



contents

- 1. Detection Algorithms
 - 1.1 Object Localization
 - 1.2 Landmark Detection
 - 1.3 Object Detection
 - 1.4 Convolution Implementation of Sliding Windows
 - 1.4.1 将fc转换为convoluton
 - 1.4.2 OverFeat(2014)
 - 1.5 Bounding Box Predictions
 - 1.6 Intersection Over Union
 - 1.7 Non-max Suppression
 - 1.8 Anchor Boxes
 - 1.9 YOLO Algorithm
 - 1.10 Region Proposals

1. Detection Algorithms

1.1 Object Localization

classification with localization通常指只识别并定位一个object，而detection指的是从图像中识别并定位多个不同类别的objects。

classification with localization一方面需要输出一个label(例如图中的4分类)，另一方面还要输出4个浮点数(b_x, b_y, b_h, b_w)，其中， (b_x, b_y) 是中心点的坐标。注意：图片的左上角坐标为(0,0)，右下角标记为(1,1)。

如果类别是1,2,3之一， $p_c = 1$ ，如果类别是4(不是要检测的object)， $p_c = 0$ 。而输出除了 $[p_c, b_x, b_y, b_h, b_w]$ 之外，还应该包括 $[C_1, C_2, C_3]$ 也就是这三类的一个one-hot。注意这里假设的是只有一个object，所以只有一个坐标。

如左图，是有object的情况。如右图，没有object的时候， $p_c = 0$ ，其他元素是什么是无所谓的啦。

损失函数，如果采用mse（均方误差）的话：

$$L(\hat{y}, y) = \begin{cases} \sum_{i=1}^8 (\hat{y}_i - y_i)^2 & \text{if } y_1 = 1 \\ \frac{\hat{y}_1}{y_1} & \text{if } y_1 = 0 \end{cases}$$

当然，这里以mse举例只是为了简化描述。在实践中，也可以对不同部分用不同的loss，例如，

- 对 p_c 用logistic regression loss
- 对 b_x, b_y, b_h, b_w 用mse
- 对 c_1, c_2, c_3 的softmax输出，可以用log likelihood loss

Defining the target label y

1 - pedestrian
2 - car ←
3 - motorcycle
4 - background ←

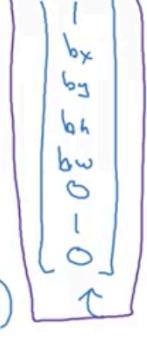
Need to output b_x, b_y, b_h, b_w , class label (1-4)

$\ell(\hat{y}, y) =$

$$\begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$

$x =$ 

(x, y) 



"don't care"

Andrew Ng

11:14 / 11:54

1.2 Landmark Detection

landmark指的是图中的重要的点，例如对于人脸识别的任务而言，可能眼角的坐标就是一个landmark。当然，还可以有一系列的landmark，例如有64个landmark，那就有128+1个输出元素。基于这些Landmark，可以做一些例如表情识别、戴帽子、面部变形等特效。

而对于pose detection的任务，也有很多类似的landmark。

1.3 Object Detection

sliding window detection方法：

例如，针对一个定位car的任务。

训练集，对原始图进行裁图，裁出的图保证car在正中间，而且基本占满整张图，图的尺寸就是window的大小。

然后对原图，拿滑动窗口慢慢滑动，每一个小区域过一遍convnet，看看是否是car。这算做一次slide。

然后把窗口调大，窗口里的图resize一下丢给同一个convnet，再看每一块是否是car。

再用更大的window，再来一次。

缺点：计算成本大。所以在还不是dnn的时代，常用比较简单的分类器，例如线性分类器来识别物体，然后走滑动窗口

1.4 Convolution Implementation of Sliding Windows

1.4.1 将fc转换为convoluton

思想来自《[Fully Convolutional Networks for Semantic Segmentation](#)》，即FCN。

- 输入是 $14 \times 14 \times 3$
- 经过16个 5×5 的filter，得到 $10 \times 10 \times 16$ ，其中， $10 = 14 - 5 + 1$
- 经过一个 2×2 的maxpooling， $s=2$, 得到 $5 \times 5 \times 16$
- 然后flatten成一个400维的向量
- 然后接一个400维的fc
- 再接一个400维的fc
- 最后输出一个4维的softmax

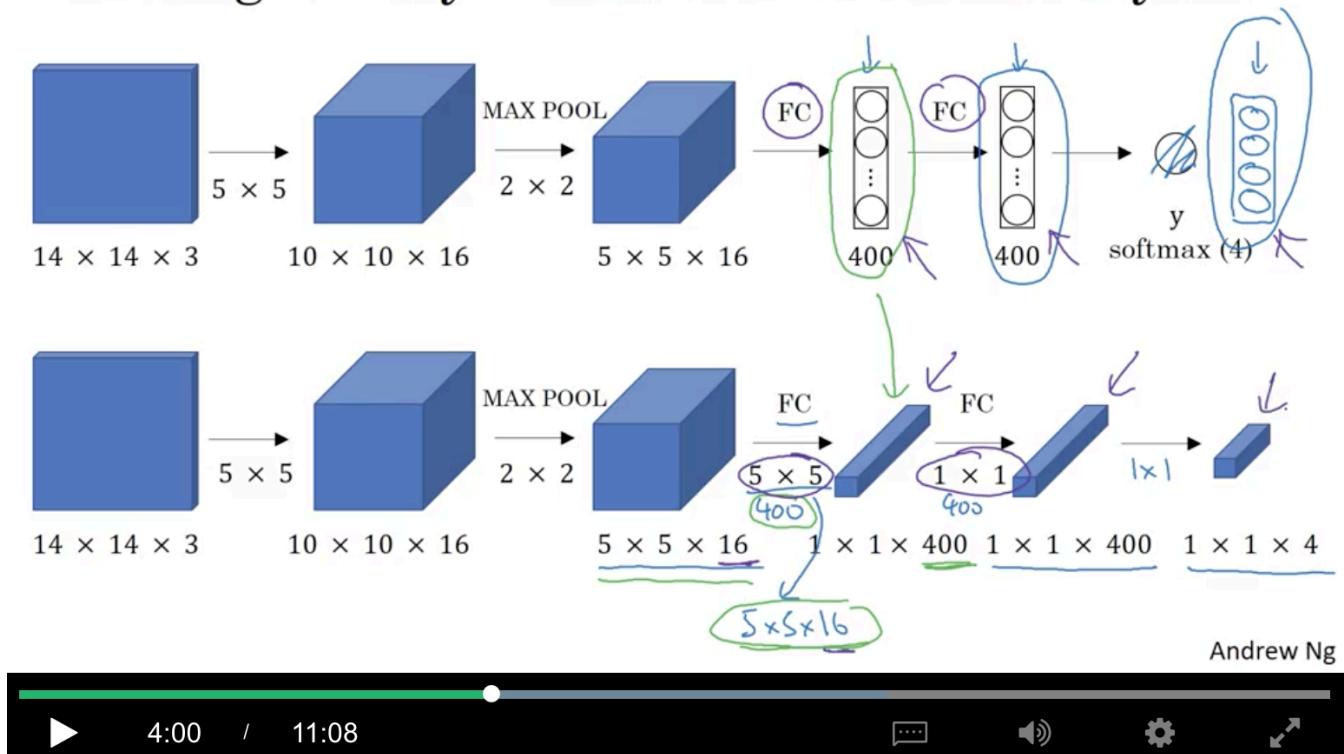
针对上面那个图，前面的conv+pool不变，我们把后面的变成卷积：

- 输入是 $5 \times 5 \times 16$ ，用400个 5×5 的filter(实际是400个 $5 \times 5 \times 16$)，得到400个 1×1 的结果，

看成 $1 \times 1 \times 400$ 的volume。【解释一下，这个操作相当于两个 $5 \times 5 \times 16$ 的东西对应相乘相加，得到一个点，然后有400个filter，就有400个点；而如果接一个fc的话，相当于一个400维的向量，经过一个 400×400 的矩阵，得到一个400维的向量，第一个元素就是原来的400维向量，和这个 400×400 的矩阵的第一列对应元素相乘再相加得到的，然后有400列，就有400个点】

- 对于第二个fc，同样地，经过400个 $1 \times 1 \times 400$ 的filter，就可以得到一个 $1 \times 1 \times 400$ 的结果
- 最后接4个 $1 \times 1 \times 400$ 的filter，替换掉softmax，得到 $1 \times 1 \times 4$ 的输出

Turning FC layer into convolutional layers



1.4.2 OverFeat(2014)

2014年的Lecun的OverFeat这篇文章中，将sliding windows转换为卷积操作。

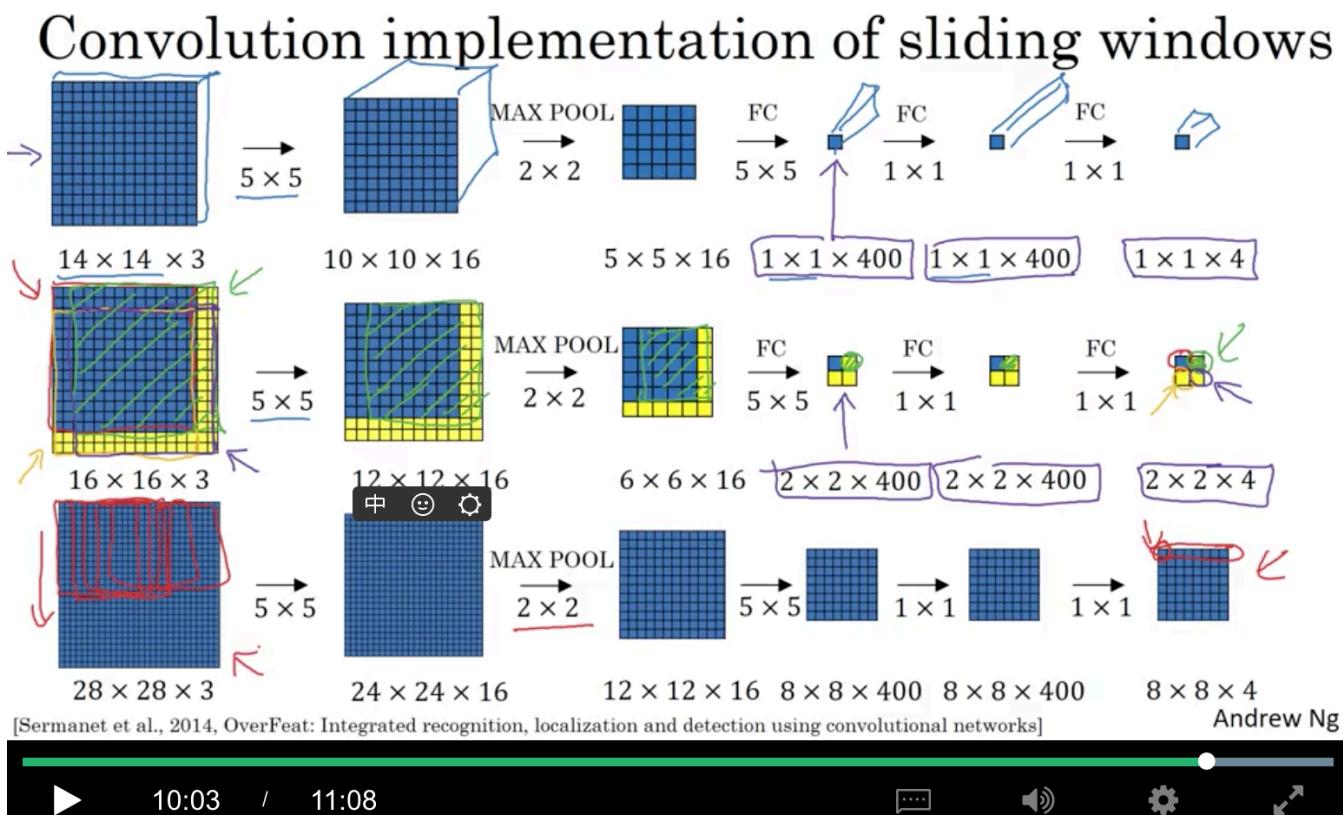
例如，训练集是 $14 \times 14 \times 3$ ，即window的大小是 14×14 ，如果测试集是 $16 \times 16 \times 3$ ，那么，例如一次滑动两格，那就需要过4次convnet，而这4次其实有大量的运算是重复的，可以通过convolution化来进行参数共享。具体如下，可以拿这个 $16 \times 16 \times 3$ 的图，经过和convnet一样的网络结构：

- 通过16个 5×5 的filter，得到 $12 \times 12 \times 16$ ，其中， $12 = 16 - 5 + 1$
- 通过16个 2×2 的maxpooling， $s=2$ ，得到 $6 \times 6 \times 16$

- 通过400个 5×5 的filter，得到 $2 \times 2 \times 400$
- 通过400个 1×1 的filter，得到 $2 \times 2 \times 400$
- 通过4个 1×1 的filter，得到 $2 \times 2 \times 4$

可以发现，最终的这个 $2 \times 2 \times 4$ ，左上角的 $1 \times 1 \times 4$ 就是原图左上角经过convnet得到的结果，右上角、左下角、右下角的类推！！

如果输入图是 $28 \times 28 \times 3$ ，最终输出的是 $8 \times 8 \times 4$ 。得到的最终结果也是滑动窗口的步长=2的结果。这是因为maxpooling使用的是 2×2 ，所以相当于在原来的图像上以步长为2运行。(再想想...是因为 2×2 还是因为stride=2呢)



但有个问题就是，bounding box并不精准，下一节来解决咯~

1.5 Bounding Box Predictions

比较有效的精确地画出bounding box的算法是YOLO算法（You Only Look Once, 2015）

- 首先对图片网格(grid)化，例如，使用一个 3×3 的网格。
- 对于每一个grid cell，输出一个8维的向量 $[p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3]$

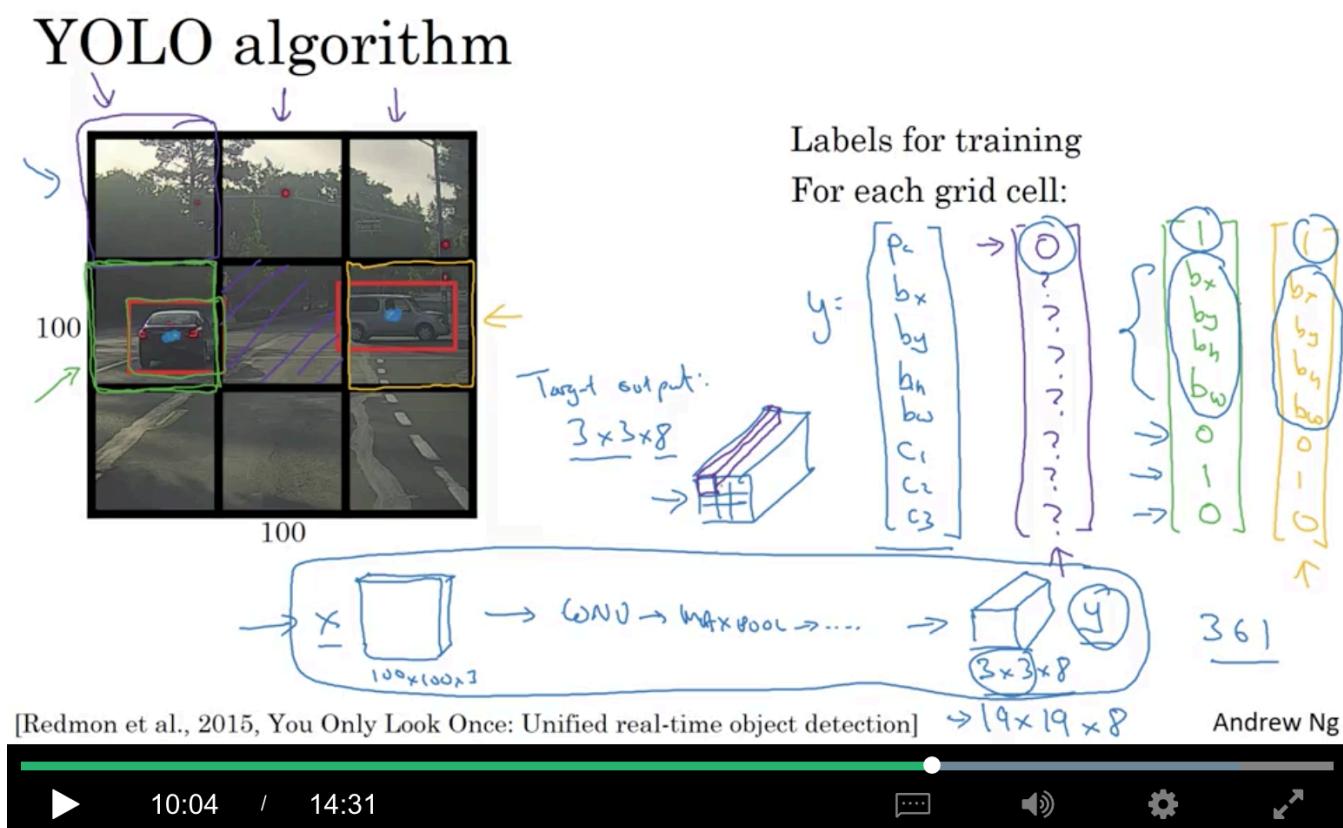
- 对于没有物体的grid cell，就是 $[0, ?, ?, ?, ?, ?, ?, ?, ?]$
- 对于中间那行，左右两个有car的中心点，所以 $p_c = 1$ ，而中间那个grid cell，虽然有一些边界，但没有中心点，所以仍然 $p_c = 0$ 。

因此，输入是 $100 \times 100 \times 3$ ，中间经过若干cnn后，输出的label就是 $3 \times 3 \times 8$ 。

实践中，往往会使用更大的grid数，例如 19×19 而不是 3×3 ，这样可以降低一个grid中有多个中心点的概率。

因为是全卷积实现的，所以速度非常快，甚至可以用在实时的目标检测上呢~

yolo-algorithm.png



对于图中的黄色的grid cell，假定左上角是 $(0,0)$ ，右下角是 $(1,1)$ ，那么 b_x, b_y 肯定都是 $0-1$ 间的float，但 b_h, b_w 可以是大于1的，比如有的车很大。

在YOLO的文章中，还有其他效果也不错的参数化方法(例如，通过sigmoid来保证在 $0-1$ 之间，用指数函数来保证非负)，但上面讲到的这种比较合理，而且应用很广。

附最新的yolov3的keras实现：<https://github.com/qzwweee/keras-yolo3>

c++版本的darknet：<https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection>

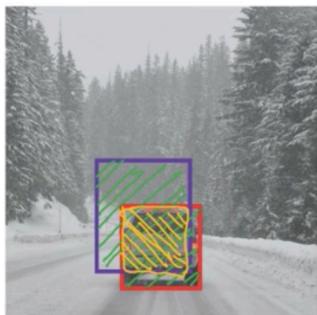
1.6 Intersection Over Union

object localization的评估指标，其实也是计算两个框的相似程度：

例如，真实的Bounding box是图中红色框，但算法预测出来是紫色的框，这个时候Intersection Over Union(IoU)就表示两个框的交集除以并集的比率。

惯例上，IoU ≥ 0.5 ，就算作correct。当然更严格的阈值也可以。

Evaluating object localization



$$\text{Intersection over Union (IoU)} = \frac{\text{Size of intersection}}{\text{Size of union}}$$

"Correct" if $\text{IoU} \geq 0.5$ ←
0.6 ←

More generally, IoU is a measure of the overlap between two bounding boxes.

Andrew Ng



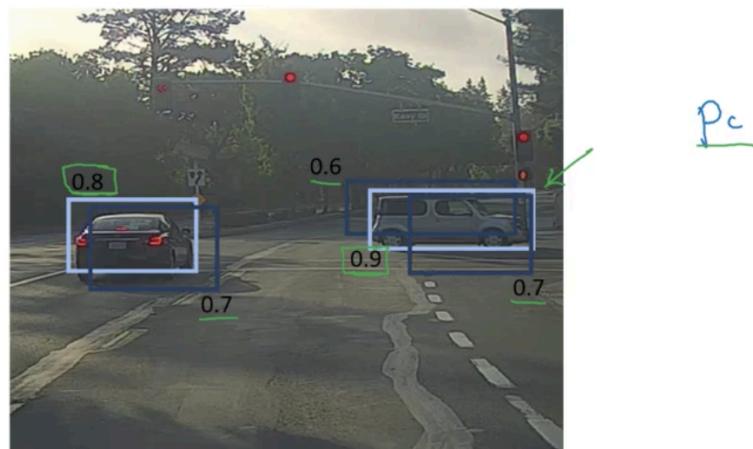
1.7 Non-max Suppression

non-max suppression(非最大值抑制)

如图，理论上应该一个中心点只属于一个grid cell，但在运行的过程中，可能多个grid cell都认为自己应该拥有这个中心点。

- 第一步，就是在这多个bounding box中，选出 p_c 最大的那个，将它变亮（如图中右边0.9的那个box）
- 然后对于那些与这个变亮的有很高IoU的box，变暗（即suppressed）
- 然后找到第二个 p_c 最大的，如图中左边那个0.8
- 然后把与它有高IoU的也变暗咯

Non-max suppression example



实现细节：

假设只要识别car，这样就可以不要 c_1, c_2, c_3 了

- 对于 $19 \times 19 = 361$ 个grid cell，会输出一个 $[p_c, b_x, b_y, b_h, b_w]$ 的向量
- 首先扔掉所有 $p_c \leq 0.6$ 的box
- 遍历所有剩下的box
 - 选择最大的 p_c ，当做预测值
 - 把与上一步有 $IoU \geq 0.5$ 的box扔掉

如果有3个类别， c_1, c_2, c_3 ，则需要对每个类别独立地进行一次如上操作。

1.8 Anchor Boxes

如果一个grid cell想要检测多个object，那就要用到anchor boxes。

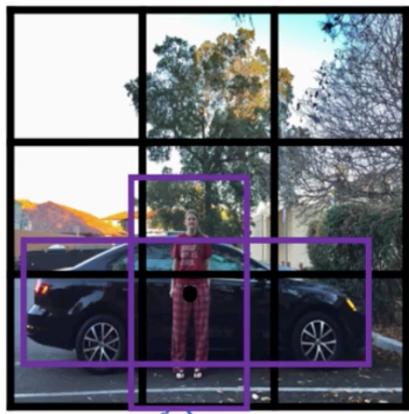
例如，一个grid cell可能即是行人的中心点，又是car的中心点，这个时候搞两个anchor box，把原来的输出由8维改成16维，前8维表示是否是第一个anchor box，后8维表示是否是第二个anchor box，衡量是否是某个anchor box的时候，会算IoU，太小的话就不算咯

如图，第一个y是既有人也有车的例子，如果只有车，就是第二个y

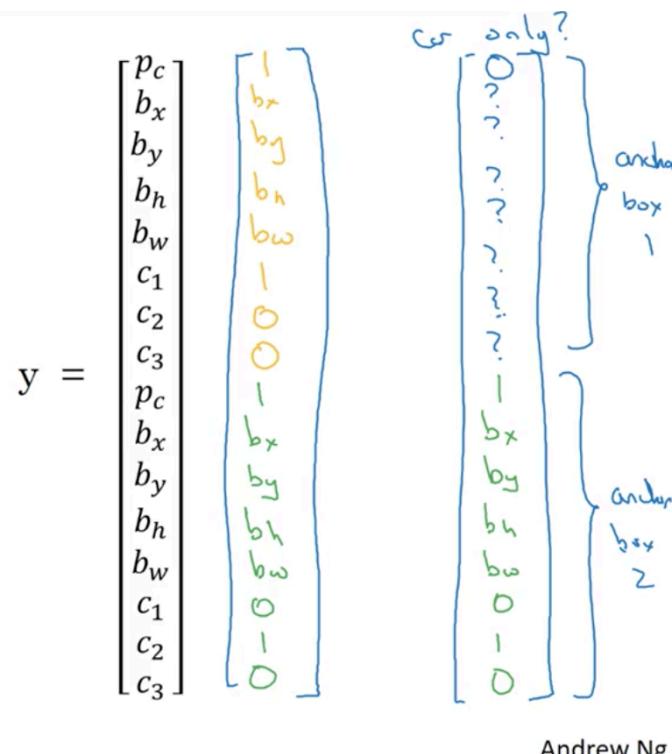
- 如果一个grid cell里有3个东西，这种算法解决不了
- 如果一个grid cell里有2个东西，他们的anchor box很像，这种算法也解决不了

对于上面的情况，需要做一些tie-breaking(挑选机制)

Anchor box example



Anchor box 1: Anchor box 2:



当然，如果格子是19x19，同一个grid cell是两个东西的中心点的概率远比3x3要小得多。

如何选择anchor box呢？传统的方法是手动挑选，5-10个不同形状的anchor box

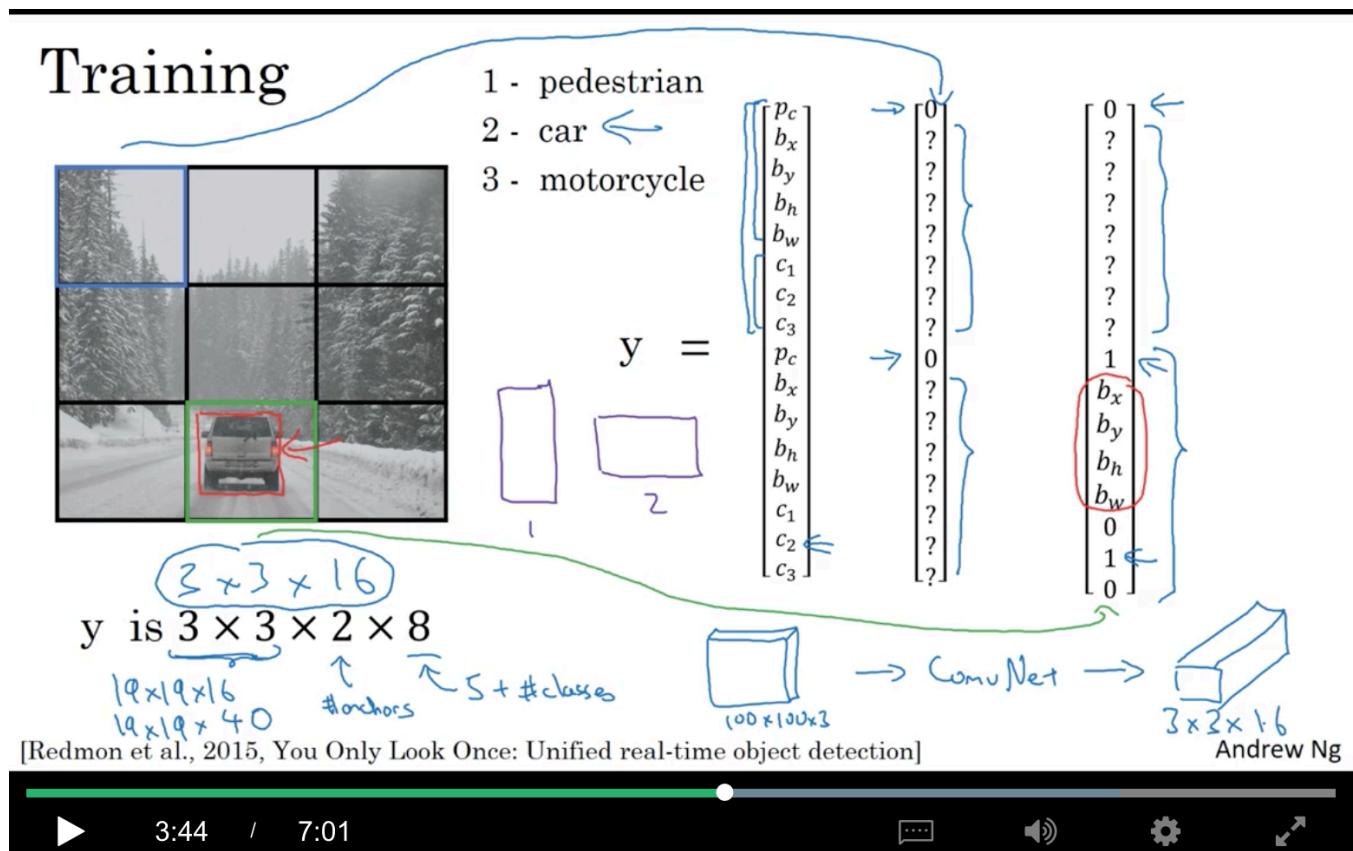
最近的yolo文章中，会用k-means，to group together two types of objects shapes you tend to

get. And then to use that to select a set of anchor boxes that this most stereotypically representative of the maybe multiple, of the maybe dozens of object classes you're trying to detect.

1.9 YOLO Algorithm

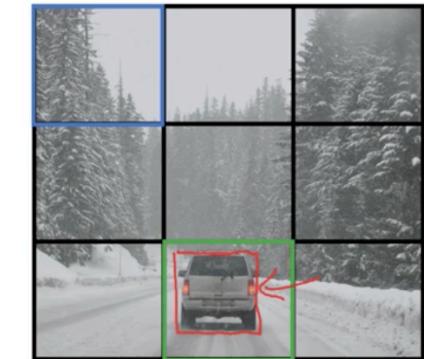
训练过程如下：

注： $8 = 1 + 4 + \# \text{classes}$



预测过程如下：

Training



- 1 - pedestrian
- 2 - car
- 3 - motorcycle

$$y =$$



p_c	0
b_x	?
b_y	?
b_h	?
b_w	?
c_1	?
c_2	?
c_3	?
p_c	0
b_x	?
b_y	?
b_h	?
b_w	?
c_1	?
c_2	?
c_3	?

0	0
?	?
?	?
?	?
?	?
?	?
?	?
?	?
?	?
?	?
?	?
1	1
b_x	1
b_y	1
b_h	1
b_w	1
0	0
1	1
0	0

$$y \text{ is } 3 \times 3 \times 2 \times 8$$

$$19 \times 19 \times 16$$

$$19 \times 19 \times 40$$

7:01

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]



→ ConvNet →



Andrew Ng

通过non-max suppression找到最终的box(图中的grid call是笔误，应该是grid cell)

Outputting the non-max suppressed outputs



- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Andrew Ng

6:22 / 7:01

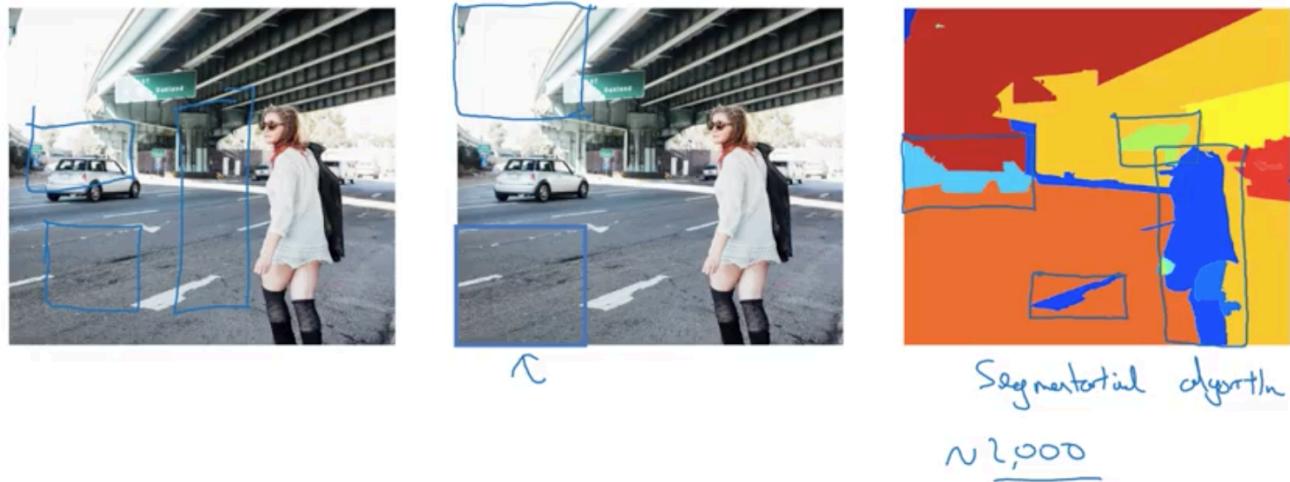
1.10 Region Proposals

region proposals(候选区域)

前面提到的sliding windows，有一个缺点就是，它会去辨别很多显然没有object的区域，因此2013年提出了R-CNN(regions with CNNs)

识别region的方法就是segmentation，如最右图，可以找到大约2000个Blocks，然后为他们分别放置一个bounding box，然后在它们上面跑分类器。这样，你的convnet就不仅是跑正方形的区域了，还可以跑矩形的。

Region proposal: R-CNN



[Girshik et. al, 2013, Rich feature hierarchies for accurate object detection and semantic segmentation] Andrew Ng



- R-CNN (2013)：找到regions，然后一次只对一个region进行分类，输出label+bounding box
- fast R-CNN(2015)：找到regions，然后使用卷积化的滑动窗口，对所有regions进行一次分类
- faster R-CNN(2016)：使用CNN而非传统的segmentation算法来找到regions。（不过据说还是没有YOLO快。。。）