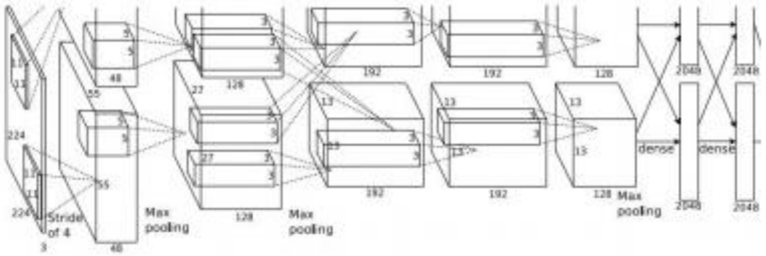
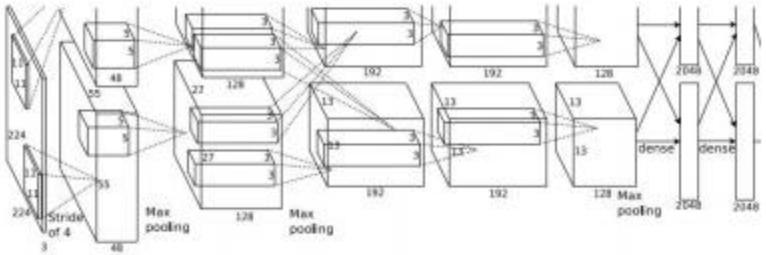
Object Detection as Regression?

Each image needs a different number of outputs!



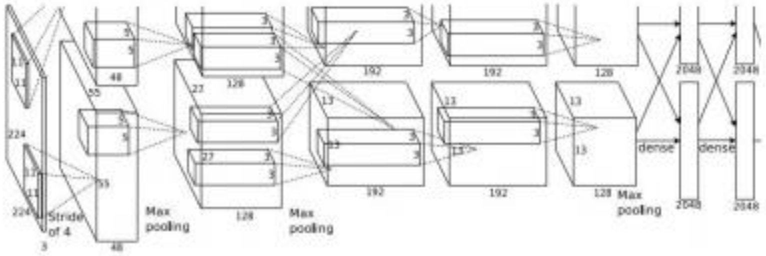
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

16 numbers

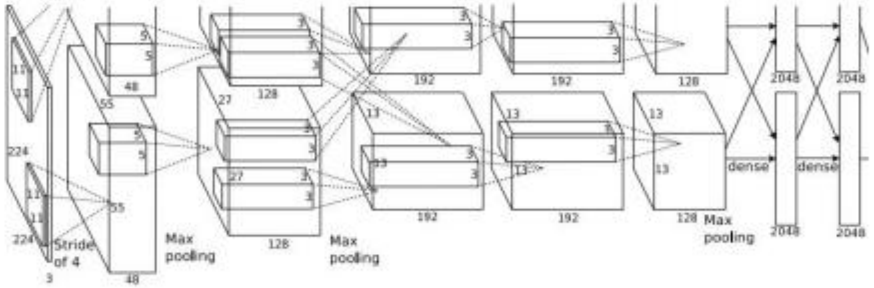


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

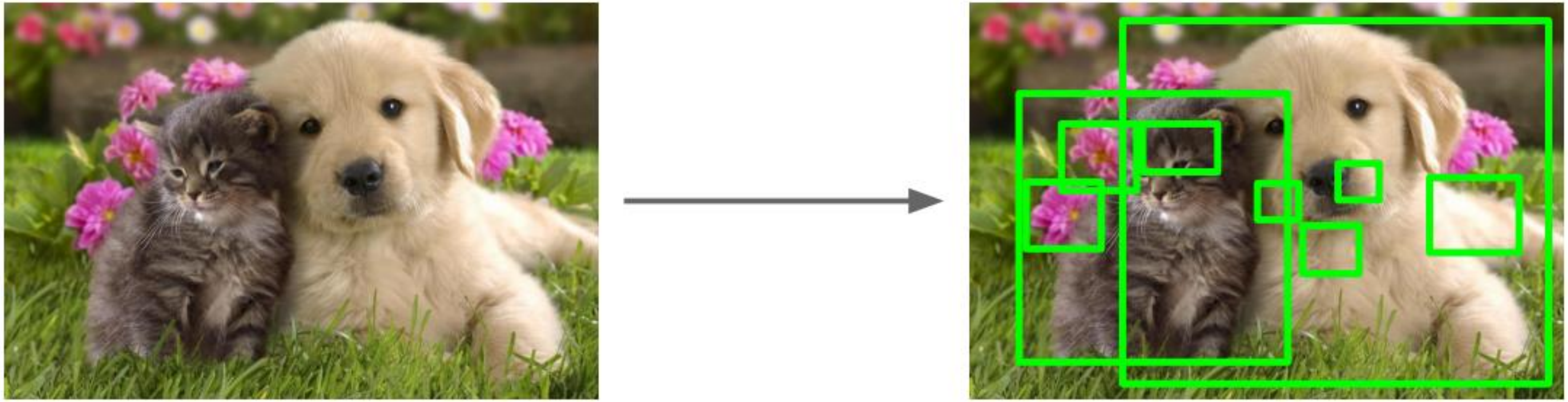


Dog? YES
Cat? NO
Background? NO

图像金字塔

Region Proposals

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 1000 region proposals in a few seconds on CPU



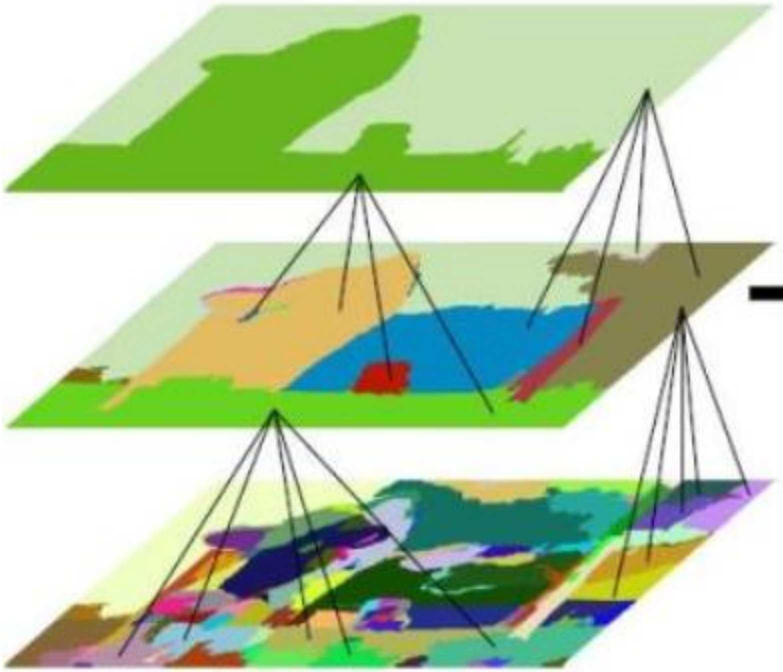
Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

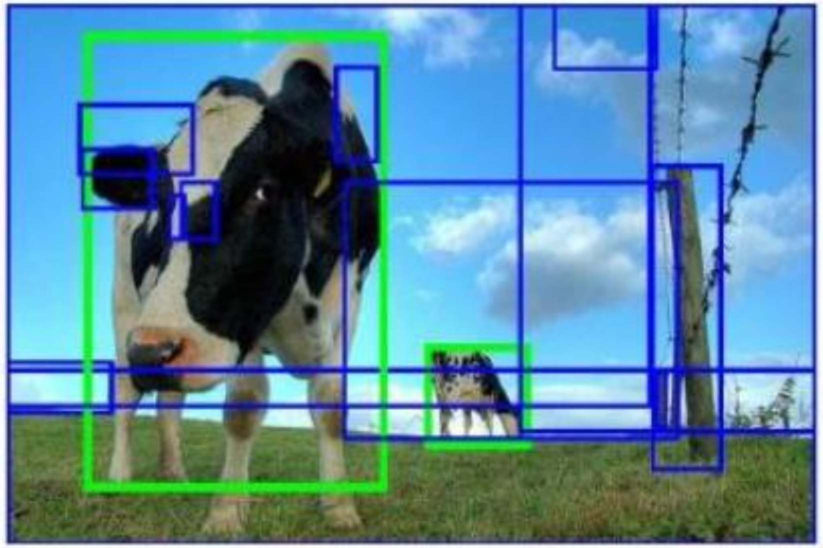
选择搜索(Selective Search)

- 选择搜索方法是最为熟知的图像bouding boxes提取算法， 由Koen E.A于2011年提出
- 选择搜索算法的主要观点：图像中物体可能存在的区域应该是有某些相似性或者连续性区域的。因此，选择搜索基于上面这一想法采用子区域合并的方法进行提取bounding boxes候选边界框。首先，对输入图像进行分割算法产生许多小的子区域。其次，根据这些子区域之间相似性(相似性标准主要有颜色、纹理、大小等等)进行区域合并，不断的进行区域迭代合并。每次迭代过程中对这些合并的子区域做bounding boxes(外切矩形)，这些子区域外切矩形就是通常所说的候选框。

选择搜索(Selective Search)

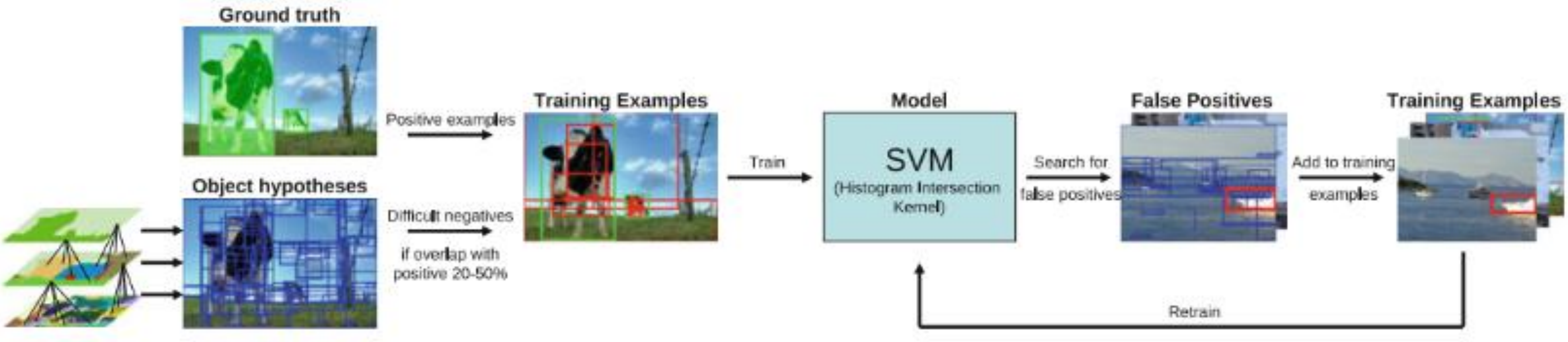
- Selective Search策略其实是借助了层次聚类的思想将层次聚类的思想应用到区域的合并上面





古典目标识别

- 第一部分: 训练集构造
  - 负样本: 使用SS方法对区域进行融合--> 计算每个候选区域与真实标记区域GT之间的重合,如果区域A与GT的重合度在20-50%之间,而且A与其他的任何一个已生成的负样本之间的重合度不大于70%,则A被采纳为负样本;
  - 正样本: 就是那些手工标记的GT区域作为正样本;

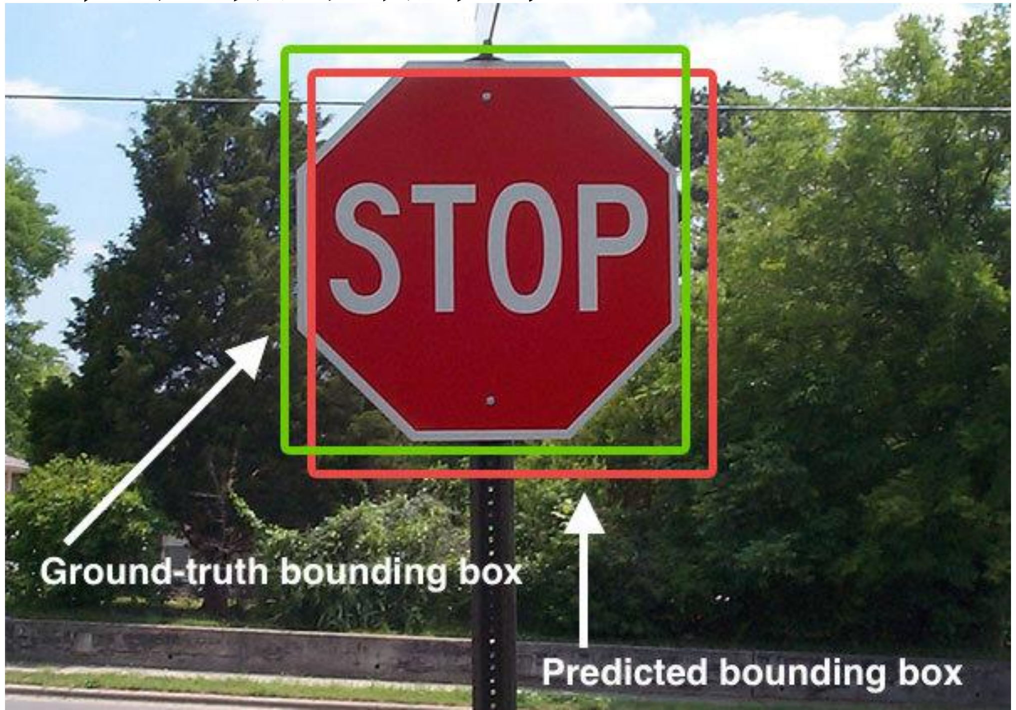


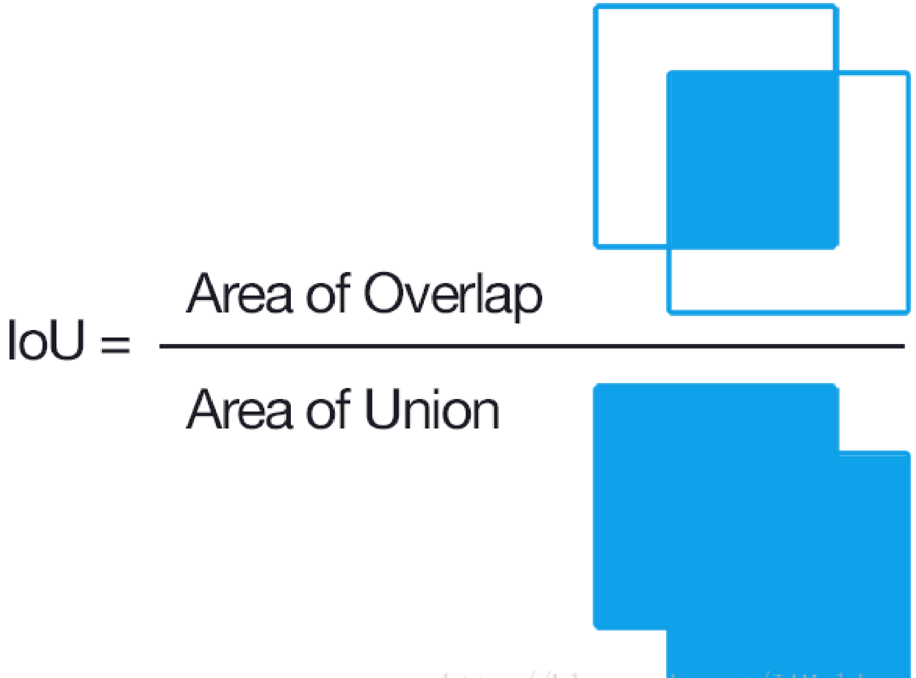
古典目标识别

- 第二部分: 提取每个正/负样本的特征
  - HOG特征 + bag-of-words特征,同时辅助性地增加了SIFT,two colour SIFT,Extended OpponentSIFT,RGB-SIFT这四种特征,这样特征加起来的维度达到了惊人的360,000
- 第三部分: 分类器SVM训练
- 第四部分: 反馈False Positive
  - 把这些"False Positives"收集起来,以刚才训练得到的SVM的权值作为其初始权值,对SVM进行二次训练,经过二次训练的SVM的分类准确度一般会有一定的提升;
- 测试过程
  - 首先用SS方法得到测试图像上候选区域 --> 然后提取每个区域的特征向量 --> 送入已训练好的SVM进行软分类 --> 将这些区域按照概率值进行排序 --> 把概率值小于0.5的区域去除 --> 对那些概率值大于0.5的,计算每个区域与比它分数更高的区域之间的重叠程度IoU,如果重叠程度大于30%,则把这个区域也去除了 --> 最后剩下的区域为目标区域.

IoU(Intersection over Union)

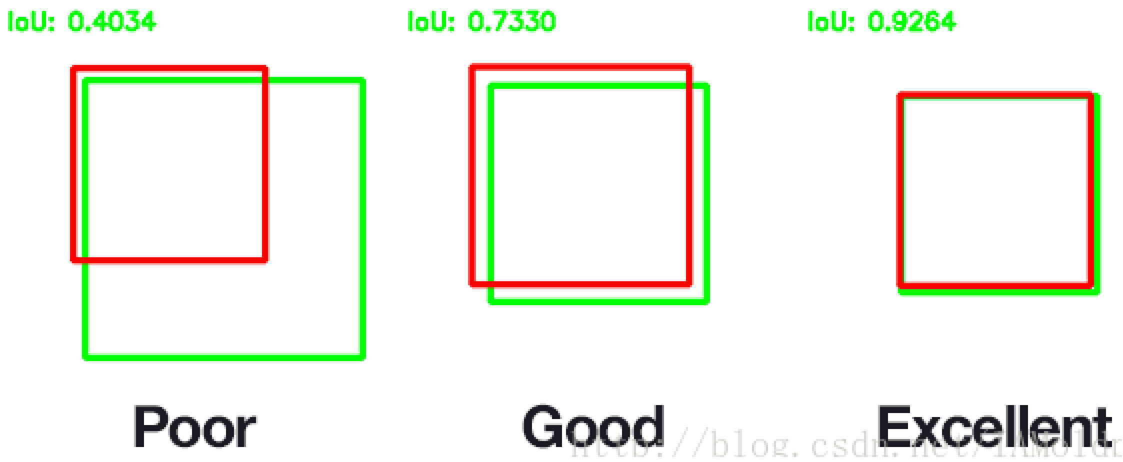
- Intersection over Union是一种测量在特定数据集中检测相应物体准确度的一个标准
- 一般来说，这个Proposal和gt求的得score ＞ 0.5 就可以被认为一个不错的结果了



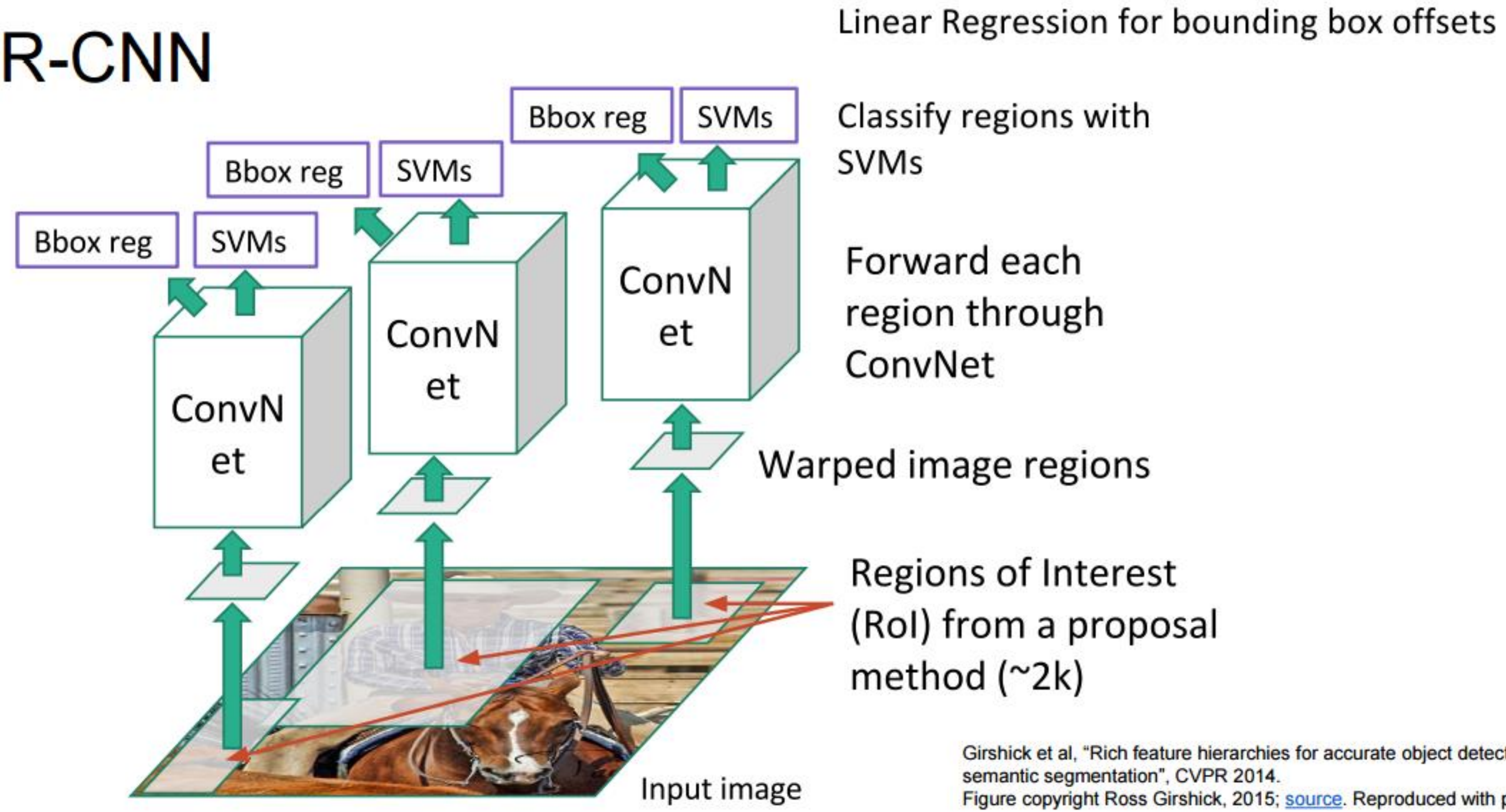


IoU的Python代码逻辑

```python
def bb_intersection_over_union(boxA, boxB):
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = (xB - xA + 1) * (yB - yA + 1)

    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou
```



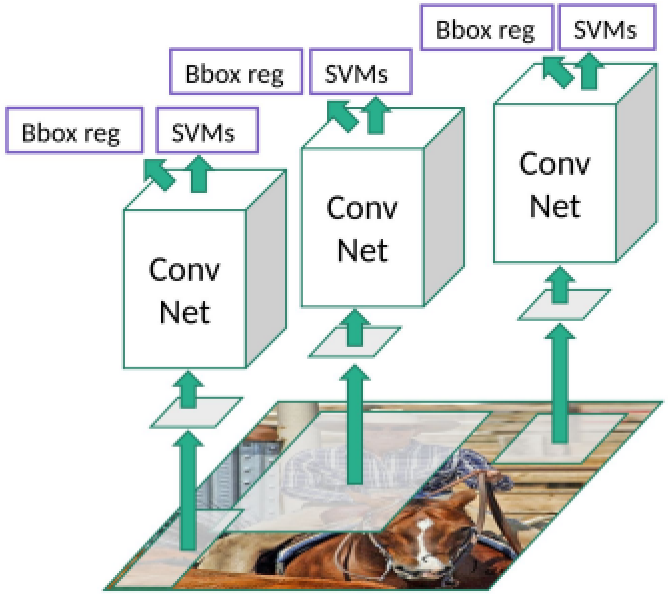
R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

R-CNN: Problems

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
  - Fixed by SPP-net [He et al. ECCV14]



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

SPP-net

Review: Spatial Pyramid Pooling (SPP) layer

From Kaiming's slides

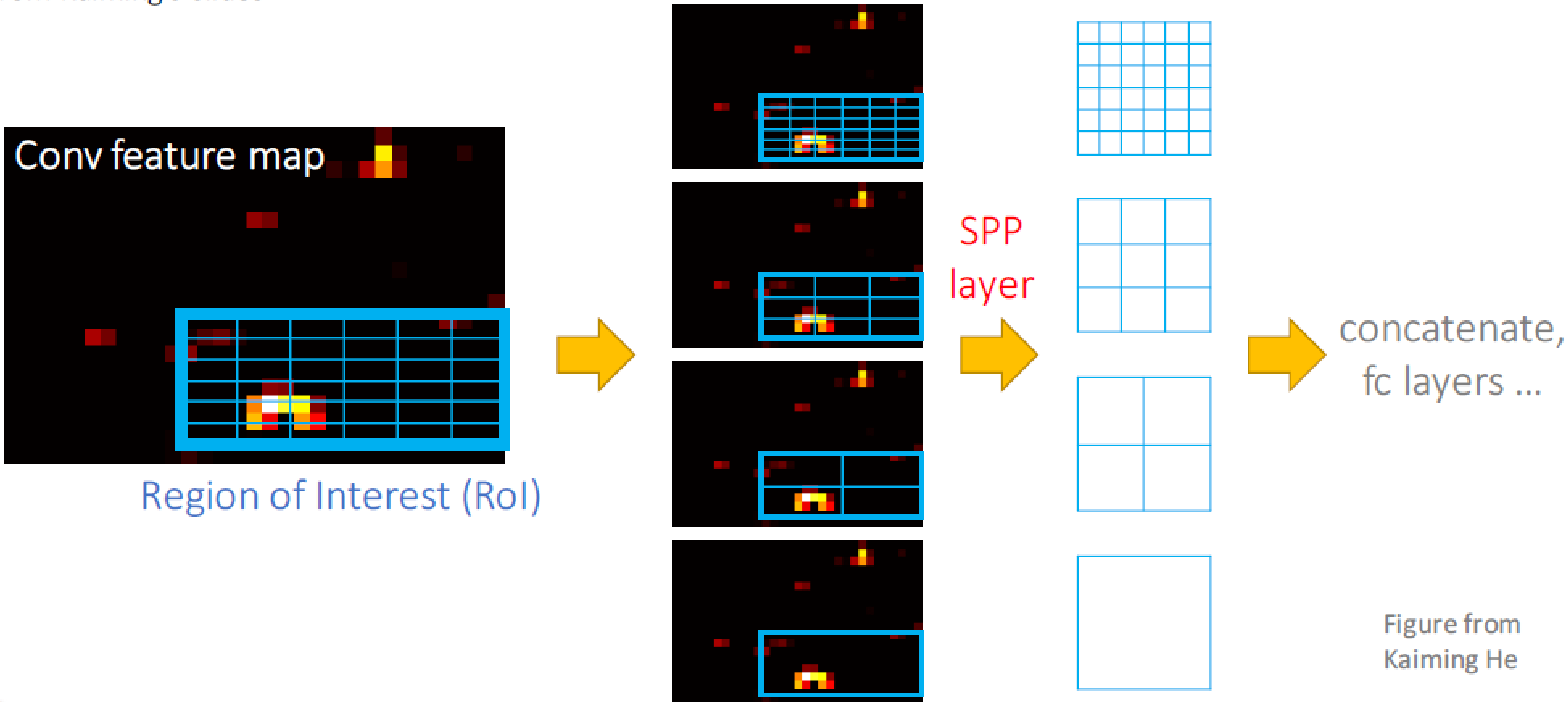
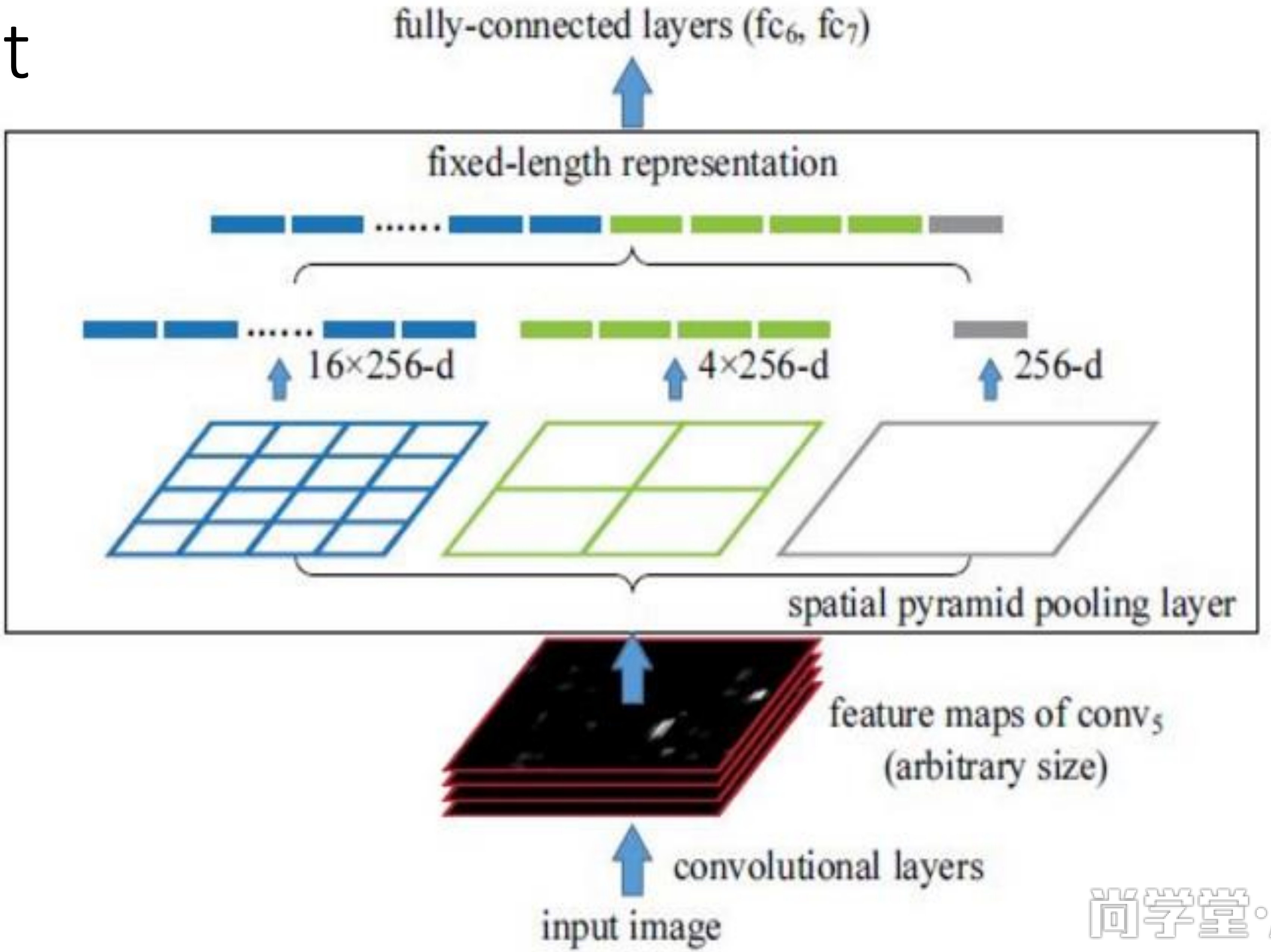


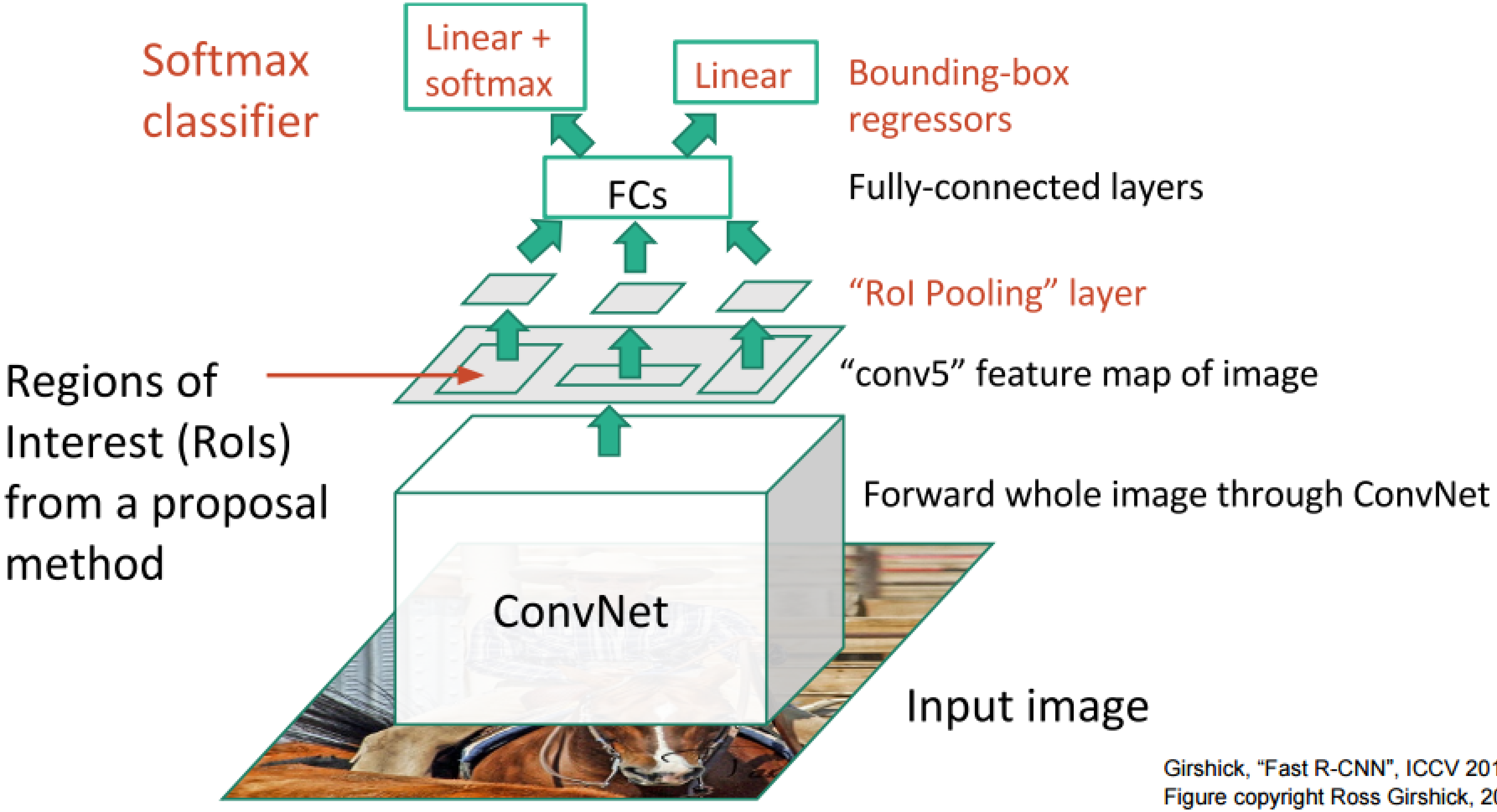
Figure from Kaiming He

SPP-net



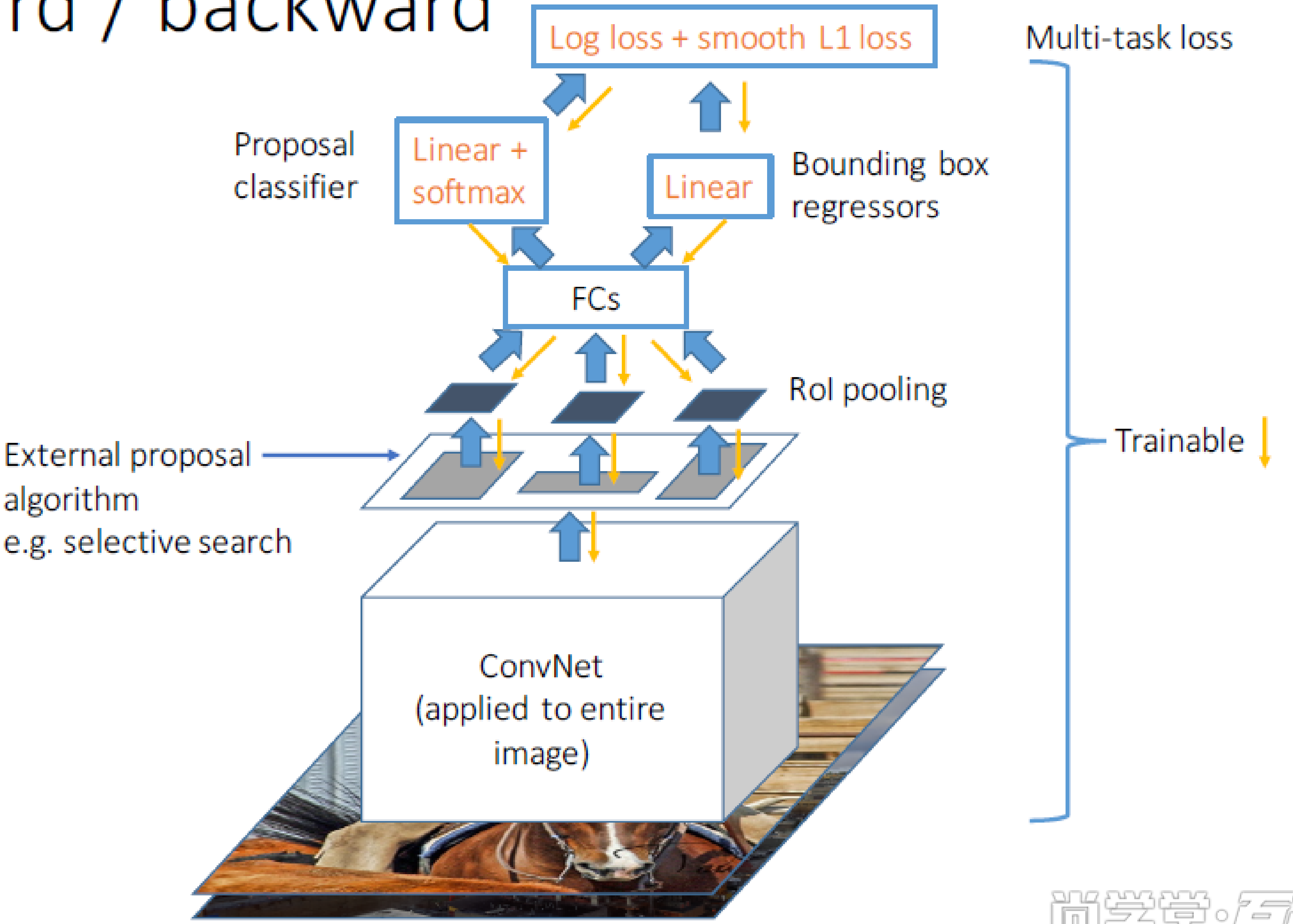
fully-connected layers (fc_6, fc_7)
fixed-length representation
16×256-d
4×256-d
256-d
spatial pyramid pooling layer
feature maps of $conv_5$
(arbitrary size)
convolutional layers
input image

Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Forward / backward



ROI池化是SPP的一个特例

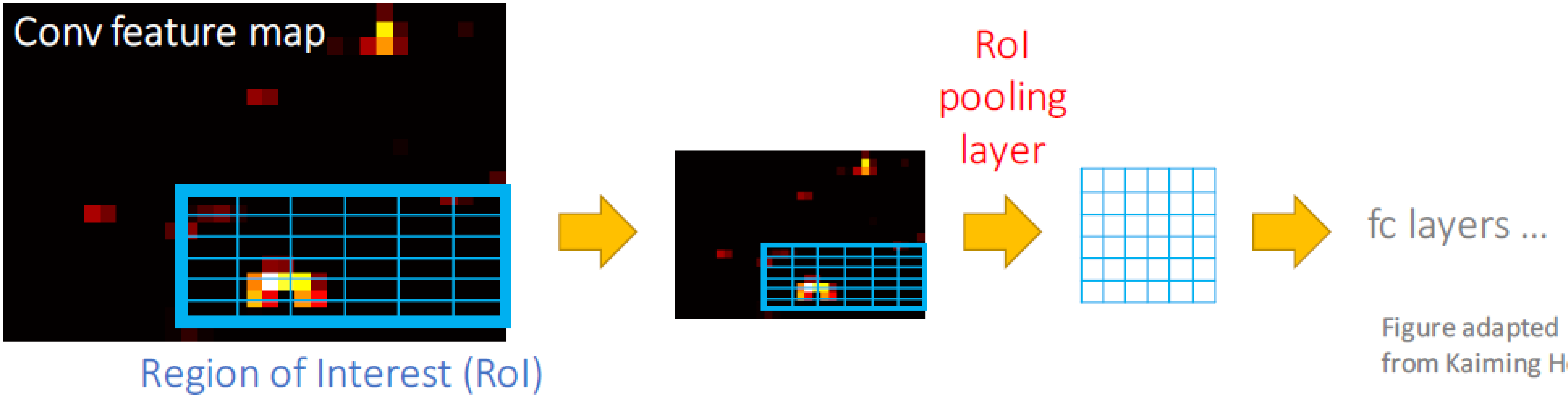
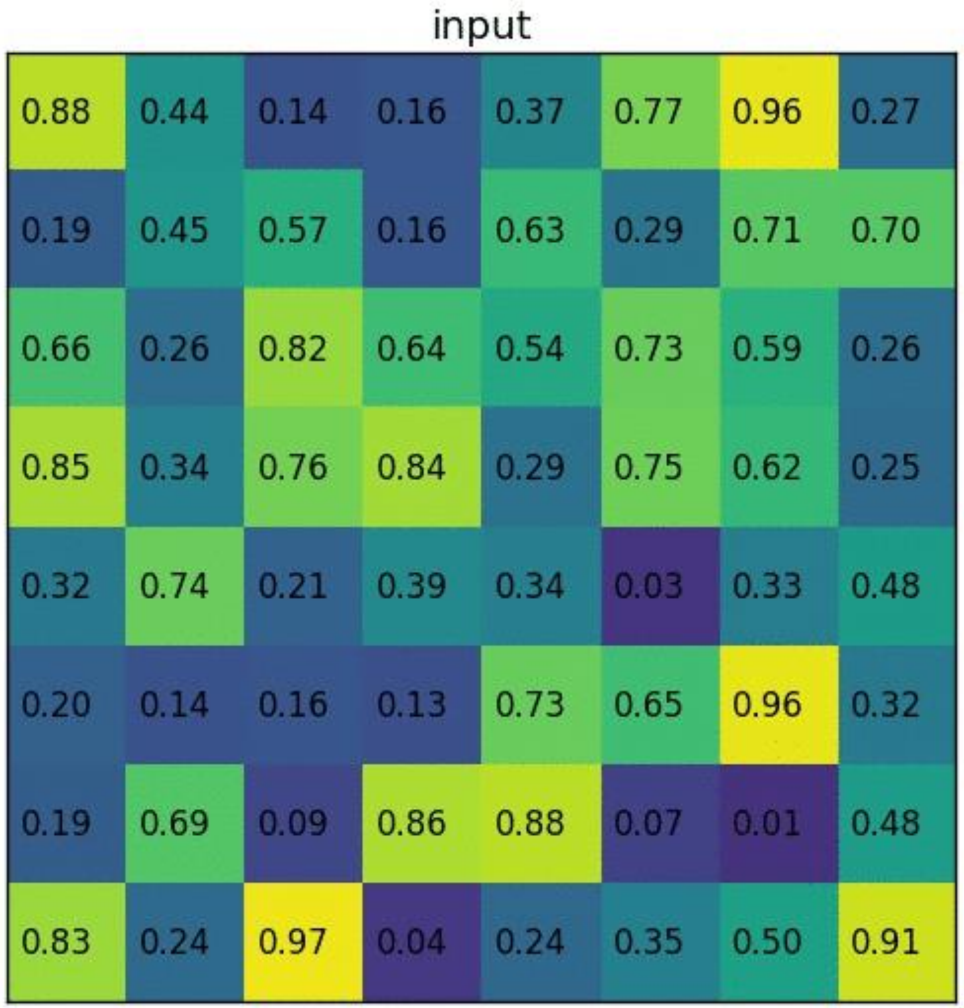


Figure adapted
from Kaiming He

Just a special case of the SPP layer with one pyramid level

ROI池化是SPP的一个特例

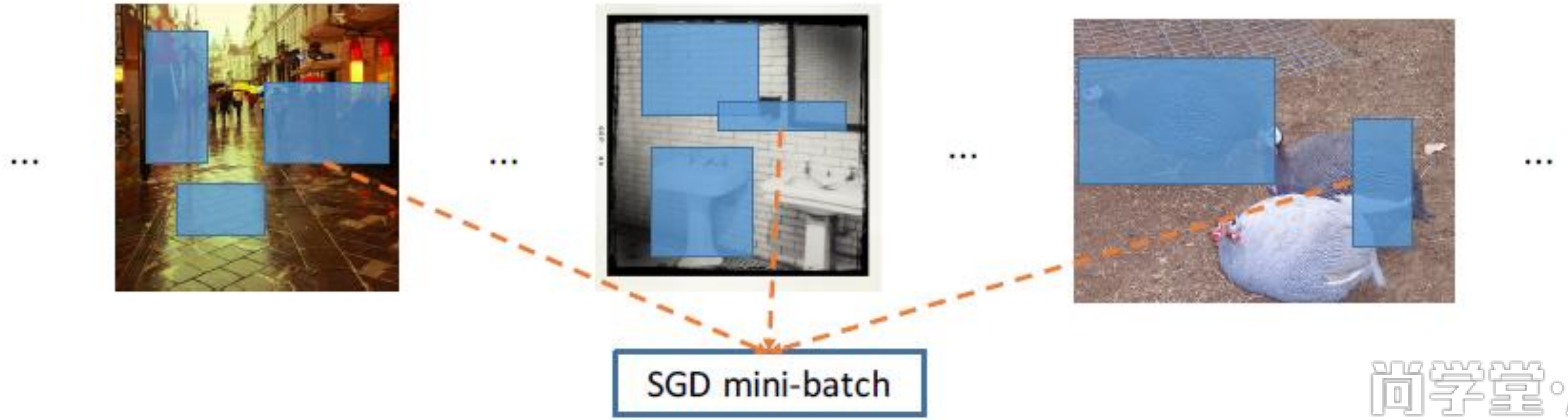
- RoI Pooling的过程就是将一个个
- 大小不同的box矩形框，
- 都映射成大小固定（w * h）
- 的矩形框



梯度下降更有效率

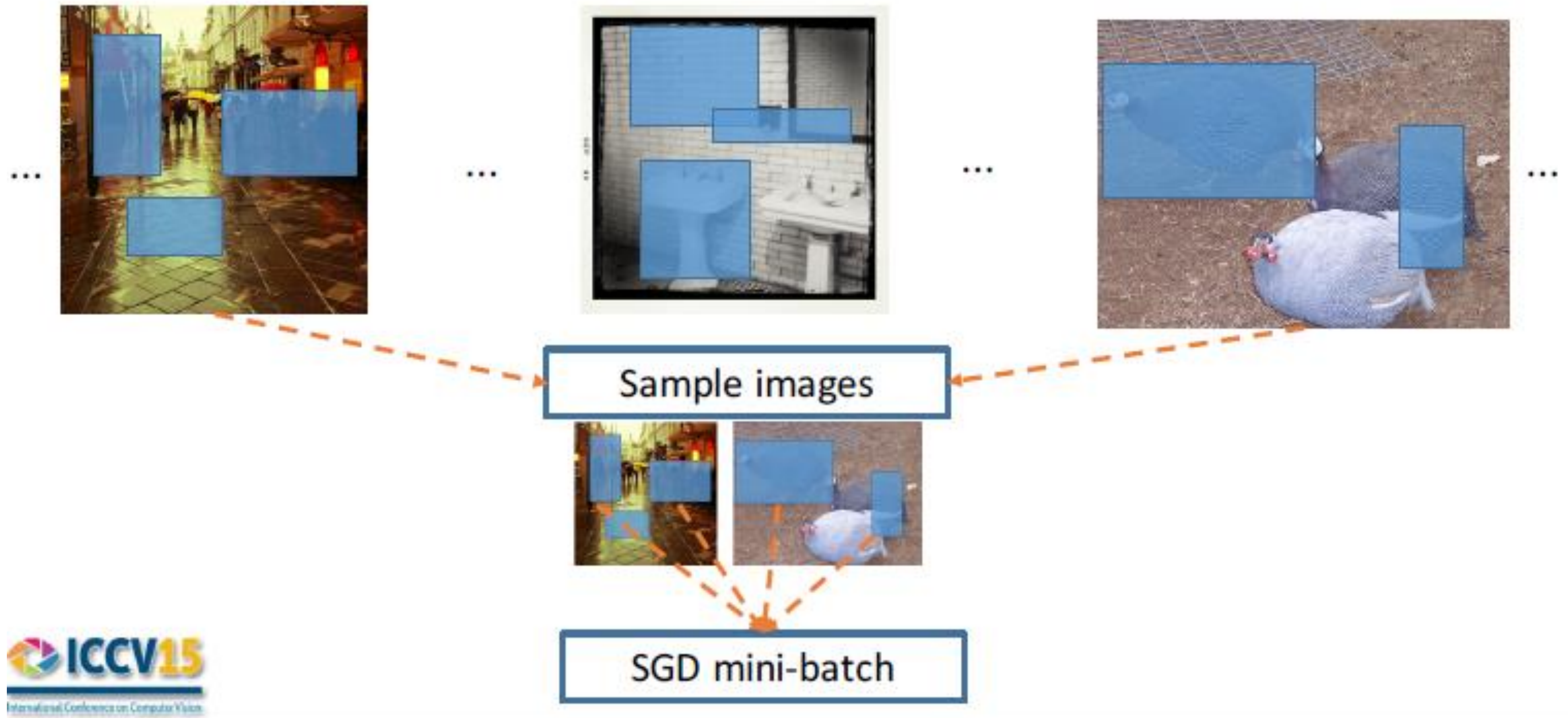
Slow R-CNN and SPP-net use region-wise sampling to make mini-batches

- Sample 128 example RoIs uniformly at random
- Examples will come from different images with high probability



梯度下降更有效率

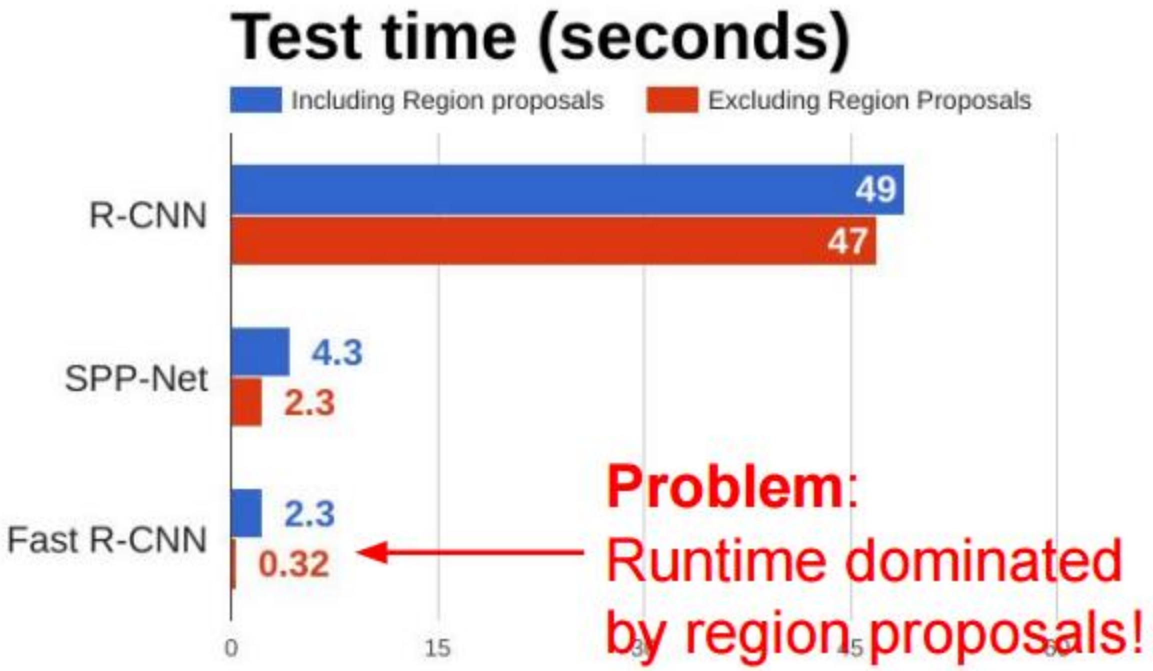
Solution: use hierarchical sampling to build mini-batches



- Sample a small number of images (2)
- Sample many examples from each image (64)

Ross Girshick. "Fast R-CNN". ICCV 2015.

R-CNN vs SPP vs Fast R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

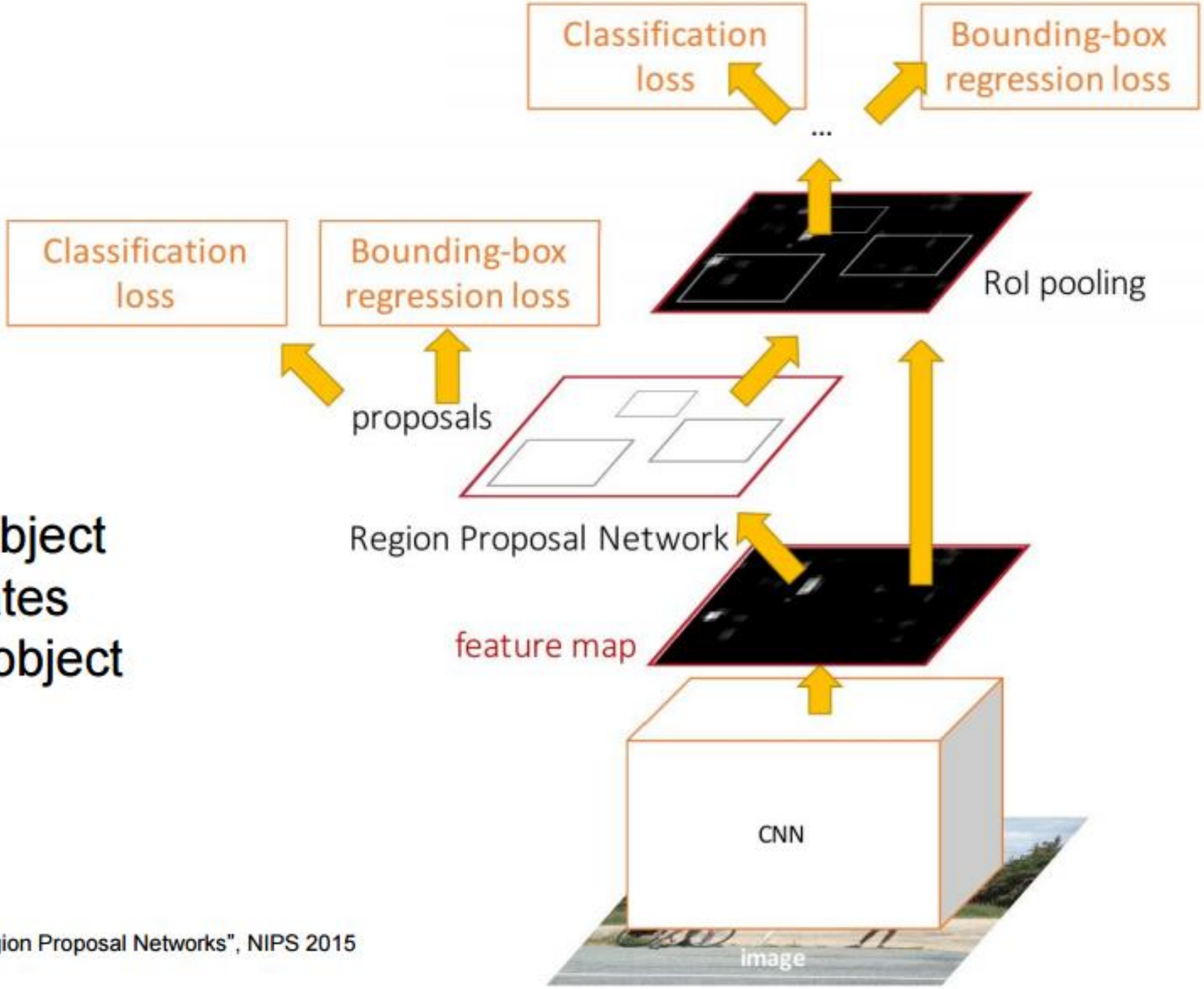
Faster R-CNN:

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

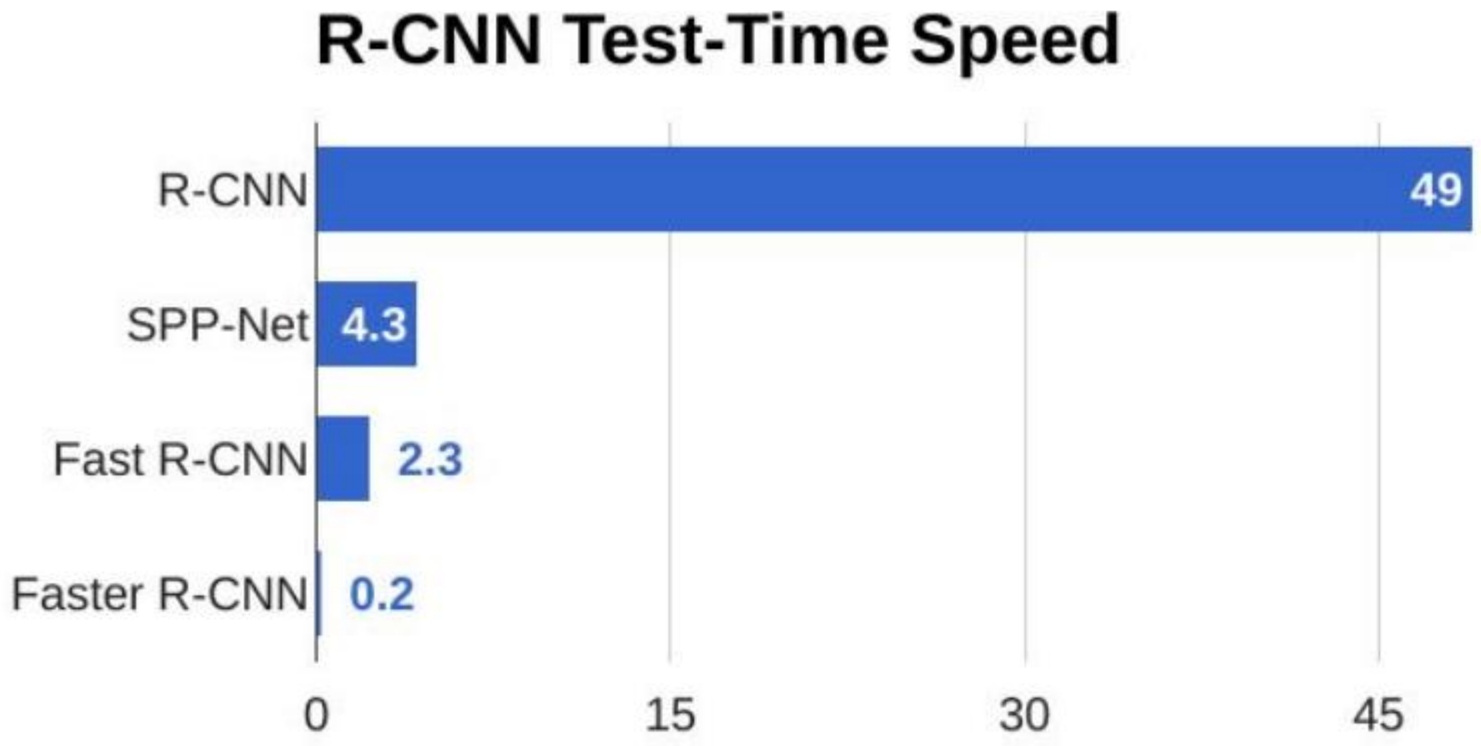
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

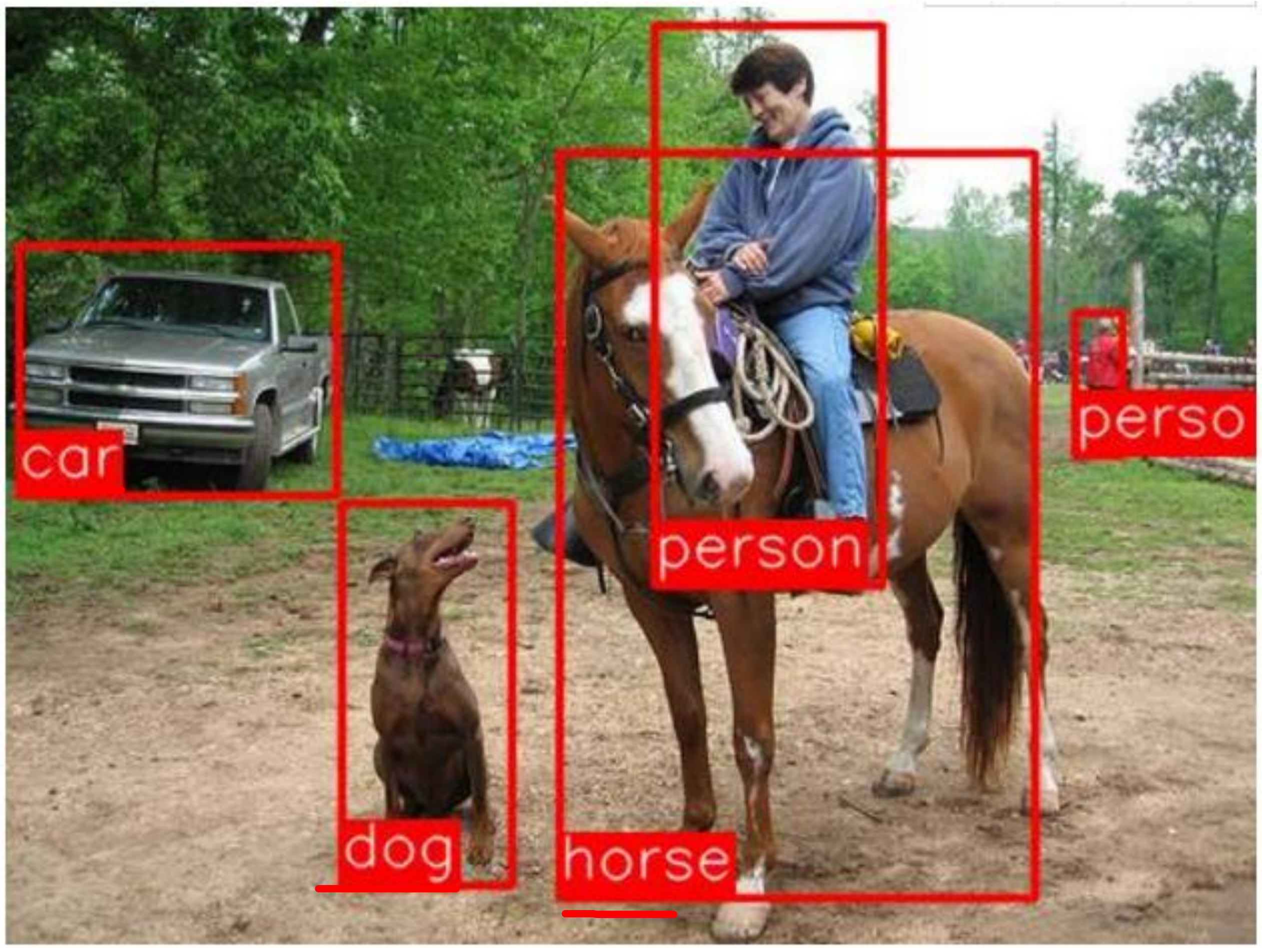
Faster R-CNN:

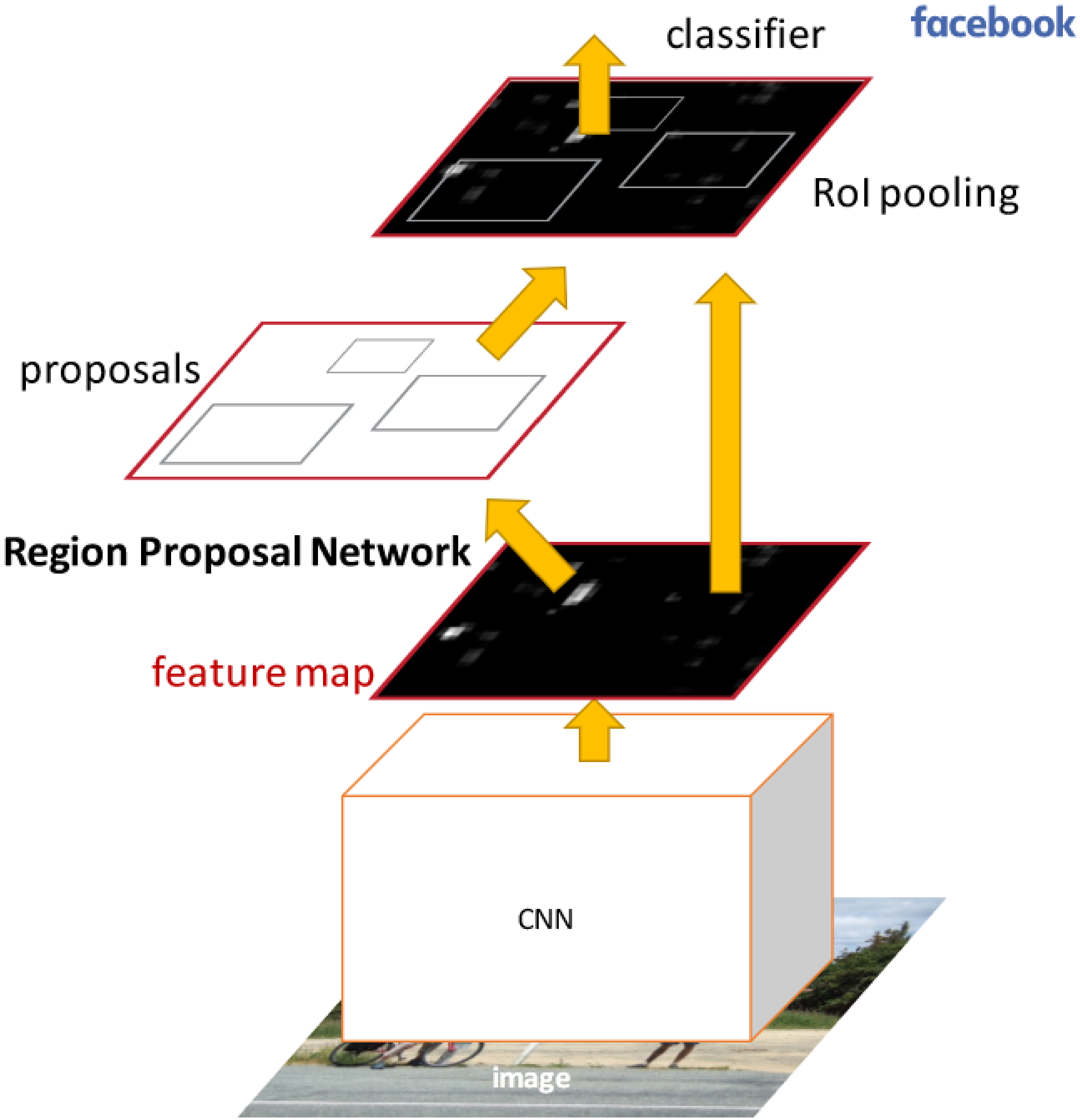
Make CNN do proposals!



FPS

- 每秒传输帧数(Frames Per Second)
- FPS”也可以理解为我们常说的“刷新率（单位为Hz）”，例如我们常在CS游戏里说的“FPS值”。我们在装机选购显卡和显示器的时候，都会注意到“刷新率”。
- 电影以每秒24张画面的速度播放，也就是一秒钟内在屏幕上连续投射出24张静止画面。有关动画播放速度的单位是fps，其中的f就是英文单词Frame（画面、帧），p就是Per（每），s就是Second（秒）。用中文表达就是多少帧每秒，或每秒多少帧。电影是24fps，通常简称为24帧。

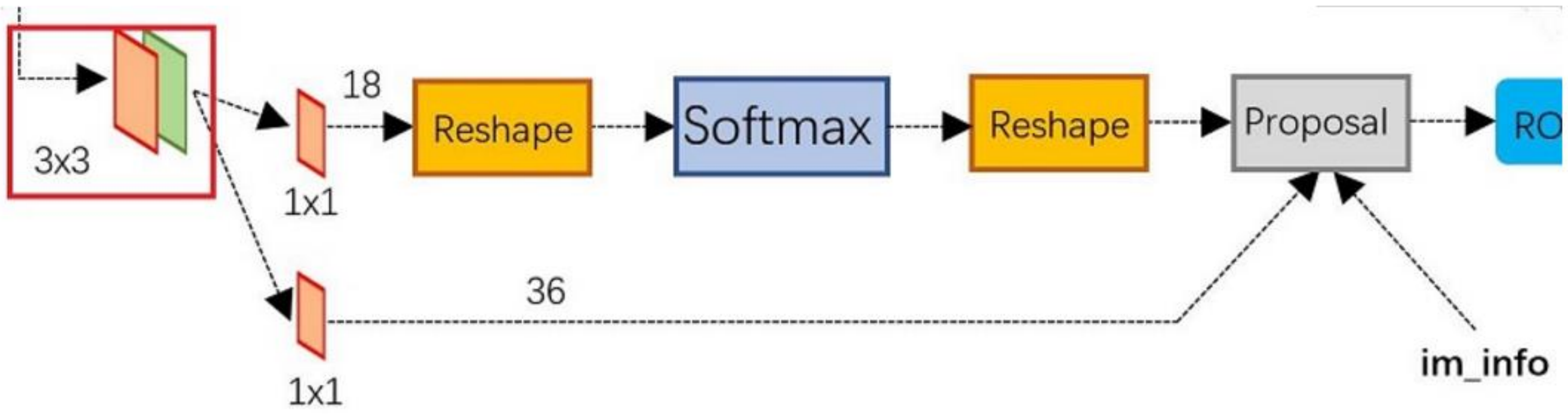

car
dog
horse
person
perso

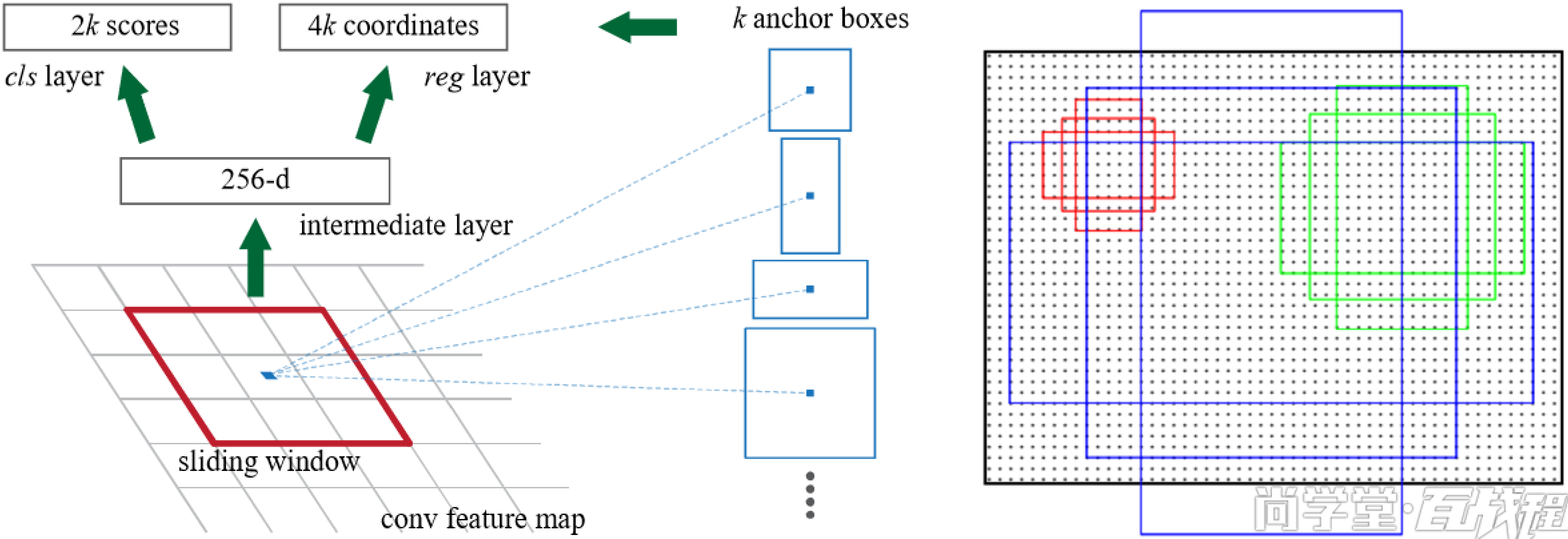

facebook
classifier
RoI pooling
proposals
Region Proposal Network
feature map
CNN
image

VGG16

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGG16


3x3
18
1x1
Reshape
Softmax
Reshape
Proposal
RO
36
1x1
im_info


2k scores
4k coordinates
cls layer
reg layer
k anchor boxes
256-d
intermediate layer
sliding window
conv feature map

anchor

- 128*128 256*256 512*512
- 1：1 2：1 1：2
- 比例只是产生框的时候初始比例，真正的框后面还会回归调整

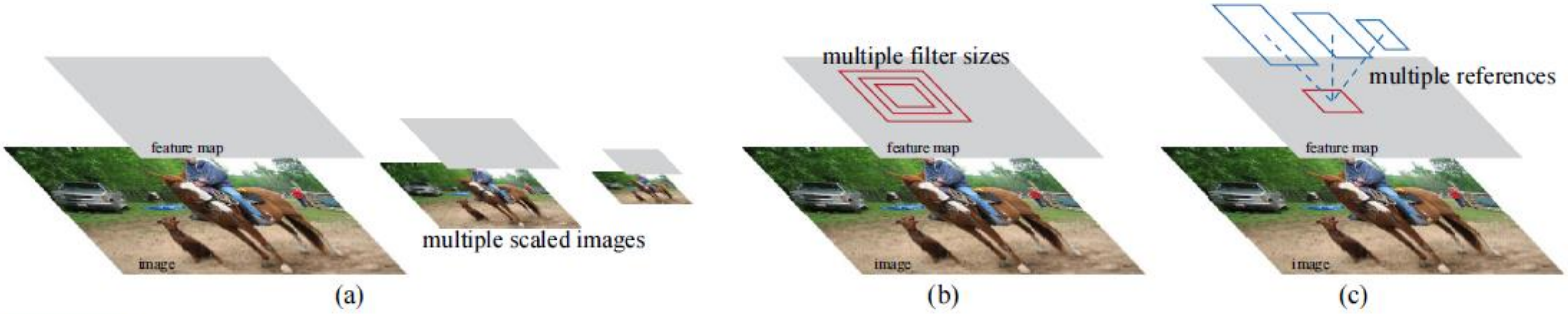
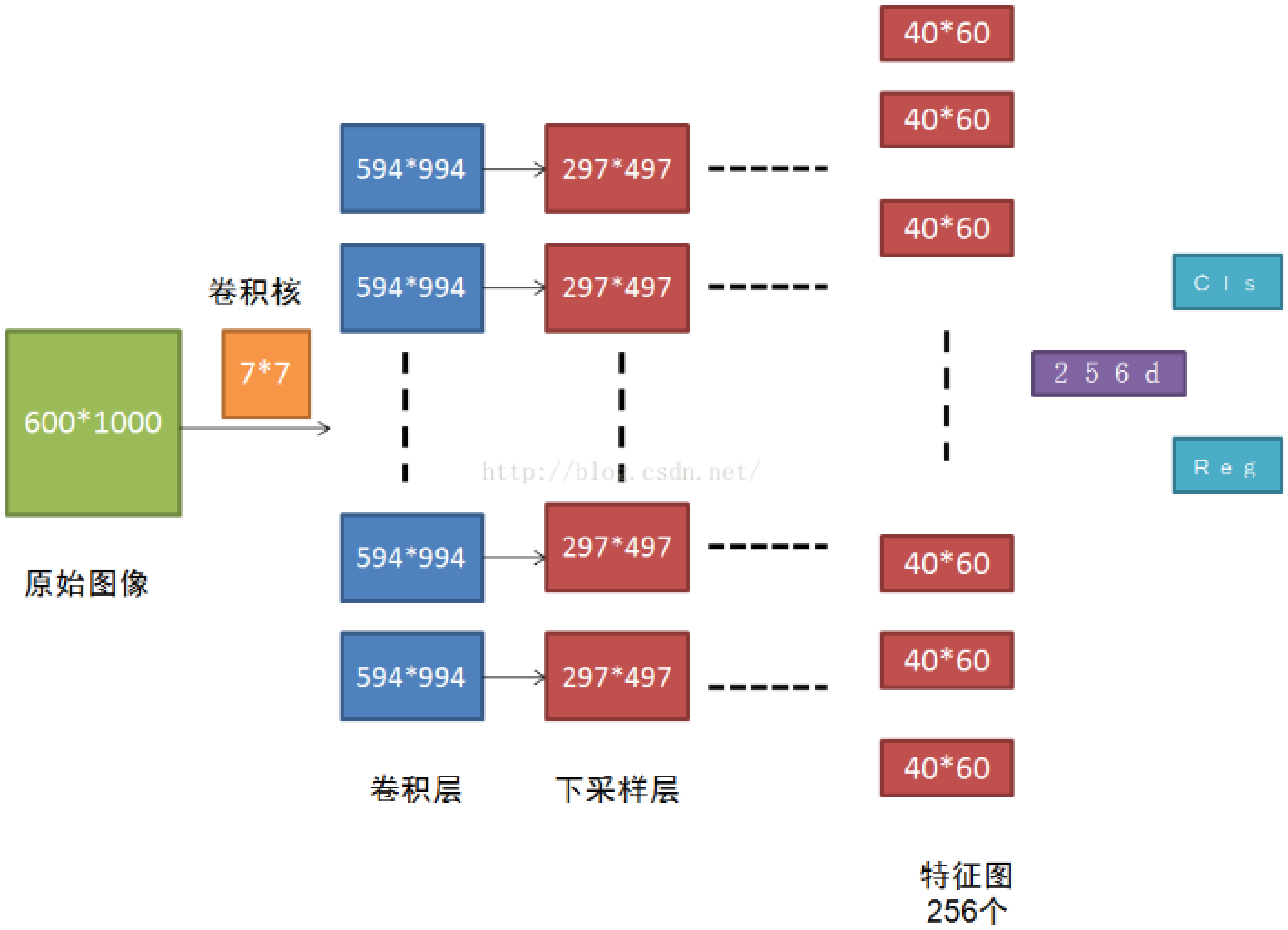


Figure 1: Different schemes for addressing multiple scales and sizes. (a) Pyramids of images and feature maps are built, and the classifier is run at all scales. (b) Pyramids of filters with multiple scales/sizes are run on the feature map. (c) We use pyramids of reference boxes in the regression functions.



原图600*1000经CNN卷积后，在CNN最后一层（conv5）得出的是40*60大小的特征图，对应文中说的典型值为2400。若特征图大小为W*H，则需要W*H*K个anchor，本文中每一个特征图需要40*60*9≈2w个。虽然一开始比较多的框，但是后面还会过滤

标记正负例标签

- 和每一个gt的重叠比例IoU最大的那个bbox是正例（一张图会有很多gt）
- 对于任意的bbox和任意gt的IoU的比例大于0.7就是正例
- 对于任意的bbox和任意gt的IoU的比例小于0.3就是负例

损失函数

- Lambda控制更重视回归还是分类
- Pi*是真实的类别标签0或1，对于回归是负例就不加调整损失了
- 一张图片有很多个anchor，用i来表示index第几个
- 早期实现及公开的代码中，λ=10，cls项的归一化值为mini-batch的大小，即Ncls=256，reg项的归一化值为anchor位置的数量，即Nreg~2,400，这样cls和reg项差不多是等权重的

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

损失函数

- 对于回归是负例就不加调整损失了，所以只有正例的才去算回归
- X预测出来的，Xa候选框的，X*是真实的框的
- 是L1 Loss，那么譬如|tx-tx*|越小，就是(X-X*)/Wa越小，即预测越接近真实的框

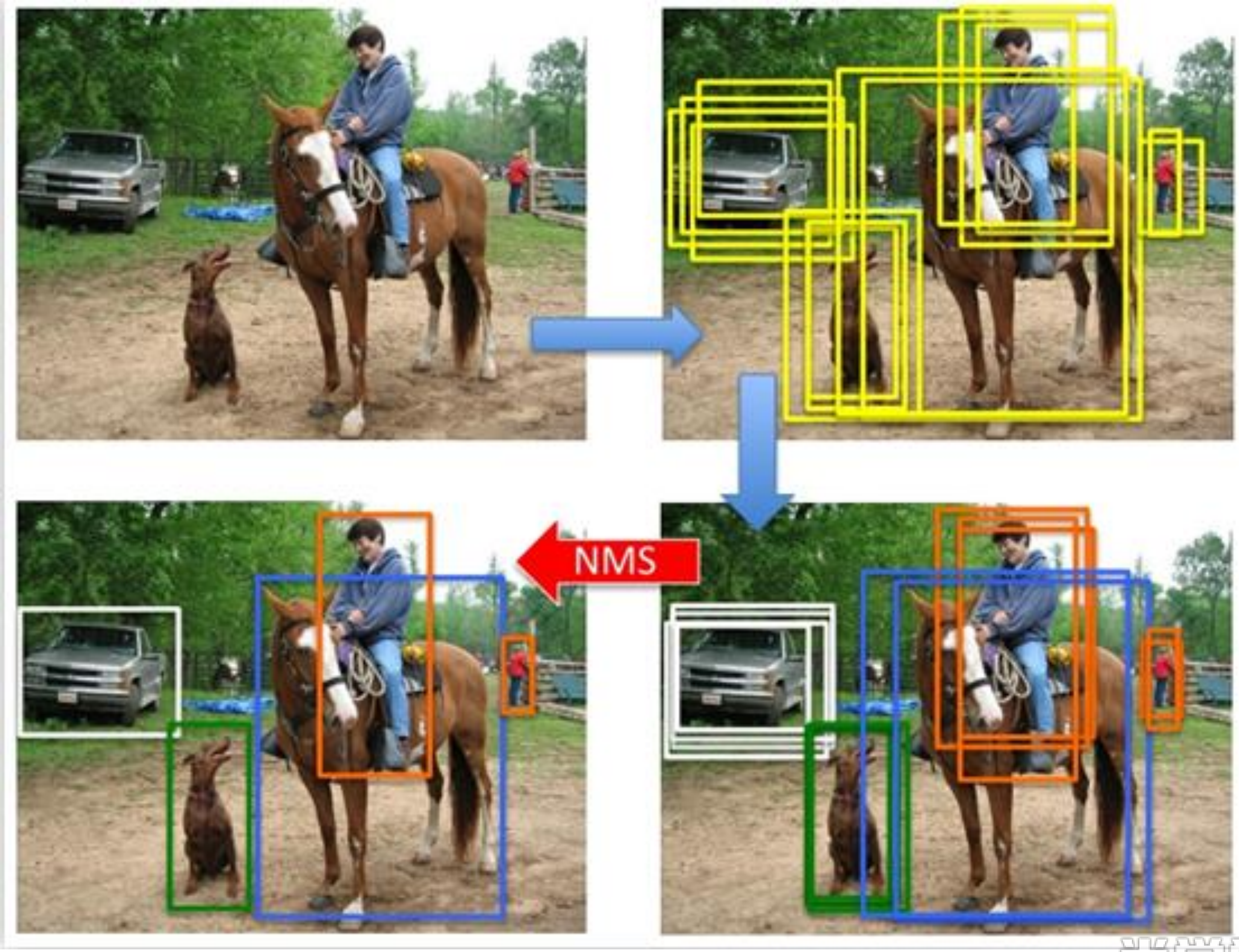
$$
\begin{aligned}
t_{\mathrm{x}} &= (x - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}} = (y - y_{\mathrm{a}})/h_{\mathrm{a}}, \\
t_{\mathrm{w}} &= \log(w/w_{\mathrm{a}}), \quad t_{\mathrm{h}} = \log(h/h_{\mathrm{a}}), \\
t_{\mathrm{x}}^{*} &= (x^{*} - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}}^{*} = (y^{*} - y_{\mathrm{a}})/h_{\mathrm{a}}, \\
t_{\mathrm{w}}^{*} &= \log(w^{*}/w_{\mathrm{a}}), \quad t_{\mathrm{h}}^{*} = \log(h^{*}/h_{\mathrm{a}}),
\end{aligned}
\tag{2}
$$

where x, y, w, and h denote the box's center coordinates and its width and height. Variables x, x_a, and x^* are for the predicted box, anchor box, and ground-truth box respectively (likewise for y, w, h). This can

每一次训练只拿一张图像

- 每一张图像进来都缩放，使得短的边是600个像素点
- 从候选框里过滤出来256个框，找128个正样本，找128个负样本，如果正样本不够，就用负样本
- 关于过滤，越界的框就不要了，60*40*9=20000.除掉越界的大概6000个，但是使用模型的时候，越界的框是clip一下后还是要的
- 6000个框还是会重叠在一起，使用NMS非极大值抑制，对于6000个候选框如果IoU的比例大于0.7就需要判断保留谁？使用分类的score看一下谁大就保留谁！
- NMS之后大概2000个，然后取一个Top-N

NMS



NMS

- 非极大值抑制的方法是：先假设有6个矩形框，根据分类器的类别分类概率做排序，假设从小到大概率分别为A、B、C、D、E、F
- (1)从最大概率矩形框F开始，分别判断A~E与F的重叠度IOU是否大于某个设定的阈值;
- (2)假设B、D与F的重叠度超过阈值，那么就扔掉B、D；并标记第一个矩形框F，是我们保留下来的。
- (3)从剩下的矩形框A、C、E中，选择概率最大的E，然后判断E与A、C的重叠度，重叠度大于一定的阈值，那么就扔掉；并标记E是我们保留下来的第二个矩形框。
- 就这样一直重复，找到所有被保留下来的矩形框。

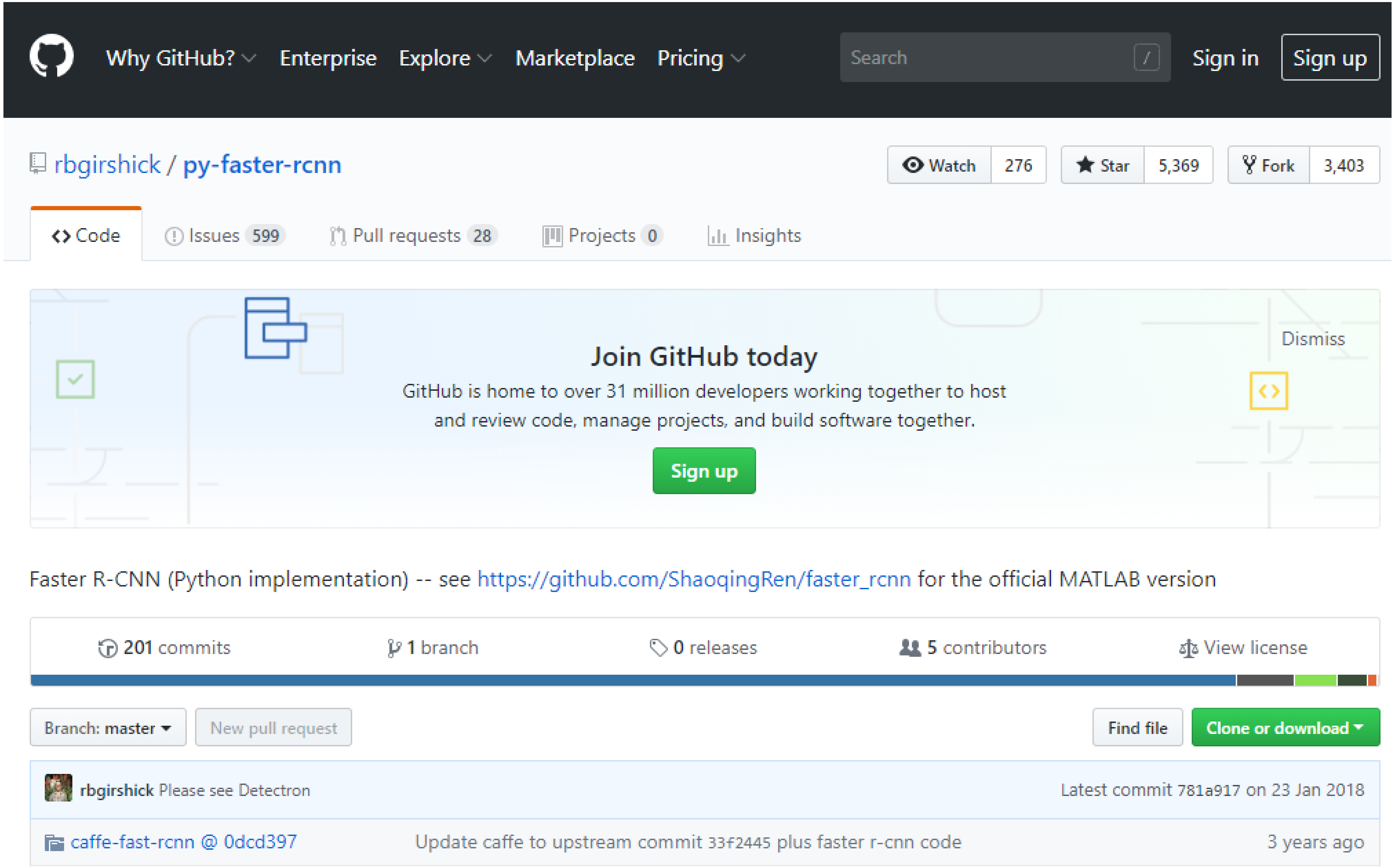
```python
def py_cpu_nms(dets, thresh):
    """Pure Python NMS baseline."""
    #x1、y1、x2、y2、以及score赋值
    x1 = dets[:, 0]
    y1 = dets[:, 1]
    x2 = dets[:, 2]
    y2 = dets[:, 3]
    scores = dets[:, 4]

    #每一个检测框的面积
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    #按照score置信度降序排序
    order = scores.argsort()[::-1]

    keep = [] #保留的结果框集合
```

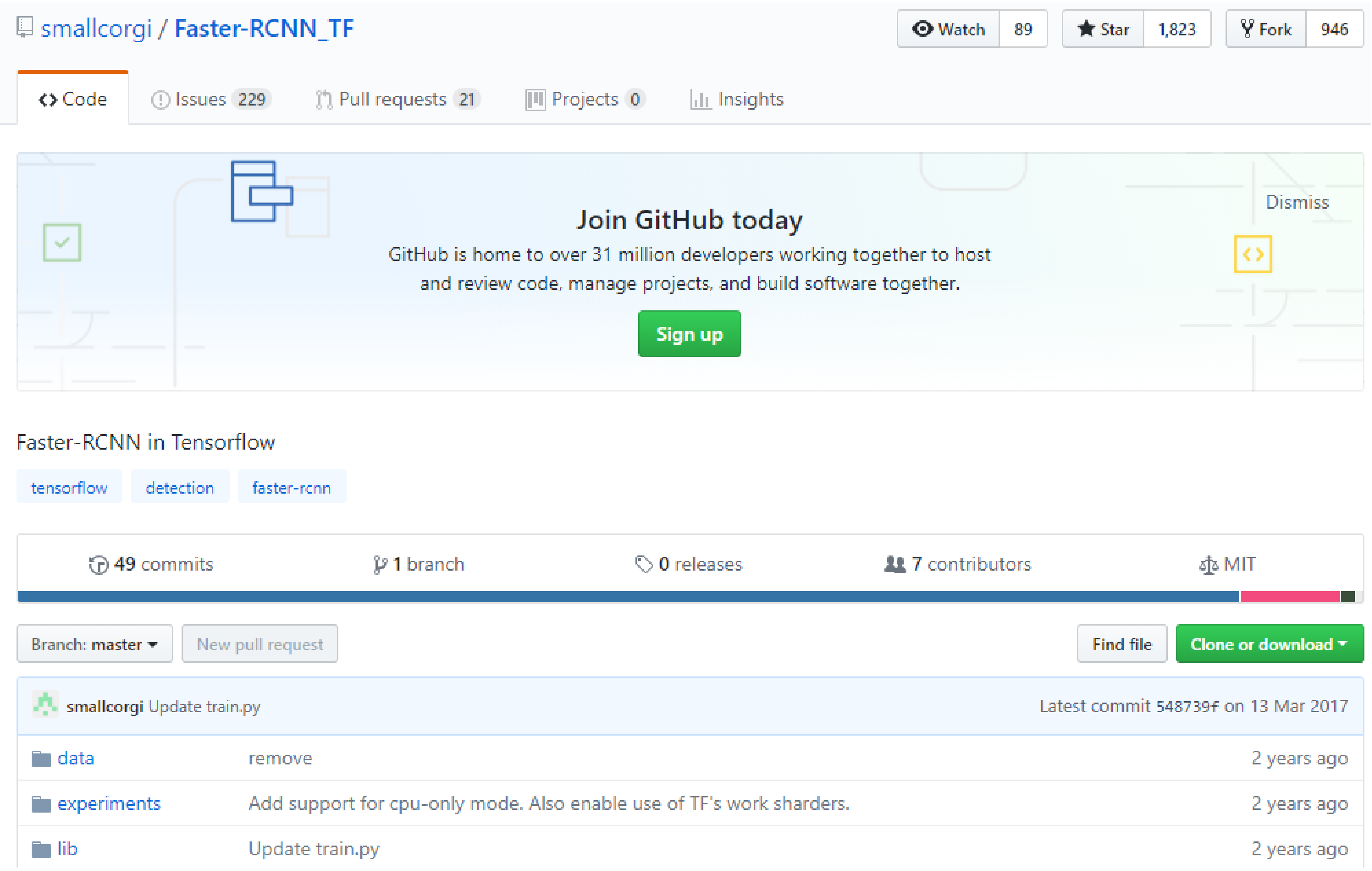
```python
while order.size > 0:
    i = order[0]
    keep.append(i) #保留该类剩余box中得分最高的一个
    #得到相交区域,左上及右下
    xx1 = np.maximum(x1[i], x1[order[1:]])
    yy1 = np.maximum(y1[i], y1[order[1:]])
    xx2 = np.minimum(x2[i], x2[order[1:]])
    yy2 = np.minimum(y2[i], y2[order[1:]])

    #计算相交的面积,不重叠时面积为0
    w = np.maximum(0.0, xx2 - xx1 + 1)
    h = np.maximum(0.0, yy2 - yy1 + 1)
    inter = w * h
    #计算IoU:重叠面积 /(面积1+面积2-重叠面积)
    ovr = inter / (areas[i] + areas[order[1:]] - inter)
    #保留IoU小于阈值的box
    inds = np.where(ovr <= thresh)[0]
    order = order[inds + 1] #因为ovr数组的长度比order数组少一个,所以这里要将所有下标后移一位

return keep
```

mAP

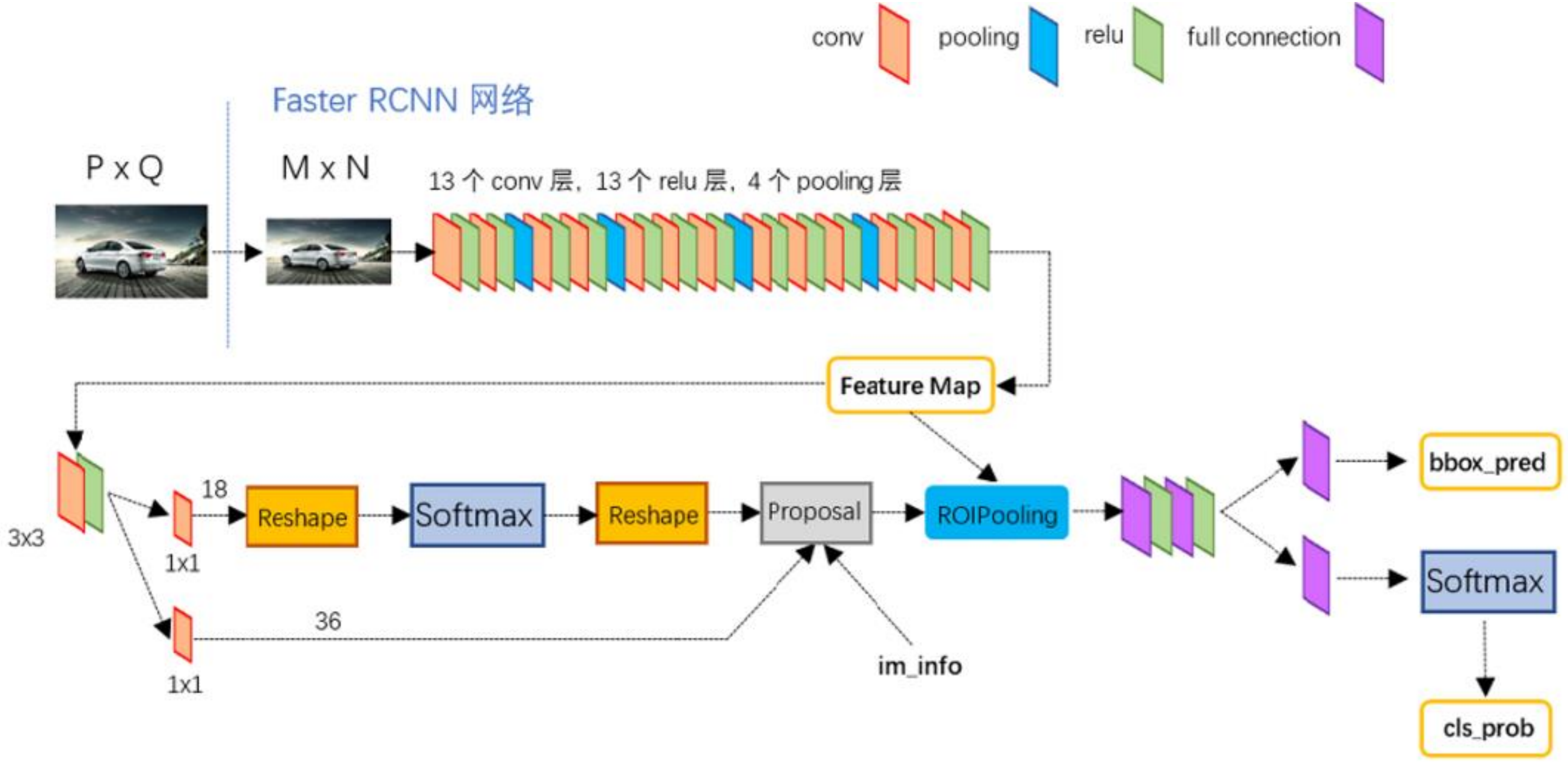
- M个precision值取平均即得到最后的AP值

top-N	Precision	Recall(r)	Max Precision for Any Recall r' >= r	Average Precision
1	1/1	1/6	1	0.6621
2	2/2	2/6	1	
3	2/3			
4	2/4			
5	2/5			
6	3/6	3/6	4/7	
7	4/7	4/6	4/7	
8	4/8			
9	4/9			
10	4/10			
11	5/11	5/6	5/11	
12	5/12			
13	5/13			
14	5/14			
15	5/15			
16	6/16	6/6	6/16	
17	6/17			
18	6/18			
19	6/19			
20	6/20			


Why GitHub?
Enterprise
Explore
Marketplace
Pricing
Search
Sign in
Sign up
rbgirshick / py-faster-rcnn
Watch 276
Star 5,369
Fork 3,403
Code
Issues 599
Pull requests 28
Projects 0
Insights
Dismiss
Join GitHub today
GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.
Sign up
Faster R-CNN (Python implementation) -- see https://github.com/ShaoqingRen/faster_rcnn for the official MATLAB version
201 commits
1 branch
0 releases
5 contributors
View license
Branch: master
New pull request
Find file
Clone or download
rbgirshick Please see Detectron
Latest commit 781a917 on 23 Jan 2018
caffe-fast-rcnn @ 0dcd397
Update caffe to upstream commit 33f2445 plus faster r-cnn code
3 years ago


smallcorgi / Faster-RCNN_TF
Watch 89
Star 1,823
Fork 946
Code
Issues 229
Pull requests 21
Projects 0
Insights
Join GitHub today
GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.
Sign up
Dismiss
Faster-RCNN in Tensorflow
tensorflow
detection
faster-rcnn
49 commits
1 branch
0 releases
7 contributors
MIT
Branch: master
New pull request
Find file
Clone or download
smallcorgi Update train.py
Latest commit 548739f on 13 Mar 2017
data
remove
2 years ago
experiments
Add support for cpu-only mode. Also enable use of TF's work sharders.
2 years ago
lib
Update train.py
2 years ago

代码剖析


conv
pooling
relu
full connection
Faster RCNN 网络
P x Q
M x N
13 个 conv 层，13 个 relu 层，4 个 pooling 层
Feature Map
3x3
18
1x1
Reshape
Softmax
Reshape
Proposal
ROIPooling
36
1x1
im_info
bbox_pred
Softmax
cls_prob