第四门课 卷积神经网络（Convolutional Neural Networks）

第一周 卷积神经网络（Foundations of Convolutional Neural Networks）

CNN卷积神经网络

卷积神经网络（ConvolutionalNeuralNetwork，CNN）最初是为解决图像识别等问题设计的，CNN现在的应用已经不限于图像和视频，也可用于时间序列信号，比如音频信号和文本数据等。

与一般的DNN模型相比，CNN模型独有两个隐藏层，分别为卷积层（Convolution Layer）和池化层(Pooling layer)，卷积层的激活函数使用的是ReLU，池化层没有激活函数。

一个典型的卷积神经网络通常有三层，一个是卷积层，我们常常用**Conv**来标注。

还有两种常见类型的层，一个是池化层，我们称之为**POOL**。

最后一个是全连接层，用**FC**表示。

虽然仅用卷积层也有可能构建出很好的神经网络，但大部分神经网络架构师依然会添加池化层和全连接层。

幸运的是，池化层和全连接层比卷积层更容易设计。

因此，想要高度CNN，只需要在搞懂DNN的基础上，把卷积层和池化层的原理搞清楚。

卷积运算是卷积神经网络最基本的组成部分。

Padding

如果我们有一个n x n的图像，用的f x f过滤器做卷积，那么输出的维度就是（n-f+1）x (n-f+1)。

如此进行的卷积操作就会出现两个问题：

1.每次卷积后输出变小

2.那些在角落或者边缘区域的像素点在输出中采用较少，意味着你丢掉了图像边缘位置的许多信息。

解决方法：在卷积操作前填充图像。【Padding】

比如6x6的图像卷积操作前填充为8x8的图像，再用3x3的过滤器进行卷积后得到就是6x6的图像。

当填充数量为p，则输出变为（n-f+2p+1）x （n-f+2p+1).

填充方法：Valid卷积和Same卷积

Valid卷积即不填充像素，即p=0

Same卷积即以上方法，求解n-f+2p+1 = n，当f是一个奇数，p = （f-1）/2

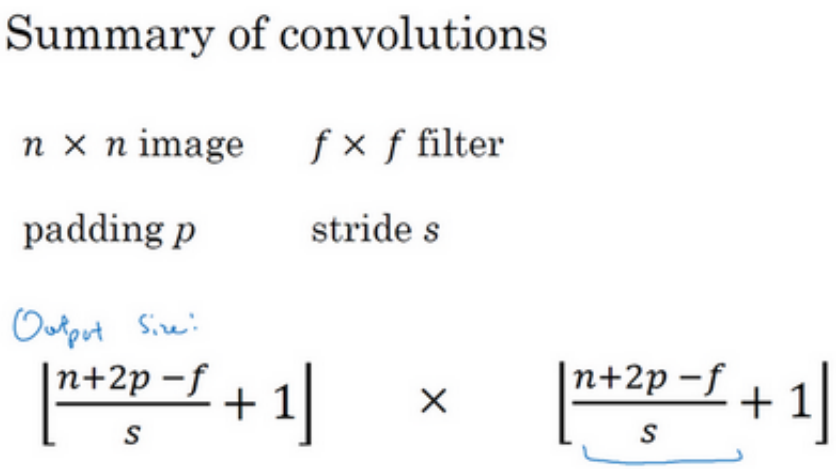
一般f都为奇数。

卷积步长（Strided convolution）

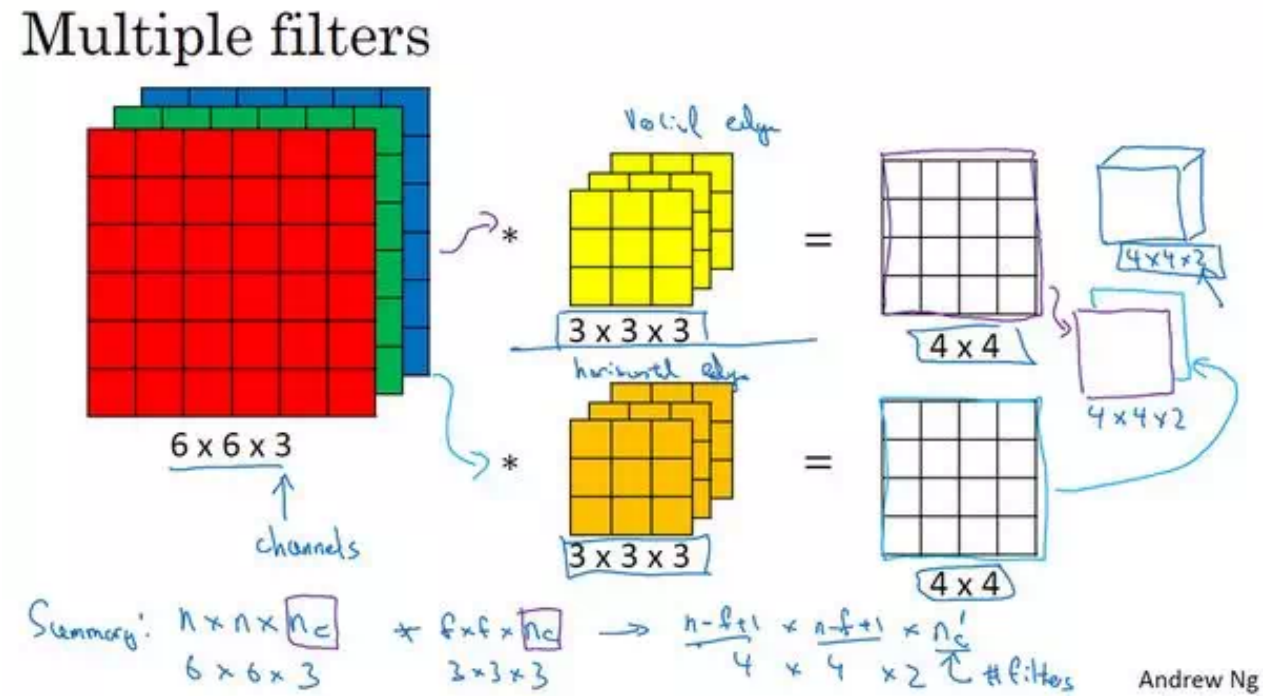
使用3x3的过滤器进行卷积7x7的图像时，设置卷积步长s=2，则得到一个3x3的输出。

即输出变为 (（n-f+2p）/s + 1) x (（n-f+2p）/s + 1)

当（n-f+2p）/s不是整数时向下取整



三维卷积（Convolutions over volumes）



如果你有一个 n x n x nc（通道数）的输入图像，在这个例子中就是6×6×3，这里的nc就是通道数目，然后卷积上一个f x f x nc，这个例子中是3×3×3，按照惯例，这个（前一个nc）和这个（后一个nc）必须数值相同。然后你就得到了（n-f+1）x （n-f+1）x nc'，这里的nc'其实就是下一层的通道数，它就是你用的过滤器的个数，在我们的例子中，那就是4×4×2。我写下这个假设时，用的步幅为1，并且没有padding。如果你用了不同的步幅或者padding，那么这个数值会变化，正如前面的视频演示的那样。

单层卷积网络（one layer of a convolution network）

--建立CNN的卷积层，了解卷积层的工作原理以及如何计算输出映射到下一层作为激活值

⚠️卷积层应用非线性函数ReLu

以上面为例，从6×6×3的输入推导出一个4×4×2矩阵，它是卷积神经网络的一层，把它映射到标准神经网络中四个卷积层中的某一层或者一个非卷积神经网络中。

在前向传播中，输入和相关过滤器等于执行线性函数，再加上偏差，作为该卷积层的输入z，然后应用于非线性函数ReLu得到4×4×2矩阵

具体：

首先执行线性函数，然后所有元素相乘做卷积，具体做法是运用线性函数再加上偏差，然后应用激活函数**ReLU**。这样就通过神经网络的一层把一个6×6×3的维度a0演化为一个4×4×2维度的a1，这就是卷积神经网络的一层。

🌟与一般全连接层对比，不考虑数据形式，实质仍是输入与参数相乘后加上偏差，最后执行sigmoid函数，但这里是ReLu。

池化层(Pooling layers)

--使用池化层缩减模型大小，提高计算速度，提高所提取特征的鲁棒性【健壮性】

假如输入是一个4×4矩阵，用到的池化类型是最大池化（**max pooling**）。执行最大池化的树池是一个2×2矩阵。执行过程非常简单，把4×4的输入拆分成不同的区域，我把这个区域用不同颜色来标记。对于2×2的输出，输出的每个元素都是其对应颜色区域中的最大元素值。

计算卷积层输出大小的公式同样适用于最大池化，即(（n-f+2p）/s + 1)，这个公式也可以计算最大池化的输出大小。

🌟还有一种池化为平均池化，选取的不是某个区域的最大值，而是该区域的平均值

最大池化只是计算神经网络某一层的静态属性。

计算神经网络有多少层时，通常只统计具有权重和参数的层。因为池化层没有权重和参数，只有一些超参数。这里，我们把**CONV1**和**POOL1**共同作为一个卷积，并标记为**Layer1**。

至于如何选定这些参数，后面我提供更多建议。常规做法是，尽量不要自己设置超参数，而是查看文献中别人采用了哪些超参数，选一个在别人任务中效果很好的架构，那么它也有可能适用于你自己的应用程序

卷积神经网络的优势

--卷积神经网络在计算机视觉任务中表现良好

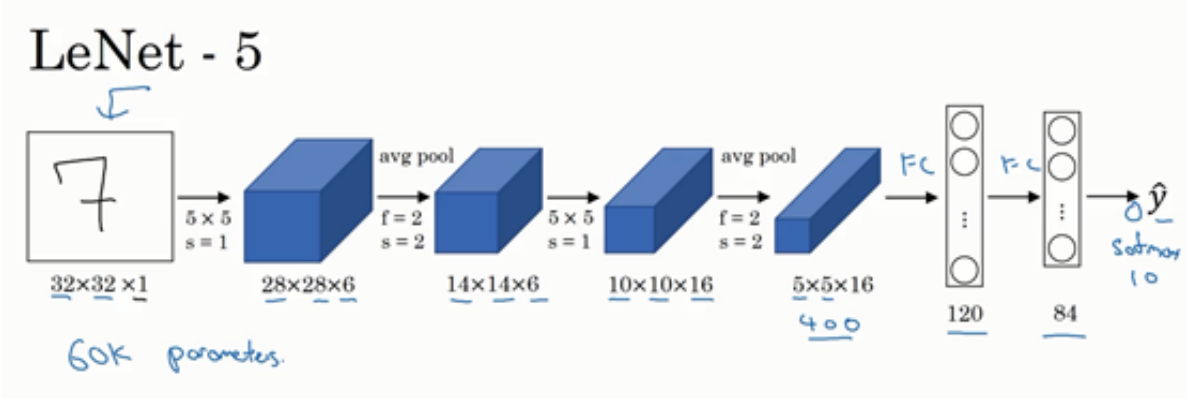
参数共享：输入矩阵共享过滤器

稀疏连接：每次卷积运算只需要用到输入矩阵中同过滤器大小相同的元素。

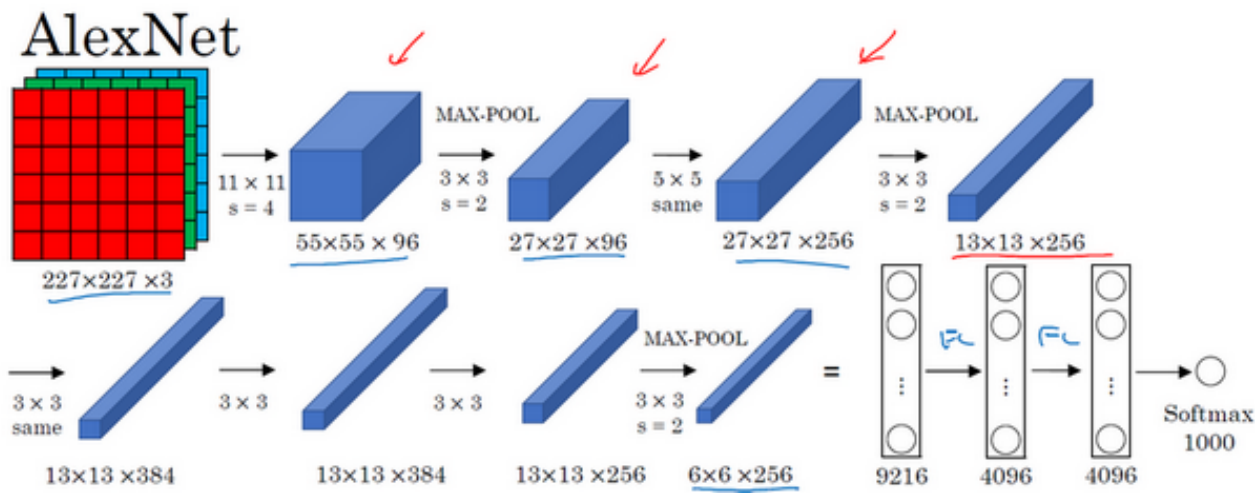
第二周 深度卷积网络：实例探究（Deep convolutional models: case studies）

经典网络（Classic networks）： LeNet-5、AlexNet和VGGNet

LeNet-5



AlexNet

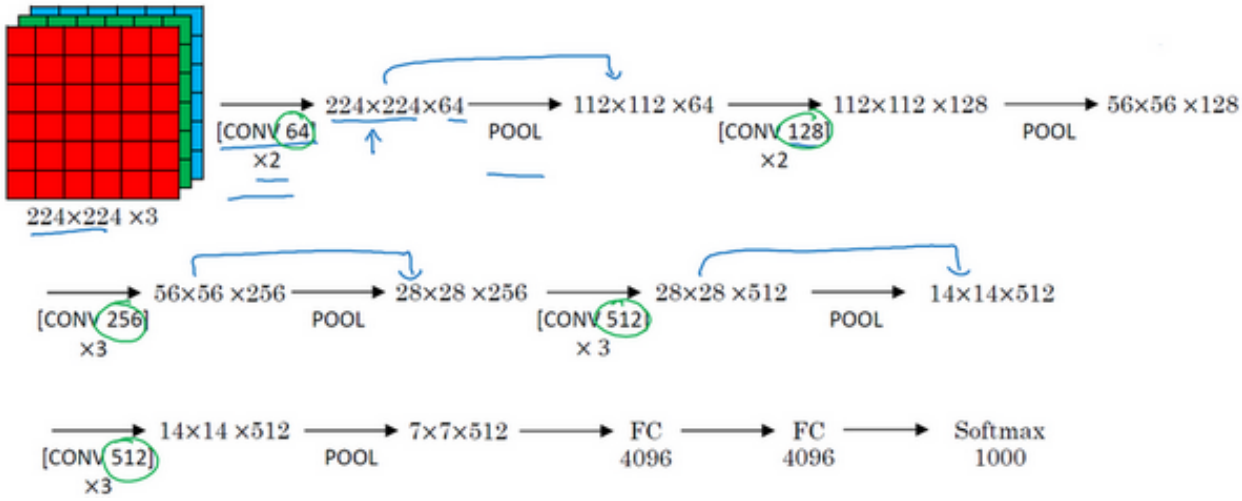


AlexNet比**LeNet-5**要大得多，能够处理非常相似的基本构造模块，这些模块往往包含着大量的隐藏单元或数据。

AlexNet比**LeNet**表现更为出色的另一个原因是它使用了**ReLu**激活函数。

VGGNet

只使用用3×3，步幅为1的过滤器构建卷积层，**padding**参数为**same**卷积中的参数。然后用一个2×2，步幅为2的过滤器构建最大池化层。



VGG网络的一大优点是它确实简化了神经网络结构

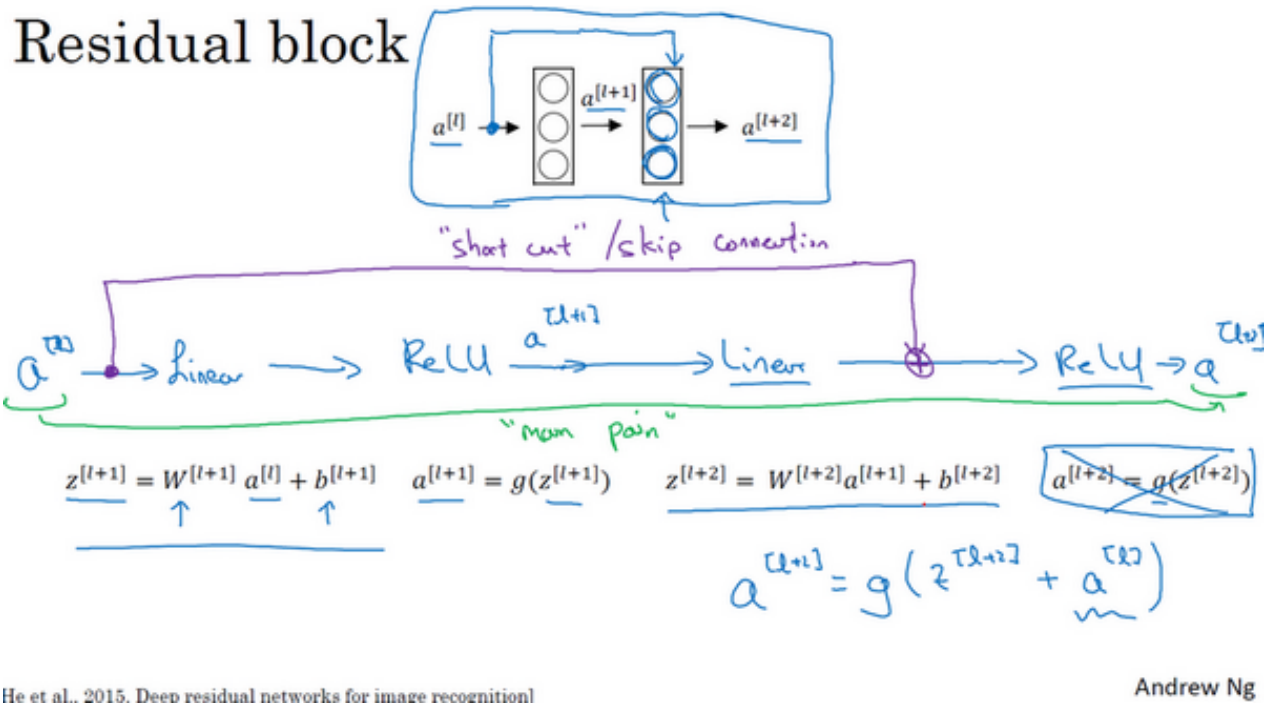
论文阅读：从介绍**AlexNet**的论文开始，然后就是**VGG**的论文，最后是**LeNet**的论文。

残差网络(ResNets)（Residual Networks (ResNets)）

非常非常深的神经网络是很难训练的，因为存在梯度消失和梯度爆炸问题。

跳跃连接（**Skip connection**）：从某一层网络层获取激活，然后迅速反馈给另外一层，甚至是神经网络的更深层。

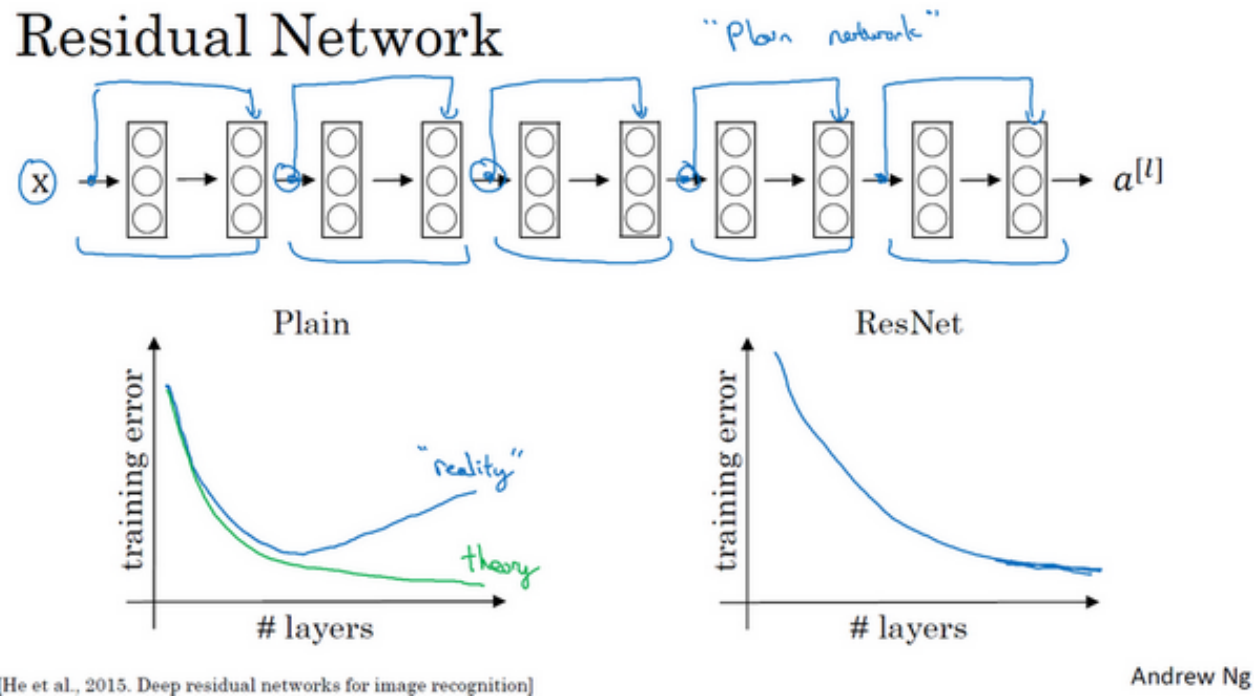
我们可以利用跳跃连接构建能够训练深度网络的**ResNets**



He et al., 2015. Deep residual networks for image recognition

Andrew Ng

某一层的特征输入到其后面的层进行线性计算Z之后，执行非线性函数relu之前。



[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

用标准优化算法训练一个普通网络，凭经验你会发现随着网络深度的加深，训练错误会先减少，然后增多。

但有了**ResNets**就不一样了，即使网络再深，训练的表现却不错，比如说训练误差减少，就算是训练深达100层的网络也不例外。

ResNets有助于解决梯度消失和梯度爆炸问题。

当添加残差块那一层的参数为0时，由公式的关系可计算出该层的输出恒等于残差块所在层的特征值，因此不论是把残差块添加到神经网络的中间还是末端位置，都不会影响网络的表现。当参数不为0时，残差网络不仅仅是保持网络的效率，还能提升它的效率。因此残差网络起作用的主要原因就是这些残差块学习恒等函数非常容易，你能确定网络性能不会受到影响，很多时候甚至可以提高效率，或者说至少不会降低网络的效率，因此创建类似残差网络可以提升网络性能。

相比一般的NN，当网络不断加深时，就算是选用学习恒等函数的参数都很困难，所以很多层最后的表现不但没有更好，反而更糟。

普通网络和**ResNets**网络常用的结构是：卷积层-卷积层-卷积层-池化层-卷积层-卷积层-卷积层-池化层……依此重复。直到最后，有一个通过**softmax**进行预测的全连接层。

网络中的网络OR 1×1 卷积（Network in Network or 1×1 convolutions）

1×1卷积层实现了一些重要功能（**doing something pretty non-trivial**），它给神经网络添加了一个非线性函数，从而减少或保持输入层中的通道数量不变，当然如果你愿意，也可以增加通道数量。

谷歌 Inception 网络简介（Inception network motivation）

Inception网络或**Inception**层的作用就是代替人工来确定卷积层中的过滤器类型，或者确定是否需要创建卷积层或池化层。

分别得出需要决定选择的通道，把得到的各个层的通道都加起来，最后得到一个28×28×256的输出。通道连接实际就是之前视频中看到过的，把所有方块连接在一起的操作。这就是一个**Inception**模块，而**Inception**网络所做的就是将这些模块都组合到一起。

结一下，如果你在构建神经网络层的时候，不想决定池化层是使用1×1，3×3还是5×5的过滤器，那么**Inception**模块就是最好的选择。我们可以应用各种类型的过滤器，只需要把输出连接起来。

第三周 目标检测（Object detection）

目标检测（对象检测）是计算机视觉领域中一个新兴的应用方向

目标定位（Object localization）

图片分类：遍历图片，判断其中的对象是不是汽车

定位分类：不仅要用算法判断图片中是不是一辆汽车，还要在图片中标记出它的位置，用边框或红色方框把汽车圈起来。

如果你正在构建汽车自动驾驶系统，那么对象可能包括以下几类：行人、汽车、摩托车和背景，这意味着图片中不含有前三种对象，也就是说图片中没有行人、汽车和摩托车，输出结果会是背景对象，这四个分类就是softmax函数可能输出的结果。

如果你还想定位图片中汽车的位置，我们可以让神经网络多输出几个单元，输出一个边界框。具体说就是让神经网络再多输出4个数字，这四个数字是被检测对象的边界框的参数化表示，即bx，by，bw，bh。

（bx，by）为目标中心坐标，bw表示边界框长度，bh表示边界框高度。

因此，特征输入x为一张图片，对应的y应为：

pc表示是否识别到对象，是则为1，否则为0。pc=0其他参数都不重要，可为?

当pc为1，需要对应边界框的参数化表示以及c1、c2、c3何值标示为1表示识别为哪个对象。

采用平方误差策略，对应的损失函数为：

$$L\left(\hat{y}, y\right)=\left(\hat{y_1}-y_1\right)^2+\left(\hat{y_2}-y_2\right)^2+\ldots\left(\hat{y_8}-y_8\right)^2$$

损失值等于每个元素相应差值的平方和。

当y=1时，平方误差策略可以减少这些元素预测值和实际输出结果之间差值的平方。

当y=0时， 矩阵中的后7个元素都不用考虑，只需要考虑神经网络评估（即pc）的准确度。

特征点检测（Landmark detection）

由目标定位我们可以识别出人脸之后，我们希望算法可以给出眼角或其他部位的具体位置。

一个位置以一个点表示，对应（lx，ly），作为眼角的坐标值

对应的y就变为了：

一个值表示是否有人脸，具体位置的坐标值。如需要标示64个点，则y需要128+1个值

目标检测（Object detection）

由对象定位和特征点检测来构建对象检测算法。以下采用基于滑动窗口的目标检测算法

适当剪切，使有汽车的图片占据基本整个照片。

由此类图片训练CNN得到预测y，0或1表示图片中有汽车或没有汽车。

训练完这个卷积网络，就可以用它来实现滑动窗口目标检测。

假设这是一张测试图片，首先选定一个特定大小的窗口，比如图片下方这个窗口，将这个红色小方块输入卷积神经网络，卷积网络开始进行预测，即判断红色方框内有没有汽车。

红色方框稍向右滑动，直到这个窗口滑过图像的每一个角落，不断对窗口图片进行识别。

思路是以固定步幅移动窗口，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，这就是所谓的图像滑动窗口操作。

选择一个更大的窗口，截取更大的区域，重复以上操作。

不论汽车在图片的什么位置，总有一个窗口可以检测到它。

缺点：在图片中剪切出太多小方块，卷积网络要一个个地处理，导致计算成本过大。

选用的步幅很大，显然会减少输入卷积网络的窗口个数，但粗粒度会影响性能。

采用小粒度或小步幅，传递给卷积网络的小窗口会特别多，这意味着超高的计算成本。

在卷积层上应用目标检测（Convolutional implementation of sliding windows）

为了构建滑动窗口的卷积应用，首先要知道如何把神经网络的全连接层转化成卷积层。

使用一定数量的过滤器得到的输出替换全连接层。

通过卷积实现滑动窗口对象检测算法：

以训练集图片维度训练出模型

一般的滑动窗口卷积网络在维度更大的测试集

不断移动分别进行卷积运行，此处进行四次

最后的输出是2x2，分别对应四次结果。

可以看到，这4次卷积操作中很多计算都是重复的。所以执行滑动窗口的卷积时使得卷积网络在这4次前向传播过程中共享很多计算。

因此：

该卷积操作的原理是我们不需要把输入图像分割成四个子集，分别执行前向传播，而是把它们作为一张图片输入给卷积网络进行计算，一次得到所有预测值，如果足够幸运，神经网络便可以识别出汽车的位置。

Bounding Box预测（Bounding box predictions）

滑动窗口法的卷积实现存在问题： 不能输出最精准的边界框。

其中一个能得到更精准边界框的算法是**YOLO**算法，**YOLO**(**You only look once**)意思是你只看一次，这是由**Joseph Redmon**，**Santosh Divvala**，**Ross Girshick**和**Ali Farhadi**提出的算法。

比如你的输入图像是100×100的，然后在图像上放一个网格。比如3×3网格，实际实现时会用更精细的网格，可能是19×19。

基本思路是使用图像分类和定位算法，然后将算法应用到9个格子上。

每个格子得到输出：

总的输出尺寸是3×3×8。

这个算法的优点在于神经网络可以输出精确的边界框，所以测试的时候，你做的是喂入输入图像，然后跑正向传播，直到你得到这个输出y。

可以看到，该算法让神经网络输出边界框，可以具有任意宽高比，并且能输出更精确的坐标，不会受到滑动窗口分类器的步长大小限制。

其次，这是一个卷积实现，你并没有在3×3网格上跑9次算法，或者，如果你用的是19×19的网格，19平方是361次，所以你不需要让同一个算法跑361次。相反，这是单次卷积实现，但你使用了一个卷积网络，有很多共享计算步骤，在处理这3×3计算中很多计算步骤是共享的，或者你的19×19的网格，所以这个算法效率很高。

交并比（Intersection over union）

使用交并比函数，可以用来评价对象检测算法：

交并比（**IoU**）函数做的是计算两个边界框交集和并集之比。

IoU>=0.5,表示检测正确。

如果预测器和实际边界框完美重叠，**IoU**就是1，因为交集就等于并集。

非极大值抑制（Non-max suppression）

非最大值抑制可以让**YOLO**算法输出效果更好。

假设你需要在这张图片里检测行人和汽车，你可能会在上面放个19×19网格，理论上这辆车只有一个中点，所以它应该只被分配到一个格子里，左边的车子也只有一个中点，所以理论上应该只有一个格子做出有车的预测。

但实践中，有一些中点附近的格子也会被认为有车

非极大值抑制就是把一些可能有车的结果清理，取概率最高的结果作为输出，即图中高亮显示。

具体：

以只做汽车检测为例，由目标检测得到pc<=0.6的边界框抛弃。

然后遍历剩下概率较高的边界框：选择pc最高的保留，去掉所有没有达到输出标准的边界框。

Anchor Boxes

实现一个格子检测出多个对象。

行人的中点和汽车的中点几乎在同一个地方，两者都落入到同一个格子中，一般的y将无法输出检测结果，所以我必须从两个检测结果中选一个。

anchor box：预先定义两个不同形状的**anchor box**，或者**anchor box**形状，你要做的是把预测结果和这两个**anchor box**关联起来。

得到的y变为：

可以把anchorbox1关联到行人的预测，把anchorbox2关联到汽车的预测。

YOLO 算法（Putting it together: YOLO algorithm）

把所有组件组装在一起构成**YOLO**对象检测算法。

假设你要训练一个算法去检测三种对象，行人、汽车和摩托车，你还需要显式指定完整的背景类别。

这里有3个类别标签，如果你要用两个**anchor box**，则y为3x3x2x8，8包括pc和边界框表示，以及是三个类型。

1.构造训练集，需要遍历9个格子，然后构成对应的目标向量y。

遍历3×3网格的所有位置，你会得到这样一个向量，得到一个16维向量，所以最终输出尺寸就是3×3×16。

2.训练一个卷积网络，输入是图片，可能是100×100×3，然后你的卷积网络最后输出尺寸是，此处是3×3×16或者3×3×2×8。

3.对不同分类运行非极大值抑制，使用两个**anchor box**，那么对于9个格子中任何一个都会有两个预测的边界框。

抛弃概率很低的预测，即神经网络预测没有对象的输出。

第四周 特殊应用：人脸识别和神经风格转换（Special applications: Face recognition &Neural style transfer）

在人脸识别的相关文献中，人们经常提到人脸验证（**face verification**）和人脸识别（**face recognition**）。

人脸验证问题，如果你有一张输入图片，以及某人的ID或者是名字，这个系统要做的是，验证输入图片是否是这个人。有时候也被称作1对1问题，只需要弄明白这个人是否和他声称的身份相符。

人脸识别问题比人脸验证问题难很多。

One-Shot学习（One-shot learning）

人脸识别所面临的一个挑战就是你需要解决一次学习问题。

这意味着在大多数人脸识别应用中，你需要通过单单一张图片或者单单一个人脸样例就能去识别这个人。即训练模型时数据库只有一个人脸的一张图片，如果有新人加入，就需要重新训练模型。

使用**Similarity**函数：以两张图片作为输入，然后输出这两张图片的差异值。

$$d(img1, img2) = degree\ of\ difference\ between\ images$$

，如果差异值大于τ，就能预测这是不同的两个人，这就是解决人脸验证问题的一个可行办法。

这时当新人加入，若本来训练的模型已经学习**Similarity**函数，则只需要将其图片放进数据库即可。

Siamese 网络（Siamese network）

使用 Siamese 网络（Siamese network）实现**Similarity**函数。

对于两个不同的输入，运行相同参数的卷积神经网络，得到一个不等同的相同维度的向量，然后比较它们。

实际要做的就是训练一个网络，神经网络的参数定义了一个编码函数f(xi)，它计算得到的编码可以用于函数d，它可以告诉你两张图片是否是同一个人。

如果是同一个人，得到的两个编码的距离就小。

如果不是同一个人，得到的两个编码的距离就很大。

Triplet 损失（Triplet 损失）

要想通过学习神经网络的参数来得到优质的人脸图片编码，方法之一就是定义三元组损失函数然后应用梯度下降。

三元组损失函数的定义基于三张图片，假如三张图片A、P、N，即**Anchor**样本、**Positive**样本和**Negative**样本，其中**Positive**图片和**Anchor**图片是同一个人，但是**Negative**图片和**Anchor**不是同一个人。

α也叫做间隔(**margin**),阻止网络输出无用的结果.

max函数的作用就是只要使绿色部分小于等于0，则模型的损失就是0。只要你能使画绿色下划线部分小于等于0，只要你能达到这个目标，那么这个例子的损失就是0。

假如你有一个10000个图片的训练集，里面是1000个不同的人的照片，你要做的就是取这10000个图片，然后生成这样的三元组，然后训练你的学习算法，对这种代价函数用梯度下降。

注意：构建训练集时，需要尽可能选择难训练的三元组A、P、N。如果随机的选择这些三元组，其中有太多会很简单，梯度算法不会有什么效果，因为网络总是很轻松就能得到正确的结果，只有选择难的三元组梯度下降法才能发挥作用，使得这两边离得尽可能远。

人脸验证与二分类（Face verification and binary classification）

将人脸识别当成一个监督学习的二分类问题。

输入x为一组图片，输出y是0或者1，取决于你的输入是相似图片还是非相似图片。

训练一个**Siamese**网络，意味着上面这个神经网络拥有的参数和下面神经网络的相同，两组参数是绑定的，这样的系统效果很好，最后把两组输出输入到逻辑回归单元。

神经风格迁移（neural style transfer?）

卷积神经网络最有趣的应用是神经风格迁移。

C来表示内容图像，S表示风格图像，G表示生成的图像。

神经风格迁移即使用S的风格来绘画C得到G。

代价函数（Cost function）：

$$J(G) = aJ_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

第一部分被称作内容代价，这是一个关于内容图片和生成图片的函数，它是用来度量生成图片G的内容与内容图片C的内容有多相似。

第二部分是一个风格代价函数，也就是关于S和G的函数，用来度量图片S的风格和图片G的风格的相似度。

随机初始化生成图像G，它可能是100×100×3，可能是500×500×3，又或者是任何你想要的尺寸,然后使用梯度下降的方法将其最小化函数J

设你从这张内容图片（编号1）和风格（编号2）图片开始，这是另一张公开的毕加索画作，当你随机初始化G，你随机初始化的生成图像就是这张随机选取像素的白噪声图（编号3）。接下来运行梯度下降算法，最小化代价函数，逐步处理像素，这样慢慢得到一个生成图片（编号4、5、6），越来越像用风格图片的风格画出来的内容图片。

内容代价函数（Content cost function）：

$$J_{\text{content}}(C, G) = \frac{1}{2}||a^{[l][C]} - a^{[l][G]}||^2$$

用隐含层l来计算内容代价,使用一个预训练的卷积模型,取l层的隐含单元的激活值，按元素相减，内容图片的激活值与生成图片相比较，然后取平方，也可以在前面加上归一化或者不加，比如1/2或者其他的，都影响不大,因为这都可以由这个超参数来调整

风格代价函数（Style cost function）：