

1 卷积神经网络

1.1 卷积基础知识

1.1.1 计算机视觉

计算机视觉 (Computer Vision) 的高速发展标志着新型应用产生的可能，例如自动驾驶、人脸识别、创造新的艺术风格。人们对于计算机视觉的研究也催生了很多计算机视觉与其他领域的交叉成果。一般的计算机视觉问题包括以下几类：

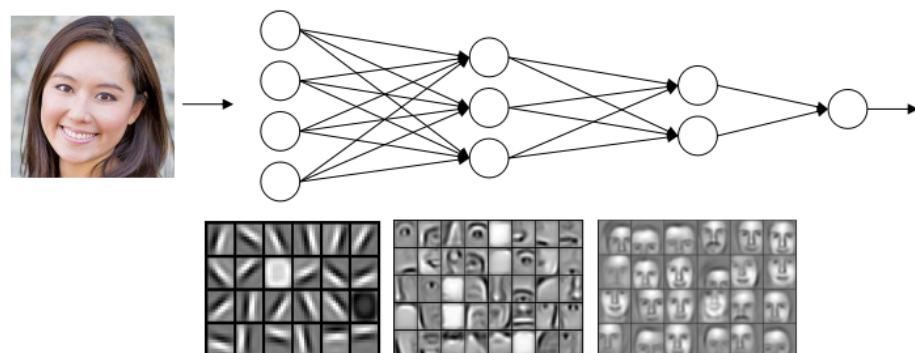
- 图片分类 (Image Classification) ；
- 目标检测 (Object detection) ；
- 神经风格转换 (Neural Style Transfer) 。

应用计算机视觉时要面临的一个挑战是数据的输入可能会非常大。例如一张 $1000 \times 1000 \times 3$ 的图片，神经网络输入层的维度将高达三百万，使得网络权重 W 非常庞大。这样会造成两个后果：

- 神经网络结构复杂，数据量相对较少，容易出现过拟合；
- 所需内存和计算量巨大。

因此，一般的神经网络很难处理蕴含着大量数据的图像。解决这一问题的方法就是使用**卷积神经网络 (Convolutional Neural Network, CNN)**。

1.1.2 基本概念



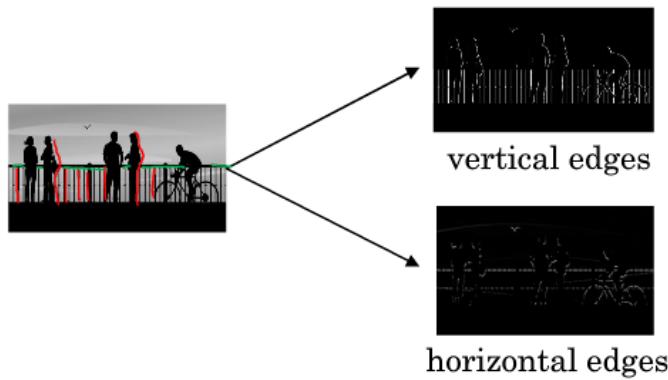
前面在神经网络中提到过，构建一个深度神经网络来进行人脸识别时，深度神经网络的前面一层可以用来进行**边缘探测**，其次一层用来探测照片中组成面部的各个特征部分，到后面的一层就可以根据前面获得的特征识别不同的脸型等等。其中的这些工作，都是依托CNN实现的。

1.1.2.1 卷积运算

卷积运算 (Convolutional Operation) 是卷积神经网络最基本的组成部分。我们以边缘检测为例，来解释卷积是怎样运算的。

边缘检测

图片最常做的边缘检测有两类：垂直边缘 (Vertical Edges) 检测和水平边缘 (Horizontal Edges) 检测。



图片的边缘检测可以通过与相应滤波器进行卷积来实现。以垂直边缘检测为例，原始图片尺寸为 6x6，中间的矩阵被称作**滤波器 (filter)**，尺寸为 3x3，卷积后得到的图片尺寸为 4x4，得到结果如下（数值表示灰度，以左上角和右下角的值为例）：

Vertical edge detection

$$\begin{array}{|c|c|c|c|c|c|} \hline
 3 & 0 & 1 & 2 & 7 & 4 \\ \hline
 1 & 5 & 8 & 9 & 3 & 1 \\ \hline
 2 & 7 & 2 & 5 & 1 & 3 \\ \hline
 0 & 1 & 3 & 1 & 7 & 8 \\ \hline
 4 & 2 & 1 & 6 & 2 & 8 \\ \hline
 2 & 4 & 5 & 2 & 3 & 9 \\ \hline
 \end{array} * \begin{array}{|c|c|c|} \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 \end{array} = \begin{array}{|c|c|c|c|} \hline
 -5 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
 & & & -16 \\ \hline
 \end{array}$$

filter

可以看到，卷积运算的求解过程是从左到右，由上到下，每次在原始图片矩阵中取与滤波器同等大小的一部分，每一部分中的值与滤波器中的值对应相乘后求和，将结果组成一个矩阵。

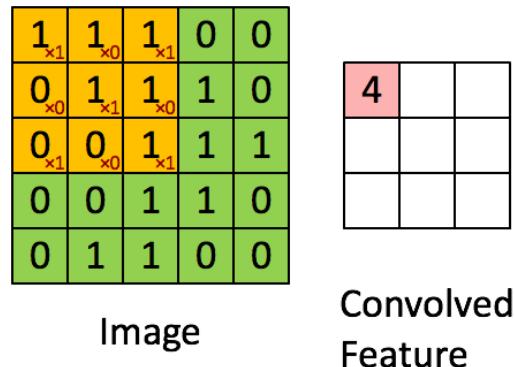
下图对应一个垂直边缘检测的例子：

$$\begin{array}{|c|c|c|c|c|c|c|} \hline
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline
 \end{array} * \begin{array}{|c|c|c|} \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 \end{array} = \begin{array}{|c|c|c|c|} \hline
 0 & 30 & 30 & 0 \\ \hline
 0 & 30 & 30 & 0 \\ \hline
 0 & 30 & 30 & 0 \\ \hline
 0 & 30 & 30 & 0 \\ \hline
 \end{array}$$

Andrew Ng

如果将最右边的矩阵当作图像，那么中间一段亮一些的区域对应最左边的图像中间的垂直边缘。

深度学习里面所称的卷积运算，和泛函分析中的卷积运算有所不同，它的求解过程只是简单将图像矩阵中，从左到右，由上到下，取与滤波器同等大小的一部分，每一部分中的值与滤波器中的值对应相乘后求和，最后的结果组成一个矩阵，其中没有经过翻转、反褶等过程。这里是一个卷积运算的动态的例子，方便理解：



图中的 * 表示卷积运算符号。在计算机中这个符号表示一般的乘法，而在不同的深度学习框架中，卷积操作的 API 定义可能不同：

- 在 Python 中，卷积用 `conv_forward()` 表示；
 - 在 Tensorflow 中，卷积用 `tf.nn.conv2d()` 表示；
 - 在 keras 中，卷积用 `Conv2D()` 表示。

更多边缘检测

如果将灰度图左右的颜色进行翻转，再与之前的滤波器进行卷积，得到的结果也有区别。实际应用中，这反映了由明变暗和由暗变明的两种渐变方式。可以对输出图片取绝对值操作，以得到同样的结果。

$$\begin{array}{|c|c|c|c|c|c|} \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 \end{array}
 *
 \begin{array}{|c|c|c|} \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 \end{array}
 =
 \begin{array}{|c|c|c|} \hline
 0 & -30 & -30 & 0 \\ \hline
 0 & -30 & -30 & 0 \\ \hline
 0 & -30 & -30 & 0 \\ \hline
 0 & -30 & -30 & 0 \\ \hline
 \end{array}$$

上面的这个滤波器 $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ 可以用来探测垂直方向的边缘，那么只要把这个滤波器翻转下，变成 $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ ，则这个新的滤波器就可以用来探测水平方向的边缘。如下图：

$$\begin{array}{|c|c|c|c|c|c|} \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 0 & 0 & 0 & 10 & 10 & 10 \\ \hline
 \end{array} * \begin{array}{|c|c|c|} \hline
 1 & 1 & 1 \\ \hline
 0 & 0 & 0 \\ \hline
 -1 & -1 & -1 \\ \hline
 \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline
 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
 30 & 10 & -10 & -30 & 0 & 0 \\ \hline
 30 & 10 & -10 & -30 & 0 & 0 \\ \hline
 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
 \end{array}$$

不同的滤波器有着不同的作用。滤波器矩阵的大小和其中的值也都不是固定不变的，可以根据需求来选择。

垂直边缘检测和水平边缘检测的滤波器如下所示：

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

其他常用的滤波器还有 Sobel 滤波器和 Scharr 滤波器。它们增加了中间行的权重，以提高结果的稳健性。

1	0	-1
2	0	-2
1	0	-1

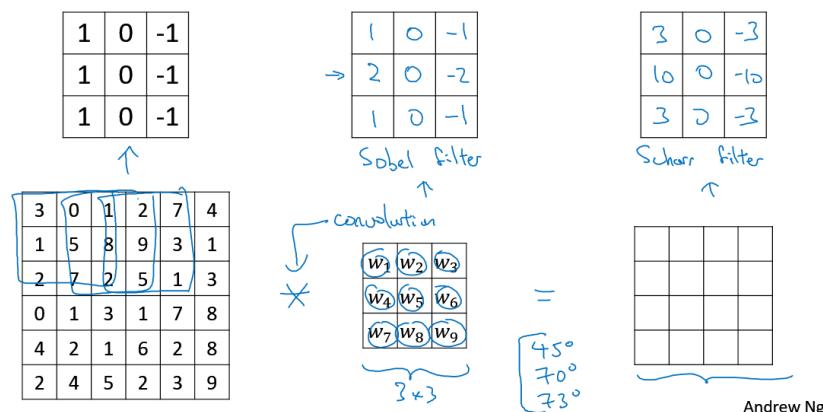
Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

滤波器中的值还可以设置为参数，通过模型训练来得到。这样，神经网络使用反向传播算法可以学习到一些低级特征，从而实现对图片所有边缘特征的检测，而不仅限于垂直边缘和水平边缘。

Learning to detect edges



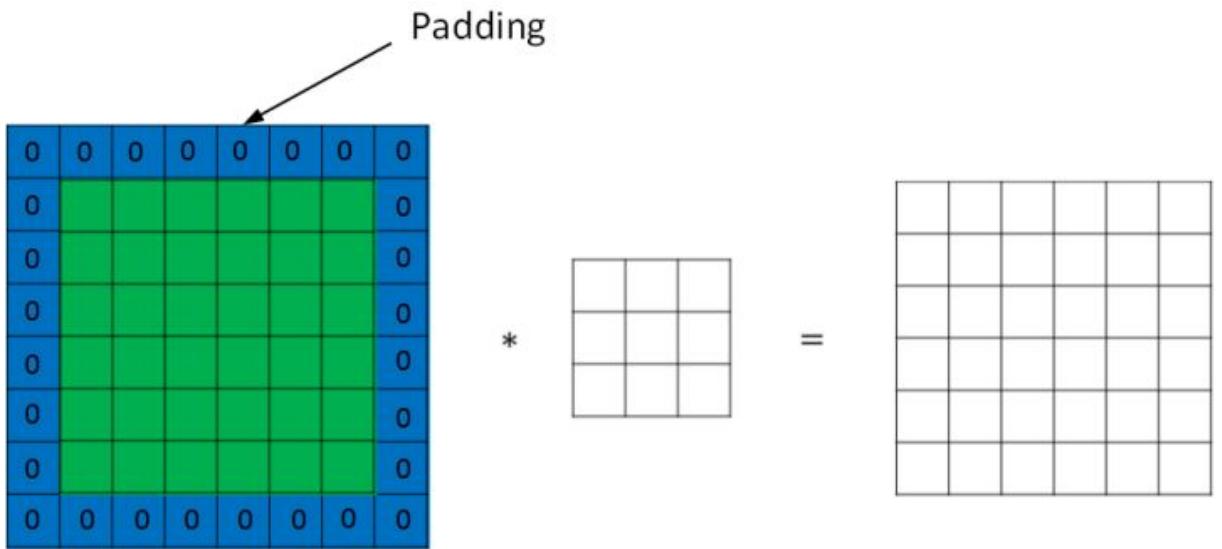
1.1.2.2 填充

假设输入图片的大小为 $n \times n$ ，而滤波器的大小为 $f \times f$ ，则卷积后的输出图片大小为 $(n - f + 1) \times (n - f + 1)$ 。

这样就有两个问题：

- 每次卷积运算后，输出图片的尺寸缩小；
- 原始图片的角落、边缘区像素点在输出中采用较少，输出图片丢失边缘位置的很多信息。

为了解决这些问题，可以在进行卷积操作前，对原始图片在边界上进行填充（Padding），以增加矩阵的大小。通常将 0 作为填充值。



设每个方向扩展像素点数量为 p , 则填充后原始图片的大小为 $(n + 2p) \times (n + 2p)$, 滤波器大小保持 $f \times f$ 不变, 则输出图片大小为 $(n + 2p - f + 1) \times (n + 2p - f + 1)$

因此, 在进行卷积运算时, 我们有两种选择:

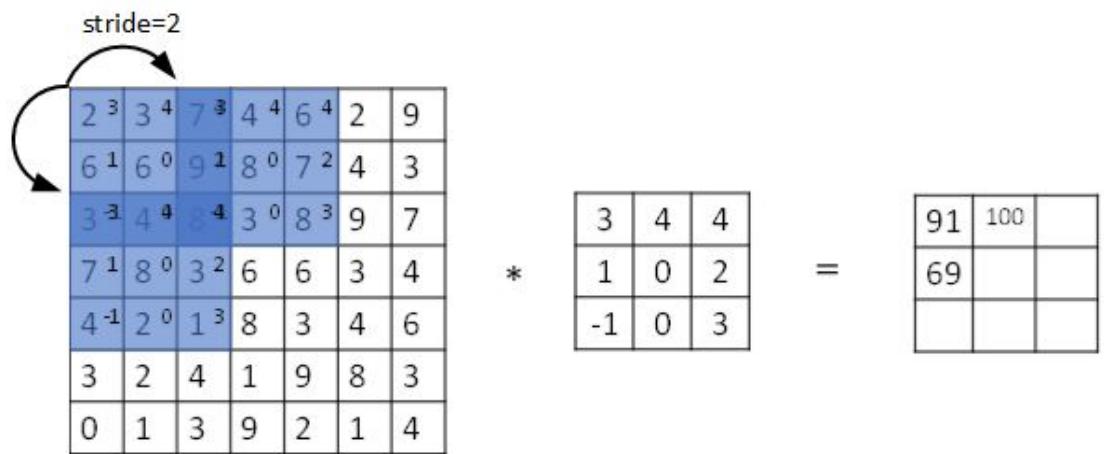
- **Valid 卷积**: 不填充, 直接卷积。结果大小为 $(n - f + 1) \times (n - f + 1)$;
- **Same 卷积**: 进行填充, 并使得卷积后结果大小与输入一致, 这样 $p = \frac{f-1}{2}$ 。

在计算机视觉领域, f 通常为奇数。原因包括 Same 卷积中 $p = \frac{f-1}{2}$ 能得到自然数结果, 并且滤波器有一个便于表示其所在位置的中心点。

1.1.2.3 步长

卷积过程中, 有时需要通过填充来避免信息损失, 有时也需要通过设置**步长 (Stride)** 来压缩一部分信息。

步长表示滤波器在原始图片的水平方向和垂直方向上每次移动的距离。之前, 步长被默认为 1。而如果我们设置步长为 2, 则卷积过程如下图所示:



设步长为 s , 填充长度为 p , 输入图片大小为 $n \times n$, 滤波器大小为 $f \times f$, 则卷积后图片的尺寸为:

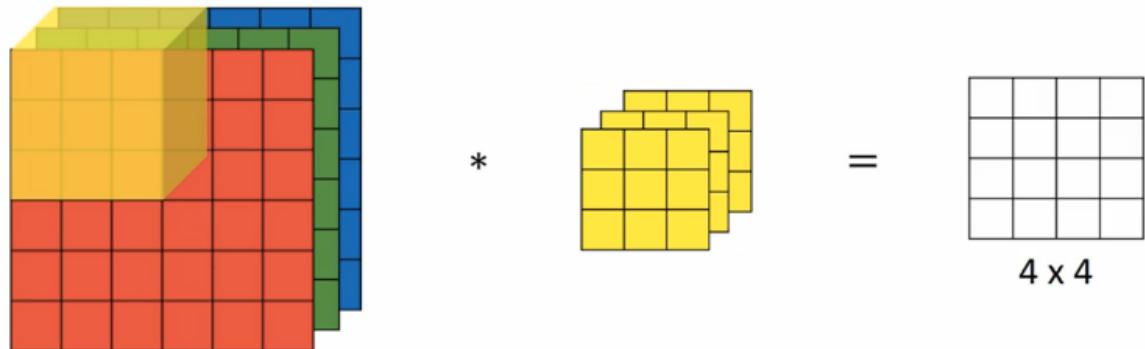
$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \quad (1)$$

注意公式中有一个向下取整的符号, 用于处理商不为整数的情况。向下取整反映着当取原始矩阵的图示蓝框完全包括在图像内部时, 才对它进行运算。

目前为止我们学习的“卷积”实际上被称为**互相关 (cross-correlation)**，而非数学意义上的卷积。真正的卷积操作在做元素乘积求和之前，要将滤波器沿水平和垂直轴翻转（相当于旋转 180 度）。因为这种翻转对一般为水平或垂直对称的滤波器影响不大，按照机器学习的惯例，我们通常不进行翻转操作，在简化代码的同时使神经网络能够正常工作。

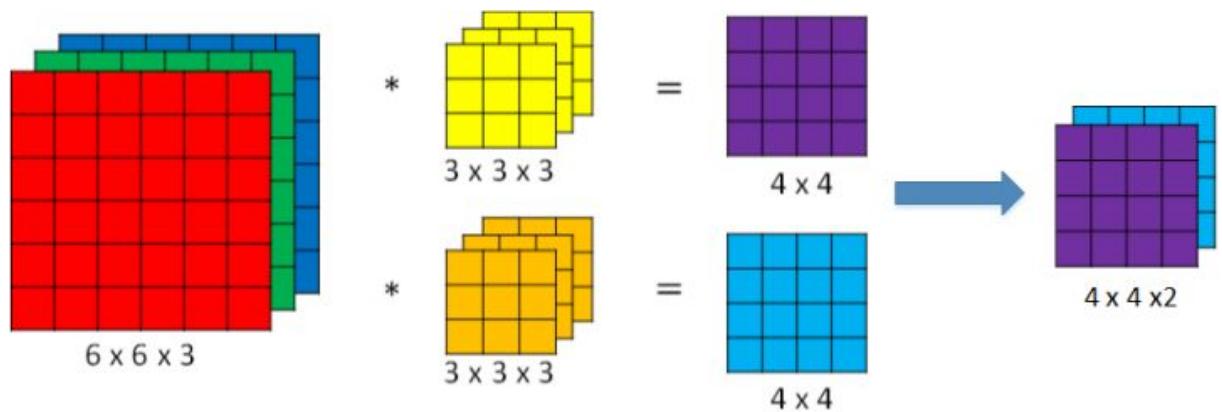
1.1.3 高维卷积

如果我们想要对三通道的 RGB 图片进行卷积运算，那么其对应的滤波器组也同样是三通道的。过程是将每个单通道 (R, G, B) 与对应的滤波器进行卷积运算求和，然后再将三个通道的和相加，将 27 个乘积的和作为输出图片的一个像素值。

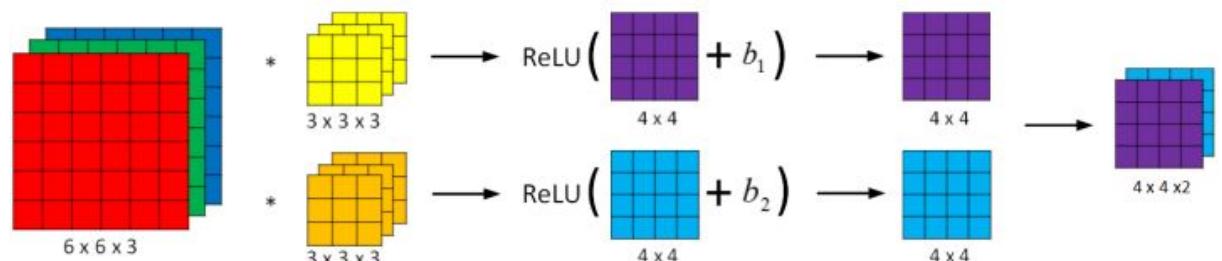


不同通道的滤波器可以不相同。例如只检测 R 通道的垂直边缘，G 通道和 B 通道不进行边缘检测，则 G 通道和 B 通道的滤波器全部置零。当输入有特定的高、宽和通道数时，滤波器可以有不同的高和宽，但通道数必须和输入一致。

如果想同时检测垂直和水平边缘，或者更多的边缘检测，可以增加更多的滤波器组。例如设置第一个滤波器组实现垂直边缘检测，第二个滤波器组实现水平边缘检测。设输入图片的尺寸为 $n \times n \times n_c$ (n_c 为通道数)，滤波器尺寸为 $f \times f \times n_c$ ，则卷积后的输出图片尺寸为 $(n - f + 1) \times (n - f + 1) \times n'_c$, n'_c 为滤波器组的个数。



1.1.4 单层卷积网络



与之前的卷积过程相比较，卷积神经网络的单层结构多了激活函数和偏移量；而与标准神经网络：

$$\begin{aligned} Z^{[l]} &= W^{[l]} A^{[l-1]} + b \\ A^{[l]} &= g^{[l]}(Z^{[l]}) \end{aligned} \quad (2)$$

相比，滤波器的数值对应着权重 $W^{[l]}$ ，卷积运算对应着 $W^{[l]}$ 与 $A^{[l-1]}$ 的乘积运算，所选的激活函数变为 ReLU。

对于一个 $3 \times 3 \times 3$ 的滤波器，包括偏移量 b 在内共有 28 个参数。不论输入的图片有多大，用这一个滤波器来提取特征时，参数始终都是 28 个，固定不变。即选定滤波器组后，参数的数目与输入图片的尺寸无关。因此，卷积神经网络的参数相较于标准神经网络来说要少得多。这是 CNN 的优点之一。

1.1.5 符号约定

设 l 层为卷积层：

- $f^{[l]}$: 滤波器的高（或宽）
- $p^{[l]}$: 填充长度
- $s^{[l]}$: 步长
- $n_c^{[l]}$: 滤波器组的数量
- 输入维度： $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ 。其中 $n_H^{[l-1]}$ 表示输入图片的高， $n_W^{[l-1]}$ 表示输入图片的宽。之前的示例中输入图片的高和宽都相同，但是实际中也可能不同，因此加上下标予以区分。
- 输出维度： $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ 。其中

$$\begin{aligned} n_H^{[l]} &= \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \\ n_W^{[l]} &= \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \end{aligned} \quad (3)$$

- 每个滤波器组的维度： $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$ 。其中 $n_c^{[l-1]}$ 为输入图片通道数（也称深度）。
- 权重维度： $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
- 偏置维度： $1 \times 1 \times 1 \times n_c^{[l]}$

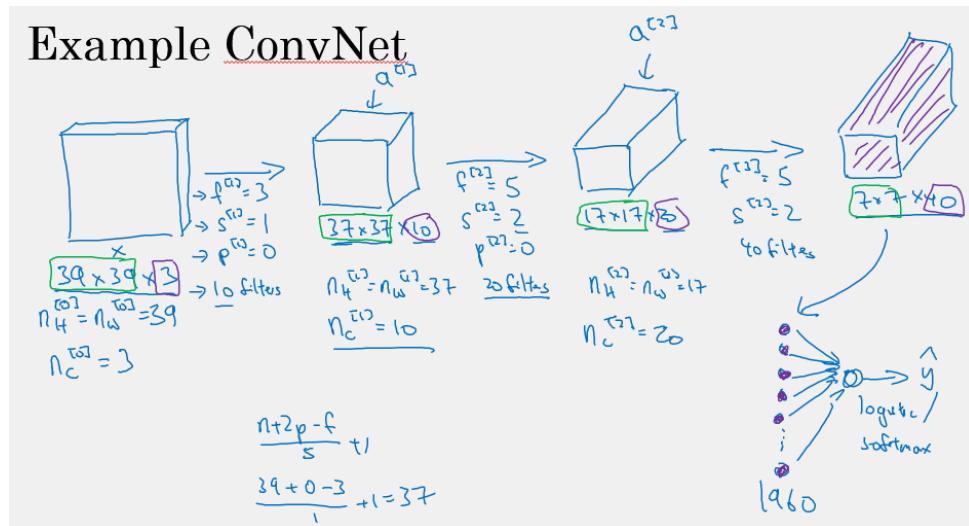
由于深度学习的相关文献并未对卷积标注法达成一致，因此不同的资料关于高度、宽度和通道数的顺序可能不同。有些作者会将通道数放在首位，需要根据标注自行分辨。

1.1.6 网络的结构

通常一个卷积神经网络是由**卷积层 (Convolution)**、**池化层 (Pooling)**、**全连接层 (Fully Connected)** 组成。

在输入层输入原始数据，卷积层中进行的是前面所述的卷积过程，用它来进行提取特征。全连接层就是将识别到的所有特征全部连接起来，并输出到分类器（如 Softmax）。

一个简单的 CNN 如下所示：



其中， $a^{[3]}$ 的维度为 $7 \times 7 \times 40$ ，将 1960 个特征平滑展开成 1960 个单元的一列，然后连接最后一级的输出层。输出层可以是一个神经元，即二元分类（logistic）；也可以是多个神经元，即多元分类（softmax）。最后得到预测输出 \hat{y} 。

随着神经网络计算深度不断加深，图片的高度和宽度 $n_H^{[l]}$ 、 $n_W^{[l]}$ 一般逐渐减小，而 $n_c^{[l]}$ 在增加

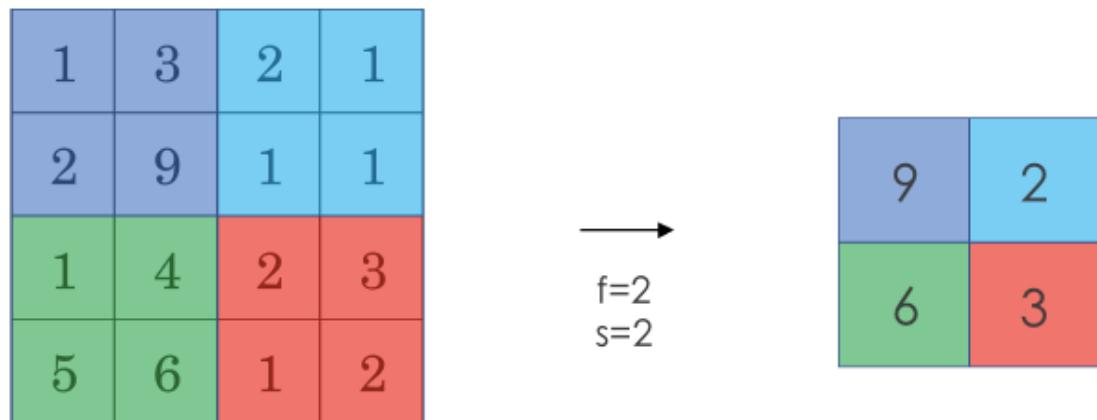
为什么使用全连接层

全连接层（fully connected layers, FC）在整个卷积神经网络中起到“分类器”的作用。如果说卷积层、池化层和激活函数层等操作是将原始数据映射到隐层特征空间的话，全连接层则起到将学到的“分布式特征表示”映射到样本标记空间的作用。在实际使用中，全连接层可由卷积操作实现：对前层是全连接的全连接层可以转化为卷积核为 1×1 的卷积；而前层是卷积层的全连接层可以转化为卷积核为 $h \times w$ 的全局卷积， h 和 w 分别为前层卷积结果的高和宽。

1.1.6.1 池化层

池化层的作用是缩减模型的大小，提高计算速度，同时减小噪声提高所提取特征的稳健性。

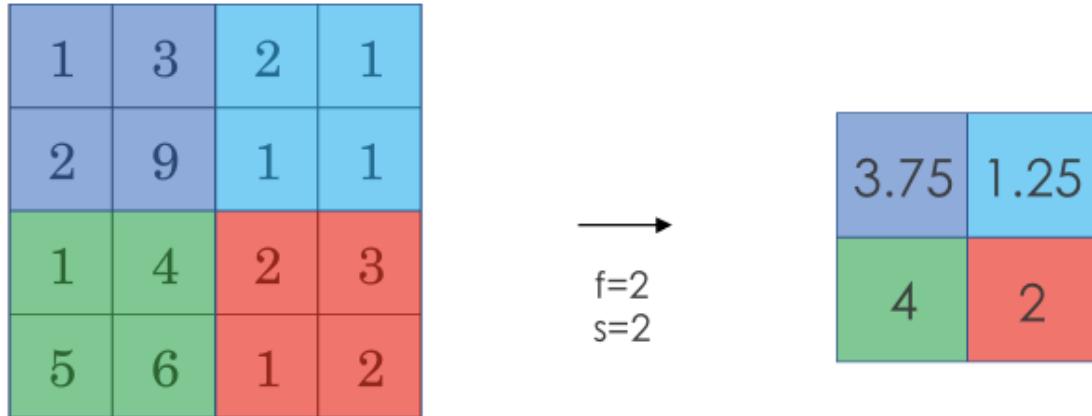
采用较多的一种池化过程叫**最大池化（Max Pooling）**，将输入拆分成不同的区域，输出的每个元素都是对应区域中元素的最大值，如下图所示：



池化过程类似于卷积过程，上图所示的池化过程中相当于使用了一个大小 $f = 2$ 的滤波器，且池化步长 $s = 2$ 。卷积过程中的几个计算大小的公式也都适用于池化过程。如果有多个通道，那么就对每个通道分别执行计算过程。

对最大池化的一种直观解释是，元素值较大可能意味着池化过程之前的卷积过程提取到了某些特定的特征，池化过程中的最大化操作使得只要在一个区域内提取到某个特征，它都会保留在最大池化的输出中。但是，没有足够的证据证明这种直观解释的正确性，而最大池化被使用的主要原因是它在很多实验中的效果都很好。

另一种池化过程是**平均池化 (Average Pooling)**，就是从取某个区域的最大值改为求这个区域的平均值：



池化过程的特点之一是，它有一组超参数，但是并没有参数需要学习。池化过程的超参数包括滤波器的大小 f 、步长 s ，以及选用最大池化还是平均池化。而填充 p 则很少用到。

池化过程的输入维度为：

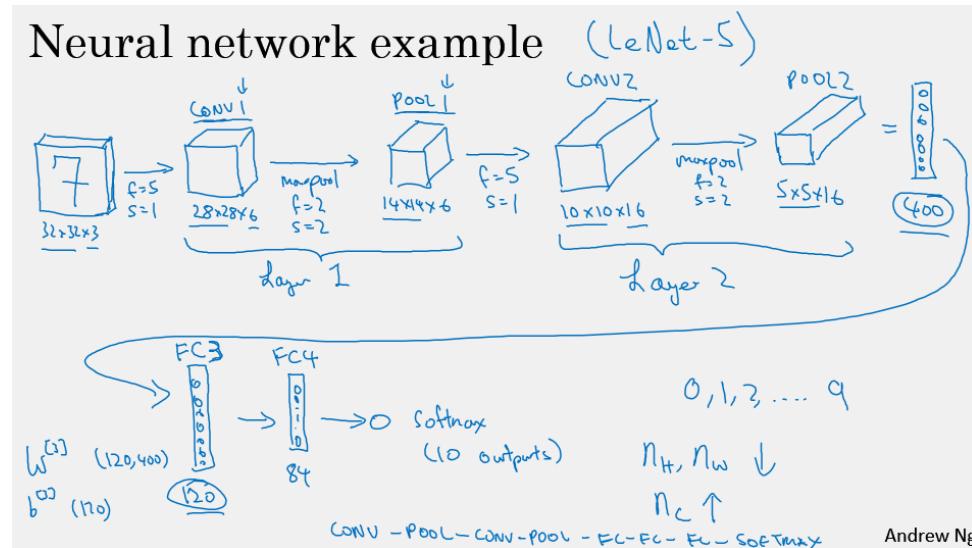
$$n_H \times n_W \times n_c \quad (4)$$

输出维度为：

$$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_c \quad (5)$$

1.1.6.2 示例结构

一种典型卷积网络结构是 LeNet-5，它是由用 LeCun 在上世纪九十年代提出的用来识别数字的卷积网络，下图的卷积网络结构与它类似：



其中，一个卷积层和一个池化层组成整个卷积神经网络中的一层，图中所示整个过程为：

$Input \rightarrow Conv1 \rightarrow Pool1 \rightarrow Conv2 \rightarrow Pool2 \rightarrow FC \rightarrow FC3 \rightarrow FC4 \rightarrow Softmax$

下面各层中的数据：

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

可以看出，激活的大小随着向卷积网络深层的递进而减小，参数的数量在不断增加后在几个全连接过程后将逐渐减少。

1.1.7 卷积使用解释

对于大量的输入数据，卷积过程有效地减少了参数数量，而这主要归功于以下两点：

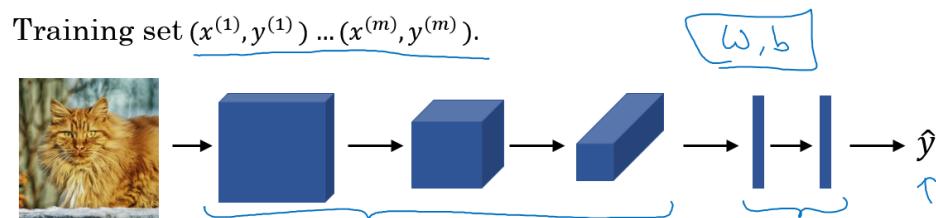
- **参数共享(Parameter Sharing)**: 特征检测如果适用于图片的某个区域，那么它也可能适用于图片的其他区域。即在卷积过程中，不管输入有多大，一个特征探测器也就前面所说的滤波器就能对整个输入的特征进行探测。
- **局部感知(Local Perception)**: 在每一层中，输入和输出之间的连接是稀疏的，每个输出值只取决于输入的一小部分值。

池化过程则在卷积后很好地聚合了特征，通过降维而减少了运算量。

由于 CNN 参数数量较小，所需的训练样本就相对较少，因此在一定程度上不容易发生过拟合现象。并且 CNN 比较擅长捕捉区域位置偏移。即进行物体检测时，不太受物体在图片中位置的影响，增加检测的准确性和系统的健壮性。

整合训练这些层

Putting it together



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce J

1.2 实例探讨

讲到的经典 CNN 模型包括：

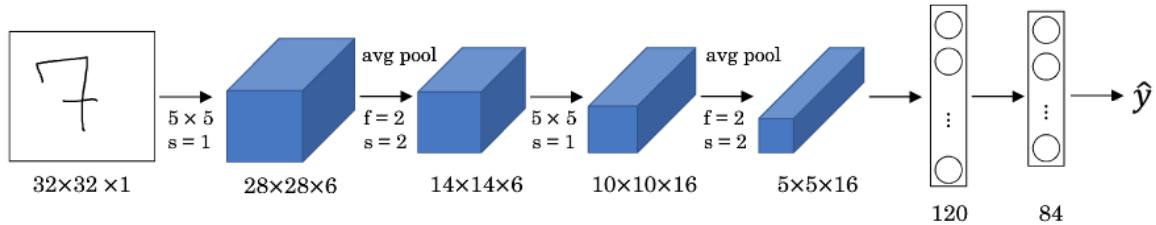
- LeNet-5
- AlexNet
- VGG

此外还有 ResNet (Residual Network, 残差网络)，以及 Inception Neural Network。

1.2.1 经典的卷积网络

1.2.1.1 LeNet-5

LeNet-5 是 LeCun 等人 1998 年在论文 [Gradient-based learning applied to document recognition](#) (<https://ieeexplore.ieee.org/document/726791>) 中提出的卷积网络。其结构如下图：

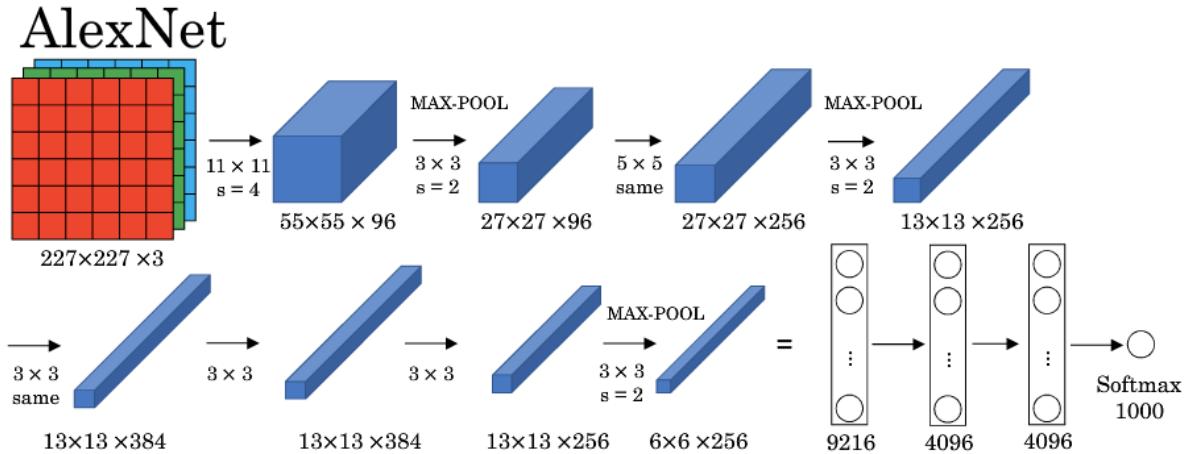


特点：

- LeNet-5 针对灰度图像而训练，因此输入图片的通道数为 1。
- 该模型总共包含了约 6 万个参数，远少于标准神经网络所需。
- 典型的 LeNet-5 结构包含卷积层 (CONV layer)，池化层 (POOL layer) 和全连接层 (FC layer)，排列顺序一般为 CONV layer->POOL layer->CONV layer->POOL layer->FC layer->FC layer->OUTPUT layer。一个或多个卷积层后面跟着一个池化层的模式至今仍十分常用。
- 当 LeNet-5 模型被提出时，其池化层使用的是平均池化，而且各层激活函数一般选用 Sigmoid 和 tanh。现在，我们可以根据需要，做出改进，使用最大池化并选用 ReLU 作为激活函数。
- 随着深度的增加， n_H 、 n_W 的值在不断减小， n_c 却在不断增加。其中的 Conv-Pool-Conv-Pool-FC-FC-Output 是现在用到的卷积网络中的一种很常见的结构。

1.2.1.2 AlexNet

AlexNet 是 Krizhevsky 等人 2012 年在论文 [ImageNet classification with deep convolutional neural networks](#) (<http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>) 中提出的卷积网络。其结构如下图：

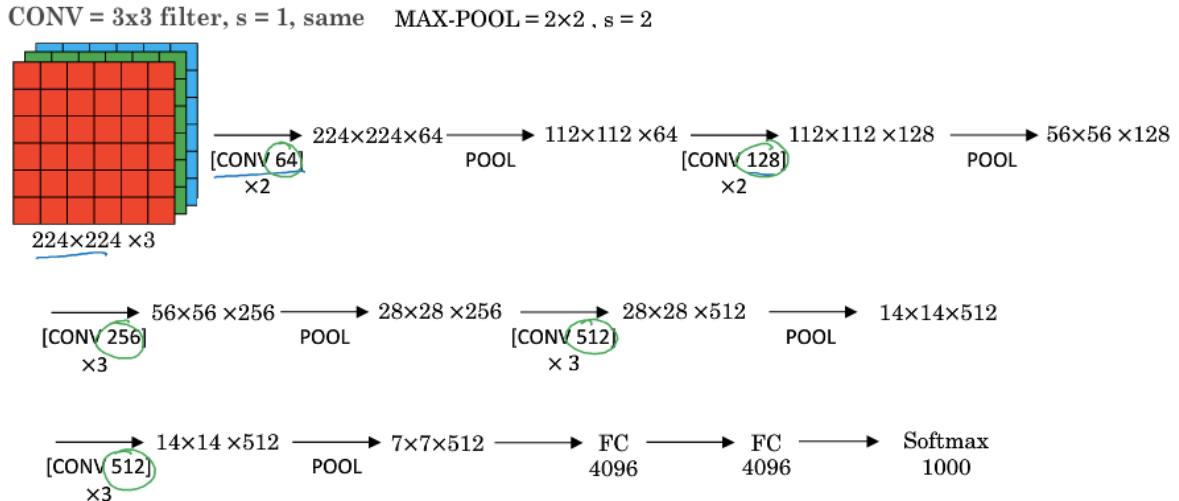


特点：

- AlexNet 模型与 LeNet-5 模型类似，但是更复杂，包含约 6000 万个参数。另外，AlexNet 模型使用了 ReLU 函数。
- 当用于训练图像和数据集时，AlexNet 能够处理非常相似的基本构造模块，这些模块往往包含大量的隐藏单元或数据。

1.2.1.3 VGG

VGG-16 是 Simonyan 和 Zisserman 2015 年在论文 [Very deep convolutional networks for large-scale image recognition](https://arxiv.org/pdf/1409.1556.pdf) (<https://arxiv.org/pdf/1409.1556.pdf>) 中提出的卷积网络。其结构如下图：



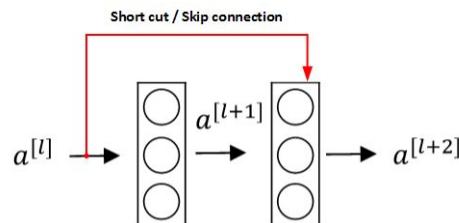
特点：

- VGG 又称 VGG-16 网络，“16”指网络中包含 16 个卷积层和全连接层。
- 超参数较少，只需要专注于构建卷积层。
- 结构不复杂且规整，在每一组卷积层进行滤波器翻倍操作。
- VGG 需要训练的特征数量巨大，包含多达约 1.38 亿个参数。

1.2.2 残差网络

当一个神经网络某个深度时，将会出现梯度消失 (Vanishing Gradient) 和梯度爆炸 (Exploding Gradient) 等问题。而 ResNets 能很好得解决这些问题。

ResNets 全称为 **残差网络 (Residual Networks)**，它是微软研究院 2015 年在论文 [Deep Residual Learning for Image Recognition](https://arxiv.org/pdf/1512.03385.pdf) (<https://arxiv.org/pdf/1512.03385.pdf>) 中提出的卷积网络。



上图的结构被称为**残差块 (Residual block)**。

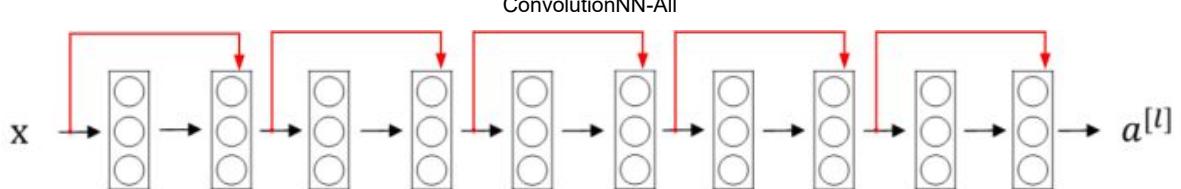
一般的前向传播的过程，也称为“主要路径 (main path)”为 $a^{[l]} \text{-linear-ReLU-linear-ReLU-} a^{[l+2]}$ ，计算过程如下：

$$\begin{aligned} z^{[l+1]} &= W^{[l+1]} a^{[l]} + b^{[l+1]} \\ a^{[l+1]} &= g(z^{[l+1]}) \\ z^{[l+2]} &= W^{[l+2]} a^{[l+1]} + b^{[l+2]} \\ a^{[l+2]} &= g(z^{[l+2]}) \end{aligned} \tag{6}$$

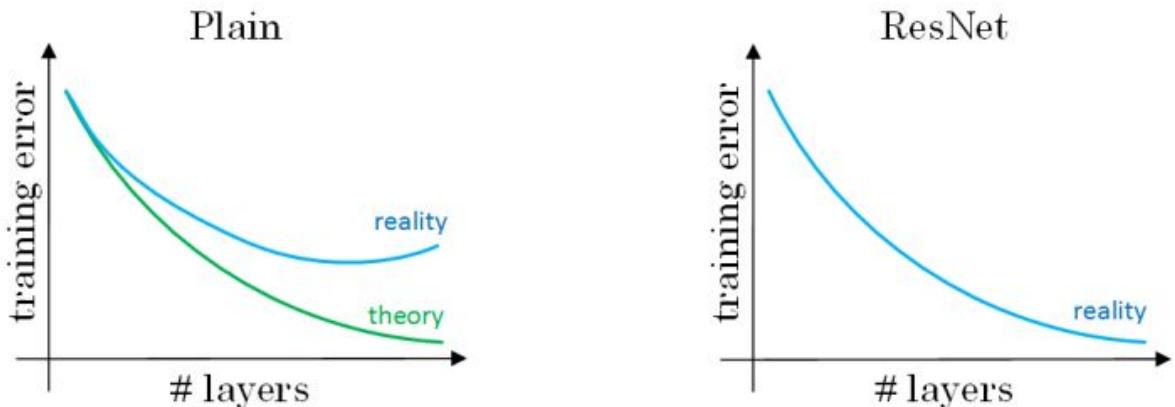
在残差网络中，通过捷径 (Short cut, 或者称跳远连接，Skip connections) 可以将 $a^{[l]}$ 添加到第二个 ReLU 过程中，直接建立 $a^{[l]}$ 与 $a^{[l+2]}$ 之间的隔层联系。也就是最后一个方程变为：

$$a^{[l+2]} = g(z^{[l+2]} + W_s a^{[l]}) \tag{7}$$

构建一个残差网络就是将许多残差块堆积在一起，形成一个深度网络。



在理论上，随着网络深度的增加，性能应该越来越好。但实际上，对于一个普通网络，随着神经网络层数增加，训练错误会先减少，然后开始增多。但残差网络的训练效果显示，即使网络再深，其在训练集上的表现也会越来越好。

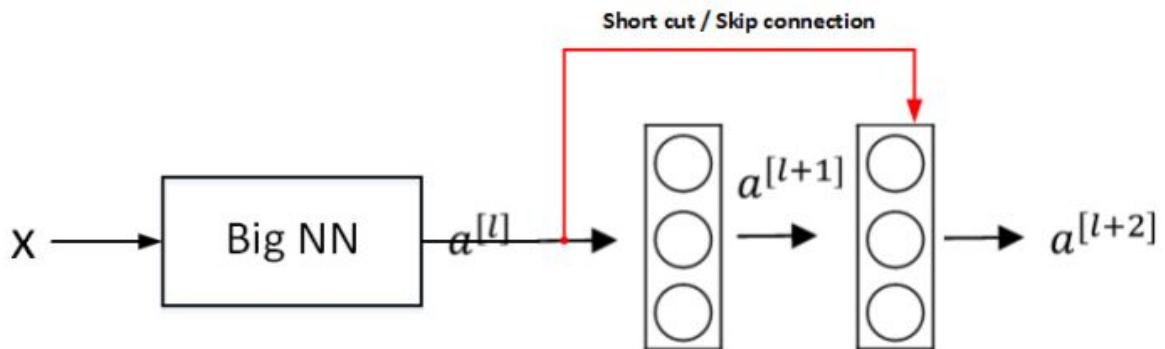


残差网络有助于解决梯度消失和梯度爆炸问题，使得在训练更深的网络的同时，又能保证良好的性能。

残差网络解释

对于一个神经网络中存在的一些**恒等函数 (Identity Function)**，残差网络在不影响这个神经网络的整体性能下，使得对这些恒等函数的学习更加容易，而且很多时候还能提高整体的学习效率。

假设有一个大型神经网络，其输入为 X ，输出为 $a^{[l]}$ 。给这个神经网络额外增加两层，输出为 $a^{[l+2]}$ 。将这两层看作一个具有跳远连接的残差块。为了方便说明，假设整个网络中都选用 ReLU 作为激活函数，因此输出的所有激活值都大于等于 0。



则有：

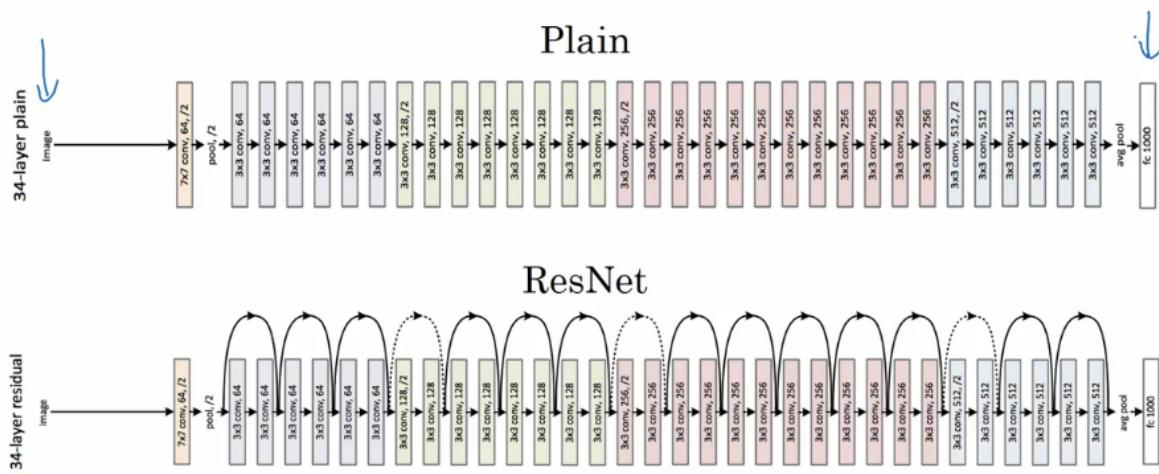
$$\begin{aligned} a^{[l+2]} &= g(z^{[l+2]} + a^{[l]}) \\ &= g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]}) \end{aligned} \quad (8)$$

当发生梯度消失时， $W^{[l+2]} \approx 0$, $b^{[l+2]} \approx 0$ ，则有：

$$a^{[l+2]} = g(a^{[l]}) = \text{ReLU}(a^{[l]}) = a^{[l]} \quad (9)$$

因此，这两层额外的残差块不会降低网络性能。而如果没有发生梯度消失时，训练得到的非线性关系会使得表现效果进一步提高。

注意，如果 $a^{[l]}$ 与 $a^{[l+2]}$ 的维度不同，需要引入矩阵 W_s 与 $a^{[l]}$ 相乘，使得二者的维度相匹配。参数矩阵 W_s 既可以通过模型训练得到，也可以作为固定值，仅使 $a^{[l]}$ 截断或者补零。



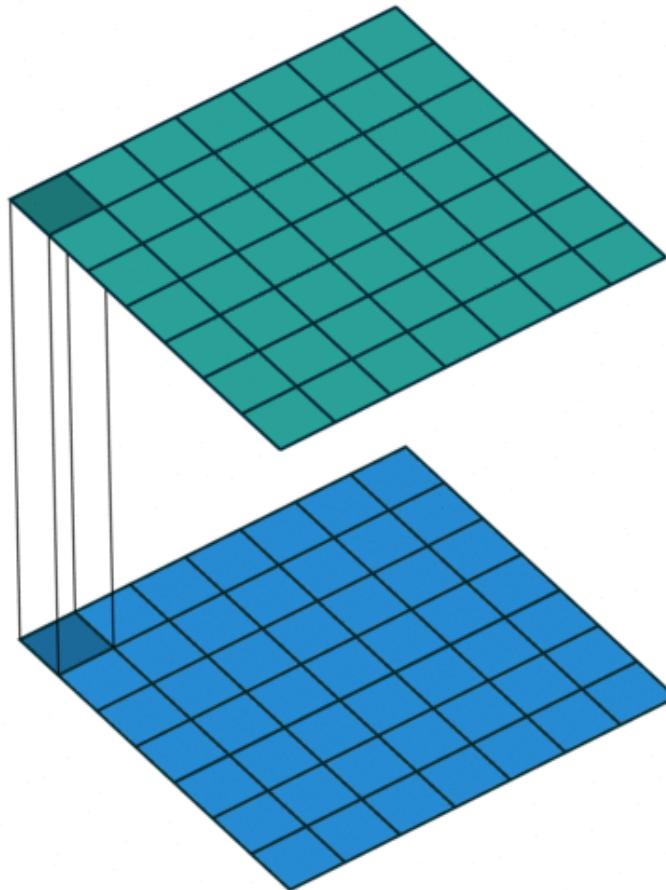
上图是论文提供的 CNN 中 ResNet 的一个典型结构。卷积层通常使用 Same 卷积以保持维度相同，而不同类型层之间的连接（例如卷积层和池化层），如果维度不同，则需要引入矩阵 W_s 。

1.2.3 1×1 卷积

1x1 卷积 (1x1 convolution, 或称为 Network in Network) 指滤波器的尺寸为 1。当通道数为 1 时，1x1 卷积意味着卷积操作等同于乘积操作。

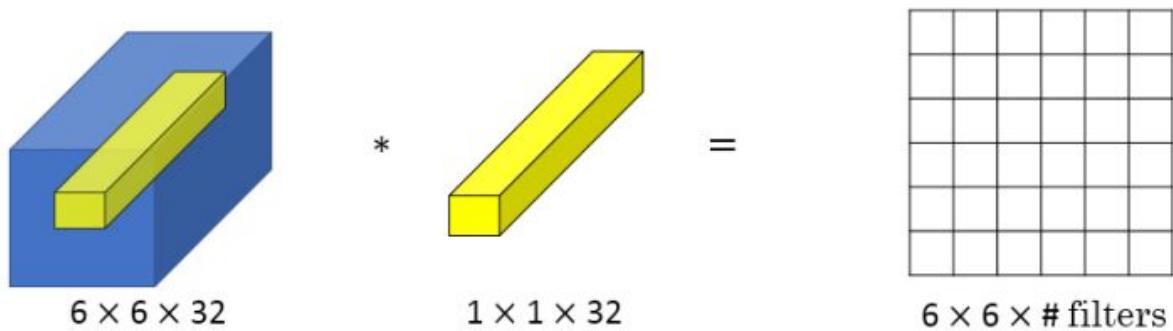
而当通道数更多时， 1×1 卷积的作用实际上类似全连接层的神经网络结构，从而降低（或升高，取决于滤波器组数）数据的维度。

2013年新加坡国立大学的林敏等人在论文 [Network In NetWork \(<https://arxiv.org/pdf/1312.4400.pdf>\)](https://arxiv.org/pdf/1312.4400.pdf) 中提出了 1×1 卷积核及 NIN 网络。



使用 1×1 卷积核进行卷积的过程如上图，它就是在卷积过程中采用大小为 1×1 的滤波器。如果神经网络的当前一层和下一层都只有一个信道，也就是 $n_C = 1$ ，那么采用 1×1 卷积核起不到什么作用的。但是当它们分别为有 m 和 n 个信道时，采用 1×1 卷积核就可以起到跨信道聚合的作用，从而降低（或升高）数据的维度，可以达到减少参数的目的。换句话说， 1×1 的卷积核操作实现的其实就是一个特征数据中的多个 Feature Map 的线性组合，所以这个方法就可以用来改变特征数据的信道数。

池化能压缩数据的高度 (n_H) 及宽度 (n_W)，而 1×1 卷积能压缩数据的通道数 (n_C)。在如下图所示的例子中，用 32 个大小为 $1 \times 1 \times 192$ 的滤波器进行卷积，就能使原先数据包含的 192 个通道压缩为 32 个。

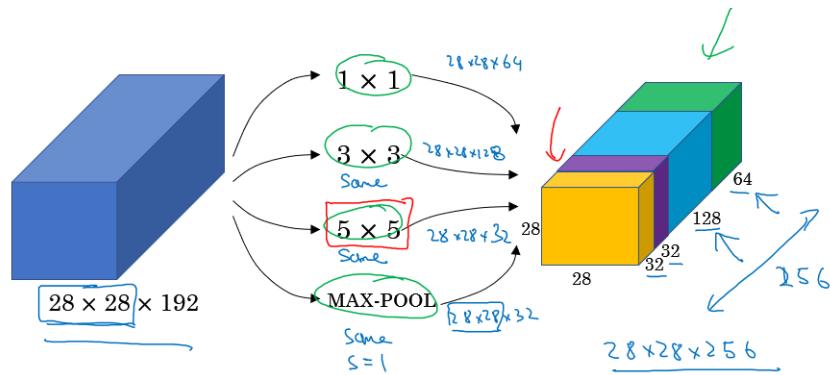


1.2.4 Inception 网络

最早的 Inception 结构的 V1 版本是由 Google 的 Szegedy 2014 年在论文 [Going deeper with convolutions](#) (<https://arxiv.org/pdf/1409.4842.pdf>) 中提出的，它是 ILSVRC 2014 中取得最好成绩的 GoogLeNet 中采用的核心结构。通过不断改进，现在已经衍生有了 V4 版本。

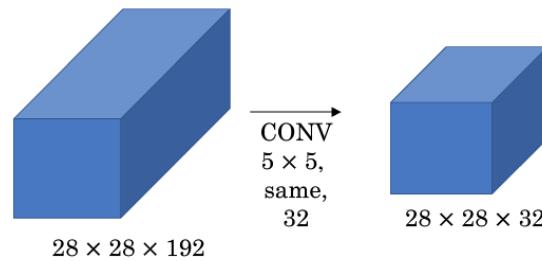
在之前的卷积网络中，我们只能选择单一尺寸和类型的滤波器。而 Inception 网络的作用即是代替人工来确定卷积层中的滤波器尺寸与类型，或者确定是否需要创建卷积层或池化层。

早期的 V1 版本的结构借鉴了 NIN 的设计思路，对网络中的传统卷积层进行了修改，其结构大致如下面的例子中所示：

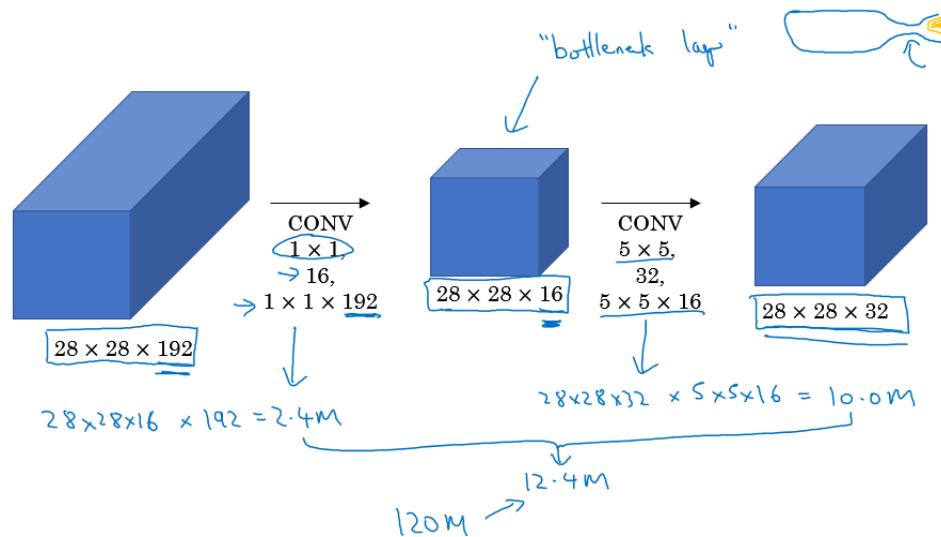


计算成本问题

通常在设计一个卷积网络的结构时，需要考虑卷积过程、池化过程的滤波器的大小，甚至是要不要使用 1×1 卷积核。在 Inception 结构中，考虑到多个不同大小的卷积核（滤波器）能够增强网络的适应力，于是分别使用三个大小分别为 1×1 、 3×3 、 5×5 的卷积核进行 same 卷积，同时再加入了一个 same 最大池化过程。最后将它们各自得到的结果放在一起，得到了图中一个大小为 $28\times 28\times 256$ 的结果。然而，这种结构中包含的参数数量庞大，对计算资源有着极大的依赖，上面的例子中光是与大小为 5×5 的滤波器进行卷积的过程就会产生 1 亿多个参数！



为了解决计算量大的问题，可以引入 1×1 卷积降维来减少其计算量。



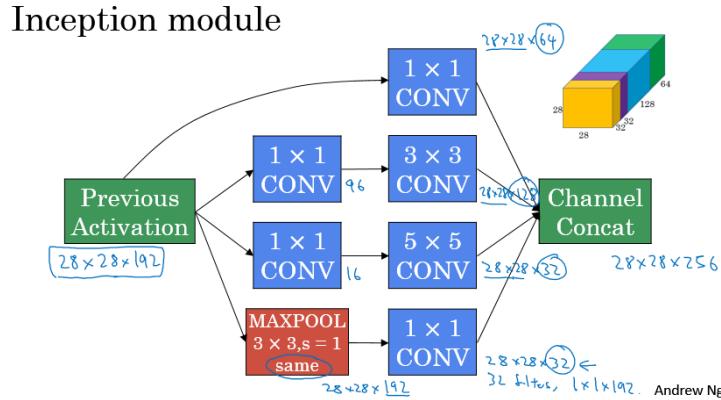
对于同一个例子，我们使用 1×1 卷积把输入数据从 192 个通道减少到 16 个通道，然后对这个较小层运行 5×5 卷积，得到最终输出。这个 1×1 的卷积层通常被称作**瓶颈层 (Bottleneck layer)**。

改进后的计算量为 $28\times 28\times 192\times 16 + 28\times 28\times 32\times 5\times 5\times 15 = 1.24$ 千万，减少了约 90%。

只要合理构建瓶颈层，就可以既显著缩小计算规模，又不会降低网络性能。

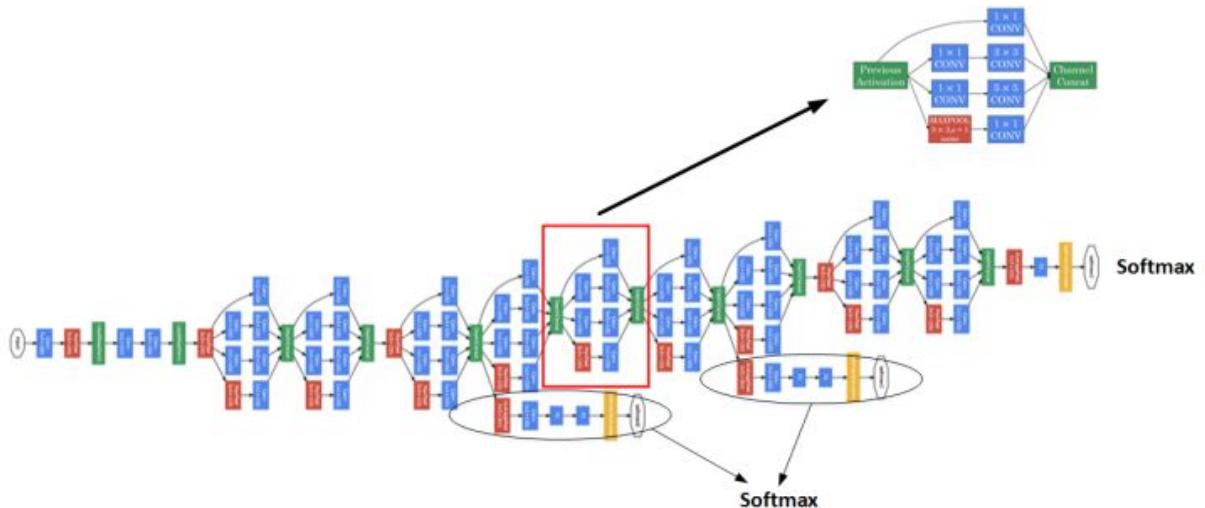
完整的 Inception 网络

在论文中提出的整个Inception模型结构如下：



上图是引入 1×1 卷积后的 Inception 模块。值得注意的是，为了将所有的输出组合起来，红色的池化层使用 Same 类型的填充 (padding) 来池化使得输出的宽高不变，通道数也不变。

多个 Inception 模块组成一个完整的 Inception 网络（被称为 GoogLeNet，以向 LeNet 致敬），如下图所示：



注意黑色椭圆圈出的隐藏层，这些分支都是 Softmax 的输出层，可以用来参与特征的计算及结果预测，起到调整并防止发生过拟合的效果。

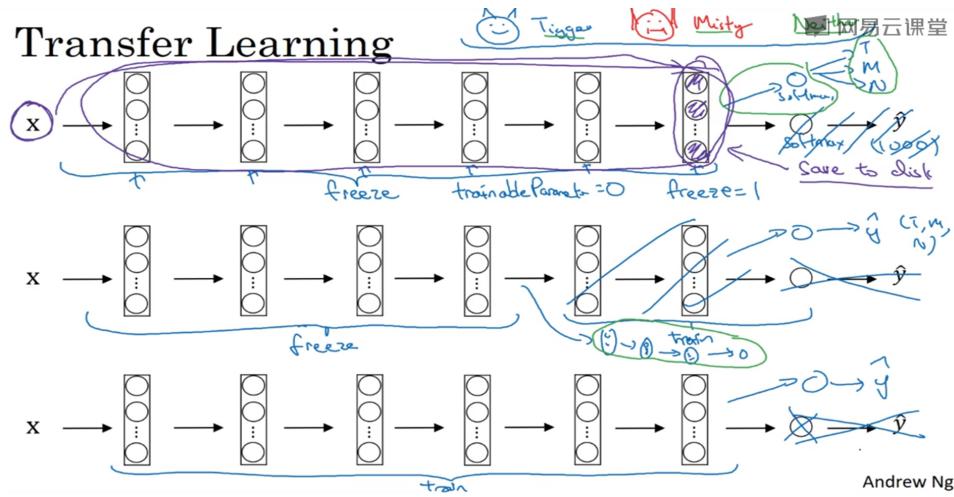
经过研究者们的不断发展，Inception 模型的 V2、V3、V4 以及引入残差网络的版本被提出，这些变体都基于 Inception V1 版本的基础思想上。顺便一提，Inception 模型的名字来自电影《盗梦空间》。

1.2.5 使用开源的实现方案

很多神经网络复杂细致，并充斥着参数调节的细节问题，因而很难仅通过阅读论文来重现他人的成果。想要搭建一个同样的神经网络，查看开源的实现方案会快很多，可以从参考Github等网站上其他人建立过的相关模型，必要时可以直接拿来根据拥有的数据量大小进行前面介绍过的迁移学习，从而减轻一些工作负担。

1.2.6 迁移学习

在“搭建机器学习项目”课程中，迁移学习已经被提到过。计算机视觉是一个经常用到迁移学习的领域。在搭建计算机视觉的应用时，相比于从头训练权重，下载别人已经训练好的网络结构的权重，用其做预训练，然后转换到自己感兴趣的任務上，有助于加速开发。



对于已训练好的卷积神经网络，可以将所有层都看作是冻结的，只需要训练与你的 Softmax 层有关的参数即可。大多数深度学习框架都允许用户指定是否训练特定层的权重。

而冻结的层由于不需要改变和训练，可以看作一个固定函数。可以将这个固定函数存入硬盘，以便后续使用，而不必每次再使用训练集进行训练了。

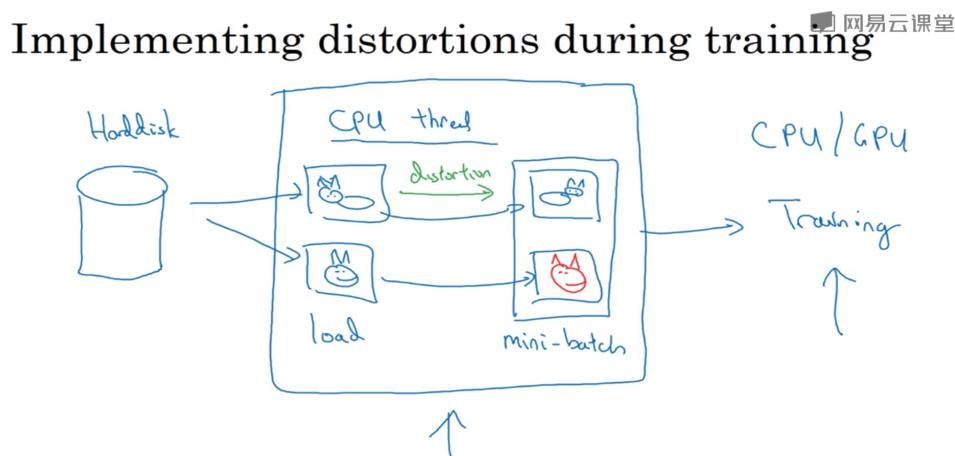
上述的做法适用于你只有一个较小的数据集。如果你有一个更大的数据集，应该冻结更少的层，然后训练后面的层。越多的数据意味着冻结越少的层，训练更多的层。如果有一个极大的数据集，你可以将开源的网络和它的权重整个当作初始化（代替随机初始化），然后训练整个网络。

1.2.7 数据扩增

计算机视觉领域的应用都需要大量的数据。当数据不够时，**数据扩增 (Data Augmentation)** 就有帮助。常用的数据扩增包括镜像翻转、随机裁剪、色彩转换。

其中，色彩转换是对图片的 RGB 通道数值进行随意增加或者减少，改变图片色调。另外，PCA 颜色增强指更有针对性地对图片的 RGB 通道进行主成分分析（Principles Components Analysis, PCA），对主要的通道颜色进行增加或减少，可以采用高斯扰动做法来增加有效的样本数量。具体的 PCA 颜色增强做法可以查阅 AlexNet 的相关论文或者开源代码。

在构建大型神经网络的时候，数据扩增和模型训练可以由两个或多个不同的线程并行来实现。



1.2.8 计算机视觉现状

通常，学习算法有两种知识来源：

- 被标记的数据
- 手工工程

手工工程 (Hand-engineering, 又称 hacks) 指精心设计的特性、网络体系结构或是系统的其他组件。手工工程是一项非常重要也比较困难的工作。在数据量不多的情况下，手工工程是获得良好表现的最佳方式。正因为数据量不能满足需要，历史上计算机视觉领域更多地依赖于手工工程。近几年数据量急剧增加，因此手工工程量大幅减少。

另外，在模型研究或者竞赛方面，有一些方法能够有助于提升神经网络模型的性能：

- 集成 (Ensembling)：独立地训练几个神经网络，并平均输出它们的输出
- Multi-crop at test time：将数据扩增应用到测试集，对结果进行平均

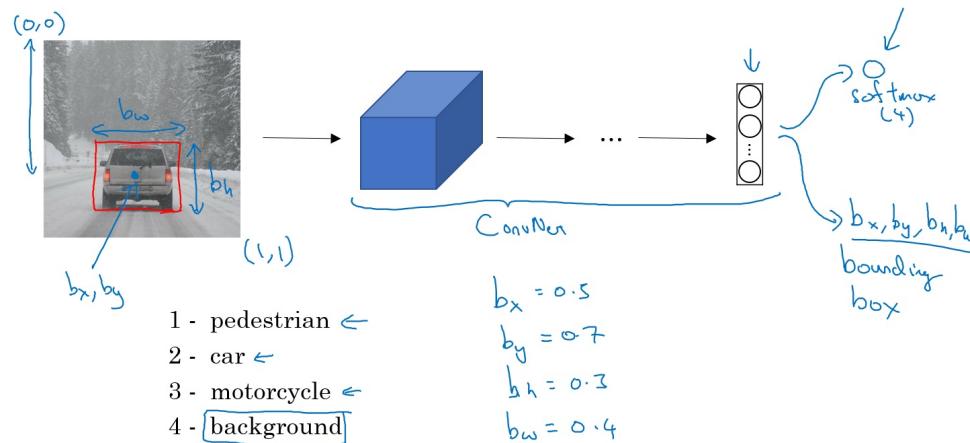
但是由于这些方法计算和内存成本较大，一般不适用于构建实际的生产项目。

1.3 目标检测

目标检测是计算机视觉领域中一个新兴的应用方向，其任务是对输入图像进行分类的同时，检测图像中是否包含某些目标，并对他们准确定位并标识。

1.3.1 目标定位

图像分类问题一般都采用 Softmax 回归来解决，最后输出的结果是一个多维列向量，且向量的维数与假定的分类类别数一致。在此基础上希望检测其中的包含的各种目标并对它们进行定位，这里对这个监督学习任务的标签表示形式作出定义。



定位分类问题不仅要求判断出图片中物体的种类，还要在图片中标记出它的具体位置，用**边框 (Bounding Box, 或者称包围盒)**把物体圈起来。一般来说，定位分类问题通常只有一个较大的对象位于图片中间位置；而在目标检测问题中，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。如上图所示，分类器将输入的图片分成行人、汽车、摩托车、背景四类，最后输出的就会是一个四维列向量，四个值分别代表四种类别存在的概率。

为了定位图片中汽车的位置，可以让神经网络多输出 4 个数字，标记为 b_x 、 b_y 、 b_h 、 b_w 。将图片左上角标记为 $(0, 0)$ ，右下角标记为 $(1, 1)$ ，则有：

- 红色方框的中心点： (b_x, b_y)
- 边界框的高度： b_h
- 边界框的宽度： b_w

因此，训练集不仅包含对象分类标签，还包含表示边界框的四个数字。定义目标标签 Y 如下：

$$\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad (10)$$

则有：

$$P_c = 1, Y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad (11)$$

其中， c_n 表示存在第 n 个种类的概率；如果 $P_c = 0$ ，表示没有检测到目标，则输出标签后面的 7 个参数都是无效的，可以忽略（用？来表示）。

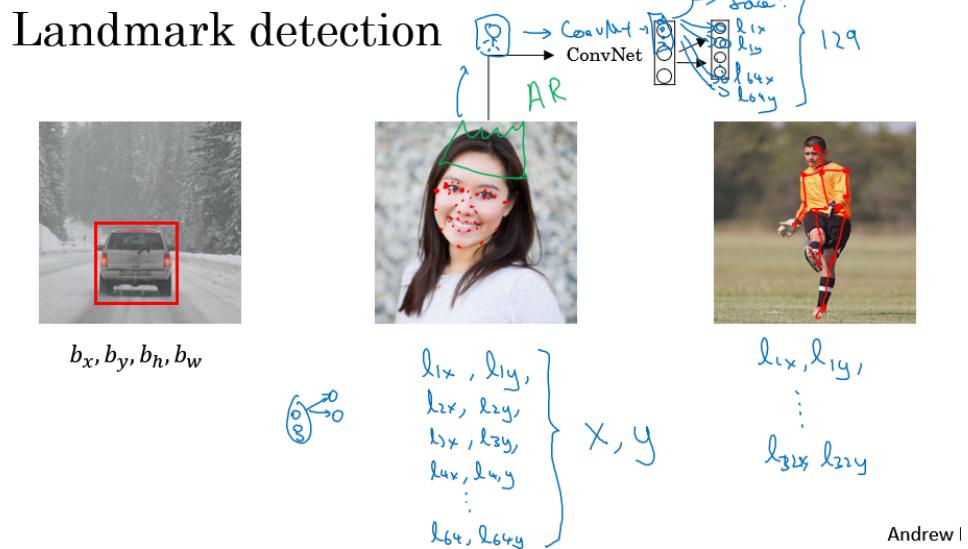
$$P_c = 0, Y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad (12)$$

损失函数可以表示为 $L(\hat{y}, y)$ ，如果使用平方误差形式，对于不同的 P_c 有不同的损失函数（注意下标 i 指标签的第 i 个值）：

$$\mathcal{L}(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_n - y_n)^2, & (y_1(p_c) = 1) \\ (\hat{y}_1 - y_1)^2, & (y_1(p_c) = 0) \end{cases} \quad (13)$$

除了使用平方误差，也可以使用逻辑回归损失函数，类标签 c_1, c_2, c_3 也可以通过 softmax 输出。相比较而言，平方误差已经能够取得比较好的效果。

特征点检测 (Landmark detection)



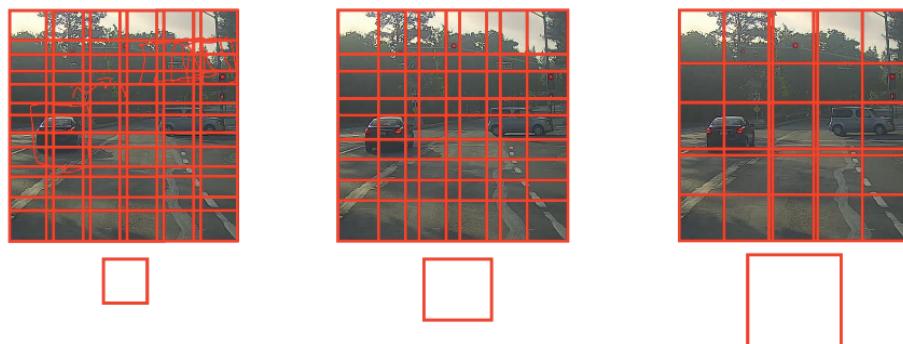
神经网络可以像标识目标的中心点位置那样，通过输出图片上的特征点，来实现对目标特征的识别。在标签中，这些特征点以多个二维坐标的形式表示。

通过检测人脸特征点可以进行情绪分类与判断，或者应用于 AR 领域等等。也可以透过检测姿态特征点来进行人体姿态检测。

1.3.2 滑窗检测

想要实现目标检测，可以采用**基于滑动窗口的目标检测（Sliding Windows Detection）**算法。该算法的步骤如下：

1. 训练集上搜集相应的各种目标图片和非目标图片，样本图片要求尺寸较小，相应目标居于图片中心位置并基本占据整张图片。
2. 使用训练集构建 CNN 模型，使得模型有较高的识别率。
3. 选择大小适宜的窗口与合适的固定步幅，对测试图片进行从左到右、从上倒下的滑动遍历。每个窗口区域使用已经训练好的 CNN 模型进行识别判断。
4. 可以选择更大的窗口，然后重复第三步的操作。

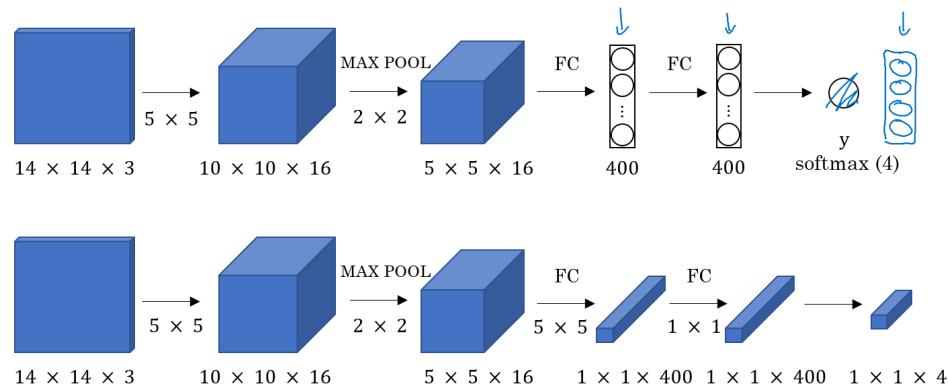


- **优点**是原理简单，且不需要人为选定目标区域；
- **缺点**是需要人为直观设定滑动窗口的大小和步幅。滑动窗口过小或过大，步幅过大均会降低目标检测的正确率。另外，每次滑动都要进行一次 CNN 网络计算，如果滑动窗口和步幅较小，计算成本往往很大。

所以，滑动窗口目标检测算法虽然简单，但是性能不佳，效率较低。

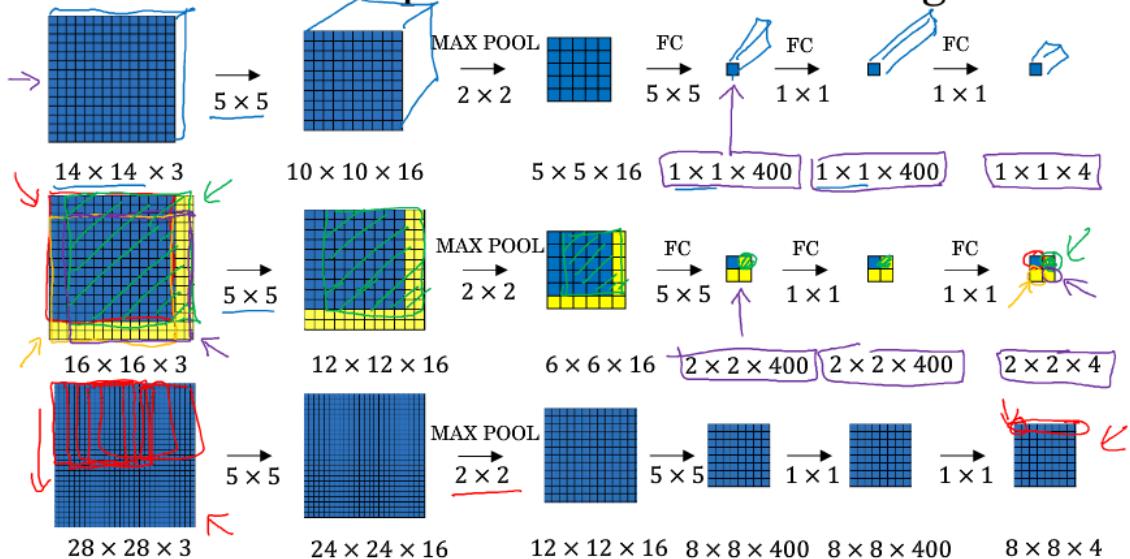
基于卷积的滑动窗口实现

相比从较大图片多次截取，在卷积层上应用滑动窗口目标检测算法可以提高运行速度。所要做的仅是将全连接层换成卷积层，即使用与上一层尺寸一致的滤波器进行卷积运算。



上图所示为一个卷积神经网络，经过卷积、池化后，全连接过程可以看作是将池化后得到的大小为 $5 \times 5 \times 16$ 的结果与 400 个大小也为 $5 \times 5 \times 16$ 的卷积核分别进行卷积，输出的结果大小为 $1 \times 1 \times 400$ ，进一步全连接再采用 Softmax 后，最后输出的结果大小为 $1 \times 1 \times 4$ 。由此，全连接过程本质上还是一个卷积过程。

Convolution implementation of sliding windows



如图，对于 $16 \times 16 \times 3$ 的图片，步长为 2，CNN 网络得到的输出层为 $2 \times 2 \times 4$ 。其中， 2×2 表示共有 4 个窗口结果。对于更复杂的 $28 \times 28 \times 3$ 的图片，得到的输出层为 $8 \times 8 \times 4$ ，共 64 个窗口结果。最大池化层的宽高和步长相等。

运行速度提高的原理：在滑动窗口的过程中，需要重复进行 CNN 正向计算。因此，不需要将输入图片分割成多个子集，分别执行向前传播，而是将它们作为一张图片输入给卷积网络进行一次 CNN 正向计算。这样，公共区域的计算可以共享，以降低运算成本。

这样一个方法，是Sermanet等人2014年在论文 [OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks](#) (<https://arxiv.org/pdf/1312.6229.pdf>) 中提出来的。

1.3.3 YOLO 算法

采用滑窗检测进行目标检测，难以选取到一个可以完美匹配目标位置的，大小合适的窗口。

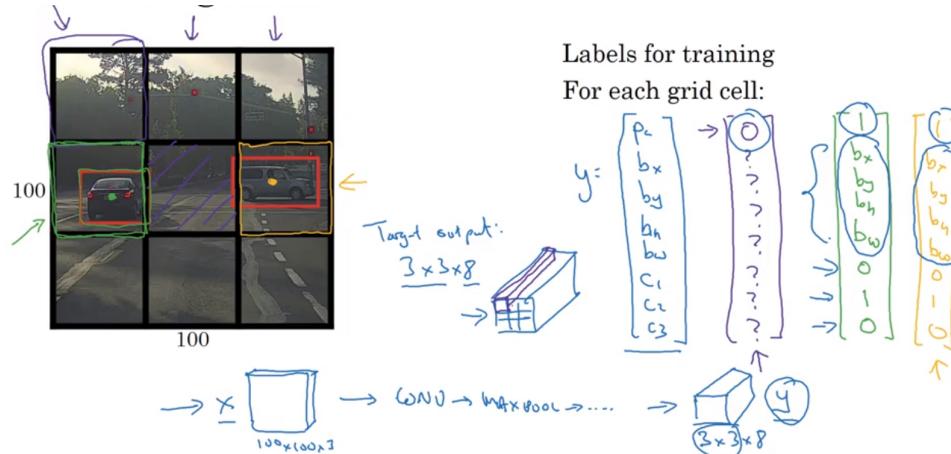
YOLO (You Only Look Once) 算法是 Redmon 等人 2015 年在论文 [You Only Look Once: Unified, Real-Time Object Detection](#) (<https://arxiv.org/pdf/1506.02640.pdf>) 中提出的另一种用于目标检测的算法。YOLO (You Only Look Once) 算法可以用于得到更精确的边框。

YOLO 算法中，将输入的图像划分为 $S \times S$ 个网格 (Grid Cell)，对这 $S \times S$ 个网格分别指定一个标签，标签的形式如前面所述：

- p_c 标识该网格中的目标存在与否。为“1”则表示存在；“0”则表示不存在，且标签中其他值都无效。

- b_x 、 b_y 表示包围盒的中心坐标值，它们相对于该网格进行了归一化，也就是它们的取值范围在 0 到 1 之间；
- b_h 、 b_w 表示包围盒的长度和宽度；
- c_n 表示第 n 个假定类别存在的概率。

若某个目标的中心点落在某个网格，则该网格负责检测该对象。



如上面的示例中，如果将输入的图片划分为 3×3 的网格、需要检测的目标有 3 类，则每一网格部分图片的标签会是一个 8 维的列矩阵，最终输出的就是大小为 $3 \times 3 \times 8$ 的结果。要得到这个结果，就要训练一个输入大小为 $100 \times 100 \times 3$ ，输出大小为 $3 \times 3 \times 8$ 的 CNN。在实践中，可能使用更为精细的 19×19 网格，则两个目标的中点在同一个网格的概率更小。

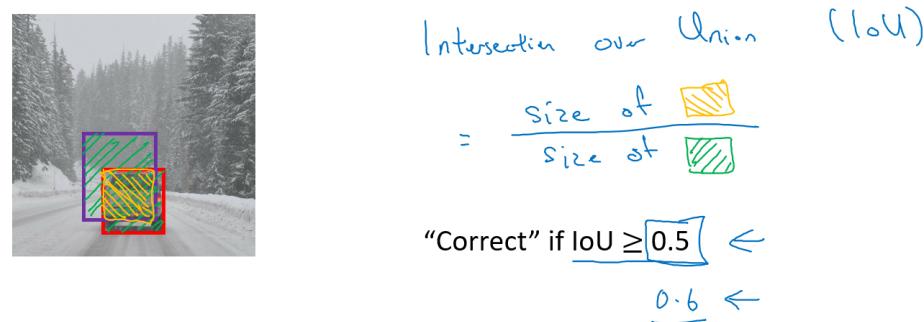
YOLO 算法的优点：

- 和图像分类和目标定位算法类似，显式输出边框坐标和大小，不会受到滑窗分类器的步长大小限制。
- 仍然只进行一次 CNN 正向计算，效率很高，甚至可以达到实时识别。

如何编码边框 b_x 、 b_y 、 b_h 、 b_w ? YOLO 算法设 b_x 、 b_y 、 b_h 、 b_w 的值是相对于网格长的比例。则 b_x 、 b_y 在 0 到 1 之间，而 b_h 、 b_w 可以大于 1。当然，也有其他参数化的形式，且效果可能更好。这里只是给出一个通用的表示方法。

交并比

预测出的目标位置的准确程度用 IOU (Intersection Over Union) 来衡量，它表示预测出的包围盒 (Bounding Box) 与实际边界 (Ground Truth) 的重叠度，也就是两个不同包围盒的交并比。如下图中所示，IOU 就等于两个包围盒的交集面积 (黄色部分) 占两个包围盒的并集面积 (绿色部分) 的比率。一般可以约定一个阈值，以此判断预测的包围盒的准确与否。



IoU 的值在 0 ~ 1 之间，且越接近 1 表示目标的定位越准确。IoU 大于等于 0.5 时，一般可以认为预测边框是正确的，当然也可以更加严格地要求一个更高的阈值。

非极大值抑制

使用 YOLO 算法进行目标检测，因为是多个网格对某些目标同时进行检测，很可能会出现同一目标被多个网格检测到，并生成多个不同的包围盒的情况，这时需要通过非极大值抑制（Non-max Suppression）来筛选出其中最佳的那个。非极大值抑制（Non-max Suppression）会通过清理检测结果，找到每个目标中点所位于的网格，确保算法对每个目标只检测一次。

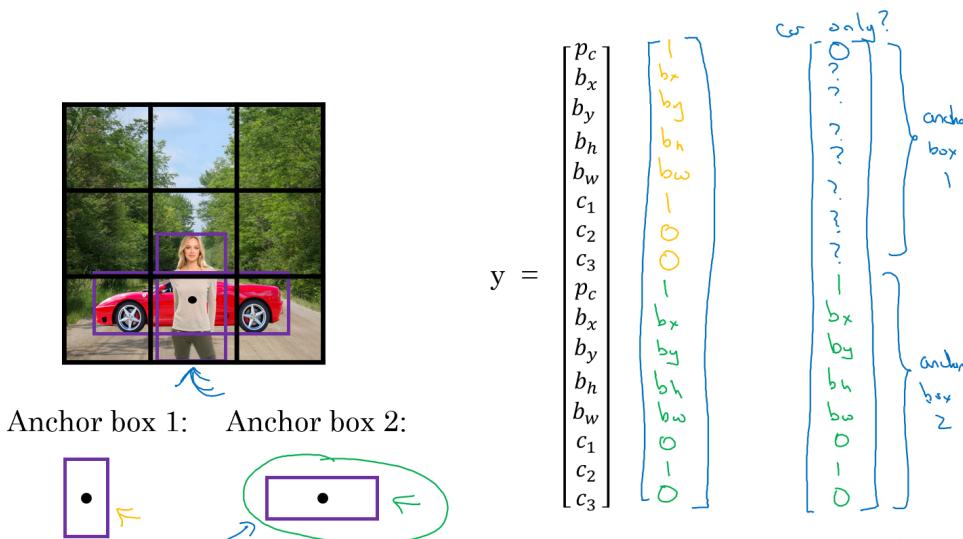
进行非极大值抑制的步骤如下：

- 将包含目标中心坐标的可信度 P_c 小于阈值（例如 0.6）的网格丢弃；
- 选取拥有最大 P_c 的网格；
- 分别计算该网格和其他所有网格的 IoU，将 IoU 超过预设阈值的网格丢弃；
- 重复第 2~3 步，直到不存在未处理的网格。

上述步骤适用于单类别目标检测。进行多个类别目标检测时，对于每个类别，应该单独做一次非极大值抑制。

Anchor Boxes

上述算法只适用于单目标检测，也就是每个网格只能检测一个对象。要将该算法运用在多目标检测上，需要用到 **Anchor Boxes**。在原单目标检测所用的标签中加入其他目标的标签，每个目标的标签表示形式都如上所述，一组标签即标明一个 Anchor Box，则一个网格的标签中将包含多个 Anchor Box，相当于存在多个用以标识不同目标的包围盒。



在上图示例中，我们希望同时检测人和汽车。因此，每个网格的的标签中含有两个 Anchor Box。输出的标签结果大小从 $3 \times 3 \times 8$ 变为 $3 \times 3 \times 16$ 。若两个 P_c 都大于预设阈值，则说明检测到了两个目标。

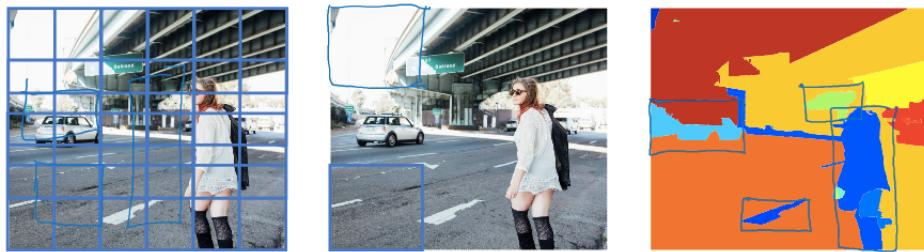
在单目标检测中，图像中的目标被分配给了包含该目标中点的那个网格；引入 Anchor Box 进行多目标检测时，图像中的目标则被分配到了包含该目标中点的那个网格以及具有最高 IoU 值的该网格的 Anchor Box。

Anchor Boxes 也有局限性，对于同一网格有三个及以上目标，或者两个目标的 Anchor Box 高度重合的情况处理不好。

Anchor Box 的形状一般通过人工选取。高级一点的方法是用 k-means 将两类对象形状聚类，选择最具代表性的 Anchor Box。

1.3.4 R-CNN

前面介绍的滑动窗口目标检测算法对一些明显没有目标的区域也进行了扫描，这降低了算法的运行效率。为了解决这个问题，R-CNN (Region CNN, 带区域的 CNN) 被提出。通过对输入图片运行**图像分割算法**，在不同的色块上找出**候选区域 (Region Proposal)**，就只需要在这些区域上运行分类器。



R-CNN意为带区域的卷积网络，类似之前所述的滑窗检测算法，先用卷积网络训练一个能够准确识别目标的分类器，但这个算法试图选出一些区域为候选区域，只在这些区域也就是只在少数的窗口上运行分类器。候选区域的选取采用的是一种称为图像分割的算法。R-CNN 的缺点是运行速度很慢，所以有一系列后续研究工作改进。例如 Fast R-CNN（与基于卷积的滑动窗口实现相似，但得到候选区域的聚类步骤依然很慢）、Faster R-CNN（使用卷积对图片进行分割）。不过大多数时候还是比 YOLO 算法慢。

相关论文：

- R-CNN: [Girshik et al., 2013. Rich feature hierarchies for accurate object detection and semantic segmentation \(<https://arxiv.org/pdf/1311.2524.pdf>\)](https://arxiv.org/pdf/1311.2524.pdf)
- Fast R-CNN: [Girshik, 2015. Fast R-CNN \(<https://arxiv.org/pdf/1504.08083.pdf>\)](https://arxiv.org/pdf/1504.08083.pdf)
- Faster R-CNN: [Ren et al., 2016. Faster R-CNN: Towards real-time object detection with region proposal networks](<https://arxiv.org/pdf/1506.01497v3.pdf> (<https://arxiv.org/pdf/1506.01497v3.pdf>))

1.4 特殊应用

1.4.1 人脸识别

人脸验证 (Face Verification) 和人脸识别 (Face Recognition) 的区别：

- 人脸验证：一般指一个一对一个问题，只需要验证输入的人脸图像是否与某个已知的身份信息对应；
- 人脸识别：一个更为复杂的一对多问题，需要验证输入的人脸图像是否与多个已知身份信息中的某一个匹配。

一般来说，由于需要匹配的身份信息更多导致错误率增加，人脸识别比人脸验证更难一些。

1.4.1.1 One-Shot 学习

人脸识别所面临的一个挑战是要求系统只采集某人的一个面部样本，就能快速准确地识别出这个人，即只用一个训练样本来获得准确的预测结果。这被称为 **One-Shot** 学习。

有一种方法是假设数据库中存有 N 个人的身份信息，对于每张输入图像，用 Softmax 输出 N+1 种标签，分别对应每个人以及都不是。然而这种方法的实际效果很差，因为过小的训练集不足以训练出一个稳健的神经网络；并且如果有新的身份信息入库，需要重新训练神经网络，不够灵活。

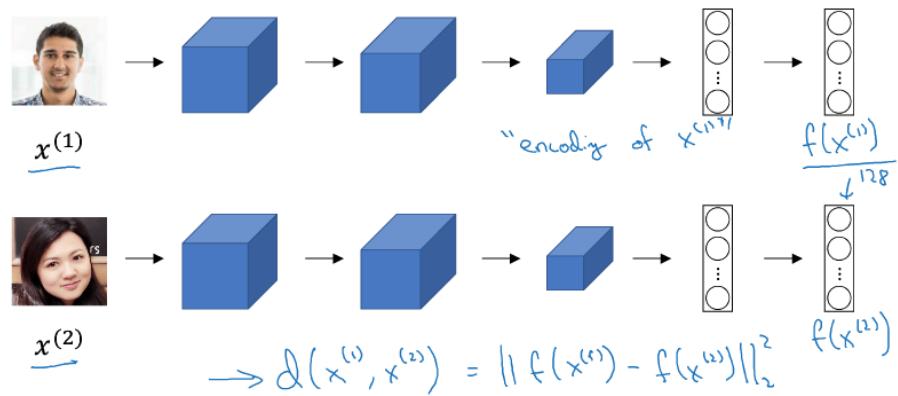
因此，我们通过学习一个 Similarity 函数来实现 One-Shot 学习过程。Similarity 函数定义了输入的两幅图像的差异度，其公式如下：

$$\text{Similarity} = d(\text{img1}, \text{img2}) \quad (14)$$

可设置一个超参数 τ ，当 $d(\text{img1}, \text{img2}) \leq \tau$ ，则两幅图片为同一人，否则为不同

1.4.1.2 Siamese 网络

实现 Similarity 函数的一种方式是使用 Siamese 网络，它是一种对两个不同输入运行相同的卷积网络，然后对它们的结果进行比较的神经网络。

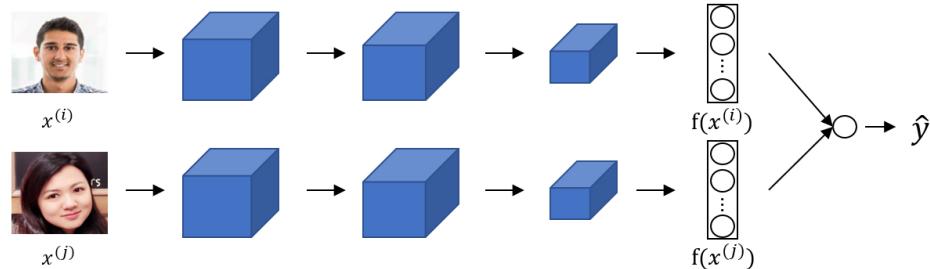


上图的示例中，将图片 $x^{(1)}$ 、 $x^{(2)}$ 分别输入两个相同的卷积网络中，经过全连接后不再进行 Softmax，得到它们的特征向量 $f(x^{(1)})$ 、 $f(x^{(2)})$ 。此时 Similarity 函数就被定义为这两个特征向量之差的 2 范数：

$$d(x^{(1)}, x^{(2)}) = \|f(x^{(1)}) - f(x^{(2)})\|_2^2 \quad (15)$$

二分类结构

利用一对相同的 Siamese 网络，可以将人脸验证看作二分类问题。



如上图中，输入的两张图片 $x^{(i)}$ 、 $x^{(j)}$ ，经过卷积网络后分别得到m维的特征向量 $f(x^{(i)})$ 、 $f(x^{(j)})$ ，将它们输入一个逻辑回归单元，最后输出的预测结果中用 1 和 0 表示相同或不同的人。

其中对最后的输出结果 \hat{y} ，如果使用的逻辑回归单元是 sigmoid 函数，表达式就会是：

$$\hat{y} = \sigma\left(\sum_{k=1}^K w_k |f(x^{(i)})_k - f(x^{(j)})_k| + b\right) \quad (16)$$

其中， w_k 和 b 都是通过梯度下降算法迭代训练得到的参数。上述计算表达式也可以用另一种表达式代替：

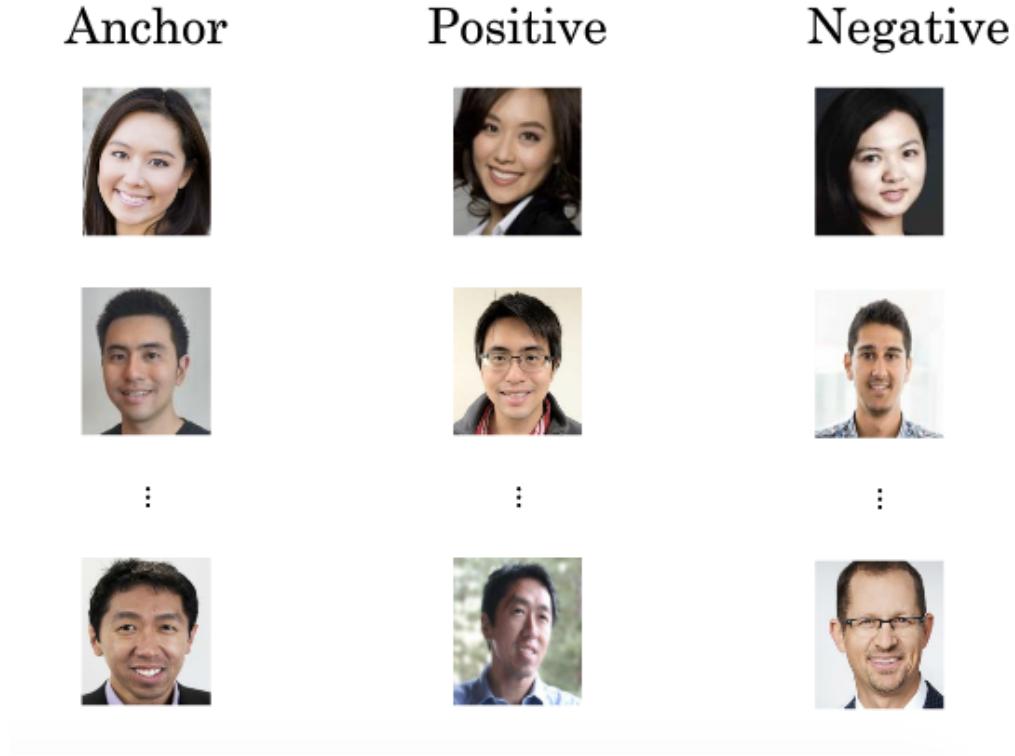
$$\hat{y} = \sigma\left(\sum_{k=1}^K w_k \frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k} + b\right) \quad (17)$$

其中， $\frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$ 被称为 χ 方相似度。

以上所述内容，都来自 Taigman 等人 2014 年发表的论文 [DeepFace closing the gap to human level performance \(\[http://www.cs.wayne.edu/~mdong/taigman_cvpr14.pdf\]\(http://www.cs.wayne.edu/~mdong/taigman_cvpr14.pdf\)\)](http://www.cs.wayne.edu/~mdong/taigman_cvpr14.pdf) 中提出的 DeepFace。

1.4.1.3 Triplet 损失

Triplet 损失函数用于训练出合适的参数，以获得高质量的人脸图像编码。“Triplet”一词来源于训练这个神经网络需要大量包含 Anchor (靶目标)、Positive (正例)、Negative (反例) 的图片组，其中 Anchor 和 Positive 需要是同一个人的人脸图像。



对于这三张图片，应该有：

$$\|f(A) - f(P)\|_2^2 + \alpha \leq \|f(A) - f(N)\|_2^2 \quad (18)$$

其中， α 被称为间隔 (margin)，用于确保 $f()$ 不会总是输出零向量 (或者一个恒定的值)。那么 Triplet 损失函数的定义：

$$L(A, P, N) = \max(\|f(A) - f(P)\|_2^2 - \|f(A) - f(N)\|_2^2 + \alpha, 0) \quad (19)$$

其中，因为 $\|f(A) - f(P)\|_2^2 - \|f(A) - f(N)\|_2^2 + \alpha$ 的值需要小于等于 0，因此取它和 0 的更大值。

这样，训练这个神经网络就需要有大量经过特定组合的包含 Anchor、Positive、Negative 的图片组。且使用 m 个训练样本，代价函数将是：

$$\begin{aligned} J &= \sum_{i=1}^m L(A^{(i)}, P^{(i)}, N^{(i)}) \\ &= \sum_{i=1}^m [\|f(A^{(i)}) - f(P^{(i)})\|_2^2 - \|f(A^{(i)}) - f(N^{(i)})\|_2^2 + \alpha] \end{aligned} \quad (20)$$

通过梯度下降最小化代价函数。

在选择训练样本时，随机选择容易使 Anchor 和 Positive 极为接近，而 Anchor 和 Negative 相差较大，以致训练出来的模型容易抓不到关键的区别。因此，最好的做法是人为增加 Anchor 和 Positive 的区别，缩小 Anchor 和 Negative 的区别，促使模型去学习不同人脸之间的关键差异。

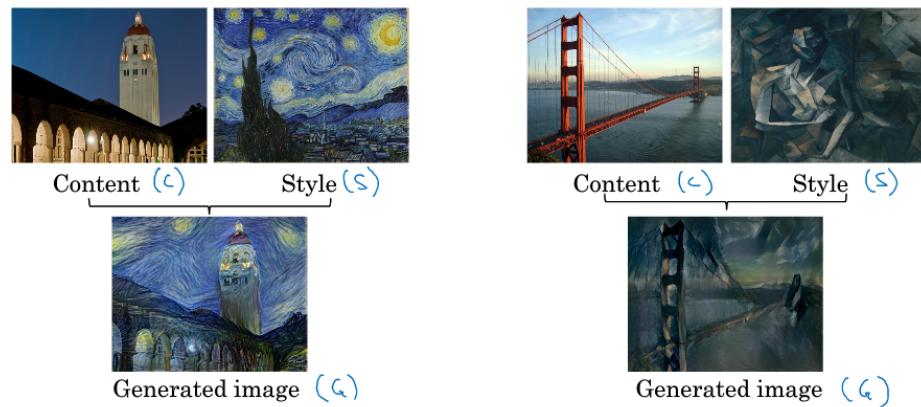
Triplet 损失的相关内容来自 Schroff 等人 2015 年在论文 [FaceNet: A unified embedding for face recognition and clustering \(<https://arxiv.org/pdf/1503.03832.pdf>\)](https://arxiv.org/pdf/1503.03832.pdf) 中提出的 FaceNet，更细节可以参考论文内容。

注意

无论是对于使用 Triplet 损失函数的网络，还是二分类结构，为了减少计算量，可以提前计算好编码输出 $f(x)$ 并保存。这样就不必存储原始图片，并且每次进行人脸识别时只需要计算测试图片的编码输出。

1.4.2 神经风格转换

神经风格迁移 (Neural Style Transfer) 是将参考风格图像的风格转换到另一个输入图像中，如下图所示。

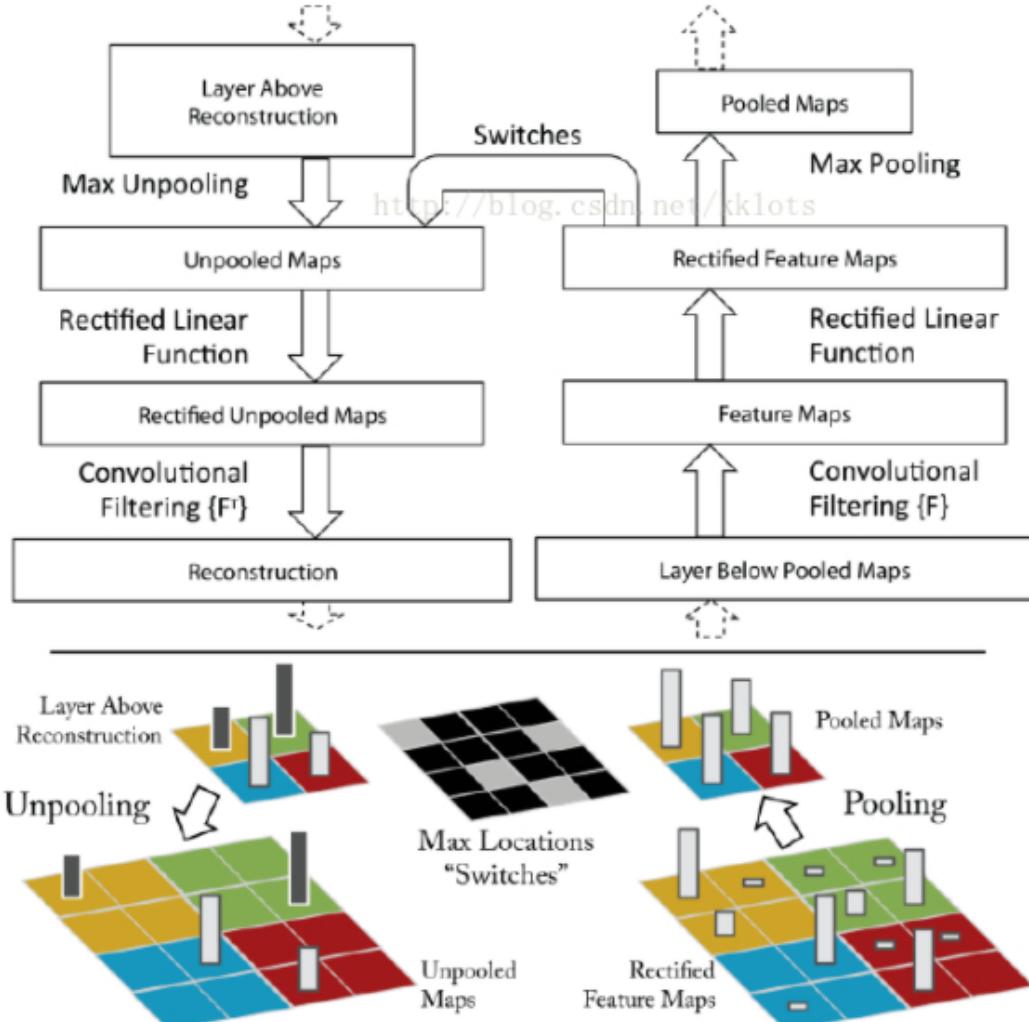


其中待转换的图片标识为 C (Content) , 某种风格的图片为 S (Style) , 转换后的图片为 G (Generated) 。

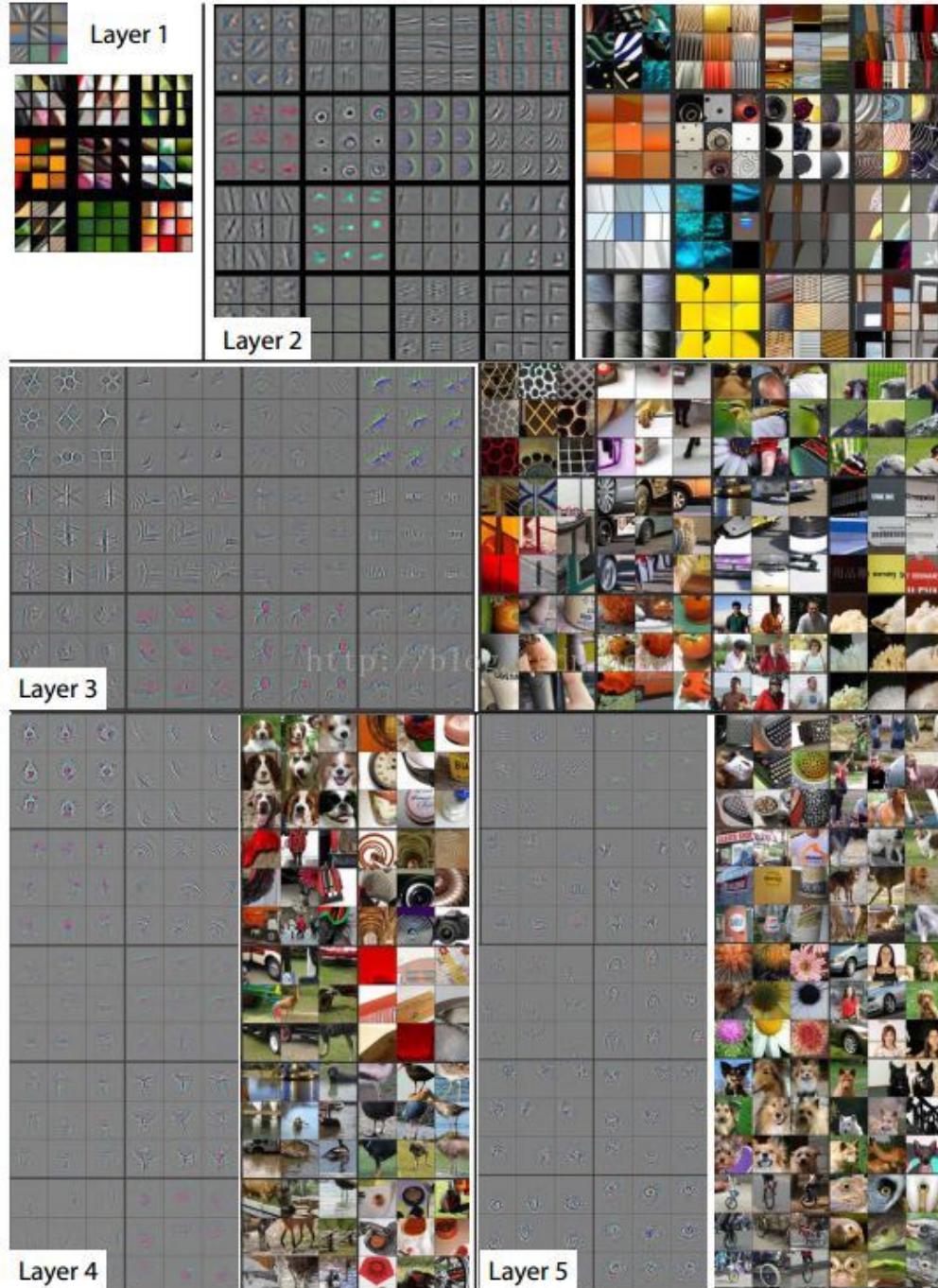
1.4.2.1 理解 CNN (可视化特征)

要理解利用卷积网络实现神经风格转换的原理，首先要理解在输入图像数据后，一个深度卷积网络从中都学到了些什么。

2013 年 Zeiler 和 Fergus 在论文 [Visualizing and understanding convolutional networks](https://arxiv.org/pdf/1311.2901.pdf) (<https://arxiv.org/pdf/1311.2901.pdf>) 中提出了一种将卷积神经网络的隐藏层特征进行可视化的方法。



上图展示的是一个 AlexNet 中的卷积、池化以及最后的归一化过程，以及实现隐藏层可视化的反卷积网络中的 Unpooling、矫正以及反卷积过程。论文中将 ImageNet 2012 中的 130 万张图片作为训练集，训练结束后提取到的各个隐藏层特征如下图：



从中可以看出，浅层的隐藏单元通常学习到的是边缘、颜色等简单特征，越往深层，隐藏单元学习到的特征也越来越复杂。

1.4.2.2 代价函数

实现神经风格转换，需要定义一个关于生成的图像 G 的代价函数 $J(G)$ ，以此评判生成图像的好坏的同时，用梯度下降法最小化这个代价函数，而生成最终的图像。

神经风格迁移生成图片 G 的代价函数如下：

$$J(G) = \alpha \cdot J_{content}(C, G) + \beta \cdot J_{style}(S, G) \quad (21)$$

其中内容代价函数 $J_{content}(C, G)$ 度量待转换的 C 和生成的 G 的相似度，风格代价函数 $J_{style}(S, G)$ 则度量某风格的 S 和生成的 G 的相似度， α 、 β 是用于控制相似度比重的超参数。

神经风格迁移的算法步骤如下：

- 随机生成图片 G 的所有像素点；
- 使用梯度下降算法使代价函数最小化，以不断修正 G 的所有像素点。

内容代价函数

$J_{content}(C, G)$ 的计算过程如下：

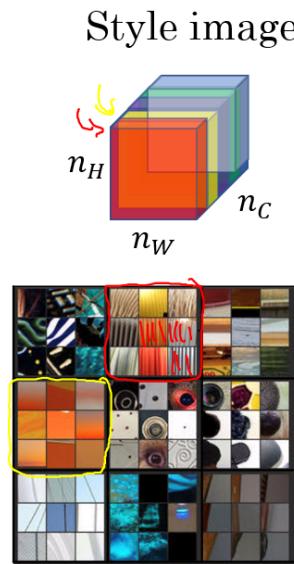
- 使用一个预训练好的 CNN (例如 VGG)；
- 选择一个隐藏层 l 来计算内容代价。 l 太小则内容图片和生成图片像素级别相似， l 太大则可能只有具体物体级别的相似。因此， l 一般选一个中间层；
- 设 $a^{(C)[l]}$ 、 $a^{(G)[l]}$ 为 C 和 G 在 l 层的激活，则有：

$$J_{content}(C, G) = \frac{1}{2} ||(a^{(C)[l]} - a^{(G)[l]})||^2 \quad (22)$$

$a^{(C)[l]}$ 和 $a^{(G)[l]}$ 越相似，则 $J_{content}(C, G)$ 越小。

风格代价函数

定义风格代价函数 $J_{style}(S, G)$ 前，首先提取出 S 的“风格”。通过之前的理解 CNN 内容，将 S 也输入那个预先训练好的卷积神经网络中，就可以将其所谓的“风格”定义为神经网络中某一层或者几个层中，各个通道的激活项之间的相关系数。如下图所示为网络中的某一层，假设其中前两个红、黄色通道分别检测出了下面对于颜色圈出的特征，则这两个通道的相关系数，就反映出了该图像所具有的“风格”。



更进一步解释为：每个通道提取图片的特征不同，比如标为红色的通道提取的是图片的垂直纹理特征，标为黄色的通道提取的是图片的橙色背景特征。那么计算这两个通道的相关性，相关性的大小，即表示原始图片既包含了垂直纹理也包含了该橙色背景的可能性大小。通过 CNN，“风格”被定义为同一个隐藏层不同通道之间激活值的相关系数，因其反映了原始图片特征间的相互关系。

对于风格图像 S，选定网络中的第 l 层，则相关系数以一个 gram 矩阵的形式表示：

$$G_{kk'}^{[l](S)} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l](S)} a_{ijk'}^{[l](S)} \quad (23)$$

其中， i 和 j 为第 l 层的高度和宽度； k 和 k' 为选定的通道，其范围为 1 到 $n_C^{[l]}$ ； $a_{ijk}^{[l](S)}$ 为激活。

同理，对于生成图像 G，有：

$$G_{kk'}^{[l](G)} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l](S)} a_{ijk'}^{[l](G)} \quad (24)$$

因此，第 l 层的风格代价函数为：

$$J_{style}^{[l]}(S, G) = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})^2 \quad (25)$$

如果对各层都使用风格代价函数，效果会更好。因此有：

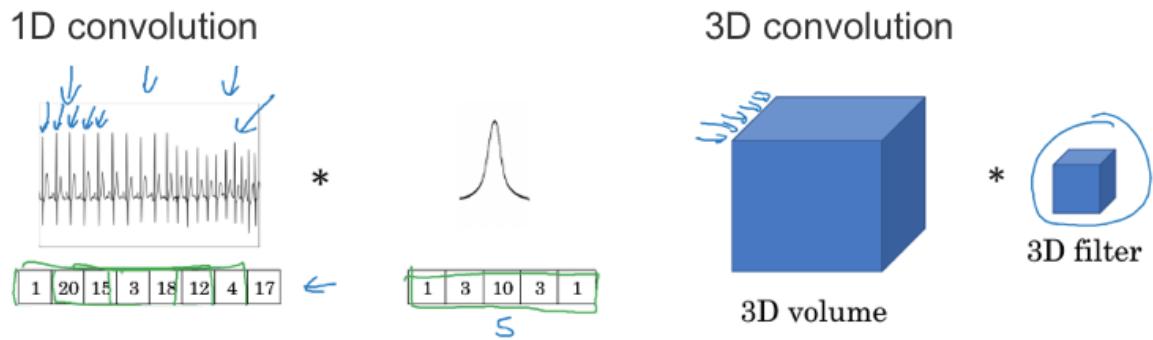
$$J_{style}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S, G) \quad (26)$$

其中， λ 是用于设置不同层所占权重的超参数。

这样一种神经风格转换的实现方法，来自 2015 年 Gatys 等人发表的论文 [A Neural Algorithm of Artistic Style](https://arxiv.org/abs/1508.06576) (<https://arxiv.org/abs/1508.06576>)。

1.4.3 扩展至一维和三维

之前我们处理的都是二维图片，实际上卷积也可以延伸到一维和三维数据。我们举两个示例来说明。



EKG 数据（心电图）是由时间序列对应的每个瞬间的电压组成，是一维数据。一般来说我们会用 RNN（循环神经网络）来处理，不过如果用卷积处理，则有：

- 输入时间序列维度： 14×1
- 滤波器尺寸： 5×1 , 滤波器个数：16
- 输出时间序列维度： 10×16

而对于三维图片的示例，有

- 输入 3D 图片维度： $14 \times 14 \times 14 \times 1$
- 滤波器尺寸： $5 \times 5 \times 5 \times 1$, 滤波器个数：16
- 输出 3D 图片维度： $10 \times 10 \times 10 \times 1$