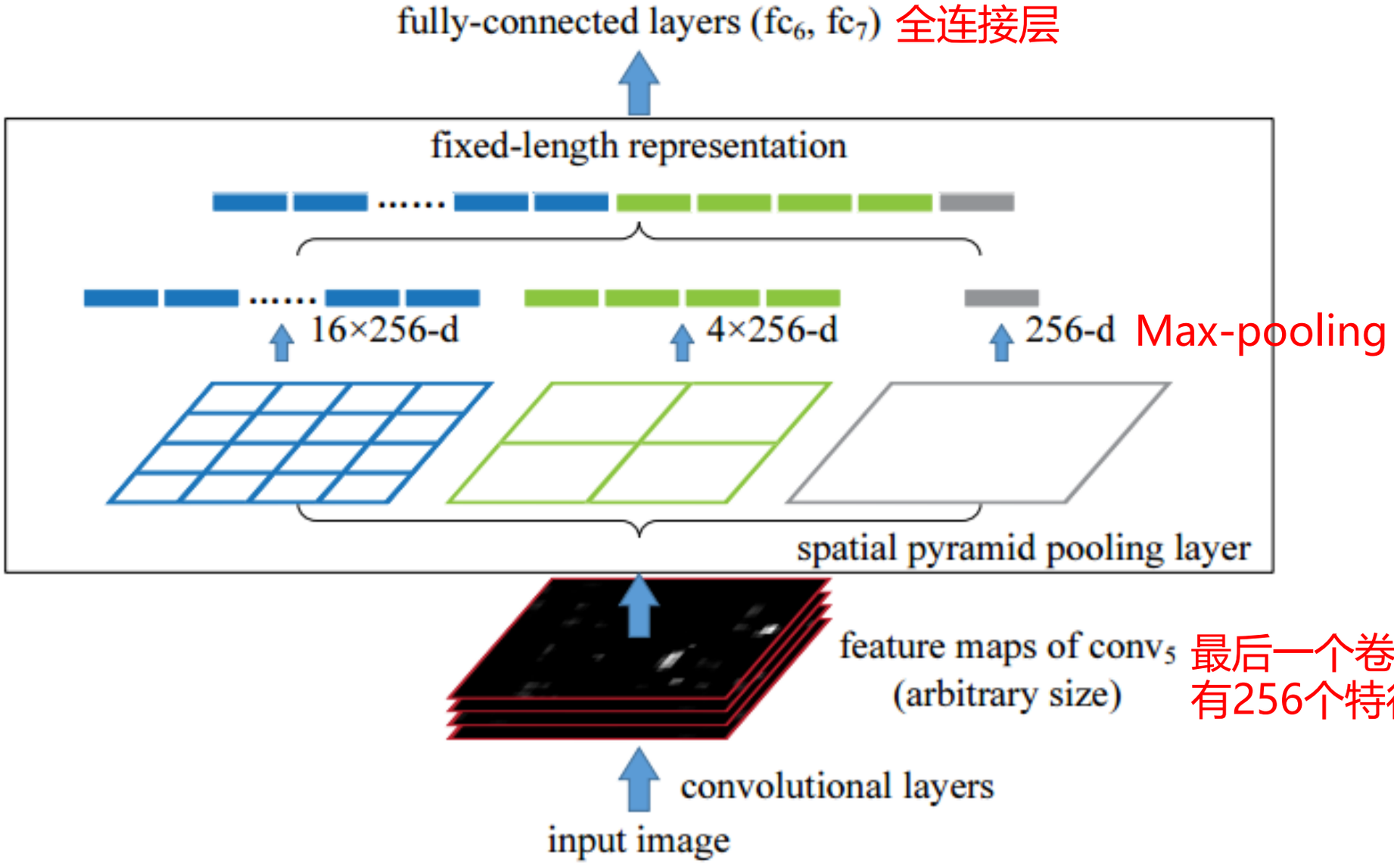
SPP-Net

RCNN的进化中SPP Net的思想对其贡献很大，这里也简单介绍一下SPP-Net。

SPP-Net的作者是何凯明。R-CNN的最大瓶颈是2k个候选区域都要经过一次CNN，速度非常慢。SPP-net最大的改进是只需要将原图做一次卷积操作，就可以得到每个候选区域的特征。

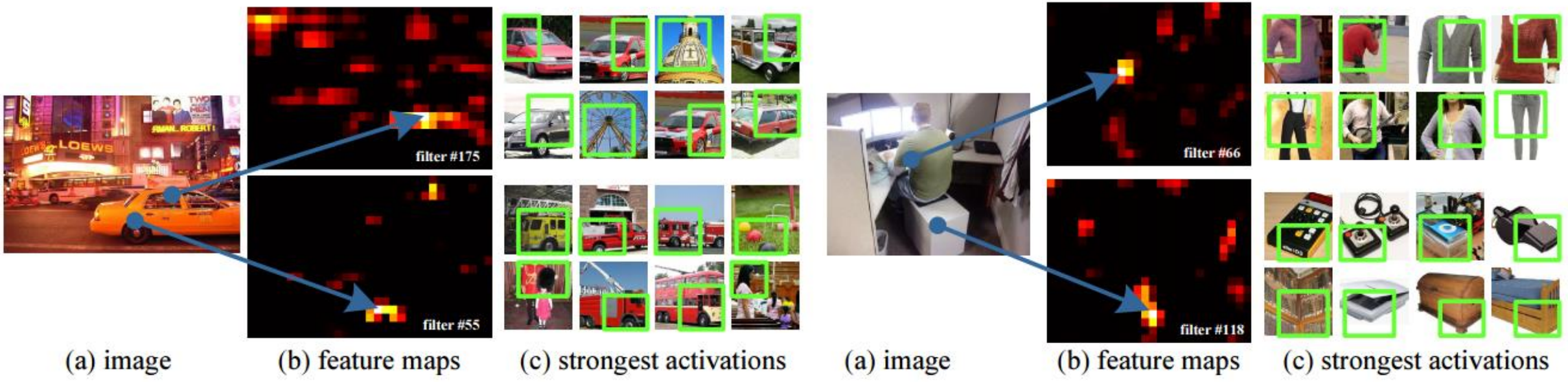


金字塔池化层(Spatial Pyramid Pooling)



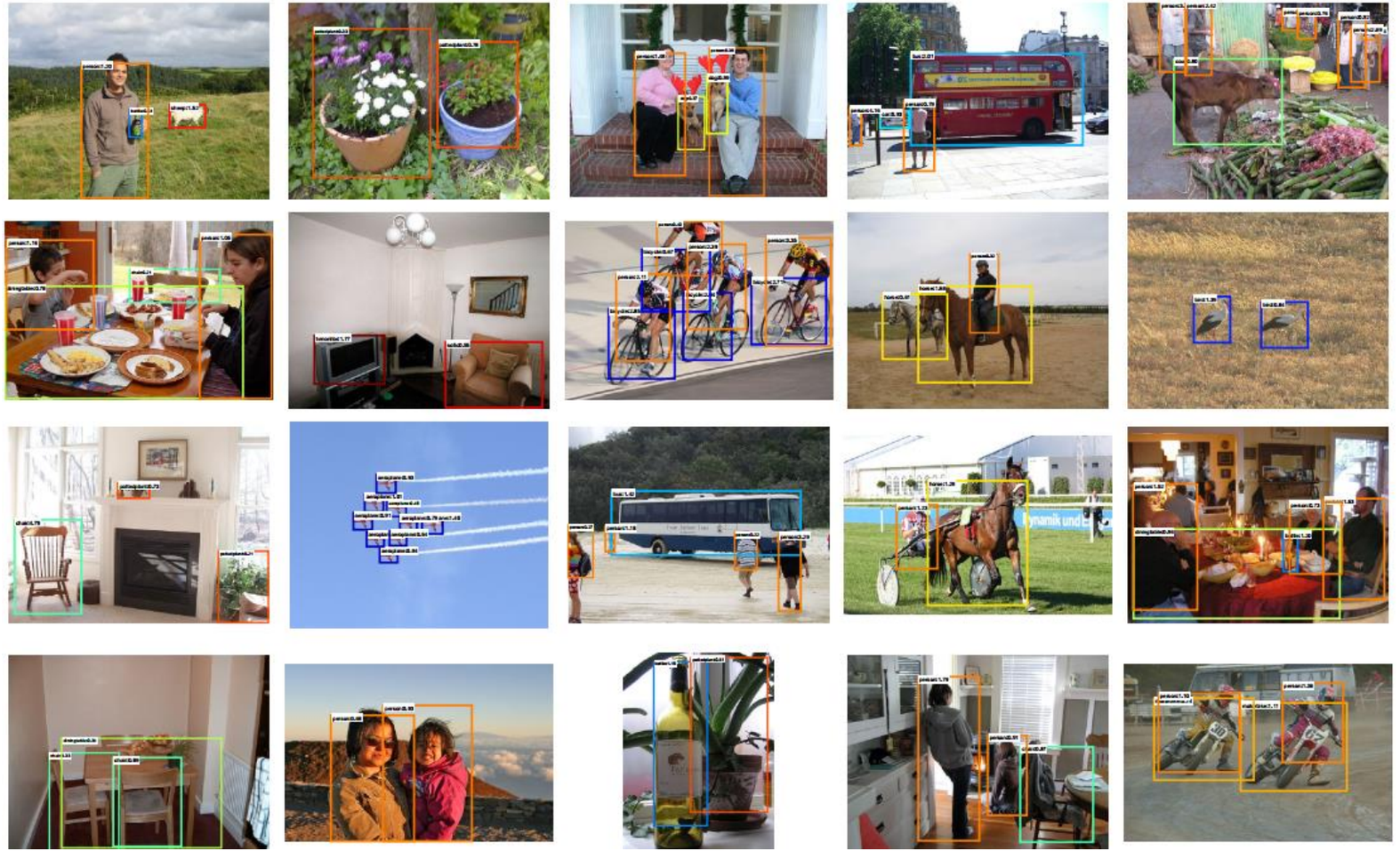
特征映射

对卷积层可视化发现：输入图片的某个位置的特征反应在特征图上也是在相同位置。
我们将ss算法提供的2000个候选区域的位置记录下来，通过比例映射到conv5输出的feature map上，提取出候选区域的特征图m，然后将m送入到金字塔池化层中进行计算。



SPP-Net效果

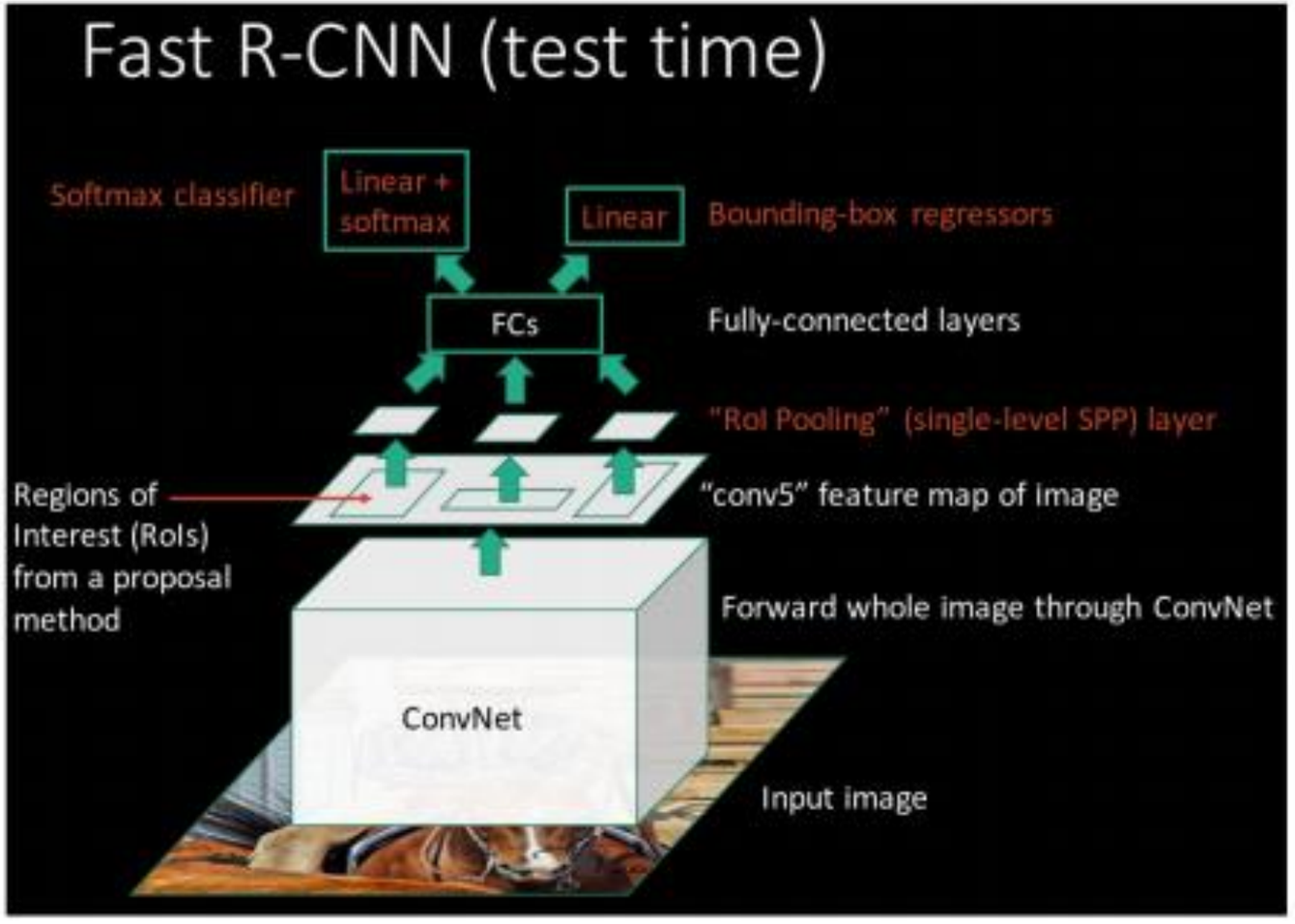
SPP-Net的图像检测速度大约比RCNN提升了100倍。



Fast-RCNN

Fast-RCNN

继2014年的RCNN之后，RBG借鉴了SPP-Net的设计思想，在15年推出了Fast RCNN。

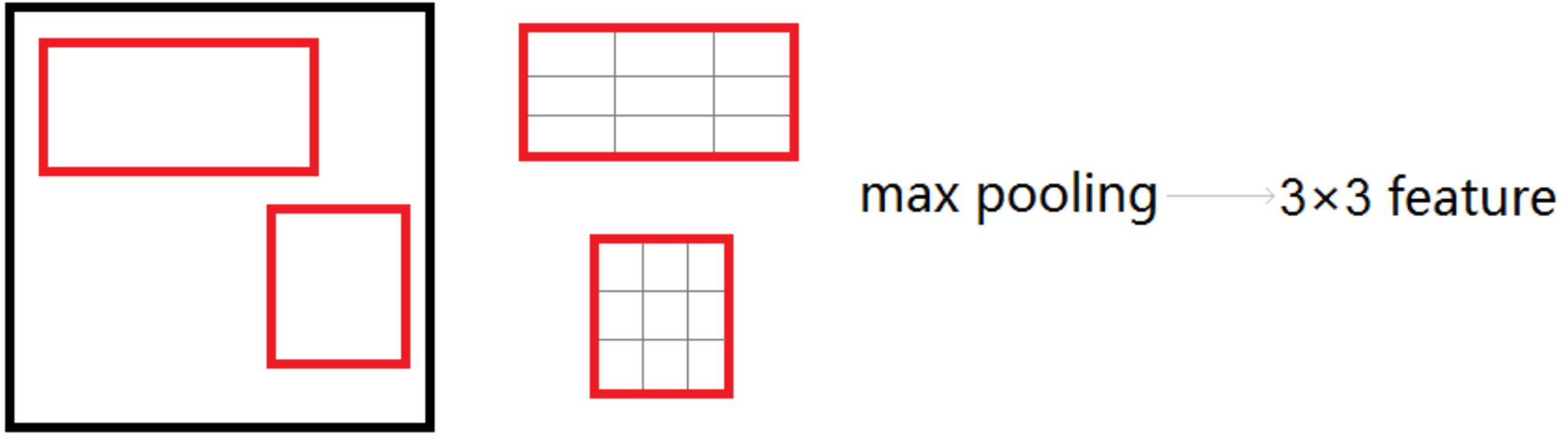


R-CNN Problem #1:
Slow at test-time due to independent forward passes of the CNN

Solution:
Share computation of convolutional layers between proposals for an image

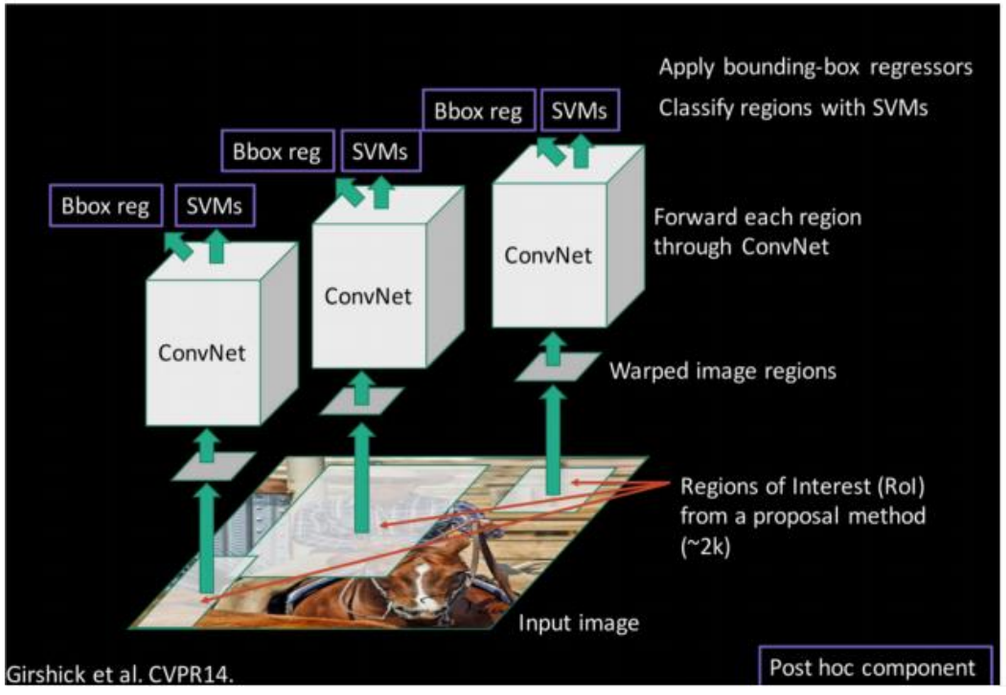
ROI Pooling(Region of Interest Pooling)

roi_pooling层其实是SPP-Net中金字塔池化层的一种简单化的形式，roi pooling层只使用一种固定输出大小的max-pooling。
将每个候选区域均匀分成M×N块，对每块进行max-pooling。将特征图上大小不一的候选区域转变为大小统一的数据，送入下一层。

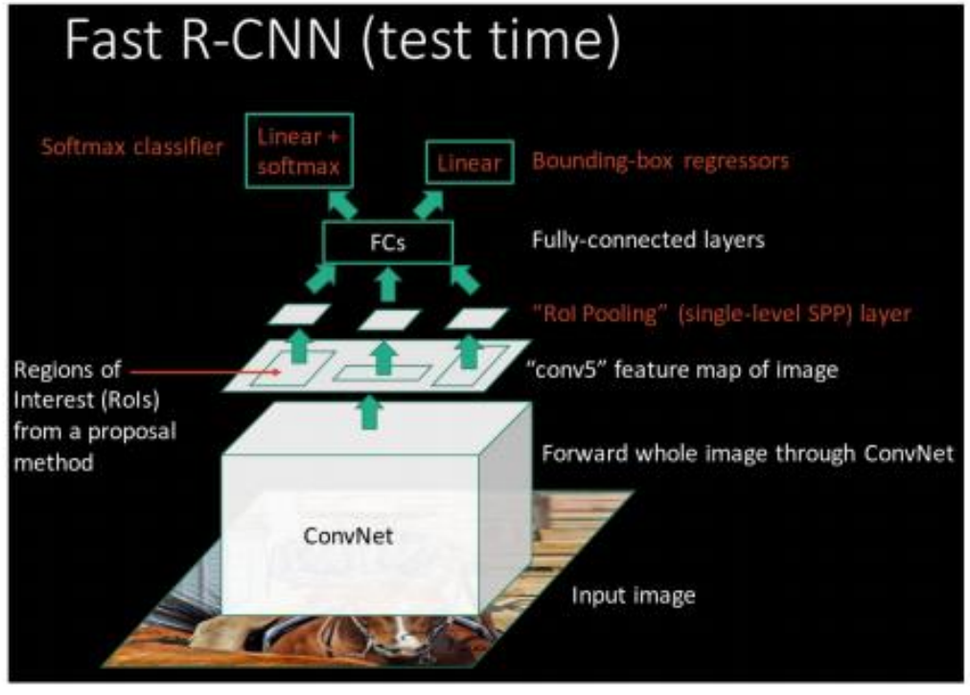


multi-task多任务学习模型

Fast-RCNN把bbox regression放进了神经网络内部，与region分类和并成为了一个multi-task模型，实际实验也证明，这两个任务能够共享卷积特征，并相互促进。这个结构的优化极大提升了模型的训练和预测速度，也为后来的Faster-RCNN做下了铺垫。



RCNN



Fast-RCNN

Fast-RCNN结果分析

	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
▷ with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
▷ with SVD	169×	150×	**213**×	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
▷ with SVD	56.5	58.7	66.6	-	-	-	-

Table 4. Runtime comparison between the same models in Fast R-CNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales specified in [11]. †Timing provided by the authors of [11]. Times were measured on an Nvidia K40 GPU.

Fast-RCNN结果分析

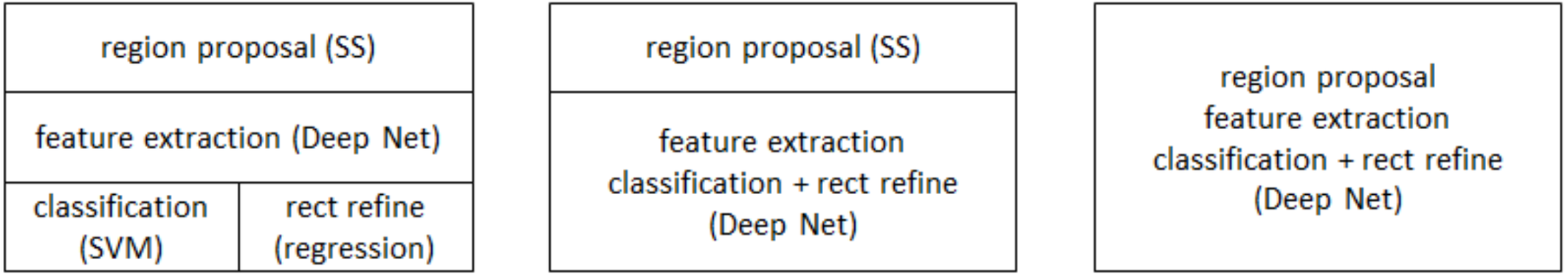
method	classifier	S	M	L
R-CNN [9, 10]	SVM	**58.5**	**60.2**	66.0
FRCN [ours]	SVM	56.3	58.7	66.8
FRCN [ours]	softmax	57.1	59.2	**66.9**

Table 8. Fast R-CNN with softmax vs. SVM (VOC07 mAP).

Faster-RCNN

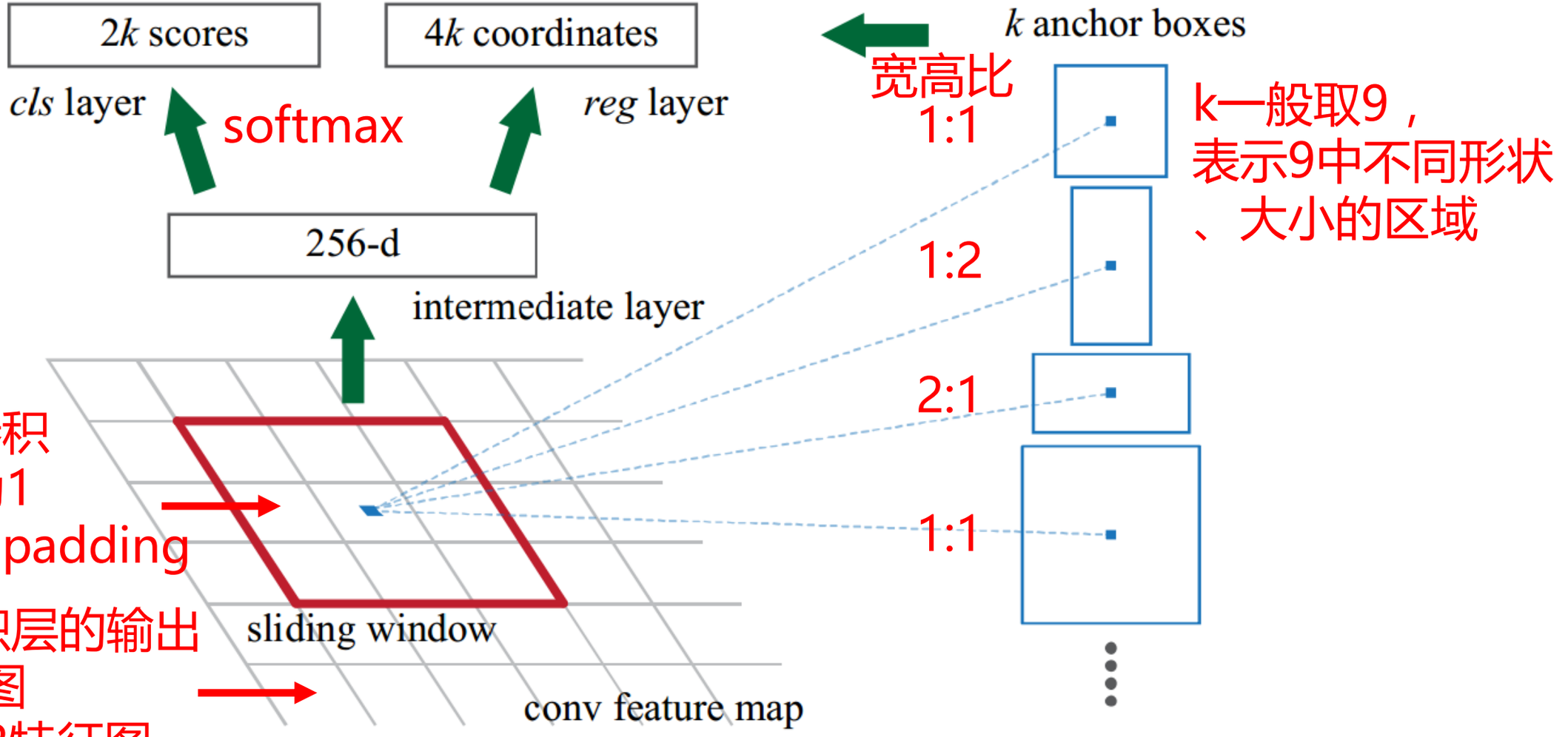
Faster-RCNN

Faster-RCNN是RBG团队在2015年的又一力作。简单网络目标检测速度达到17fps，复杂网络达到5fps。
Fast-RCNN已经很优秀了，但是还存在一个比较大的问题，就是selective search。Faster-RCNN加入了一个专门生成候选区域的神经网络，也就是说找到候选框的工作也交给了神经网络来做了。做这个任务的网络叫做Region Proposal Network(RPN)
Faster-RCNN可以简单地看做“区域生成网络+fast-RCNN“的系统，用区域生成网络代替fast-RCNN中的Selective Search方法

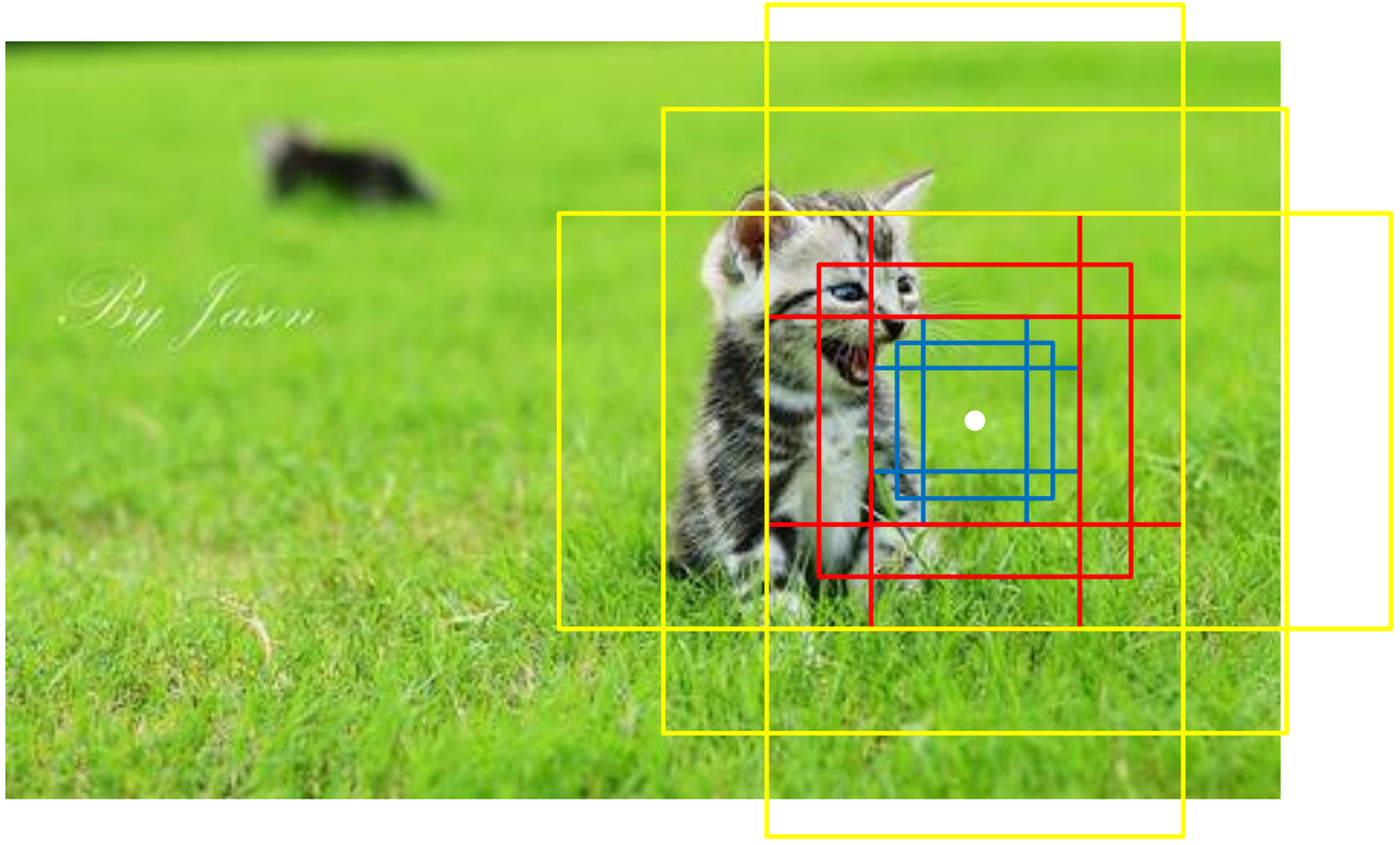


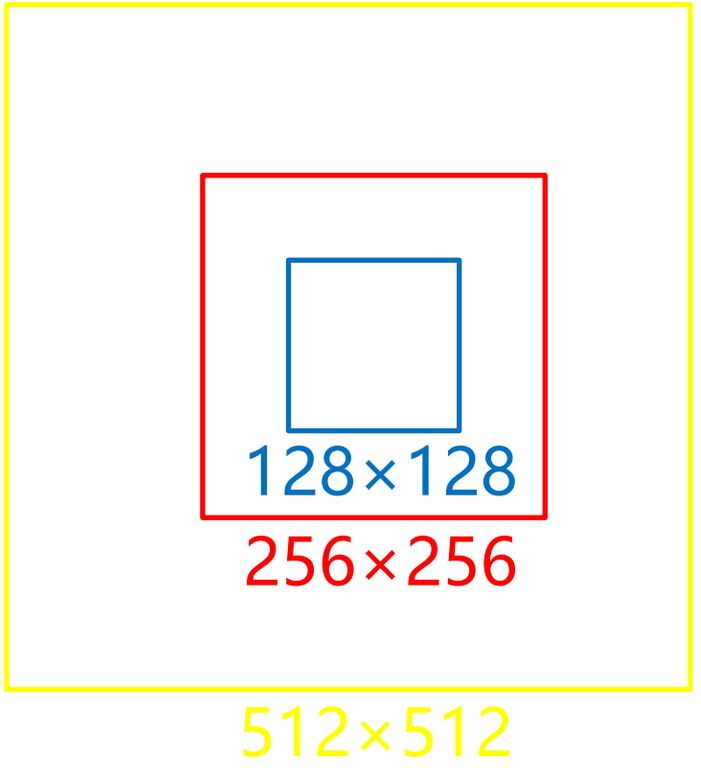
Region Proposal Network(RPN)

target | not a target | x | y | w | h

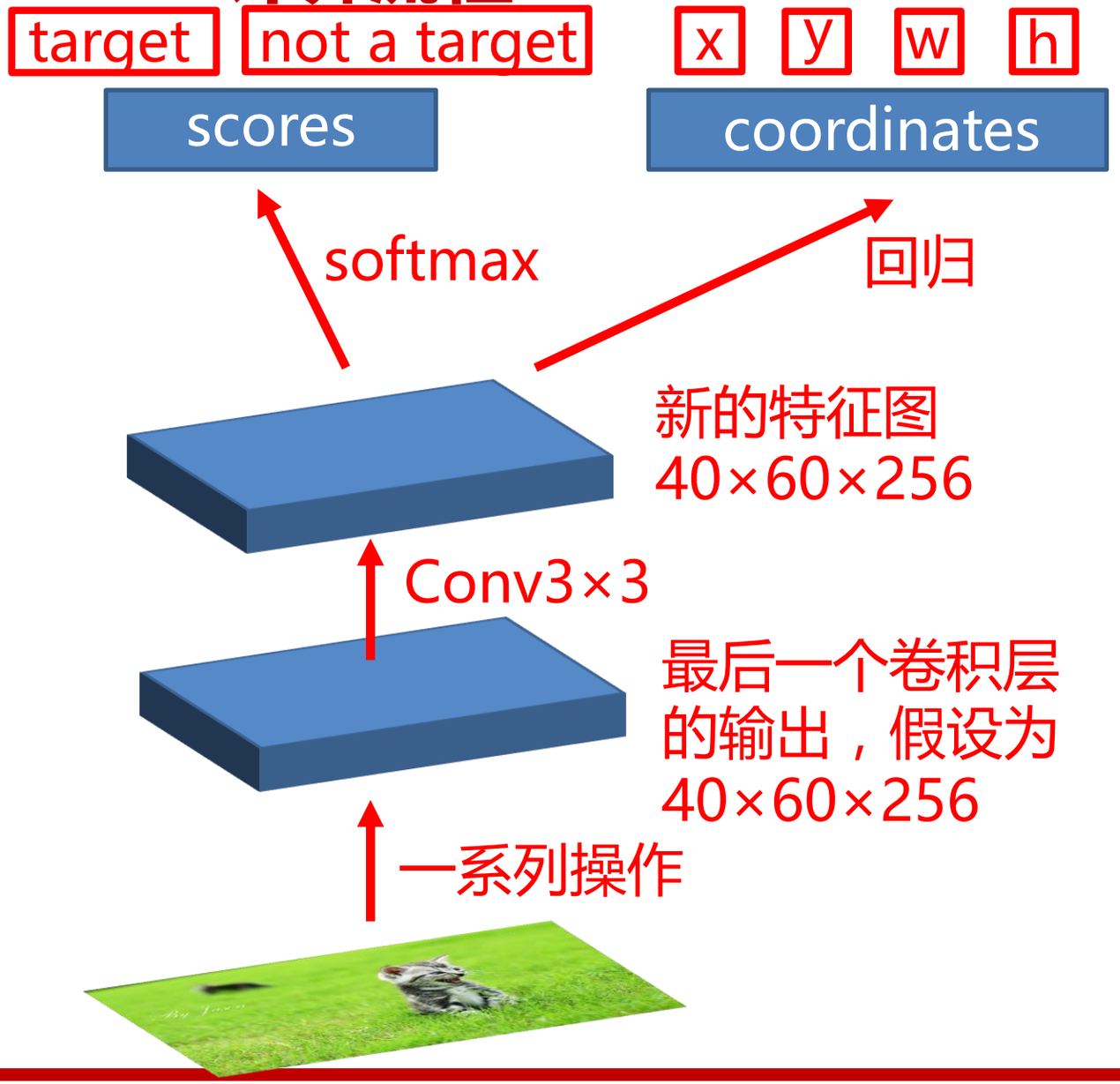


anchor





RPN计算流程1



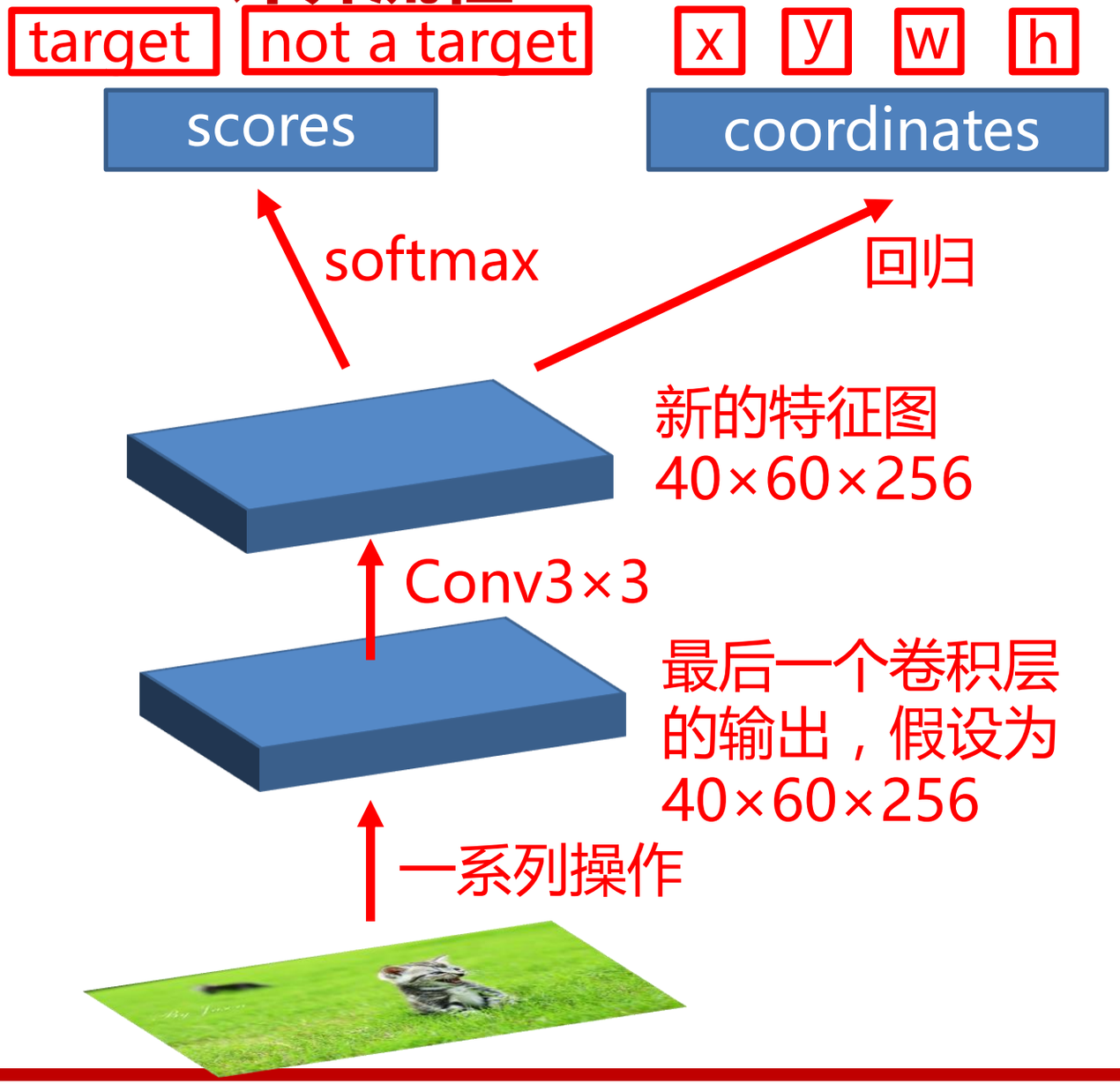
1.最后一个卷积层输出的特征图再进行一次卷积操作得到新的特征图。

2.新的特征图的平面上有40×60共2400个点，每个点都可以对应到原始图片上，得到9个anchor，所以一共可以得到40×60×9大约20000个候选区域。

3.计算所有候选区域的scores。

4.把所有超出图片的候选区域都限制在图片区域内，选scores最大的前12000个候选区域。

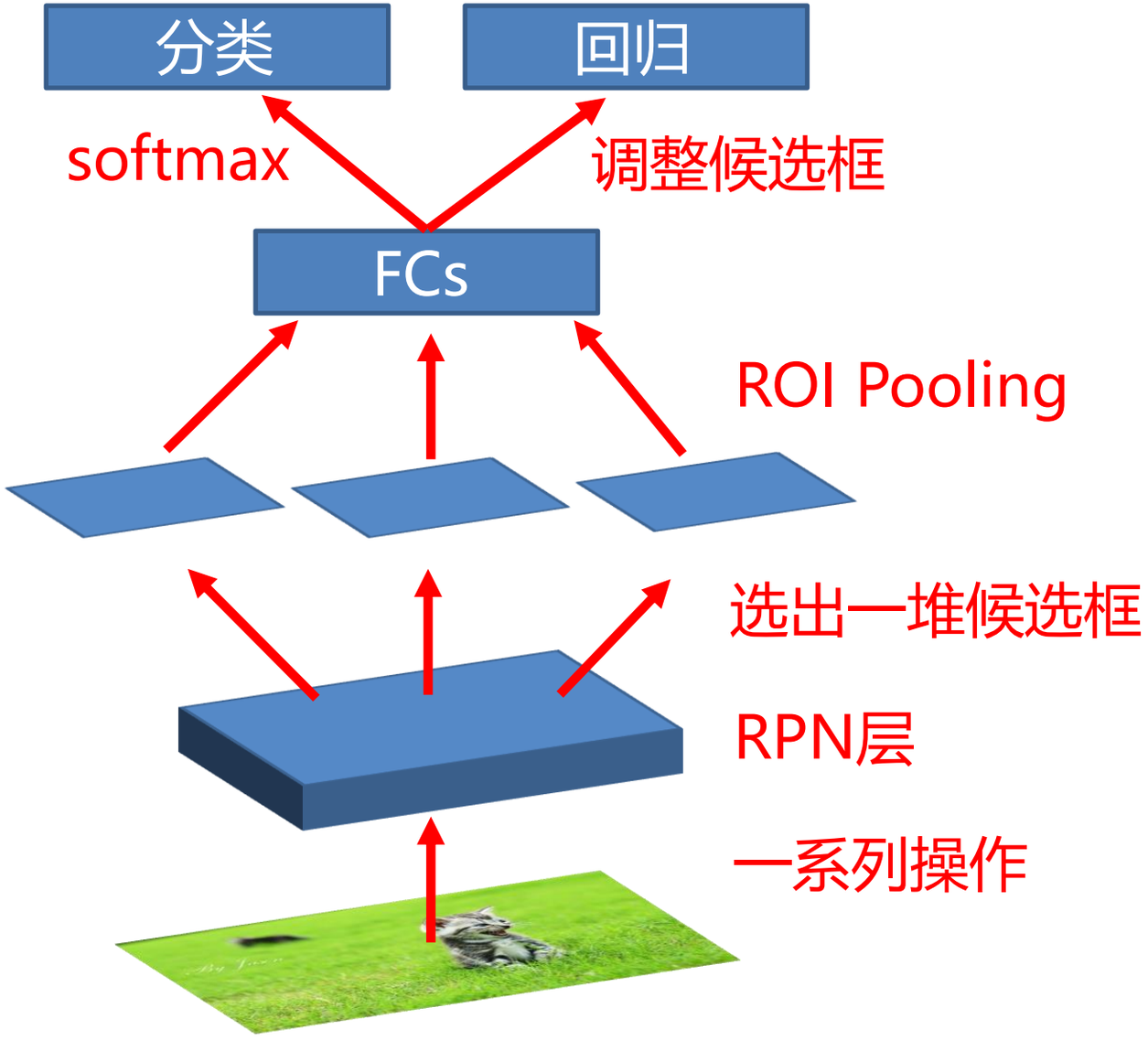
RPN计算流程2



5. 剩余的区域中有些候选区域跟其他候选区域有大量重叠，我们可以基于第4步计算的scores，采用非极大值抑制(NMS)。我们固定NMS的IoU阈值为0.7。然后再选出scores最大的前2000个候选区域。这2000个候选区域如果与某个标定区域重叠比例大于0.7，记为正样本，如果与任意一个标定区域重叠比例都小于0.3，记为负样本。其余区域不作为样本。

6.在训练RPN层分类回归任务时，我们会随机地抽取256个区域来训练，正负候选区域比例为1:1，如果正样本数小于128，用负样本填充。

Faster-RCNN算法流程



7.训练最后输出的分类回归任务时，我们随机抽取64个与真实标注框IoU≥0.5的区域作为前景，256-64个IoU＜0.5且≥0.1的区域作为背景来训练。

Faster-RCNN-RPN_loss定义

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

函数由两部分组成，一部分是目标预测的loss，一部分是回归预测的loss。

考虑分类的loss：

p_i为anchor i是目标的预测概率。

如果anchor为正，则GT标签p^*为1；anchor为负，GT标签p^*为0。

$L_{cls}(p_i, p_i^*) = -log[p_i^* p_i + (1-p_i^*)(1-p_i)]$

当p_i^*为0时，$L_{cls}(p_i, p_i^*) = -log(1-p_i)$

当p_i^*为1时，$L_{cls}(p_i, p_i^*) = -logp_i$

N_{cls}分类loss的系数

Faster-RCNN-RPN_loss定义

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

考虑回归的loss：

$t_i = \{t_x, t_y, t_w, t_h\}$

当p_i^*为0时，回归的loss为0

当p_i^*为1时，才需要考虑回归loss

$L_{reg}(t_i, t_i^*) = \mathrm{R}(t_i - t_i^*)$，R为：

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

在程序中的计算为：$w_{out} \times Smooth_{L1}(w_{in} \times (t_i - t_i^*))$

λ和N_{reg}回归loss的系数

Faster-RCNN结果

Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

train-time region proposals		**test-time region proposals**		
method	# boxes	method	# proposals	mAP (%)
SS	2000	SS	2000	58.7
EB	2000	EB	2000	58.6
RPN+ZF, shared	2000	RPN+ZF, shared	300	**59.9**
ablation experiments follow below				
RPN+ZF, unshared	2000	RPN+ZF, unshared	300	58.7
SS	2000	RPN+ZF	100	55.1
SS	2000	RPN+ZF	300	56.8
SS	2000	RPN+ZF	1000	56.3
SS	2000	RPN+ZF (no NMS)	6000	55.2
SS	2000	RPN+ZF (no *cls*)	100	44.6
SS	2000	RPN+ZF (no *cls*)	300	51.4
SS	2000	RPN+ZF (no *cls*)	1000	55.8
SS	2000	RPN+ZF (no *reg*)	300	52.1
SS	2000	RPN+ZF (no *reg*)	1000	51.3
SS	2000	RPN+VGG	300	59.2

Faster-RCNN结果

data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9†
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	**73.2**
RPN+VGG, shared	300	COCO+07+12	**78.8**

Table 4: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07++12": union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

Table 5: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. "Region-wise" includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

Faster-RCNN程序

https://github.com/dBeker/Faster-RCNN-TensorFlow-Python3.5

标注工具：https://github.com/tzutalin/labelImg

Thanks!