

Contents

1	1. Detection Algorithms	1
1.1	1.1 Object Localization	1
1.2	1.2 Landmark Detection	2
1.3	1.3 Object Detection	2
1.4	1.4 Convolution Implementation of Sliding Windows	3
1.4.1	1.4.1 将 fc 转换为 convoluton	3
1.4.2	1.4.2 OverFeat(2014)	3
1.5	1.5 Bounding Box Predictions	4
1.6	1.6 Intersection Over Union	5
1.7	1.7 Non-max Suppression	5
1.8	1.8 Anchor Boxes	6
1.9	1.9 YOLO Algorithm	8
1.10	1.10 Region Proposals	8

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 1. Detection Algorithms

1.1 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:

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

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)

$\hat{y}_i =$

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

$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ is there an object?

(x, y)

Figure 1: object-localization.png

1.2 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 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 1.4 Convolution Implementation of Sliding Windows

1.4.1 1.4.1 将 fc 转换为 convolution

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

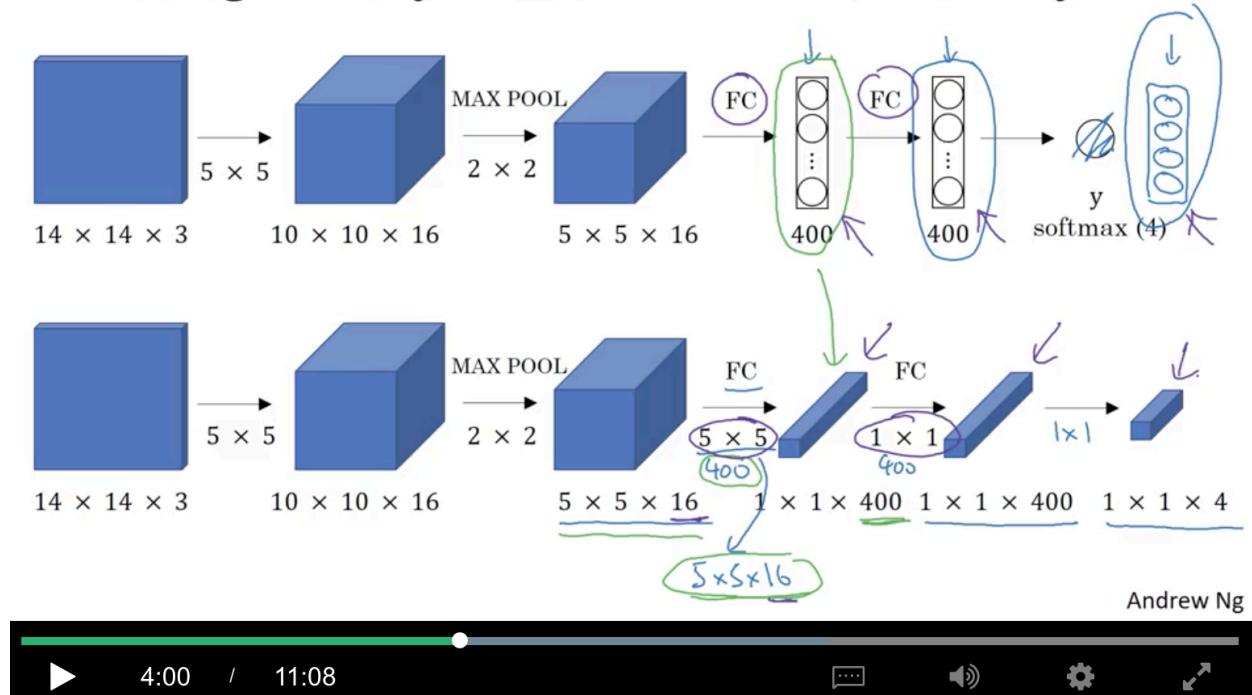


Figure 2: fc-to-conv.png

1.4.2 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 $\text{stride}=2$ 呢）

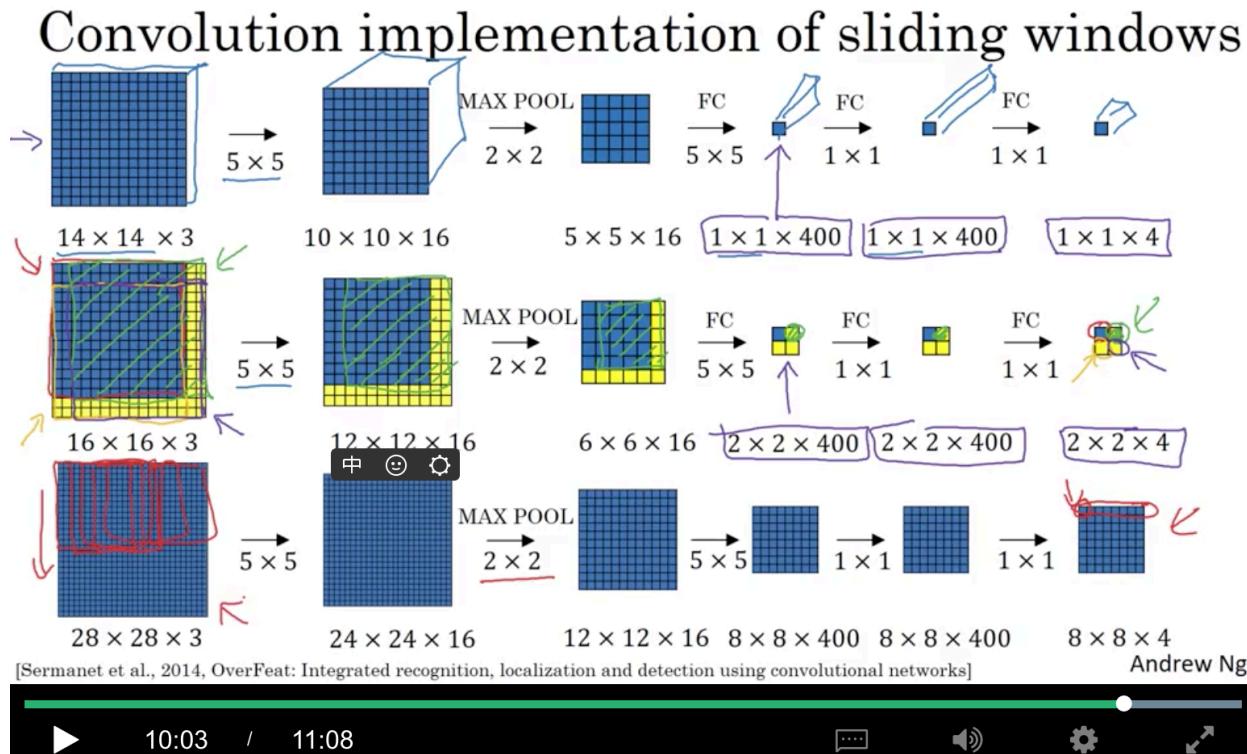


Figure 3: conv-sliding-window.png

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～

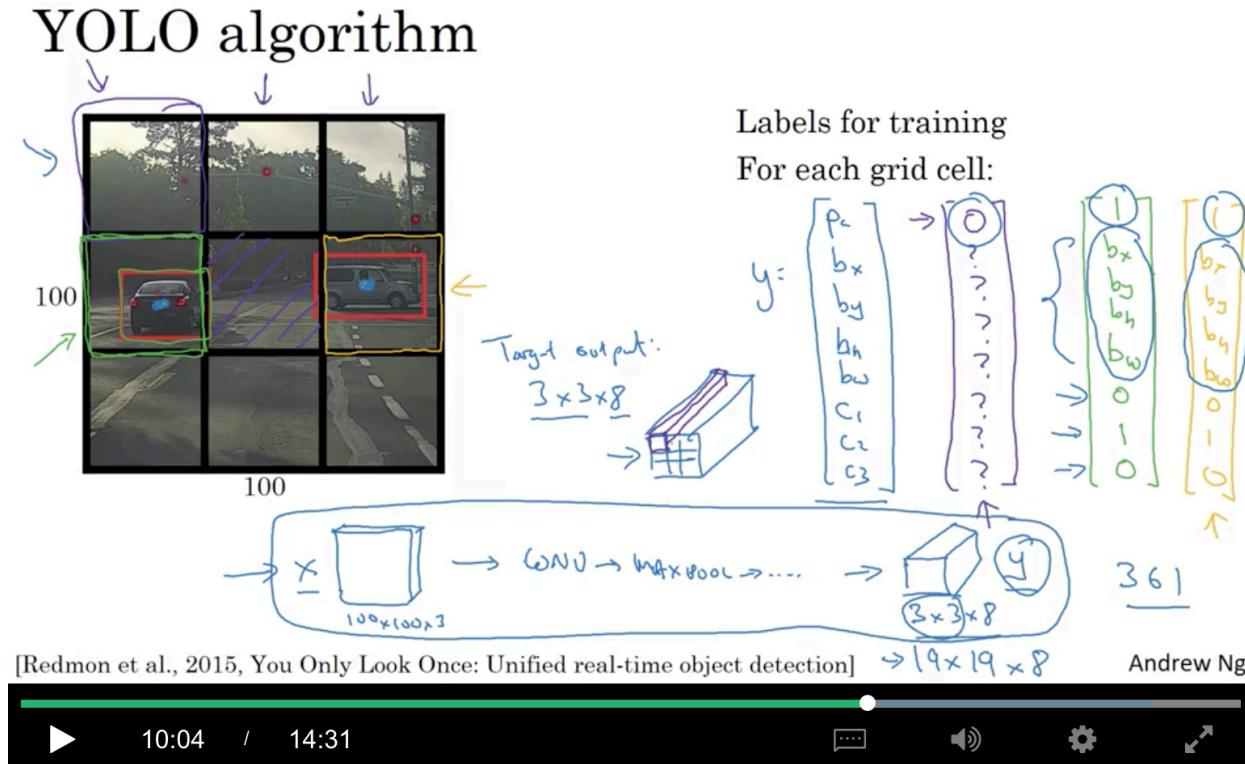


Figure 4: yolo-algorithm.png

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 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， $\text{IoU} \geq 0.5$ ，就算作 correct。当然更严格的阈值也可以。

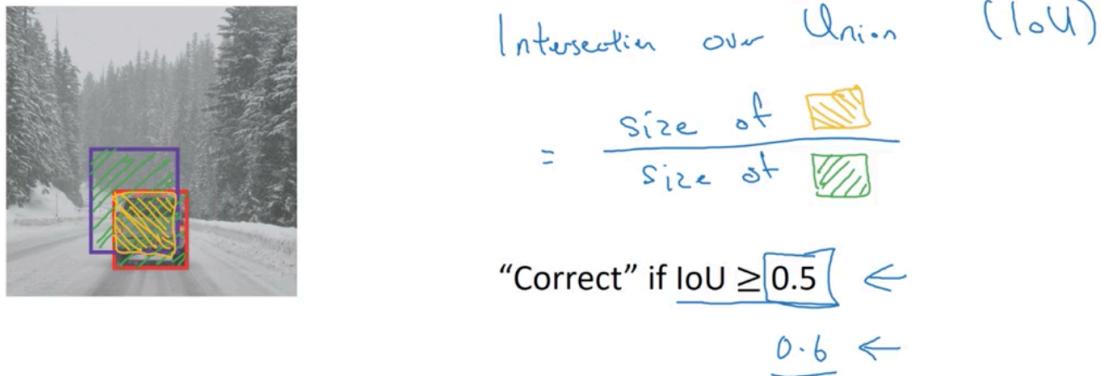
1.7 1.7 Non-max Suppression

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）

Evaluating object localization



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



Figure 5: iou.png

- 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

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 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)

Non-max suppression example

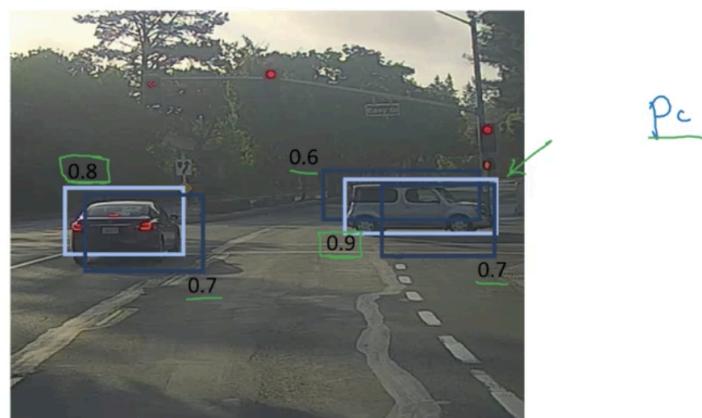
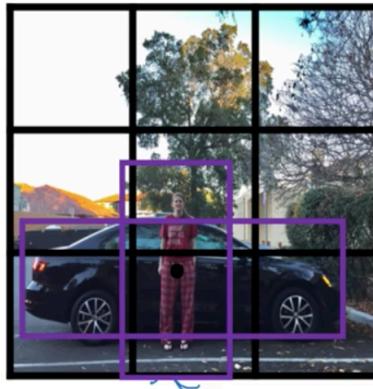


Figure 6: non-max-supression.png

Anchor box example



Anchor box 1: Anchor box 2:

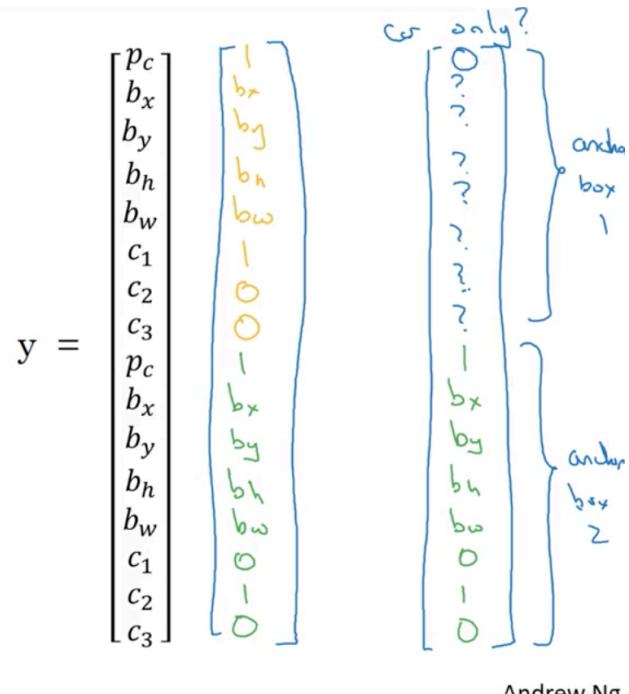
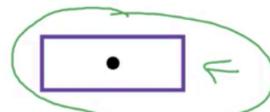
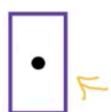


Figure 7: anchor-box.png

当然，如果格子是 19×19 ，同一个 grid cell 是两个东西的中心点的概率远比 3×3 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 1.9 YOLO Algorithm

训练过程如下：

注： $8 = 1 + 4 + \# \text{classes}$

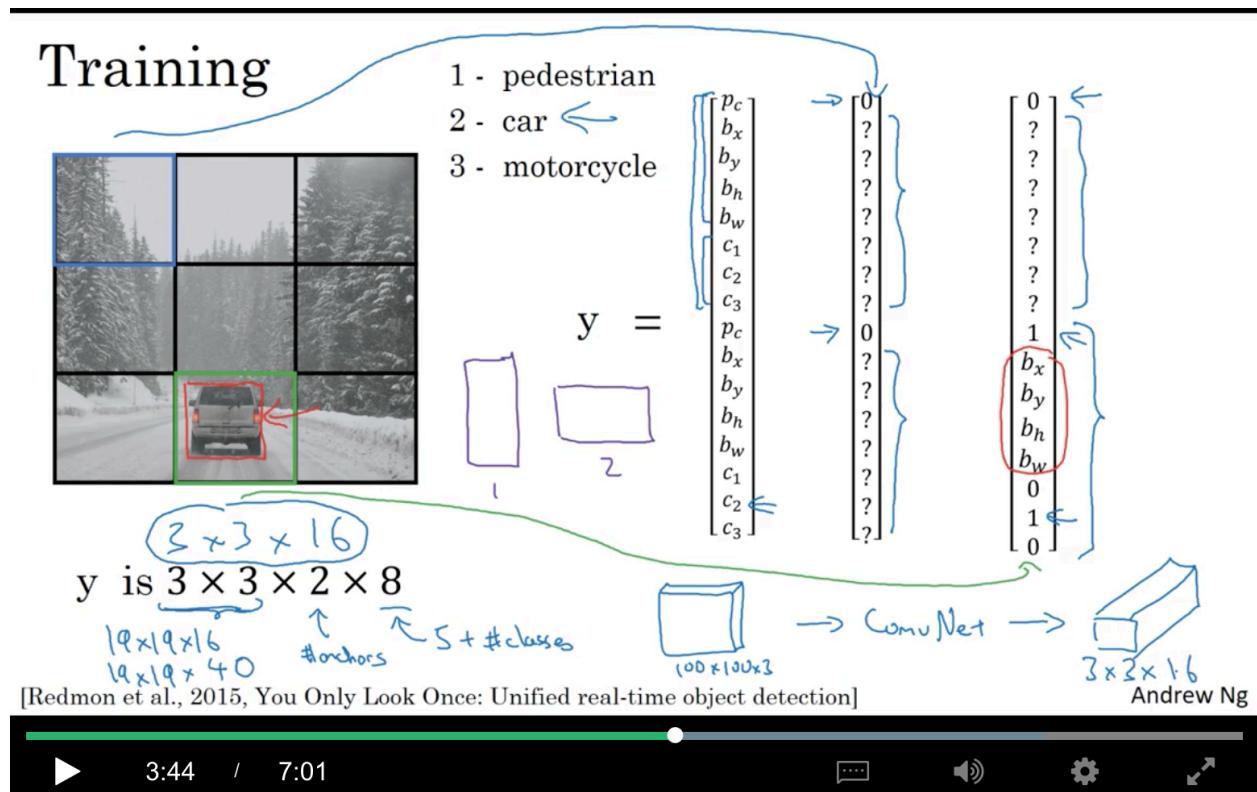


Figure 8: yolo-training.png

预测过程如下：

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

1.10 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。

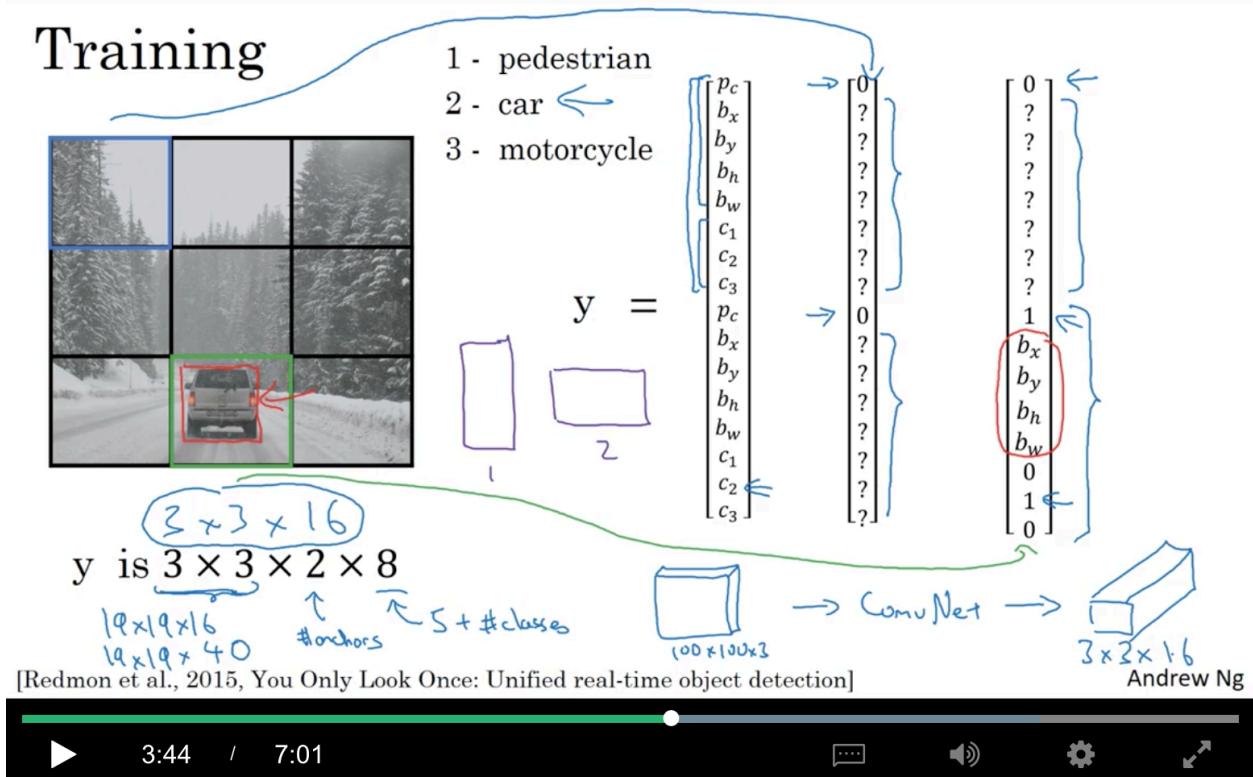


Figure 9: yolo-training.png

Outputting the non-max suppressed outputs

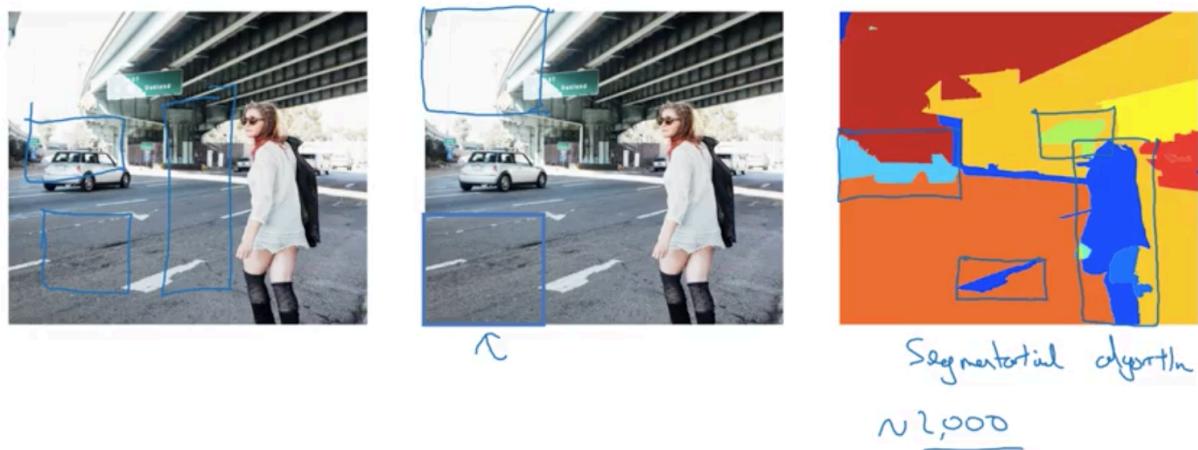


- For each grid cell, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.



Figure 10: yolo-non-max-suppression.png

Region proposal: R-CNN



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



Figure 11: R-CNN.png

- 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。。)