4. 卷积神经网络

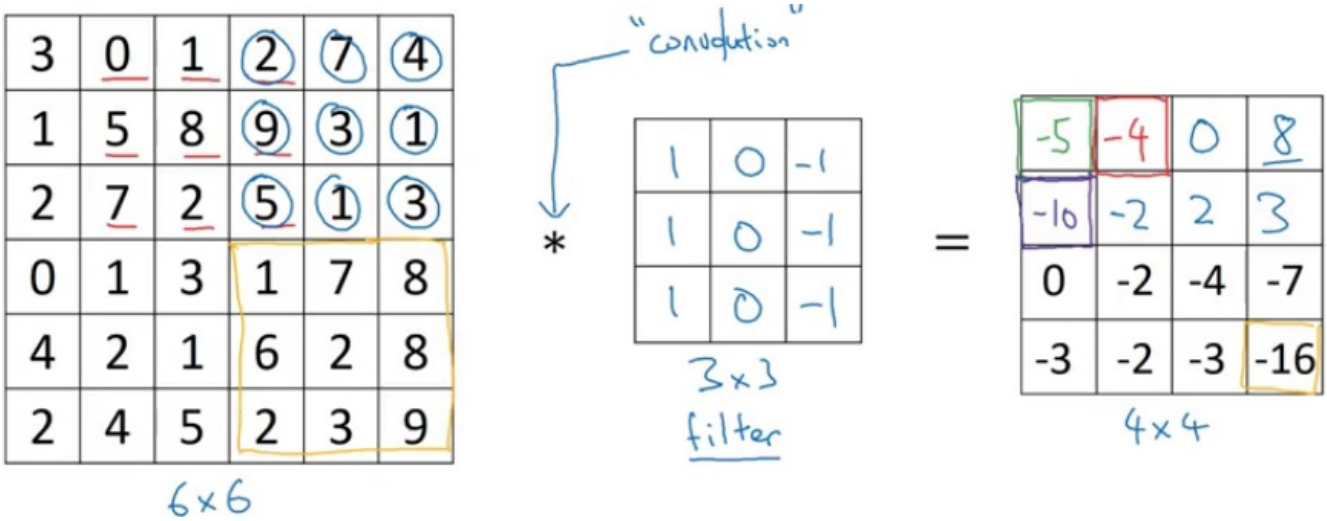
深度学习

1. 卷积神经网络基础

1.1 卷积计算

过滤器

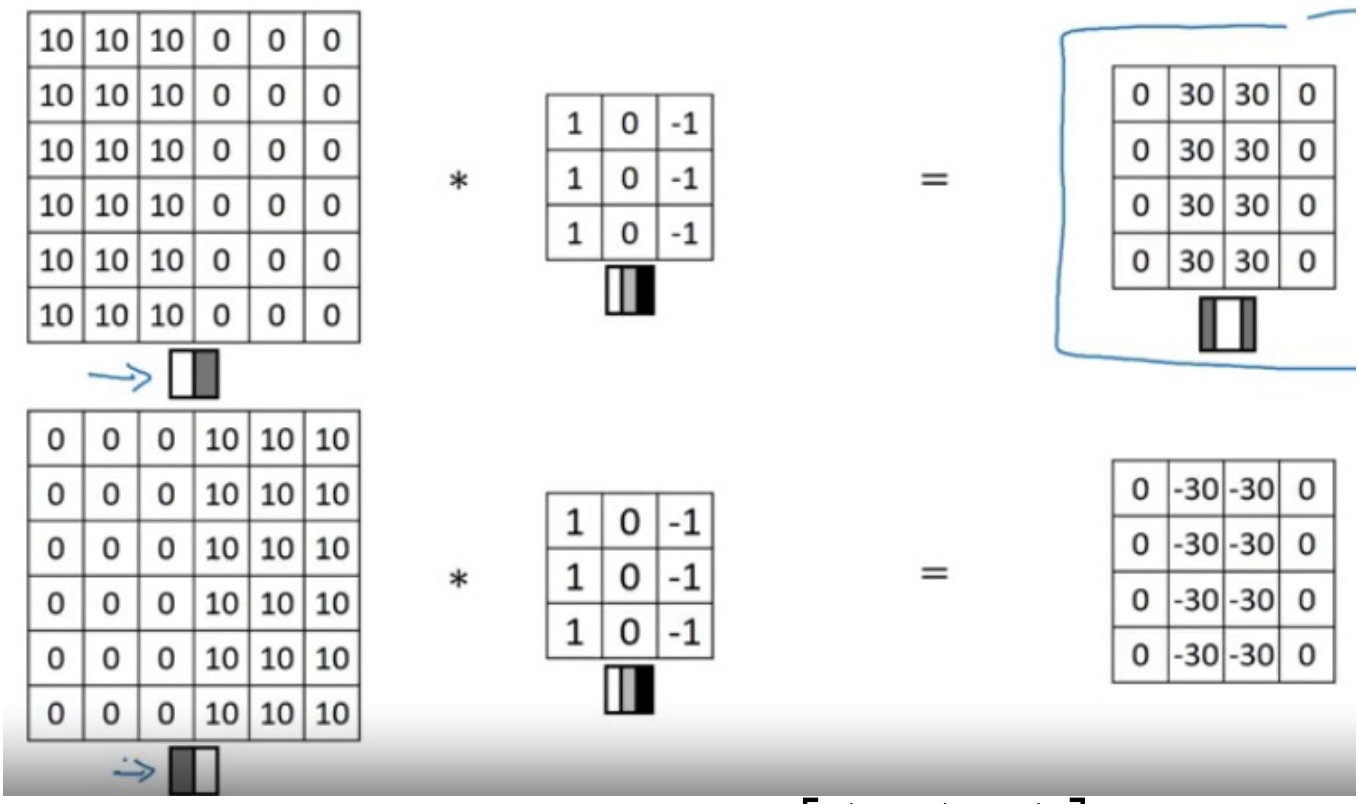
卷积的基本计算形式如下图所示：



计算原理是：图像矩阵中每个与过滤器形状对应部分分别与过滤器进行点积计算，输出结果放置于输出矩阵的对应位置上。例如，首先计算图像矩阵左上角3*3的元素，结果放置在输出矩阵左上角的位置上(绿色)；将元素窗口向右平移，计算下一个位置的结果(红色)；依次类推，直至计算完所有3*3窗口的(最后位置为黄色)。图像为6*6的特征矩阵，过滤器(fliter)为3*3，两者的卷积计算会形成4*4的输出矩阵。

边缘检测效果

合理的设置过滤器可以起到边缘检测的作用。例如，下图左上方图像矩阵相当于一个左白右黑的图像，其垂直边界在正中间。使用过滤器 $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ 计算后，输出结果显示了边界的所在位置(数字30所表示的部分)。



同理，如果图像在水平方向出现边界，可以使用过滤器 $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ 进行检测。

过滤器的形式还有很多，例如sobel过滤器 $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$，Scharr过滤器 $\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$ 等。在使用卷积神经网络的过程中，过滤器一般不是直接指定的，而是通过反向传播进行学习的，相当于人工神经网络中的权重W。

填充

由之前卷积计算方法可知，卷积计算形成的输出矩阵一般会小于输入矩阵。若输入矩阵为n*n，过滤器为f*f，则输出结果为(n-f+1)*(n-f+1)。这样的计算方法有一个问题，边界像素的部分信息被丢失了，例如使用3*3过滤器，图像中间的像素一般会被计算9次，而边界像素(如左上角像素)可能仅被使用了一次。

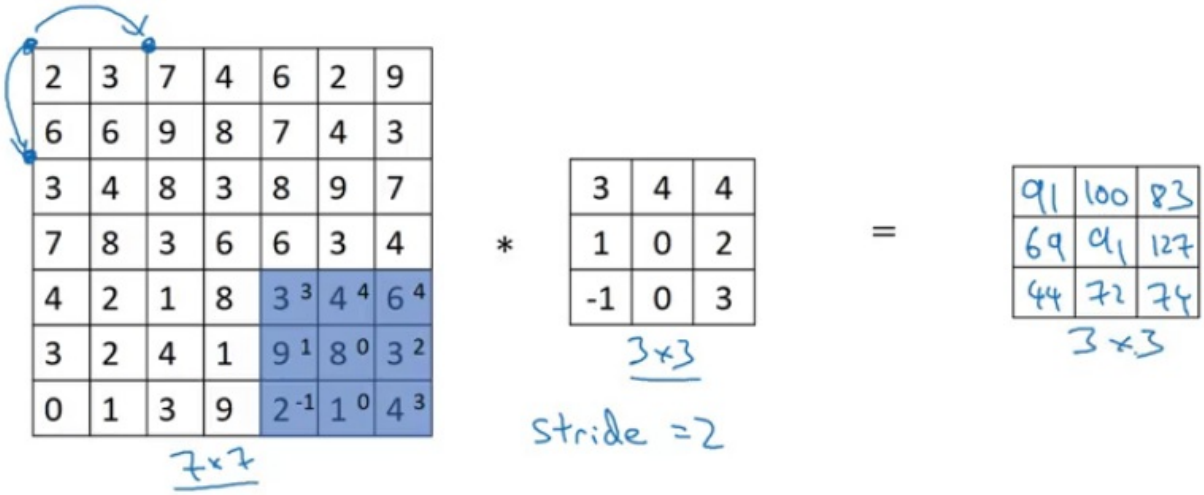
一个解决方法是在卷积计算前对原图像矩阵进行填充，将原图像矩阵大小变为(n+2p)*(n+2p)，这样计算后的输出矩阵大小为(n+2p-f+1)*(n+2p-f+1)。

填充方法一般有两种选择：

- valid：不进行填充
- same：填充后，输出矩阵与输入矩阵形式一样，也就相当于n=n+2p-f+1，此时 $p = \frac{f-1}{2}$

步长

卷积计算时平移的步长记做s。如下图例子，7*7的矩阵与3*3的过滤器进行计算，如果步长为2，那么在计算 $\begin{bmatrix} 2 & 3 & 7 \\ 6 & 6 & 9 \\ 3 & 4 & 8 \end{bmatrix}$ 后，直接横向平移两个长度，计算 $\begin{bmatrix} 7 & 4 & 6 \\ 9 & 8 & 7 \\ 8 & 3 & 8 \end{bmatrix}$，因此最终的输出结果是3*3。



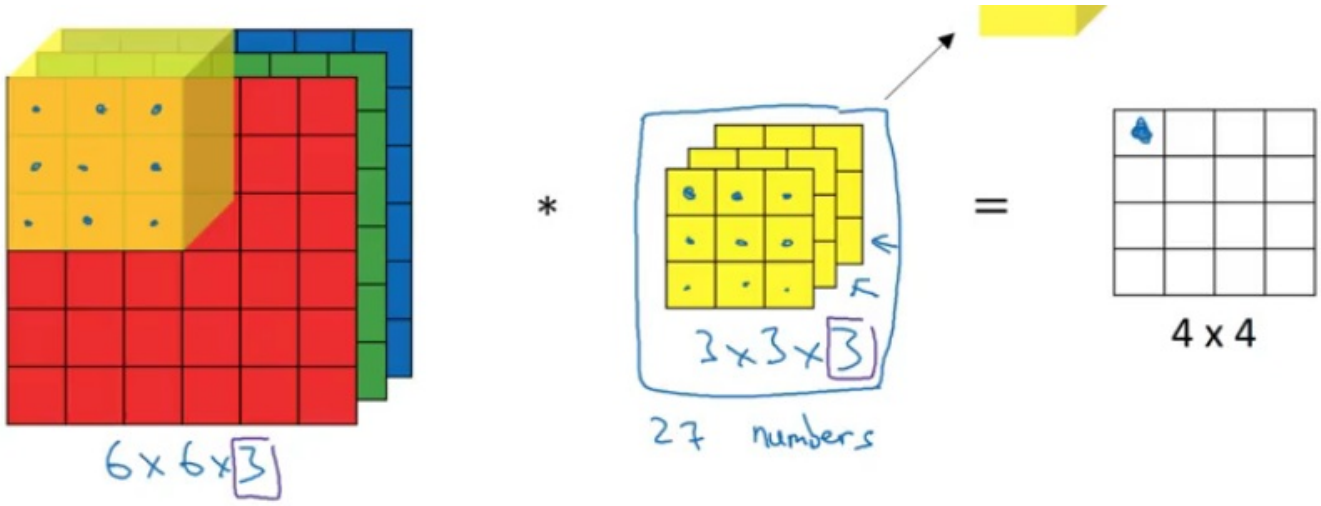
同时考虑步长s和填充p，n*n的输入矩阵，经f*f的过滤器计算后，输出为 $[\frac{n+2p-f}{s}+1]*[\frac{n+2p-f}{s}+1]$。计算结果如果不是整数，需要向下取整。

1.2 单层卷积神经网络计算

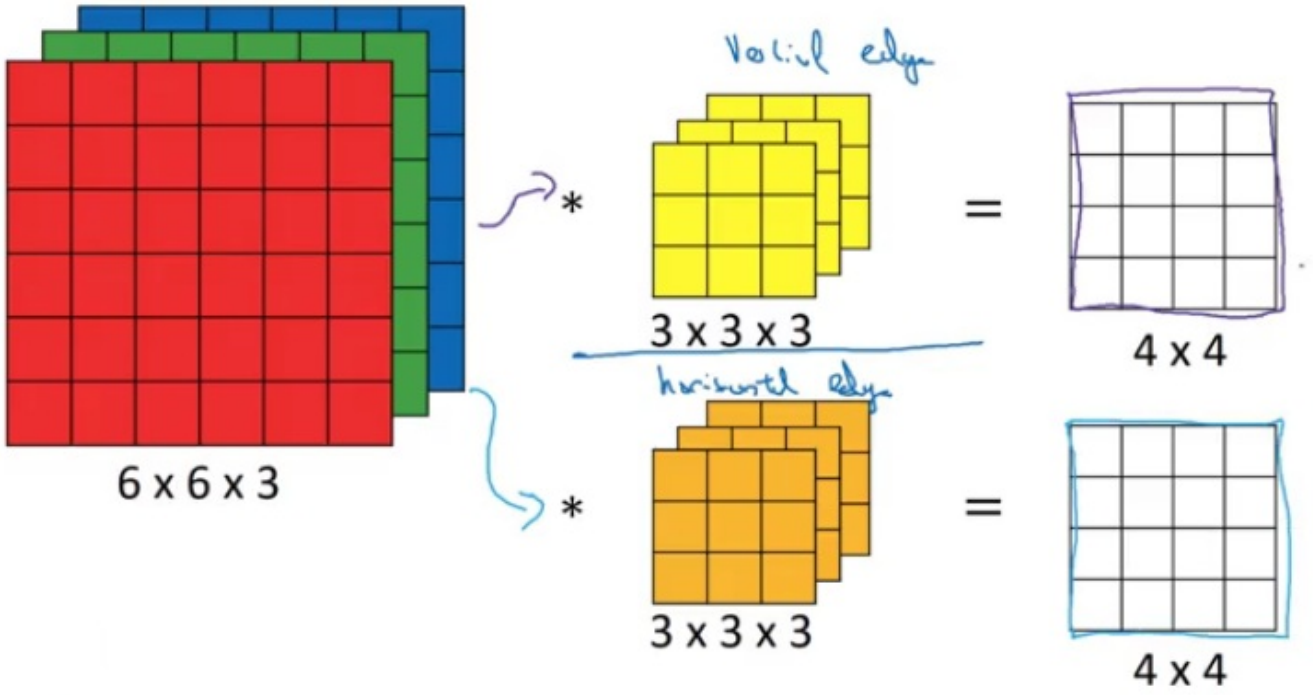
多信道

实际的图像一般会有红/绿/蓝三个颜色通道，因此卷积的计算也不是只有二维，而应该是三维的。

如果输入图像矩阵为6*6*3，代表总共有3个信道，每个信道的图像色素都为6*6，那么过滤器的形式应为f*f*3，信道数要和图像矩阵保持一致。如果过滤器形式为3*3*3，最终计算结果为4*4，也就是说在信道这个维度上，计算结果被加法合并了。



如果希望卷积计算后仍有多个信道，就需要设置多个过滤器。如下图，黄色和橙色代表两个不同的3*3*3过滤器，分别计算出两个4*4的结果。多个过滤器可以视为网络的权重W，其形式为3*3*3*2，而输出结果为4*4*2。



偏置项和激活函数

卷积神经网络中也有偏置项b，对于W中的每一个过滤器，有一个对应的偏置项，也就是说3*3*3*2的权重矩阵W，有2个偏置项b，形式为1*1*1*2。

激活函数的效果与人工神经网络是一致的，对WX+b的计算结果利用激活函数进行非线性转换。

符号约定

对于神经网络的中的一层l：

- $f^{[l]}$代表过滤器大小
- $p^{[l]}$代表填充大小
- $s^{[l]}$代表步长
- $n_c^{[l]}$代表过滤器个数

因此：

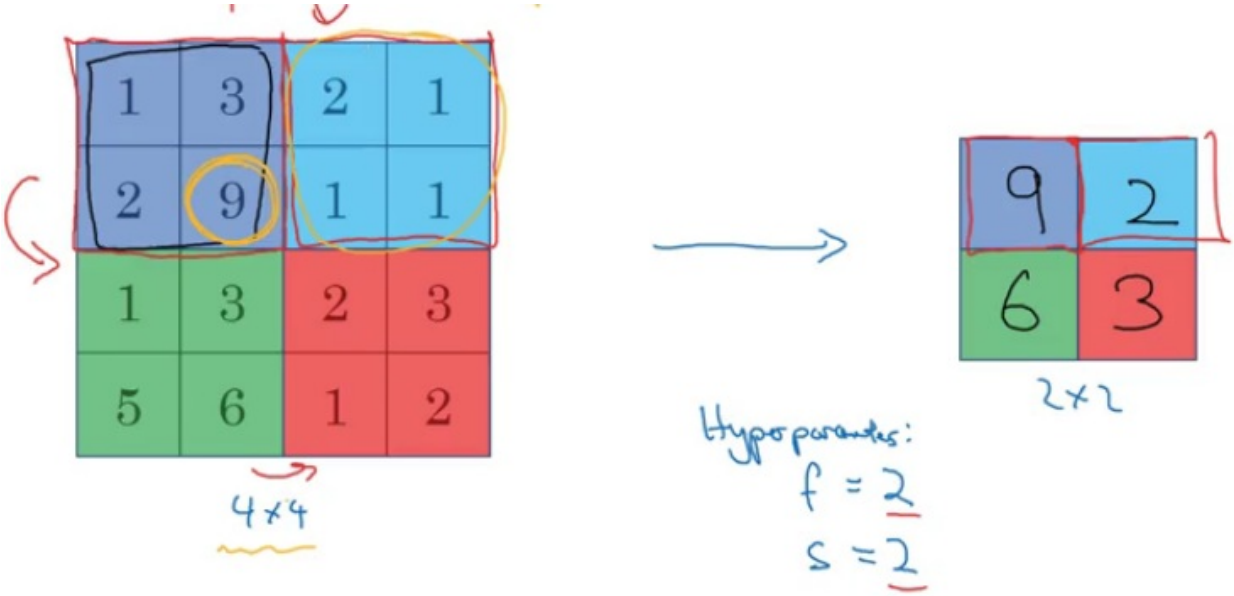
- 权重W(过滤器)的形式为：($f^{[l]}$, $f^{[l]}$, $n_c^{[l-1]}$, $n_c^{[l]}$)
- 偏置项b的形式为：(1, 1, 1, $n_c^{[l]}$)
- 输入矩阵形式为：($n_H^{[l-1]}$, $n_W^{[l-1]}$, $n_c^{[l-1]}$)
- 输入矩阵形式为：($n_H^{[l-1]}$, $n_W^{[l-1]}$, $n_c^{[l]}$)，其中$n_H^{[l]} = [\frac{n_H^{[l-1]}+2p^{[l]}-f^{[l]}}{s^{[l]}}+1]$，$n_W^{[l]} = [\frac{n_W^{[l-1]}+2p^{[l]}-f^{[l]}}{s^{[l]}}+1]$

1.3 卷积神经网络

卷积神经网络一般包括卷积层、池化层和全连接层。

池化层

池化层也包含f和s的概念，不过不是通过含有权重的过滤器计算的，而是直接将原像素矩阵中的一块转换成一个值。下图展示的池化过程中，f为2，步长为2，对于每个2*2的窗口，计算窗口中的最大值，也称最大池化。

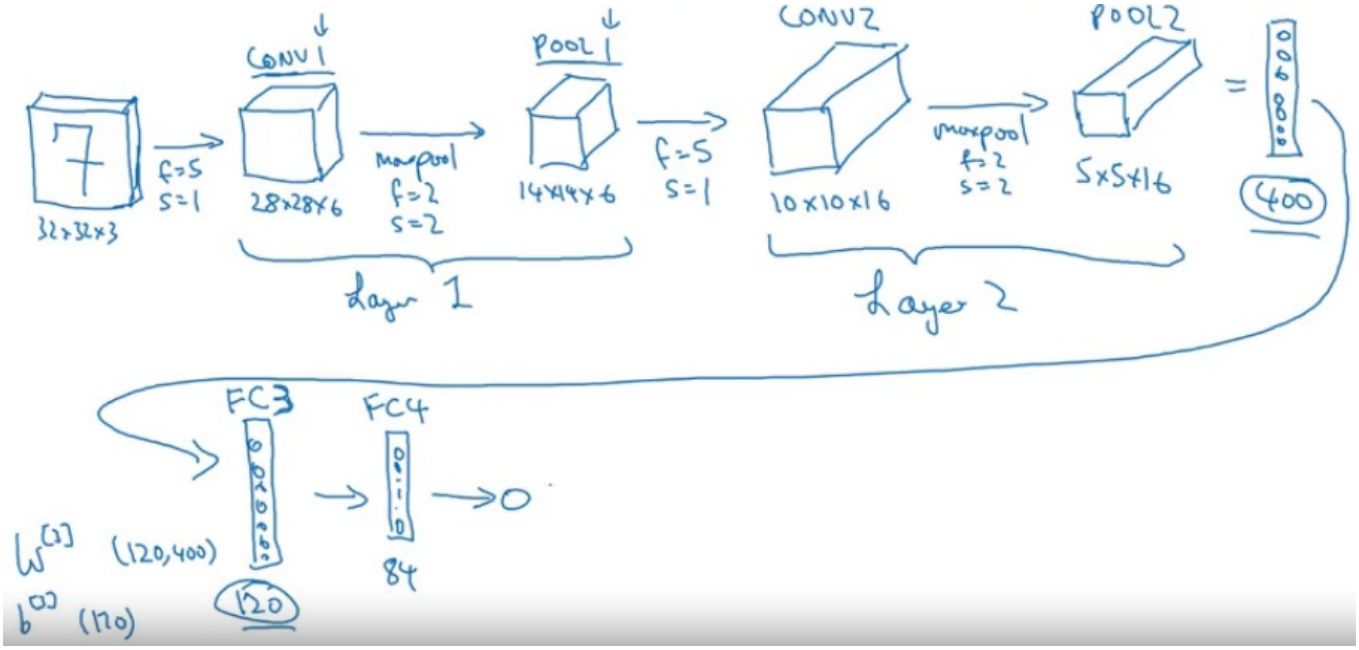


池化一般有两种：最大和平均。其中最大池化层是最常用的，卷积层相当于提取图像特征，而

最大池化是将特征集合中最显著的提取出来。

卷积神经网络

下图是一个完整的卷积神经网络：输入(32*32*3)->卷积层1(28*28*6)->池化层1(14*14*6)->卷积层2(10*10*16)->池化层2(5*5*16)->全连接层2(5*5*16=400)->全连接层3(120)->全连接层4(84)->输出层。



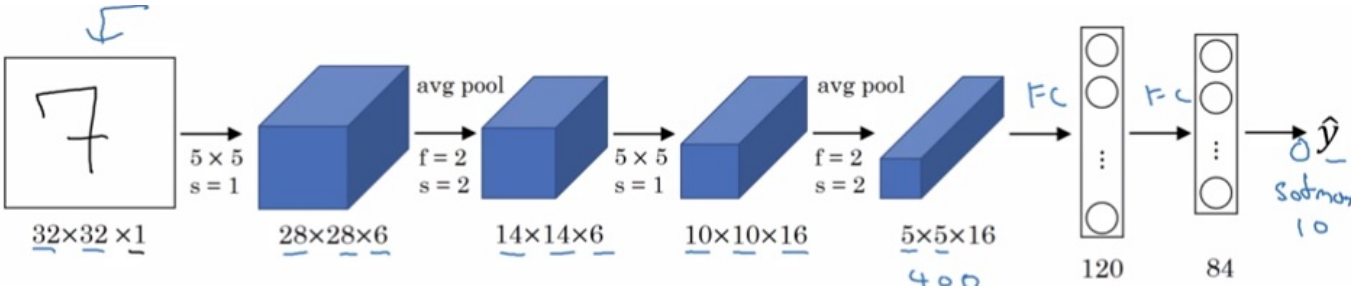
卷积神经网络的优点

- 参数共享：图像特征矩阵的维度可能很大，但卷积计算时它们是共享过滤器(权重)的，参数就没有那么多。
- 局部感知：卷积计算时，下一层只与前一层的部分节点相连，因此相比全连接，卷积层是比较稀疏的。

2. 深度卷积模型

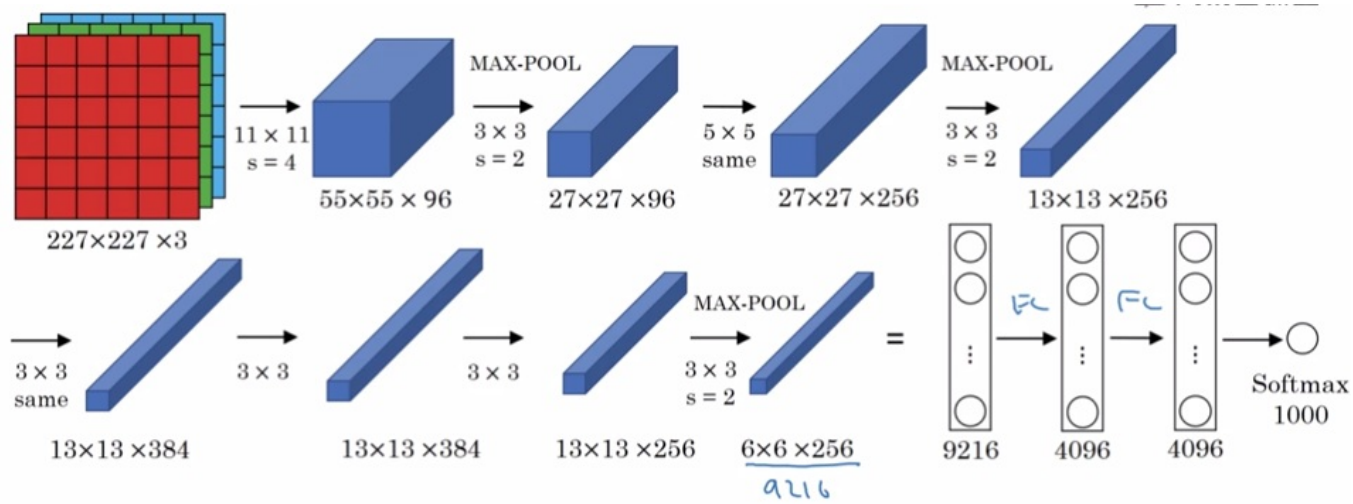
2.1 经典卷积神经网络

LeNet-5



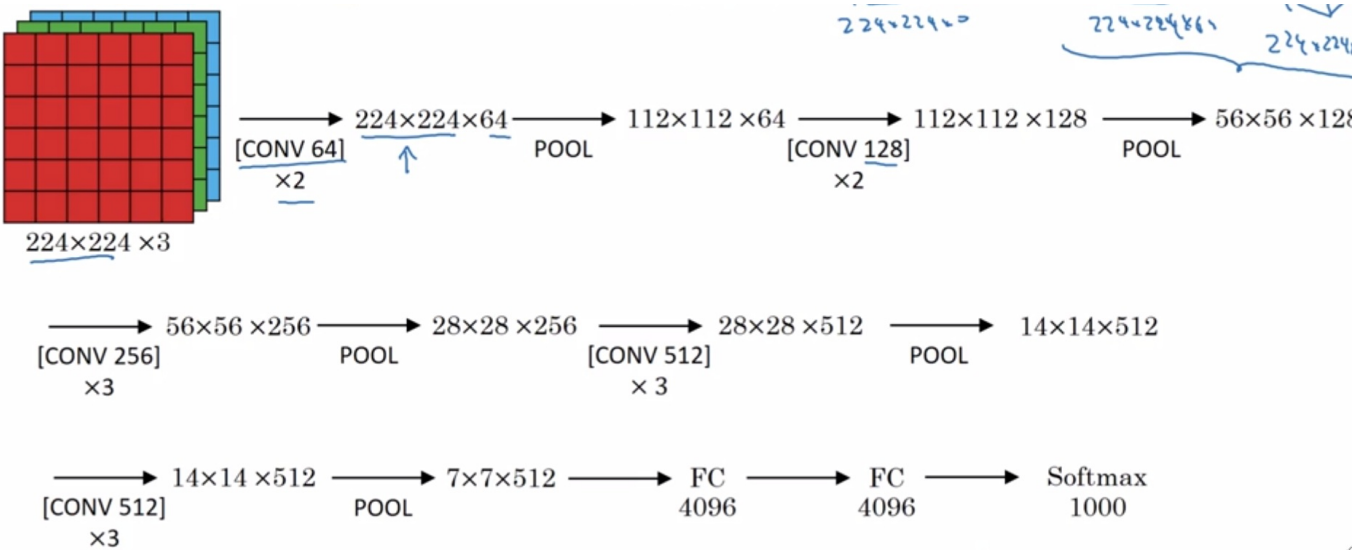
输入->卷积层->池化层->卷积层->池化层->全连接层->全连接层->输出

AlexNet



输入->卷积层->池化层->卷积层->池化层->卷积层->卷积层->卷积层->池化层->全连接层->全连接层->全连接层->输出

VGG-16

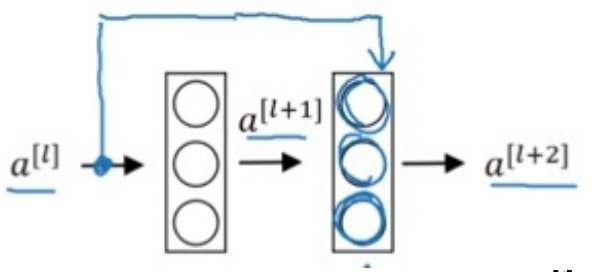


输入->卷积层*2->池化层->卷积层*2->池化层->卷积层*3->池化层->卷积层*3->池化层->卷积层*3->池化层->

全连接层*2->输出

2.2 残差网络 Residual Networks

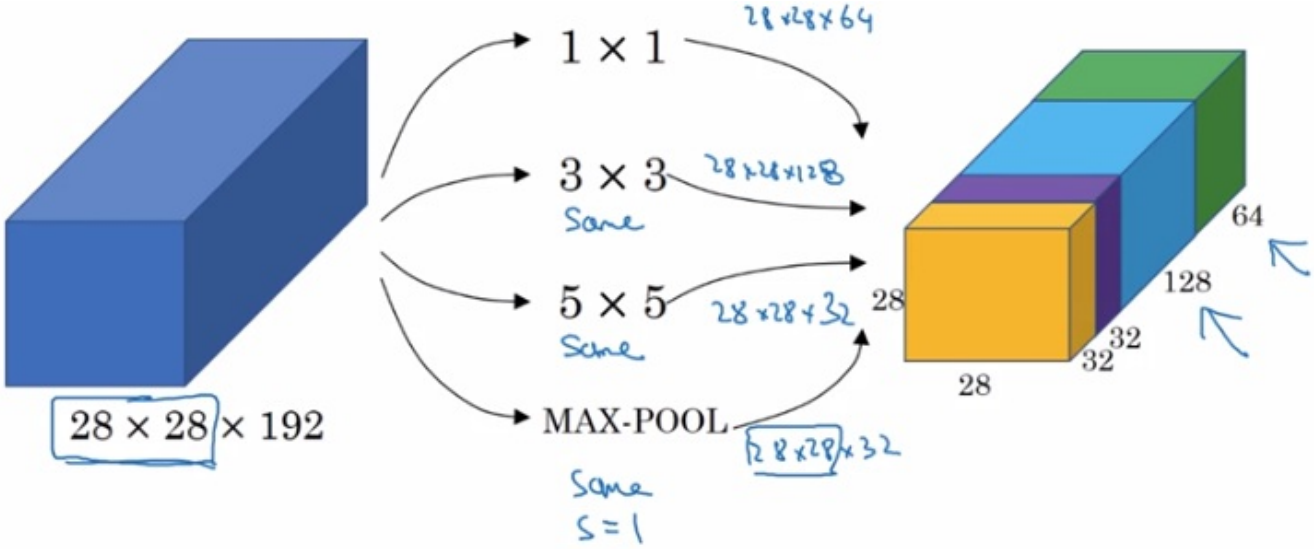
当神经网络很深时，经常会出现梯度消失或梯度爆炸等问题，很影响训练效果。ResNet能有效的缓解这种问题。



ResNet中包含残差块结构，也就时为$a^{[l]}$提供一个捷径，直接输入到$a^{[l+2]}$处，不需要在l+1层中计算。l+2激活后的值为$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$。
将$z^{[l+2]}$和$a^{[l]}$相加是基于两者形式相同的假设，实际上两者的形式可能并不相同。此时需要学习一个W_s参数矩阵，使两者保持一致：$a^{[l+2]} = g(z^{[l+2]} + W_s a^{[l]})$。

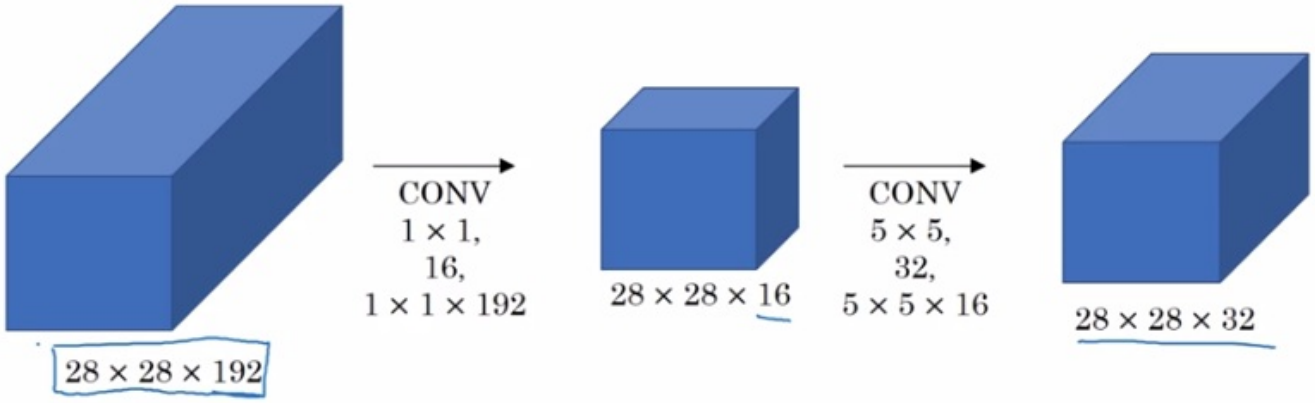
2.3 Inception网络

在卷积神经网络中，有非常多的参数可供选择，例如卷积核大小、是否添加池化层等，这使模型设计变得很复杂。Inception网络是将多种选择所产生的结果结合再一起。

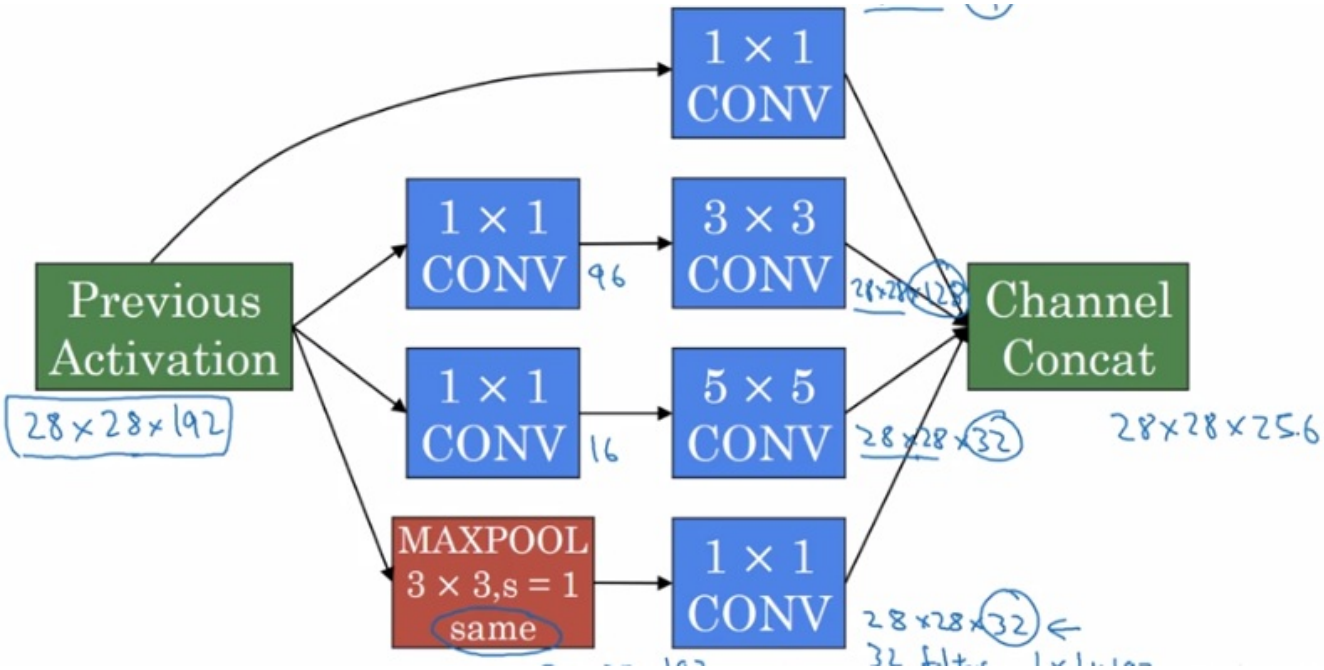


Inception网络结构十分复杂，参数很多。假设输入特征的维是28*28*192，过滤器为5*5，共有32个。所需要的乘法运算次数：每次乘法计算需要5*5*192次，一共需要28*28*32次乘法计算，总计算次数约1.2亿。

为了尽量简化计算过程，可以考虑使用1*1的过滤器简化计算。先使用16个1*1*192的过滤器将输入转换为28*28*16形式，再用32个5*5*16的过滤器，输出依然是28*28*32。此时的计算次数为：(1*1*192)*(28*28*16)+(5*5*16)*(28*28*32)，总次数约1200万次，明显简化了计算。



Inception层的构成大致如下图所示，对于非1*1过滤器的部分，首先使用1*1过滤器做中间层，再用f*f过滤器。无论使用什么形式的过滤器，都应添加padding来保持输出的高度和宽度不变(信道数可以不同)，最后合并所有过滤器产生的输出，共同构成新的网络层。



2.4 卷积神经网络的实际应用

- 使用开源代码：很多paper中提出的网络结构复杂，有效利用开源代码能节省时间。
- 迁移学习：相比于其他深度学习任务，迁移学习在计算机视觉应用中的有效率是最高的。
- 数据增强：计算机视觉任务一般需要以大量的数据为基础，当数据量不足时，可以考虑利

用数据增强来获取更多图像数据，如镜像对称、随机剪切、旋转、色彩转换等方法。

3. 目标检测

3.1 目标定位

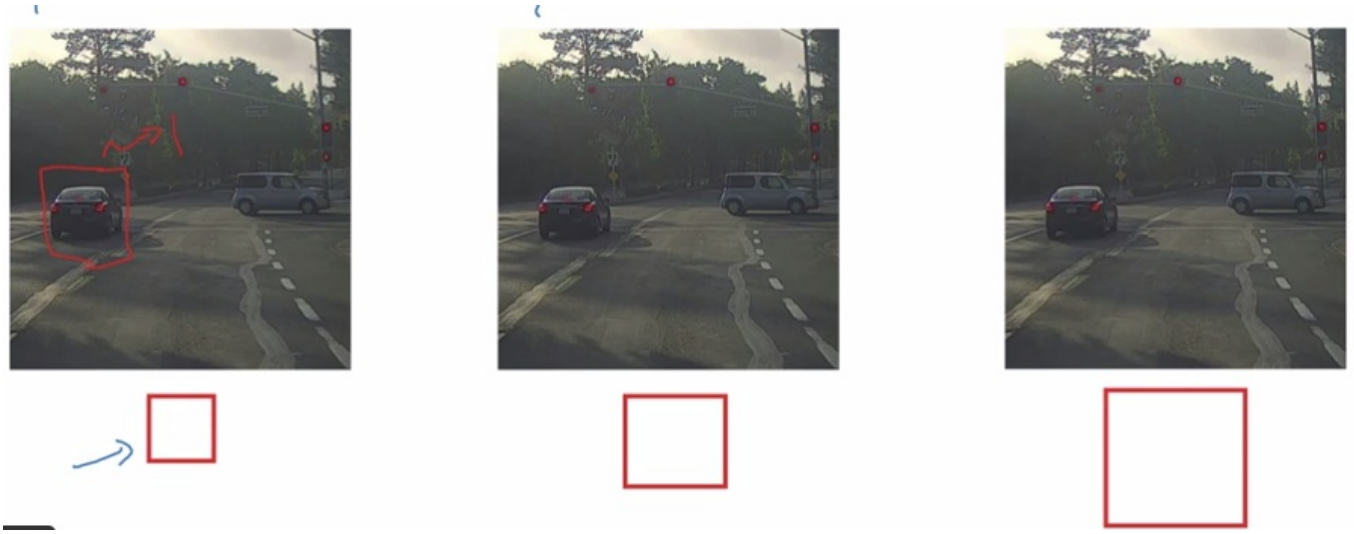
一般的图像识别问题中，我们只需要对图像进行分类，例如图片中有或者没有车。在目标检测问题中，我们不仅需要识别是否有车，还需要定位出车的具体位置。
更一般地，目标检测任务也许不只有一类目标，我们可能需要检测出是否有车、行人、红绿灯等，并且定位出每一个目标的位置。因此，目标检测任务的标签也比较复杂。假设每个图像中只有一个目标，那么标签为：$y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, \ldots, c_n]$。$p_c$用来表示图像中是否存在目标；$b_x, b_y, b_h, b_w$用来定位目标，分别表示目标在x和y轴的位置以及高度和宽度；c_i用来表示目标属于哪一类。当$p_c = 0$时，标签中的其他项不存在。
目标检测的损失函数可以用平方损失，对于$p_c! = 0$的情况，
$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 + (\hat{y}_2, y_2)^2 + \ldots + (\hat{y}_n, y_n)^2$；对于$p_c = 0$的情况，
$L(\hat{y}, y) = (\hat{y}_1, y_1)^2$。当然，损失函数也可以更为复杂，例如对$p_c$使用交叉熵损失函数，对$b_x, b_y, b_h, b_w$使用平方损失函数，对$c_i$使用softmax损失函数。

3.2 卷积的滑动窗口实现

滑动窗口检测是实现目标检测的一种方法。滑动窗口检测是有效的，但其运算速度很慢，而且很难找准目标的边界。

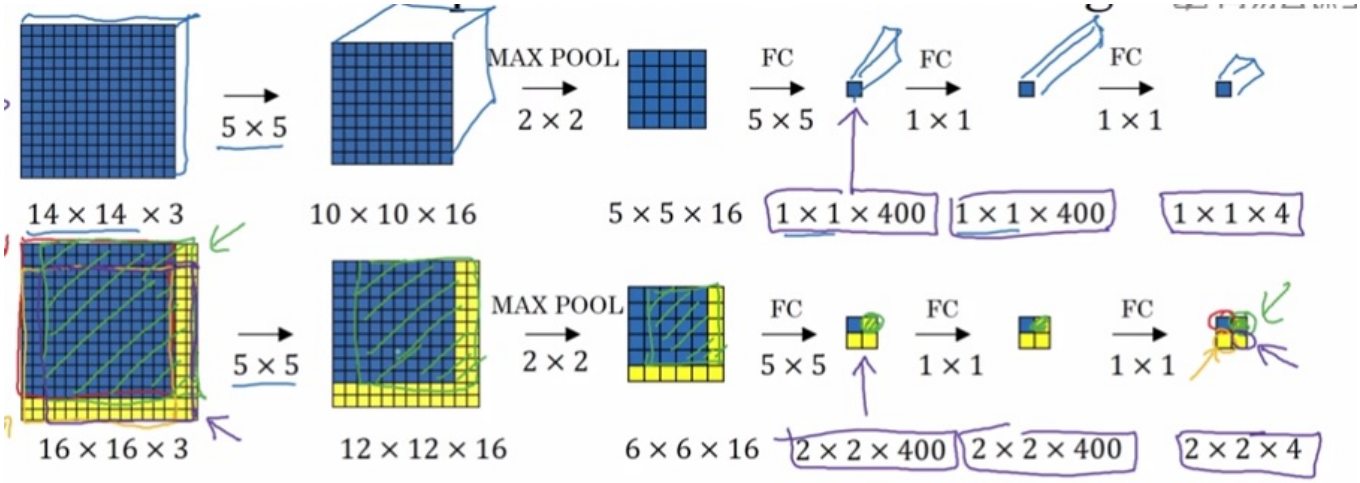
基本思路

首先训练出一个用于识别图像中是否包含目标的分类器，样本都是包含或者不包含目标的图片，目标需要占据图片的大部分并居于中心。
测试阶段，对于每一张图片，用一个小的滑窗提取图片的一小部分，使其通过模型，并判断是否包含目标。不断移动滑窗通过图片的每一个角落，识别图片每一个位置是否包含目标。完成一次滑窗遍历后，可以扩大滑窗重新遍历。只要遍历次数足够多，一定能找到图片中的目标。



卷积实现

卷积实现有助于加速滑窗检测的速度。如果网络允许的输入形式是14*14*3的图像，输出结果是0或1，表示是否存在目标。假设输入的图像是16*16*3的形式，对于一般的滑窗算法而言，共包含9个14*14*3的滑窗，需要9次执行。卷积化则是让16*16*3的图像直接通过训练出的卷积网络，输出的形式为3*3(当步长为2时，输出形式为2*2)，其中每一个位置都应为0或1，代表对应滑窗中是否检测出实体。这样输入数据只需要通过卷积网络一次，就可以计算出9个滑窗的输出结果。



R-CNN

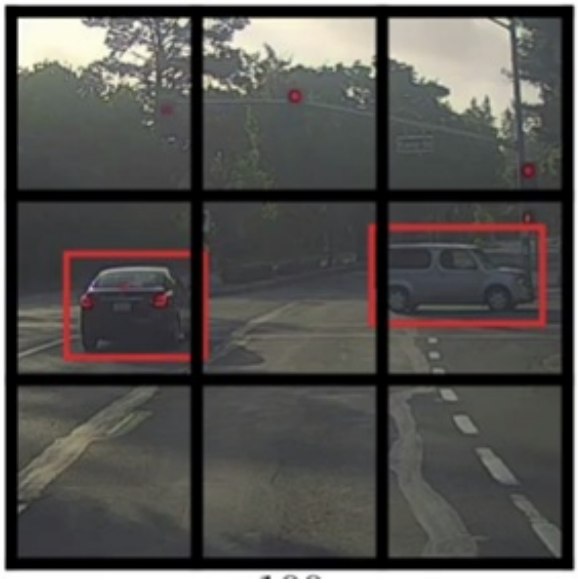
R-CNN方法首先利用图像分割方法找出图像中的一些区域作为候选区域，然后只在候选区域进行滑窗运算，来加快速度。

3.3 YOLO算法

YOLO也是实现目标检测的一种方法，是You Only Look Once的简写形式。

标签设定

YOLO算法首先将图像分割成m*m个小块，对于每个小块，分别有标签
$y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, \ldots, c_n]$。
假设将图像分为9块，识别目标共有3类，那么标签实际上为3*3*(1+4+3)形式。



目标可能存在于多个图块里，一般只选取其中一个标记为1，其余为0。
YOLO算法的标签适用于一个图像块里最多只有一个目标的情况，如果存在两个或以上则不适用了。不过当图像被分为特别多的小块时(如19*19)，这种假设基本是成立的。

交并比

交并比IoU是用于评估模型效果的，即检测出的边界框与实际边界框交集面积与并集面积的比值。一般来说，IoU高于0.5时，就可以认为检测结果是正确的。

非极大值抑制

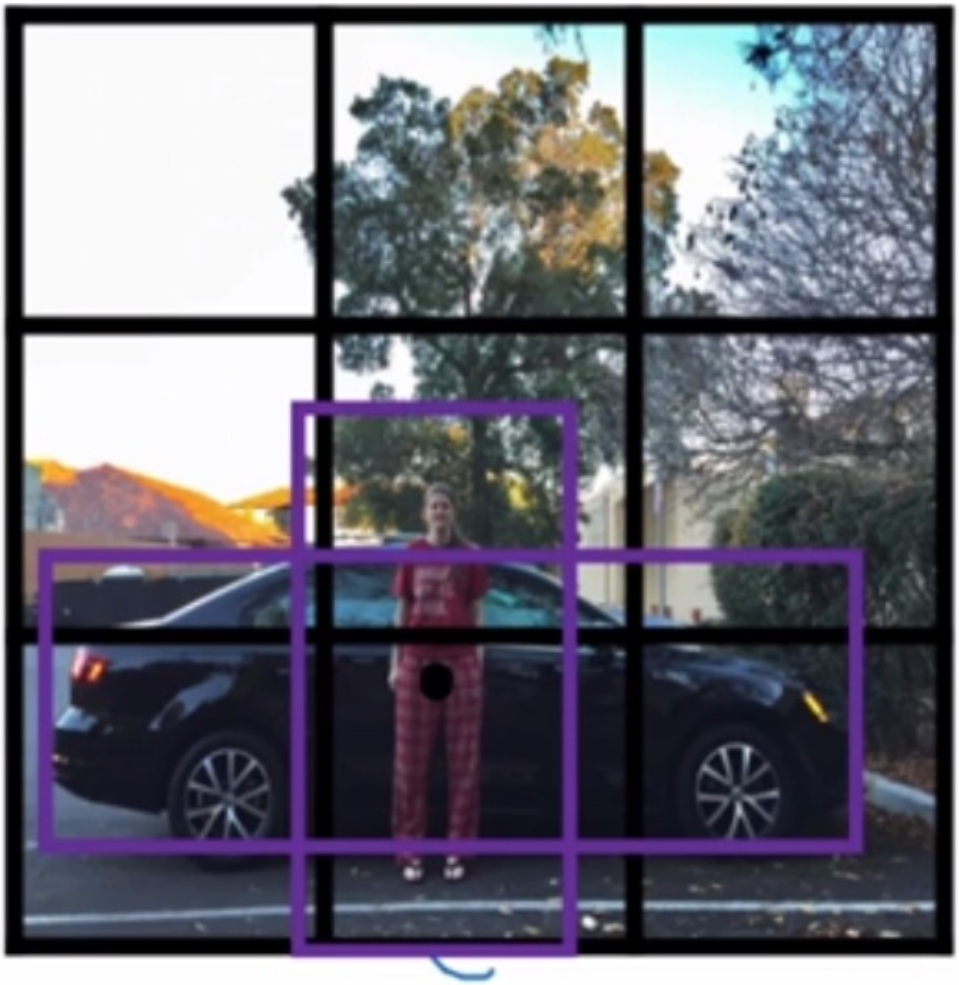
在进行YOLO算法时，我们将图像分为多个小格，可能有多个小格里包含目标，但我们只将目标标记在其中一个小格中，因此实际预测时，可能有多个小格都显示它们检测出了目标。
对此，我们通过非极大值抑制来选取这些小格中的一个。

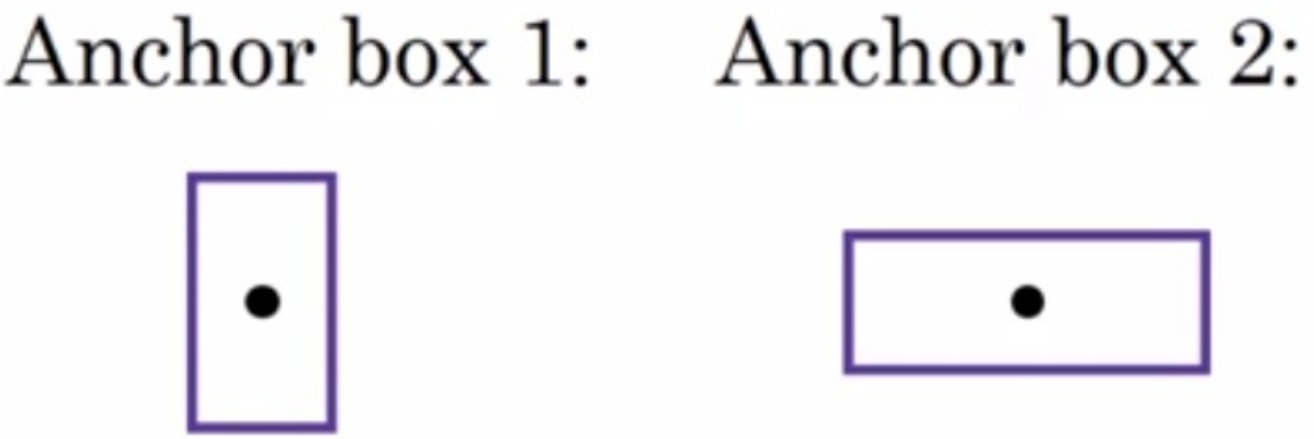
- 首先，通过设置$p_c * c_n$的阈值来排除预测概率过小的目标，仅保留概率大的预测结果。
- 选择所有目标中$p_c * c_n$最大的目标，计算与其他识别出的目标的IoU，超过一定的阈值的其他目标视为与最大目标重复，将被消除。
- 重复以上过程，直至遍历所有超过$p_c * c_n$阈值的目标

Anchor Box

为解决每个小格中只包含一个目标的问题，可以使用Anchor Box。我们首先绘制好几个Anchor Box的形状，例如纵向的为Anchor Box 1，横向的为Anchor Box 2。在标记样本时，计算目标的边界框与每个Anchor Box的交并比来判断目标应该被标记到哪一个Anchor Box。样本标签向量中的元素个数也会增加，对于原来3*3*8的样本，如果有两个Anchor Box，标签为3*3*8*2。

Anchor Box形状是人工定制的，最好可以具有代表性，能够匹配目标的形状。例如，当我们已知目标检测任务中包含人和车时，设定的Anchor Box如下图，Anchor Box 1比较接近人的形状，而Anchor Box 2比较接近车的形状。





对于目标被标记到哪一个Anchor Box中，可以用交并比来确认。

YOLO算法流程

- 设置k个Anchor Box
- 将图像切分为m*m的小窗格，对每个窗格进行标记，假设一共有n类，那么标签中的元素个数为m*m*(1+4+n)*k个。对于1+4+n，1代表包含目标的概率，4代表目标的位置，n代表目标属于每一类的概率。

- 预测结果可能出现多个窗口检测出同一个目标的情况，此时需要用非极大值抑制方法保留概率最高的一个。注意非极大值抑制仅作用于多个图像窗口识别出同一目标的情况，不同目标之间是不需要抑制的。

4. 人脸识别和神经风格转换

4.1 人脸识别

- **人脸验证**：一对一问题，只需要验证输入的人脸图像是否与某个的身份信息对应；
- **人脸识别**：一对多问题，需要验证输入的人脸图像是否与多个身份信息中的某一个匹配。

One-Shot 学习

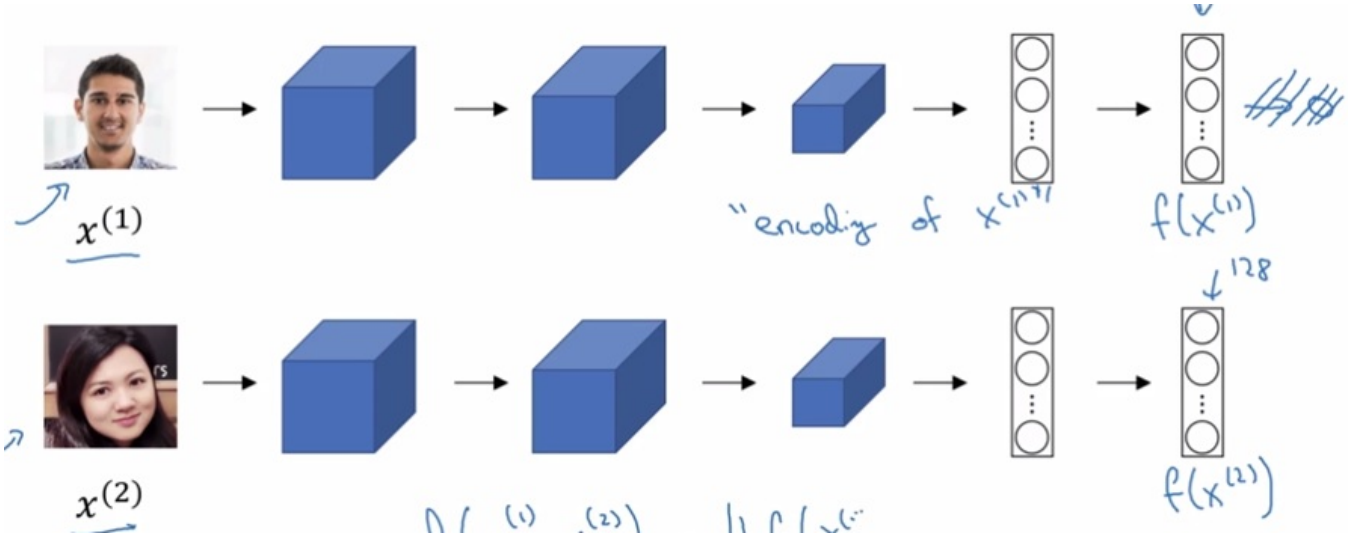
人脸识别的一个难点是需要只采集某人的一个面部样本，就可以识别出这个人，这也被称作One-Shot学习。

我们通过一个相似函数来实现One-Shot学习过程：$Similarity = d(img_1, img_2)$。可以设置一个阈值$\tau$，当两张图片的相似度小于$\tau$时，我们认为两张图片是同一个人。

Siamese网络

为了计算两张图片的相似度，我们训练一个卷积网络——Siamese网络。对于不同的图像，Siamese网络可以输出一个向量预测结果，我们比较不同图像输出向量的相似度，来确认相似度。

下图的Siamese网络有128个输出层单元，相似函数即为两个输出向量的L2范数：$d(x^{(1)}, x^{(2)}) = ||f(x^{(1)}) - f(x^{(2)})||_2^2$。



Triplet 损失

为训练Siamese网络，我们需要设置一个合适的损失函数。这个损失函数希望可以使两张相同人的图像的距离尽量小、两张不同人的图像的距离尽量大。因此训练样本需要是个三元组，包括锚样本(Anchor, A)，正样本(Positive, P)和负样本(Negative, N)，其中锚样本与正样本是同一个人，与负样本不是。

对于三张图片，应该有：$||f(A)-f(P)||_2^2+\alpha \leq ||f(A)-f(N)||_2^2$，其中$\alpha$是一个软间隔确保$||f(A)-f(P)||_2^2$可以严格小于$||f(A)-f(N)||_2^2$。

经转化可得Triplet损失函数：

$L(A,P,N)=max(||f(A)-f(P)||_2^2-||f(A)-f(N)||_2^2+\alpha,0)$。

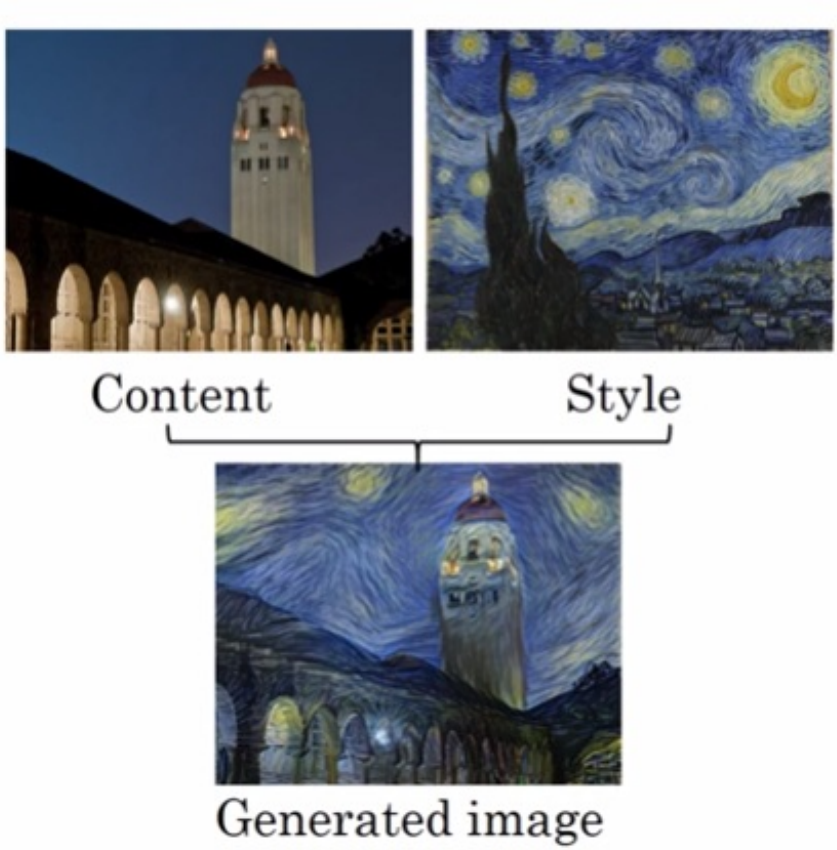
二分类人脸识别

人脸识别问题也可以转化成一个二分类问题。当训练好Siamese网络之后，可以使两个图片分别通过同一个Siamese网络，将输出的两个向量再输入到一个Sigmoid单元里：

$\hat{y}=\sigma(\sum_{k=i}^{K}w_k|f(x^{(i)})_k-f(x^{(j)})_k|+b)$。可以通过Sigmoid单元的输出结果来判断两张图片是否为同一个人。

4.2 神经风格迁移

神经风格迁移是将一张图片中的内容(content)迁移到另一张图片的风格(style)上，生成全新的图片。



代价函数

神经风格迁移任务包含内容图片和风格图片，并利用它们生成新的图片。因此我们希望新的图片与内容图片的内容差距尽量小，与风格图片的风格差距尽量小。因此代价函数为：
$J(G) = \alpha \cdot J_{content}(C,G) + \beta \cdot J(S,G)$，其中$\alpha$和$\beta$是控制两者比重的超参数。
在神经风格迁移任务中，我们首先随机产生一个图像G，再不断更新参数使代价函数最小化，来改进G，使其接近我们所期望的图像。

内容代价成分

对于一个已经预训练好的神经网络(例如VGG等)，我们选择一个隐藏层l来计算内容代价。l最好处于网络的中间部分，不宜过深或过浅。内容代价函数为
$J_{content}(C,G) = ||a^{(C)[l]} - a^{(G)[l]}||_2^2$。

风格代价成分

在卷积网络中，风格被定义为同一隐藏层中不同通道的相关系数。具体地，对于图片的隐藏层l，i和j分别代表图像的高度和宽度，k为通道(k'和k一样也是通道，为了区分于k)，因此$a_{ijk}^{[l]}$为一个激活结果。Gram矩阵形式为：$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^l} \sum_{j=1}^{n_W^l} a_{ijk}^{[l]} a_{ijk'}^{[l]}$。

假设风格图像的Gram矩阵为$G^{(S)}$，生成图像的Gram矩阵为$G^{(G)}$，则第l层的风格代价函数为：$J_{style}^{[l]}(S,G)=||G^{[l](S)}-G^{[l](G)}||_2^2$。如果对各层都是使用风格代价函数，则有$J_{style}(S,G)=\sum_{l=1}^{L}\lambda^{[l]}J_{style}^{[l]}(S,G)$，其中$\lambda$用于调节各层的权重。