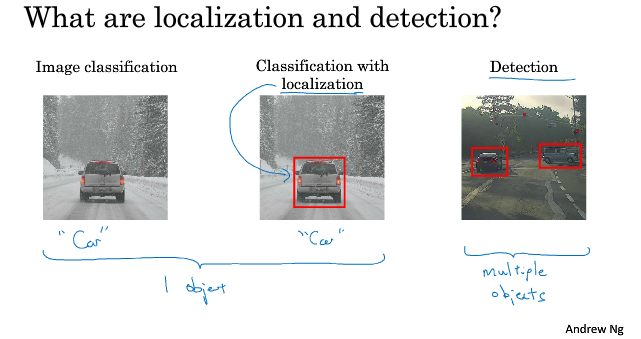
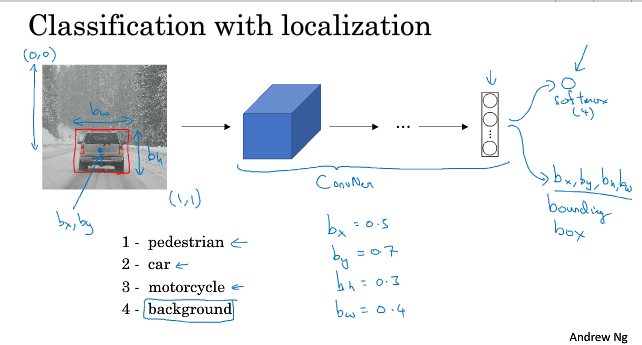
第三章 目标检测

3.1 目标定位

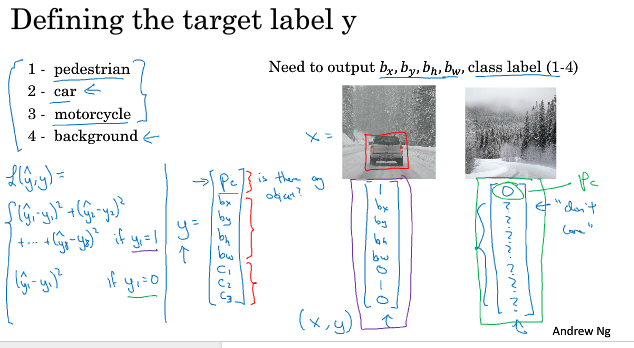
定位分类问题，判断汽车在图片上的具体位置，再进行分类。通常只有一个较大的对象位于图片中间位置，进行识别和定位。而对象检测问题中，图片可以含有多个对象，甚至单张图片中会有多个不同分类的对象，因此图片分类的思路可以帮助学习分类定位，而对象定位又有助于对象检测。



首先对于标准的对图片中的内容是人，还是车，还是自行车，或者是背景等的分类，只需要把图片输入到卷积网络中，使用softmax层进行分类，还想定位图片中汽车的位置应该如何做呢？可以让神经网络多输出几个单元，输出一个边框界，具体来说就是多输出四个数字，bx，by,bh,bw被检测对象的边框界的参数化表示。



对于一张图片，左上角为(0,0)，右下角为(1,1)，中心位置为(bx,by)，bh为整个图片的高度，bw为宽度。因此，训练数据不仅要包括分类的标签，还要包含边界框的这四个数字，接着采用监督式学习。



如何为监督学习任务定义目标标签y，第一个数Pc表示被检测对象属于某一分类的概率，也就是图片里是否含有下面的对象，如果检测到对象，就输出四个边框界参数，后面再输出c1,c2,c3…来表明是属于分类里面的哪一类。这里的例子，每个样本的y向量就是一个(8,1)的向量，c1,c2,c3…里面最多只有一个等于1，这是图片只有一个检测对象的情况。如果图片里没有要检测的目标，那么剩下的参数都毫无意义，表示成？，这些一起构成了训练集。

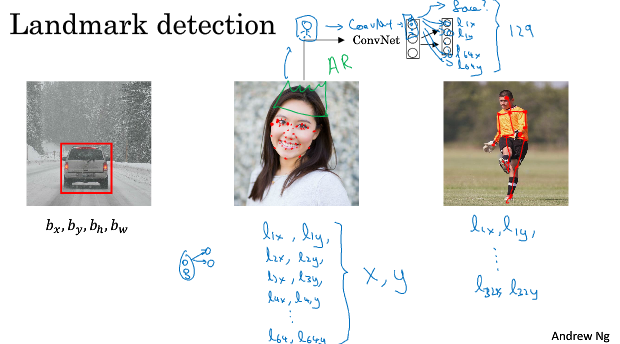
再说说代价函数，如果采用平方误差策略：

损失函数等于每个对应的差值的平方和，这是的情况，而对于，

结果证明，利用神经网络输出批量实数来识别图片中的对象是个非常有用的算法。下一节分享另一种思路，就是把神经网络输出的实数集作为一个回归任务，也被应用于计算机视觉的其他领域。

3.2 特征点检测

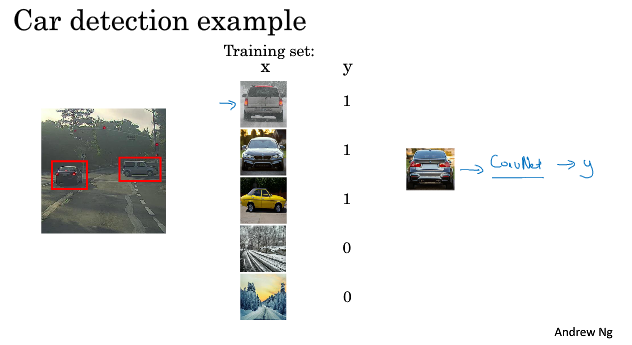
假设在人脸识别中，你想输出多少个特征点确定位置，你就可以输出多少个点，有的点甚至可以帮助你定义脸部轮廓或者下颌轮廓，选定特征点的个数，并生成包含这些特征点的标签训练集，然后利用神经网络输出脸部关键特征点的位置，具体做法是，准备一个卷积网络和一些特征集，将人脸图片输入卷积网络，输出1或0来表示是否有人脸，然后输出（l1x,l1y）…（l64x,l64y）这分别表示了64个特征点的坐标位置。所以一共就有129个输出单元，由此实现人脸检测和定位，这只是一个识别脸部表情的基本构造模块。



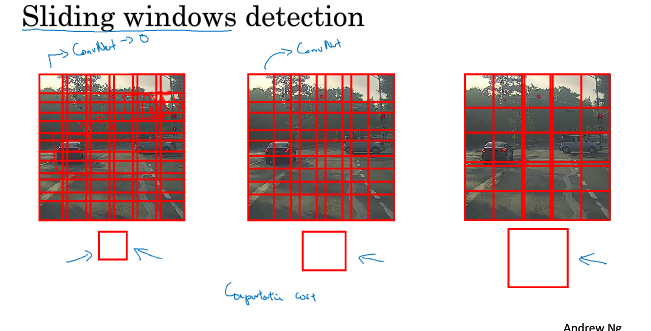
实现这样一个网络，需要准备一个标签训练集，也就是图片X和标签Y的集合，这些特征点都是人为辛苦标注的。另一个例子，比如对人体姿态有兴趣，也可以定义一些关键特征点，比如胸部的重点，腰，左肩，右肩等，通过神经网络标注人物姿态的关键特征点，再输出这些标注过的特征点，就相当于输出了任务的姿态。一旦了解如何用二维坐标系定义人物姿态，操作就简单了，批量添加输出单元，用以输出要识别的各个特征点的（X,Y）坐标值。要明确一点，特征点1的特性在所有图片中必须保持一致，就好比，特征点1都是代表左眼外眼角，特征点2代表左眼右眼角等等，所有标签在图片中必须保持一致。

3.3 目标检测

学过对象定位和特征点检测，构建一个对象检测算法，如何通过卷积网络进行对象检测，采用的是基于滑动窗口的目标检测算法sliding windows detection algorithm，思路是以固定步幅滑动窗口，遍历图像的每个区域，把这些剪切后的小图像输入卷积网络，对每个位置按0或1进行分类，这就是图像滑动窗口操作。



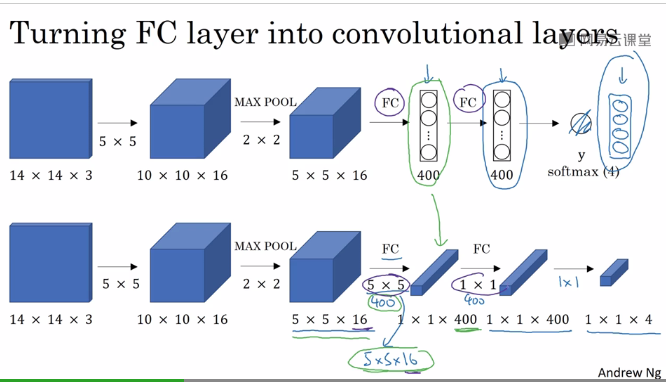
假设想构建一个汽车检测算法，首先建立一个标签训练集，也就是X和Y，表示一个适当剪切的汽车图片样本，然后训练一个卷积网络，这个网络的作用只是用来判断是不是汽车，使用的训练样本都是剪切过的，汽车大概在图片中间的一些样本数据。再根据滑动窗口的操作对图片的每一个小部分来判断是否是汽车，可以根据卷积网络对输入大小的要求来调整这个区域，输入给卷积网络，输出0和1。再以某个步幅滑动窗口，直到遍历完整个图像，输出结果。第三次可以选择一个更大的窗口来进行遍历。这样做的话，不论汽车在图片上的什么位置，总有一个窗口可以检测到它。



滑动窗口检测算法有明显的缺点就是计算成本，因为你在图片中剪切出太多小方块，卷积网络要一个一个的处理，如果步幅太大，显然会减少出入卷积网络的窗口个数，粗粒度可能影响性能，如果太小又会需要很高的计算成本。人们通常采用更简单的分类器进行对象检测，比如简单的线性分类器，至于误差，因为每个分类器的计算成本都很低，只是一个线性函数，所以滑动窗口目标检测表现良好是个不错的算法，然而卷积网络运行单个分类任务的成本却高得多。像这样滑动窗口太慢了，除非使用粒度特别小，步幅特别小，否则无法准确定位图片中的对象。不过幸好，计算成本问题已经有了很好的解决方案，大大地特高了在卷积层上应用滑动窗口目标检测器的效率。

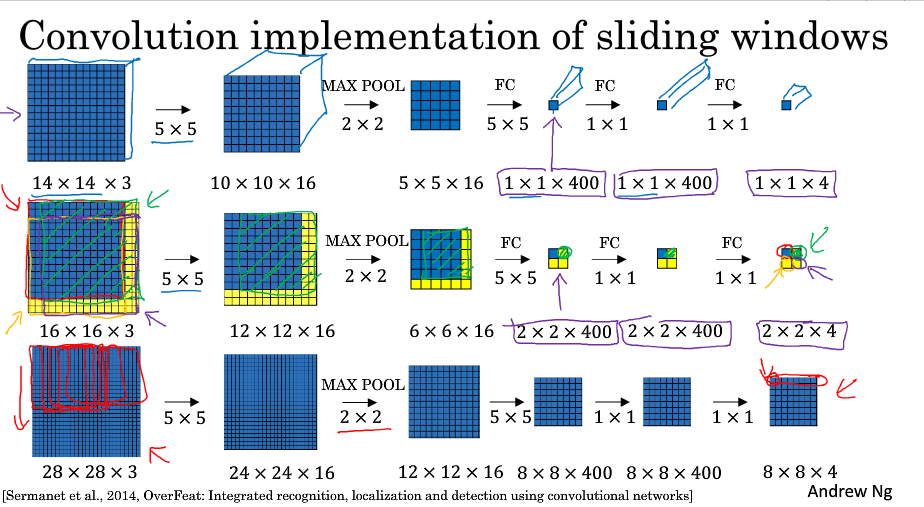
3.4 卷积的滑动窗口实现

前面是使用卷积网络完成滑动窗口探测算法效率很低，下面说如何在卷积层上应用这个算法。首先我们要知道如何把全连接层转化成卷积层：

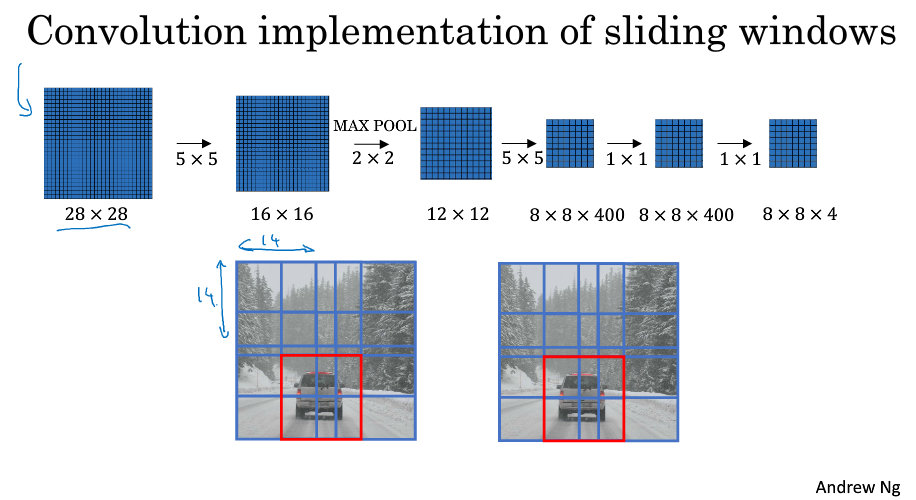


使用卷积层代替全连接层，中间使用过滤器以及对应的shape自己研究，最后的到一个1\*1\*4的矩阵其实就是4个类别对应的概率值并且应用400个5\*5\*16的过滤器其实就是对上一层的所有参数都进行过线性运算。

接下里看如果通过卷积来实现滑动窗口目标对象检测算法，其实最主要的思路在于，对一张图片使用一个窗口剪切一个小图片输入卷积网络，一次一次的计算，跟卷积操作也是按过滤器的大小在原始图片上进行一次一次计算的相似性。过滤器移动步幅就相当于窗口在移动，并且我们之所以使用卷积层能解决计算问题就是因为，每次将一个图片放入卷积网络，参数都是一样的，进行了重复的操作运算，通过卷积可以一次性完成生成对一张图片的滑动检测结果。



在16\*16\*3的原始图片上滑动过滤器，即滑动窗口，卷积运算就相当于进行卷积网络运行了4次，输出了四个标签。这4次卷积操作的很多计算都是重复的，滑动窗口的卷积应用使得卷及网络在这4次操作有很多重复步骤。

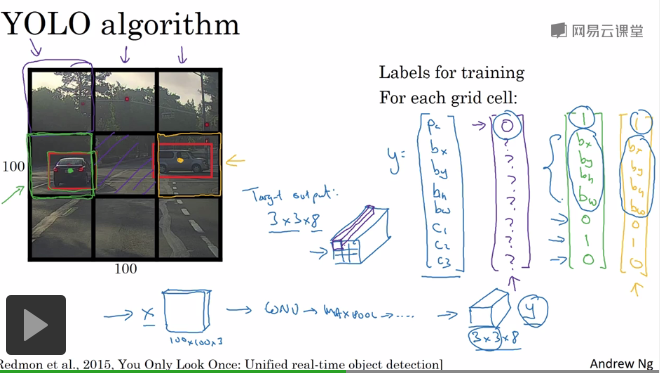


总结一下滑动窗口的实现过程在图片上剪切出来一块，大小为14\*14，输入卷积网络，重复操作，直到某一步识别出来汽车。但现在不用依靠连续的㡿操作来识别图片中的汽车，比如，我们直接对大小为28\*28的整张图片进行卷积操作，一次得到所有预测值，足够幸运的话，神经网络就能识别出来汽车位置，提高了效率。

但这种方式也有缺点，缺点就是边框界的位置可能不够准确，下一节看怎么解决。

3.5 Bounding Box检测

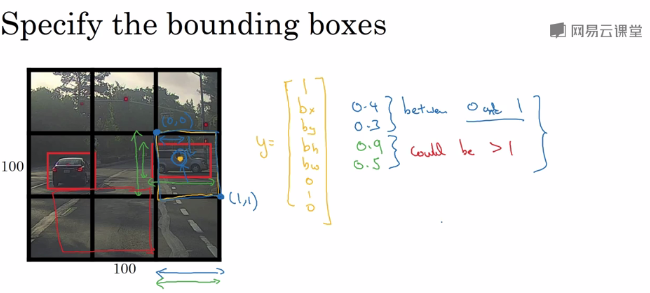
滑动窗口的卷积实现，效率已经很高了，但是边框界不够准确。比如汽车的边界根本不是一个正方形，又或者你的正方形过滤器刚好没有圈住图片中的车。解决这个问题的一个算法：YOLO算法，意思是只看一次。先将图片网格化，基本思路是使用图像分类和定位算法，然后将算法应用到每个格子上，对于训练集的标记方法Y则是将每个格子的定位分类结果（8,1）向量堆叠在一起，所以最后得到的三维的结果矩阵。



例如对于一个100\*100\*3的图片，我们采用3\*3的格子，也就是9个格子划分，实际上应该会更多，我们在制作标签Y的时候，首先考对象的中心，这张图片里面有两辆车，将两个中心分配到格子里面，就算作是这个格子里面有对象出现，并给对每个格子打标签的时候就输入正确的bx,by,bh,bw，所以训练集就制作好了，最后我们将训练集放入构建的卷积网络中进行学习，通过反向传播等，学习从输入到输出的映射，通过这种方式，不再有滑动窗口，但是也对图片的每一部分都进行了探测，并且还能得到准确的边框界。

对于一个格子内有两个对象，实际上操作时会用更小的网格，那么多个对象分配到一个格子的概率就小得多。强调，把对象分配到一个格子的过程，是观察对象的中点，然后将这个对象分配到中心所在的格子，所以即使对象横跨多个格子，也只会被分配到9个格子其中之一。

这和图像分类和定位算法非常像，就是它显示地输出边界框坐标，能让神经网络输出可以具有任意宽高比，并且能输出更精确的坐标，不会受滑动窗口分类器的步长大小限制，其次这是一个卷积实现，并没有在3\*3上跑9次算法，相反这是单词卷积实现，但你使用了一个卷积网络，有很多共享计算步骤。所以这个算法效率很高。实际上，YOLO算法受欢迎，这是一个卷积实现，运行速度非常快，可以达到实时识别。

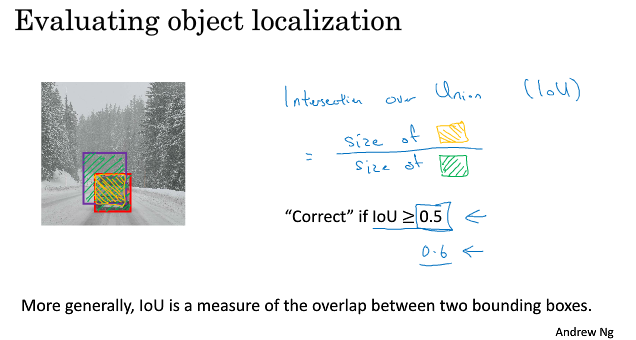


小细节：如何编码这些边框界bx,by,bh,bw，在YOLO算法中，我们根据一个格子的左上角为(0,0),右下角为(1,1)，然后看中心点为(bx,by),bw bh就用整个格子的比例大小表示。也就是说，这四个单位是相对格子尺度的比例，所以bx by必须在0到1之间，如果不在0~1之间，在方块外，就应该分配到另一个格子上。至于宽和高，是可能大于1的，因为可能在格子外面。不过还有其他更复杂的参数化方式，涉及到sigmoid函数，确保这个值介于0~1之间，然后使用指数参数化，来确保这些都是非负数，还有其他更高级的参数化方式。

Yolo算法，只看一次算法。

3.6 交并比intersection over union

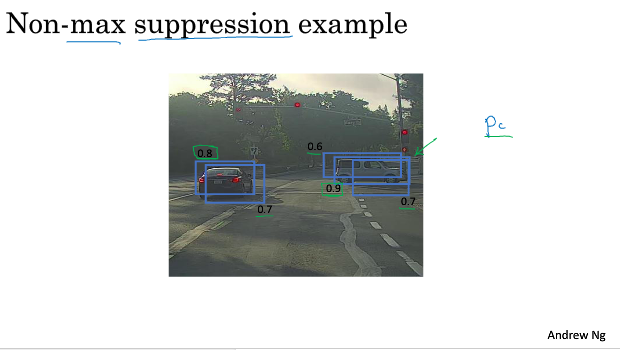
如何判断对象检测算法运作良好，这节学习交并比函数intersection over union function（IoU函数），可以用来评价对象检测算法。IoU做的是计算两个边界框交集和并集之比。一般约定，在计算机检测任务中，如果IoU大于或等于0.5，就说检测正确。如果预测器和实际边界框完美重叠，IoU=1，单一般来说，只要超过0.5，就是可以接受的。IoU越高，边框越精确。所以这是衡量定位精确度的一种方式，只需要统计算法正确检测和定位对象的次数。



3.7 非极大值抑制non-max suppression

到目前学到的对象检测中的一个问题是，算法可能对同一个对象作出多次检测，非最大值抑制non-max suppression 这个方法可以确保你的算法只对每个对象检测一次。

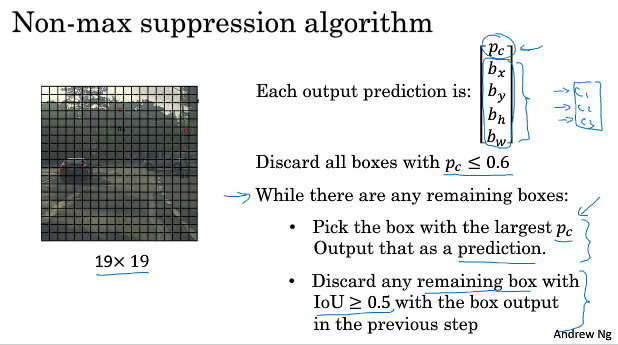
比如举个例子，下图中加入分层19\*19的格子，那么中心点只分配在一个格子里，但是其他的周围的格子就会认为这个格子里有车。因为你要在361个格子上都跑一遍定位分类算法，很多格子都会说我的Pc值，即这个格子里有车的概率很高。所以跑算法的时候，最后可能会对同一对象做出多次检测。所以非最大值抑制做的就是清理这些检测结果，使一辆车只会检测一次。



非最大值抑制，就是只输出概率最大的分类结果，但抑制很接近但不是最大的其他预测结果。

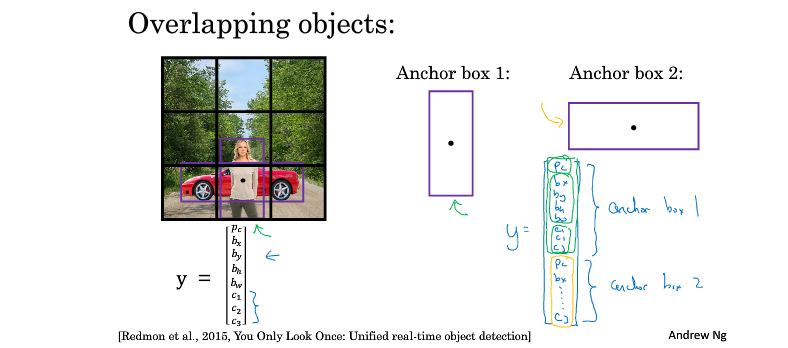
举个例子，在下图中，检测图片中的汽车，这里我们只检测有没有汽车，所以就没有c1,c2,c3三个值，所以对于格子的每一个标签都应该是(5\*1)的，首先我们跑一遍算法，对所有的预测，所有的边界框，Pc小于某个阈值的去掉，这样就抛弃了所有概率比较低的输出边界框，接下来是个while循环，在剩下的边界框中，就一直选择概率最大的Pc最高的边界框，然后把它输出成预测结果，接下来，去掉所有剩下的边界框中任何没有达到输出标准的边界框和上一步输出边界框有很高交并比的边界框全部抛弃。一直这么判断，直到每个边界框都判断过了，有的作为输出结果，有的抛弃了。如果是检测多个物体的话，那么就要做多次非最大值抑制，对每个输出类别都做一次。

非极大值抑制，来抑制那些冗余的框：抑制的过程是一个迭代-遍历-消除的过程，**（1）**将所有框的得分排序，选中最高分及其对应的框 **（2）**遍历其余的框，如果和当前最高分框的重叠面积(IOU)大于一定阈值，我们就将框删除 **（3）**从未处理的框中继续选一个得分最高的，重复上述过程。

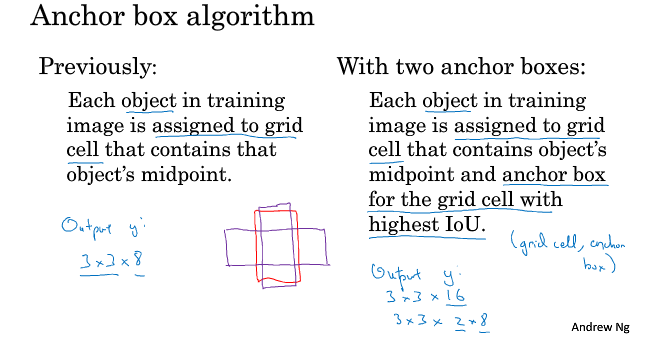


3.8 Anchor Boxes

目前对象检测还有一个问题就是每个格子只能检测出一个对象，如果要让一个格子检测出多个对象，就要使用Anchor Boxes这个概念。



在之前的对象检测中，做的是对于训练集图像中的每个对象都根据那个对象重点位置分配到对应的格子中，然后输出y是一个3\*3\*8的值，因为是3\*3个格子。现在使用anchor boxes的话，现在每个对象都和之前一样分配到同一个格子中，分配到对象中点所在的格子中，但是它还分配到一个anchor box和那个格子形状交并比最高的anchor box。所以在这个例子中有两个anchor box，就取这个对象，看对象形状跟两个anchor box中，哪一个的交并比最高，不管选的是哪一个，最后这个对象都不止分配到一个格子，而是分配到一对（格子，anchor box）。这就是对象在目标标签中的编码方式。所以后来的输出就是3\*3\*16，现在的y是（16,1）的。



如果是在同一个格子里有三个对象，这种情况算法处理不好，如果这种情况发生，并没有很好的处理办法，就引入一种打破僵局的默认手段，两个对象都分配到一个格子中，并且它们的anchor box形状也一样，这是算法处理不好的另一种情况。

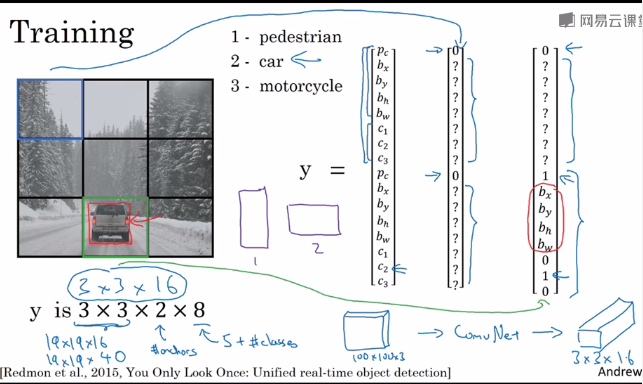
Anchor box就是为了解决两个对象出现在一个格子里，实际上，如果网格划分的很小的话，出现这种情况的概率很低，但确实会出现，anchor box可以让你的学习算法能够更有针对性，特别是如果你的数据集有一些很高很瘦的对象，比如说行人，还有像汽车这样很宽的对象，这样你的算法就能更有针对性的处理。

那如何选择anchor box的大小呢，人们一般手工指定anchor box的形状，可以选择5到10个anchor box形状，覆盖到多种不同的形状，可以涵盖你要检测的对象的各种形状，还有一个更高级的版本，YOLO后期有更好的做法就是所谓的k-平均算法，可以将两类对象形状聚类，如果我们用它来选择一组anchor box，选择最具代表性的一组anchor box，可以代表你试图检测的十几个对象类别，这其实是自动选择anchor box的高级方法。

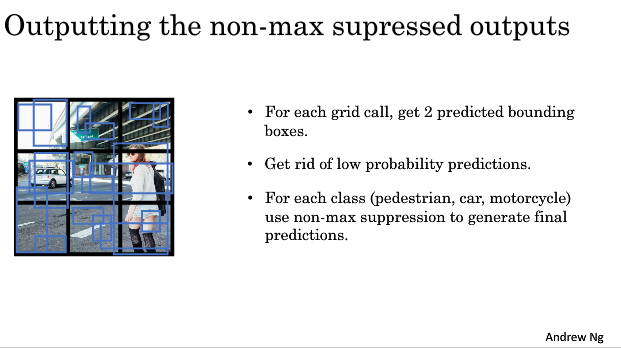
3.9 YOLO算法

已经学到对象检测算法的大部分零件了，接下来把全部零件组装在一起，构成YOLO对象检测算法。

先看如何构造训练集，假设要训练一个算法去检测三种对象，行人，汽车和摩托车，还需要显示指定完整的背景类别，那么总共就有三个类别，输出y就是3\*3\*16，要构造训练集，需要遍历9个格子，然后构造对应的变量y。



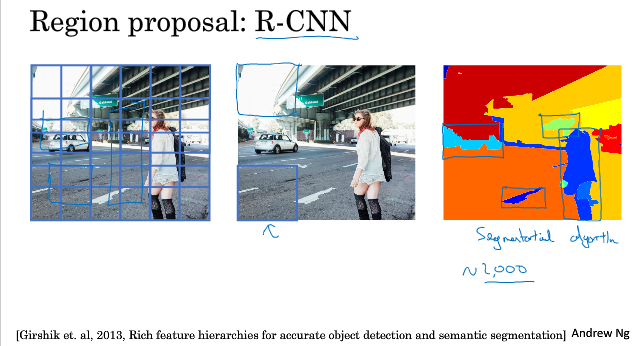
接下来看算法是怎么做出预测的，输入图像，神经网络的输出尺寸是3\*3\*16，对于像左上角这种没有对象的格子，输出y应该是Pc为0，而边框位置不能是？而是一些数字，尽管这些数字是不是对应一个类别的边界框无关紧要，所以基本上是一组数字，多多少少都是噪音，最后我们就得到关于输出y，这就是神经网络做出预测的过程。



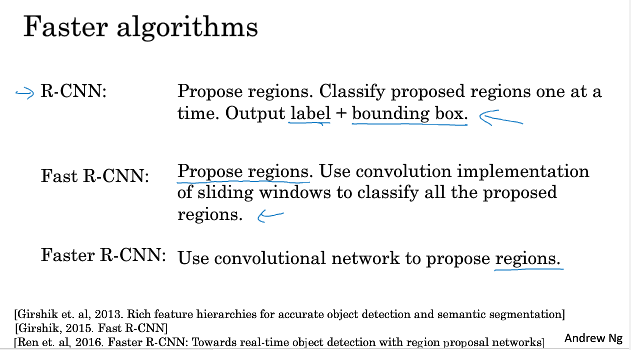
最后要跑一下非最大值抑制，如果你使用了两个anchor box，那么对于9个格子任何一个都会有两个预测的边界框，如上图所示，其中一个的Pc概率很低，注意有一些边界框可以超出所在格子的高度和宽度，接下里，抛弃概率低的预测，说明神经网络测出来告诉你，这里没有什么对象。最后如果你检测的很多对象，那么你就需要对每一类单独运行非最大值抑制，处理预测结果是那个类别的边界框，用非最大值抑制处理行人类别，以及车类别，运行三次得到预测结果。最好能够检测出图像里所有的车子，所有的行人。这就是YOLO对象检测算法，实际上是最有效的对象检测算法之一。

3.10 RPN网络

R-CNN算法，带区域的卷积网络，这个算法尝试选出一定区域，在这些区域上运行卷积网络分类器是有意义的，所以不再针对每个滑动窗口跑检测算法，而是只选择一些，然后在这些窗口上运行分类器。选出候选区域的方法是运行图像分割算法，为了找出可能存在对象的区域，分割色块。先找出可能的2000个色块，在这个色块放边框界，最后运行分类器的次数肯定会比窗口滑动算法少。



现在看来这个算法还是很慢，所以有一系列的研究工作改进，基本的R-CNN是使用某种算法求出候选区域，然后对每个区域跑分类器，每个区域都会有输出，并输出一个边框界，这样在确实存在对象的区域得到一个精确的边界框。缺点就是太慢了，快速的R-CNN，用卷积实现滑动窗口。快速的问题是取得候选区的速度还是很慢，更快的R-CNN，使用的是卷积神经网络而不是传统的分割算法，来获得候选区域色块，结果比前者快的多。



但是R-CNN还是比YOLO慢的多，但是候选区域这种想法还是很好的。