

Lecture5 Convolutional Neural Networks

1. CNN 介绍

- CNN 可视化: <http://scs.ryerson.ca/~aharley/vis/>
- 在浏览器中构建 CNN: <https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>
- Keras 中的各种神经网络: <https://transcranial.github.io/keras-js>

MLP 的局限



- 电脑是如何看这张图片的?



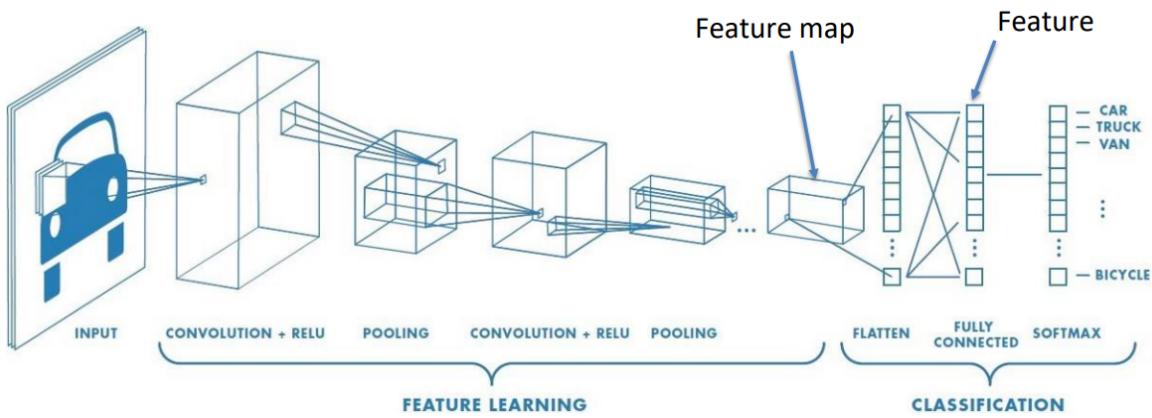
前 $20 \times 20 = 400$ 个像素

$1920 \text{ (宽度)} \times 1080 \text{ (高度)} = 2,073,600$ 个输入变量

$1920 \text{ (宽度)} \times 1080 \text{ (高度)} \times 3 \text{ (RGB)} = 6,220,800$ 个输入变量

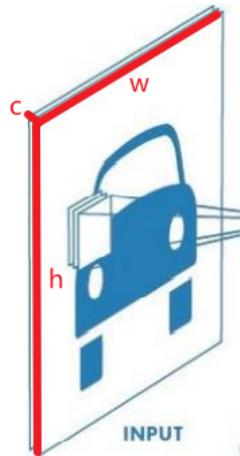
- 每一个像素有多少个通道 (图片中的通道数是 RGB 三种颜色) 来表示
- 如果你使用 MLP，并用一个隐藏层加上 100 个单元的话，那么有上亿个参数！！

什么是卷积神经网络 CNN



- 连接模式类似于人类大脑中的神经元
 - 单个神经元只对视觉范围内一个被称为**接受区域的受限区域**的刺激作出反应
 - 不同通道的卷积核 kernel, 可以理解为一个受限的神经元
 - 这些字段的集合重叠覆盖整个视觉区域

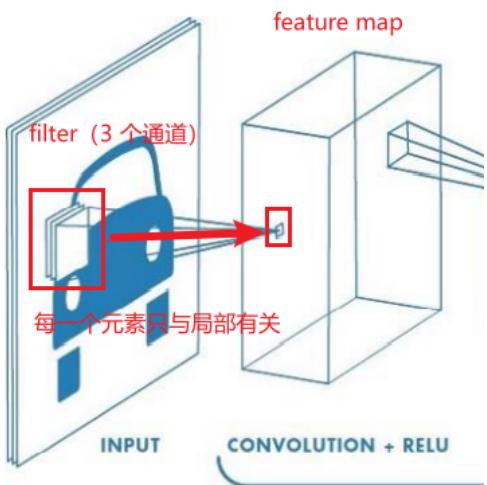
输入



由于图片有 RGB 三个通道, 所以输入大部分都是三个矩阵 (3 个维度)

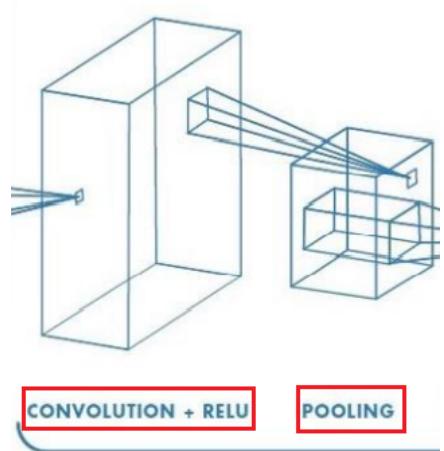
- h : 高度
- w : 宽度
- c : 通道数

卷积层



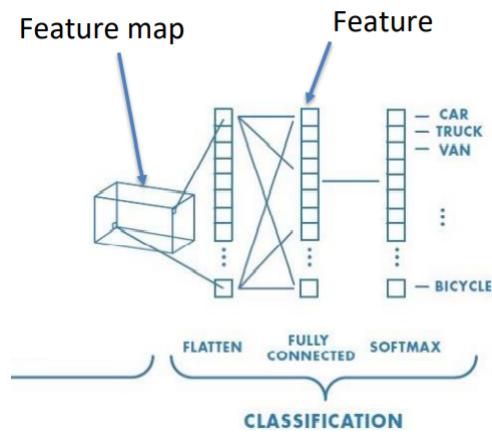
- feature map
 - 通道随着**卷积核 kernel**的通道，逐步变宽
 - 长和宽逐渐变小
 - 但是尽量要保证 w/h 与原来的图像的比例一样

特征学习



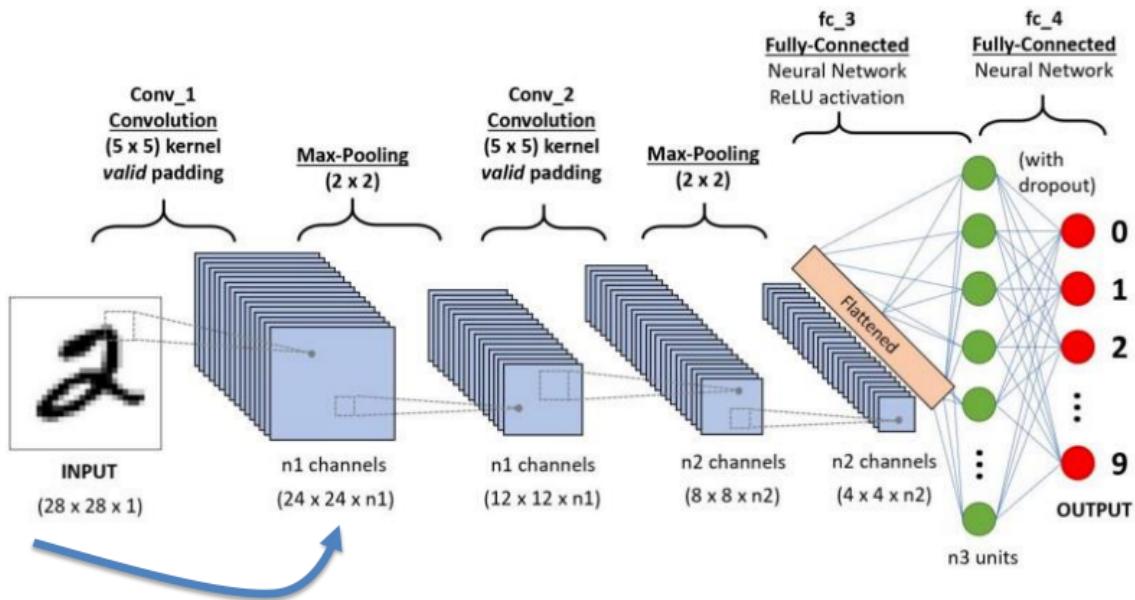
- 反复完成上面三个操作：**卷积 + 激活函数 + 池化**

Flatten

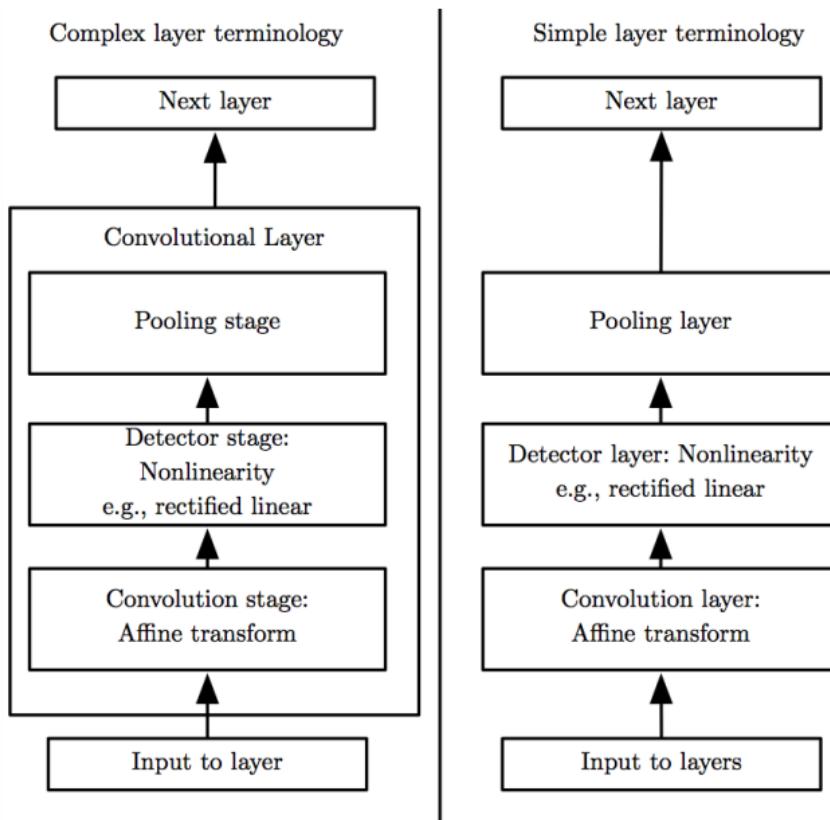


- 将最后通过卷积层出来的 feature map 拉平，拉平后称为 feature
- 最后通过全连接层接到输出层进行分类

另一种 CNN 的描述



CNN 的组件



- **卷积层 Convolution layers**
 - 卷积核大小 Kernel size
 - 步长 Stride
 - 填补 Padding
 - 特征映射 Feature map
- **池化层 Pooling layers**
 - 最大池化 Maximum pooling
 - 平均池化 Average pooling
- **全连接层 Fully connected layers**

组件的优点

- 通过以下方法保持空间结构
 - 卷积核
- 使用下面的组件扩展到处理非常大的输入（布局在网格上）
 - 松散连接 Sparse Connection
 - 参数共享（需要学习的参数变少）
- 对局部方差的鲁棒性
 - 池化

CNN 的核心想法

- 将第一层的卷积乘积替换为卷积操作
- 其它仍然是一样的
 - 最大似然估计
 - 反向传播

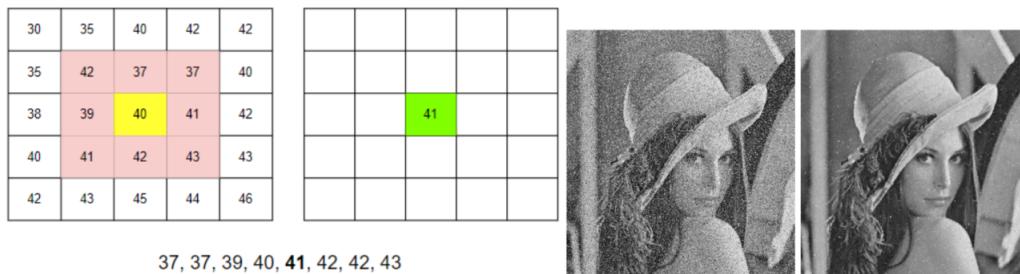
2. CNN 算法

滤波器 Filter

滤波技术用于增强和修改数字图像。此外，图像滤波器用于模糊和降噪，锐化和边缘检测

- 它来源于人类的经验，filter 里面的值是固定的
- 而 CNN 的 kernel（卷积核）与滤波器非常像，区别在于 kernel 是可学习的

中值滤波器 Median Filter



- 中值滤波器是一种非线性滤波器
- 它用相邻像素的中间值替换每个像素值
- 这是去除椒盐杂音的有效方法

均值滤波器 Mean Filter

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



- 均值滤波是一种简单、直观、易于实现的图像平滑方法，即减少一个像素与下一个像素之间的强度变化量，它常被用来降低图像中的噪声

高斯滤波器 Gaussian Filter

1	2	1
2	4	2
1	2	1

3x3 kernel

1/173

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

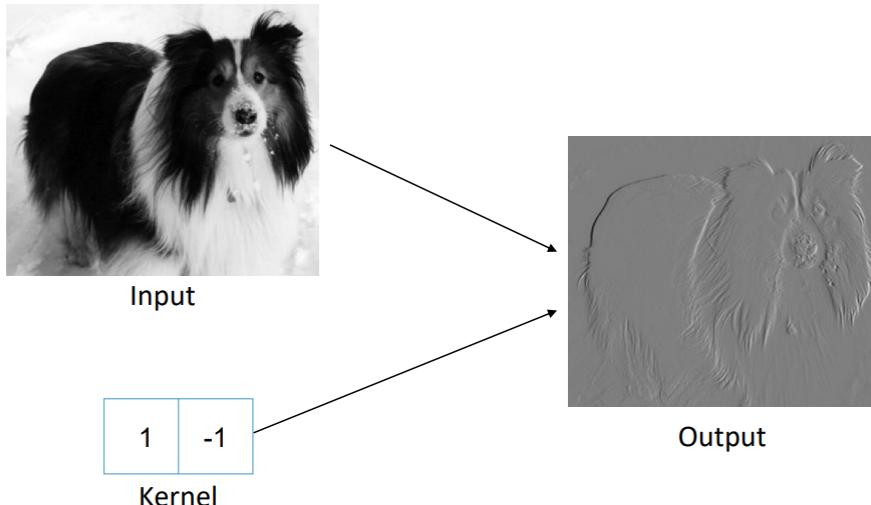
5x5 kernel



3X3 Kernel 5x5 Kernel

- 这个滤波器是一个二维卷积算子
- 它用来模糊图像，此外，它还去除细节和噪音
- 高斯滤波器类似于均值滤波器，但主要的区别是，高斯滤波器采用核函数。核的形状是一个高斯驼峰。高斯核对其中心像素的加权比其边界强得多

索贝尔滤波器 Sobel filter



- 可以用来检测图像的边界

还有一些索贝尔滤波器的变种，例如下面的一种

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

- \mathbf{G}_x : 发现图像的竖着的边缘
- \mathbf{G}_y : 发现图像的横着的边缘

在 Matlab 中也有使用



拉普拉斯滤波器 Laplacian Filter

0	-1	0
-1	4	-1
0	-1	0

Kernel 1

-1	-1	-1
-1	8	-1
-1	-1	-1

Kernel 2



Kernel 1

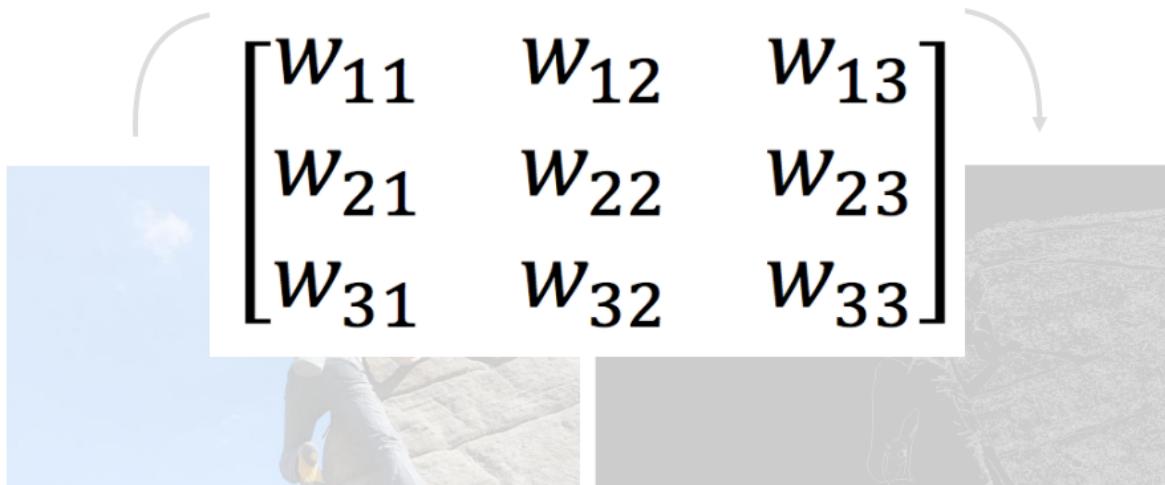
Kernel 2

$$\text{Laplace}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- 拉普拉斯平滑技术主要用于图像边缘的检测，它突出了灰色层次的不连续性
- 它是基于图像的二次空间倒数
- 拉普拉斯边缘检测器只使用一个核
- 为了检测图像的边缘，该核通过单次检测图像强度水平的二阶导数
- 拉普拉斯方法的计算速度比其他方法快

卷积 Convolution

可学习滤波器 Learnable filter -> Kernel



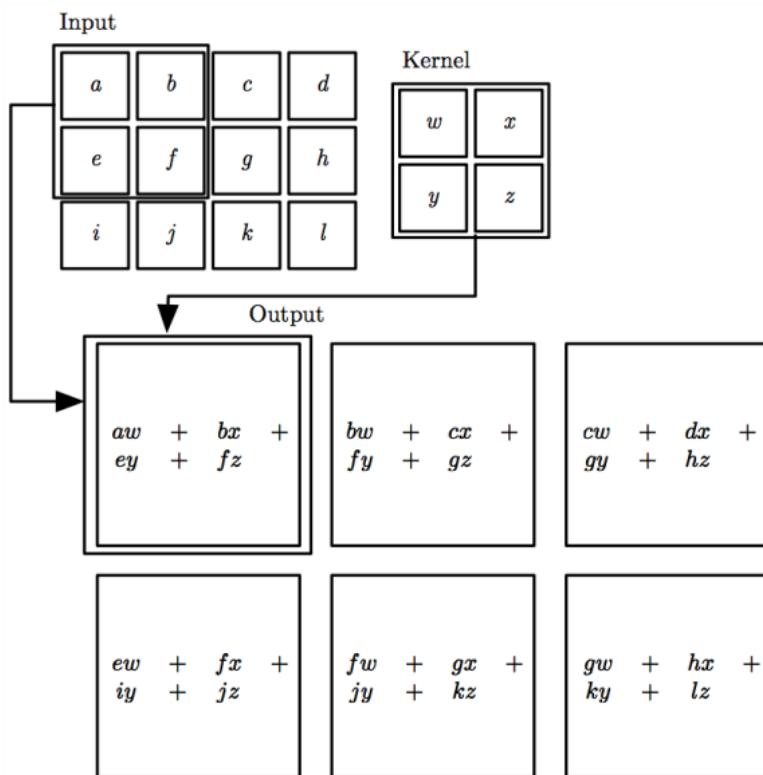
- 上面讲的滤波器，参数都是固定的
- 在深度学习中，滤波器，也就是卷积核，它的参数是**可以学习的**

$$a_{rc} = \sum_{i=-a}^a \sum_{j=-b}^b x_{r-i,c-j} \cdot w_{ij}$$

$$\frac{\partial a_{rc}}{\partial w_{ij}} = \sum_{i=-a}^a \sum_{j=-b}^b x_{r-i,c-j}$$

- 卷积本质上是一个点积，就像线性层一样

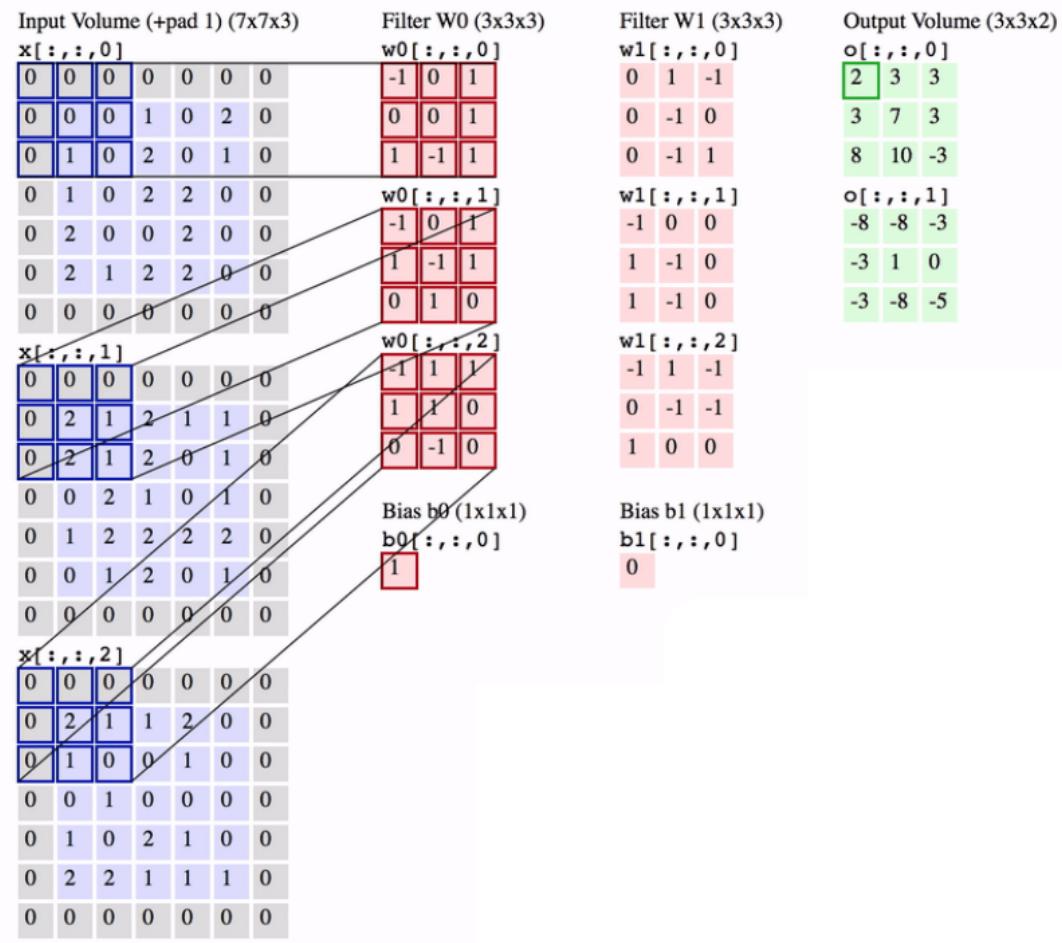
二维卷积



- 输入: $4 \times 4 \times 1$
- Kernel: $2 \times 2 \times 1$ (步长在 x, y 方向上都为 1)
- 卷积输出: $3 \times 2 \times 1$

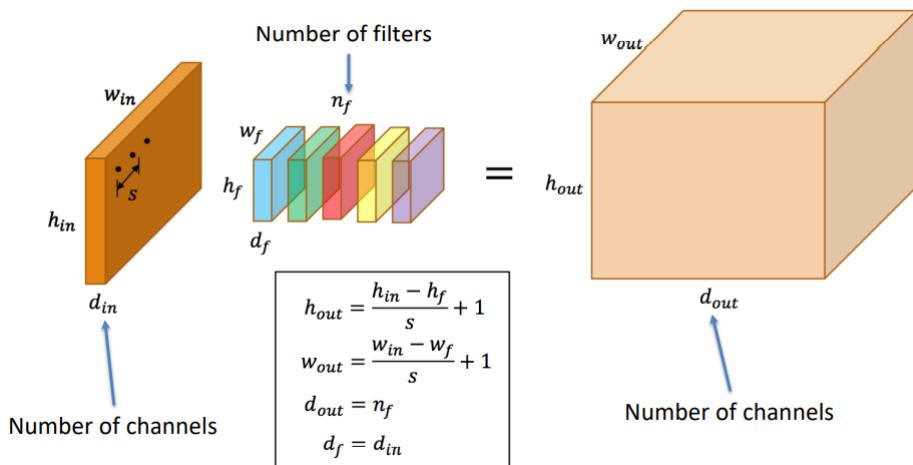
卷积的计算

CS231n Convolutional Neural Networks for Visual Recognition



- 输入
 - 原图片: $5 \times 5 \times 3$
 - padding = 1
 - 预处理后: $7 \times 7 \times 3$
- 卷积核 chennel = 2 (有两个卷积核 = 有两个通道 W1, W2)
 - 大小: $3 \times 3 \times 3$
 - stride = 2
- 输出
 - $3 \times 3 \times 2$ (两个通道的输出结果)

计算公式



- $(5 + 1 \times 2 - 3)/2 + 1 = 3$ (上面的示例中，长和宽都相同)
- $w_{out} = \frac{w - \text{kernel size}_w + 2 \times \text{padding}_w}{\text{stride}_w} + 1$
- $h_{out} = \frac{h - \text{kernel size}_h + 2 \times \text{padding}_h}{\text{stride}_h} + 1$
- $c_{out} = \text{filter size}$ (输出的通道数 = 卷积核的个数)
- $c_{input} = c_{filter}$ (卷积核的通道数 = 输入的通道数)

O = Size (width) of output image.

I = Size (width) of input image.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

S = Stride of the convolution operation.

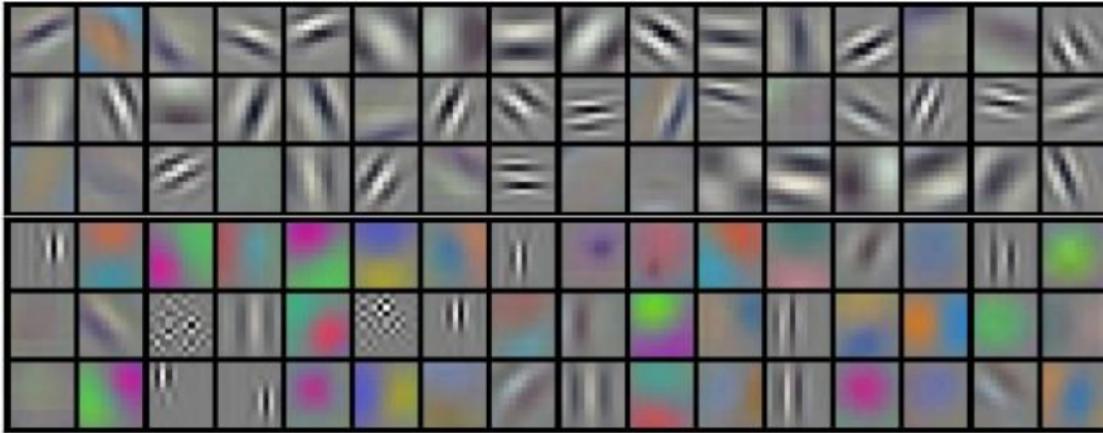
P = Padding.

The size (O) of the output image is given by

$$O = \frac{I - K + 2P}{S} + 1$$

The number of channels in the output image is equal to the number of kernels N .

卷积效果示例

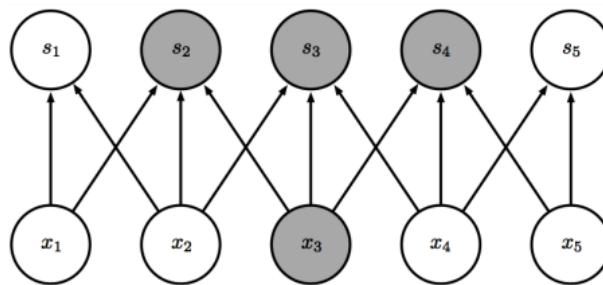


- Krizhevsky 等人学习的过滤器示例
- 这里显示的 96 个卷积核
- 大小都是 11x11x3

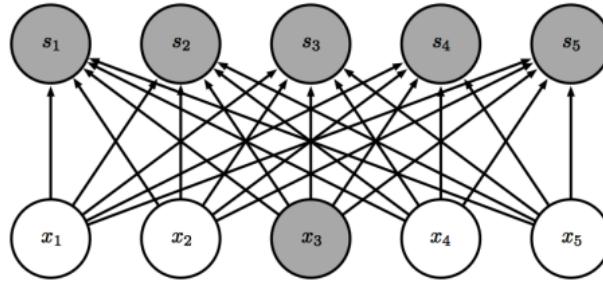
稀疏连接 Sparse Connectivity

From input view

Sparse
connections
due to small
convolution
kernel

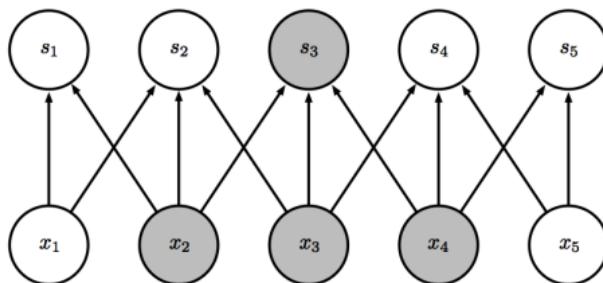


Dense
connections

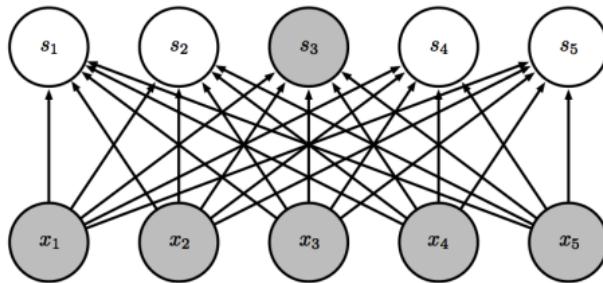


From output view

Sparse
connections
due to small
convolution
kernel

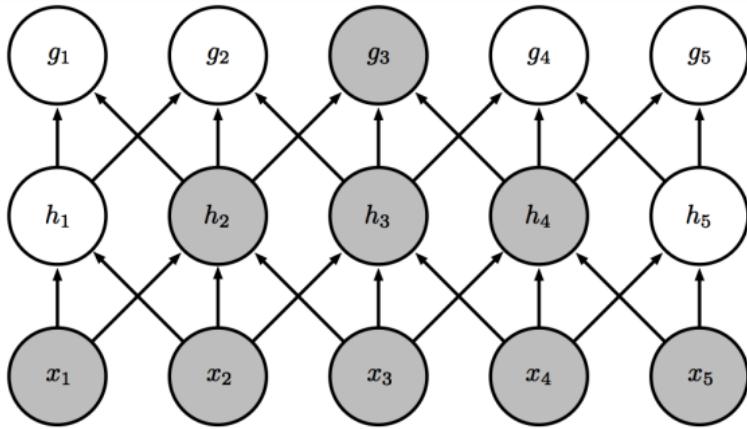


Dense
connections



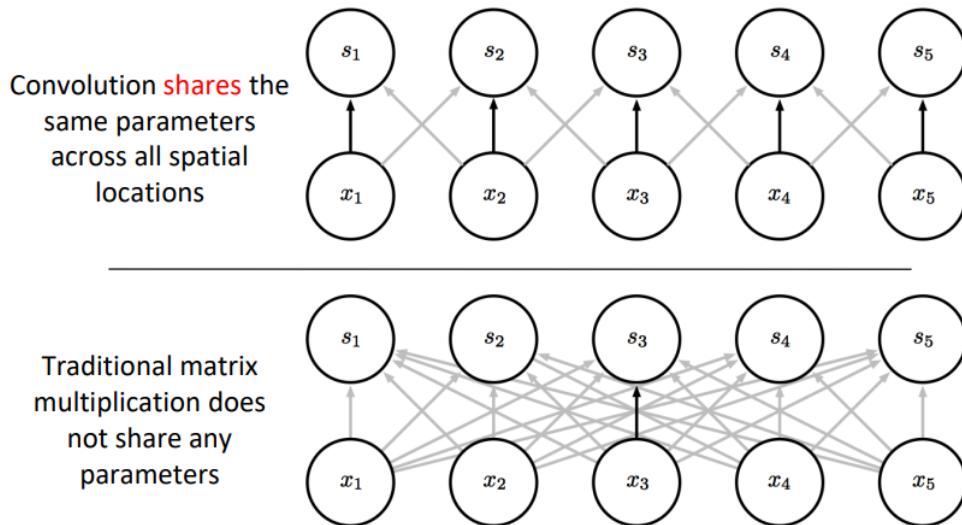
- 从 input 和 output 看，都只跟局部的信息有关

为什么要 deep?

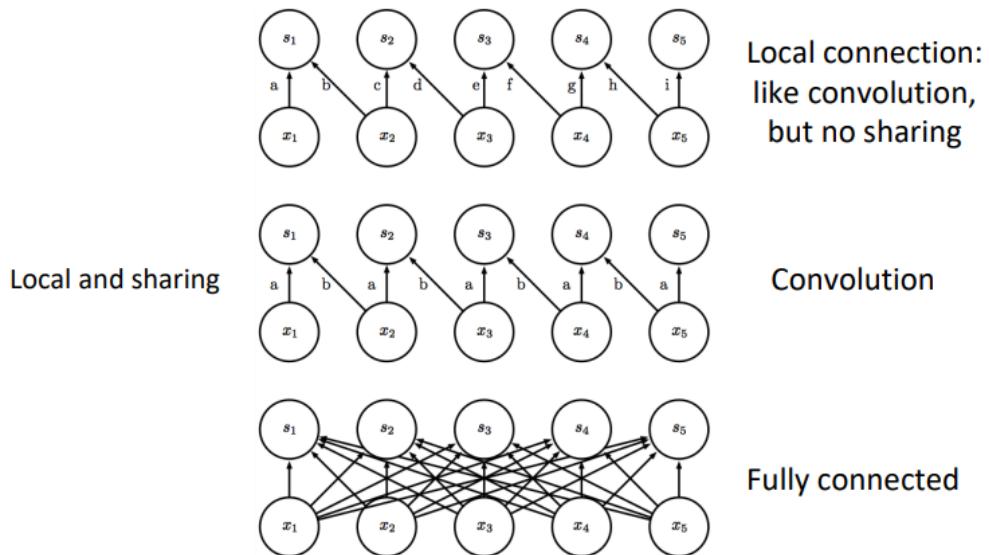


- 当网络层数够深的时候，更深层次的 layer 可以包含浅层的更多信息

参数共享

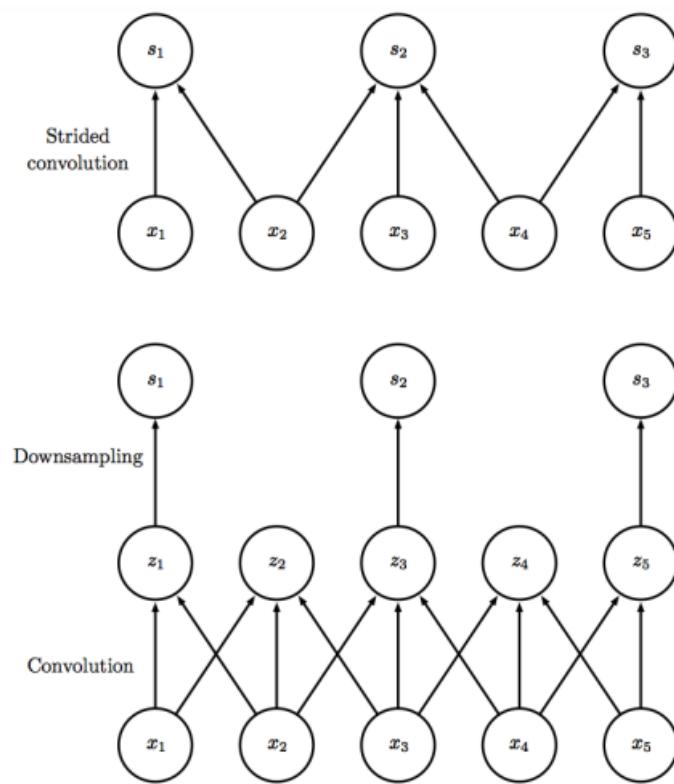


局部连接 vs 卷积



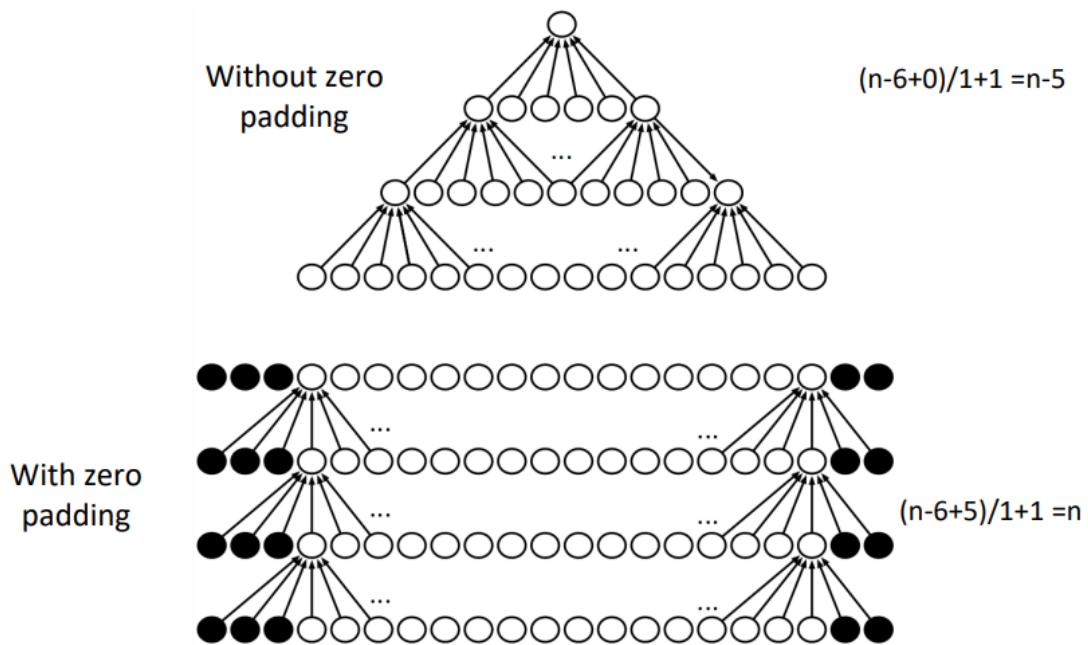
- 局部连接：参数不共享
- 卷积：参数共享且局部
- 全连接：参数过多

步长和池化



- 都可以起到降维的作用

为什么要 padding



$$(5-3+0)/1+1 = 3$$

Original

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

After 1st conv

4	3	4
2	4	3
2	3	4

$$(3-3+0)/1+1 = 1$$

After 2nd conv

18

- 没有 padding, 图像会收缩

$$(5-3+2)/1+1 = 5$$

Original

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

* =

0	0	1
0	1	1
1	1	1

After 1st conv

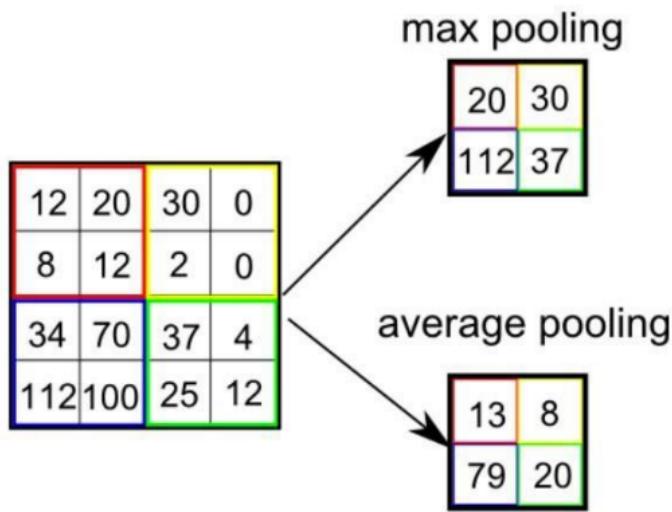
1	1	2	0	0
0	1	1	1	0
0	0	1	2	1
1	0	2	1	0
0	1	1	3	0

- 有 padding, 图像没有过分收缩

池化层 Pooling

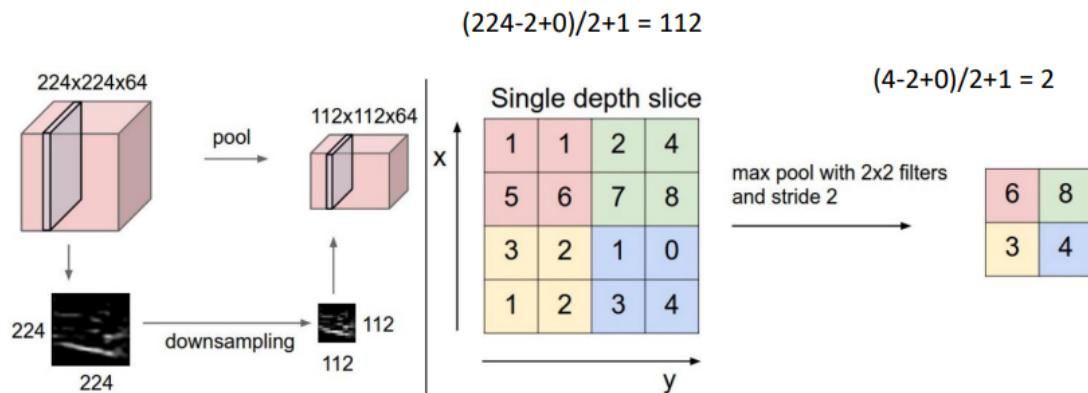
- 将多个值聚合为一个值
 - 减少层的大小 (加快计算速度)
 - 把最重要的信息留到下一层
 - 小变换的具有不变性和鲁棒性

池化的种类



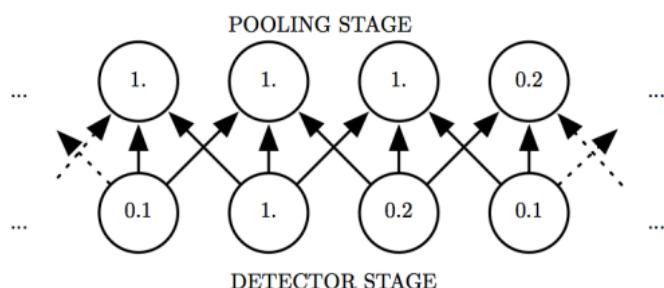
- 最大池化 Max Pooling: 找最大的值
- 平均池化 Average Pooling: 计算平均值

计算公式

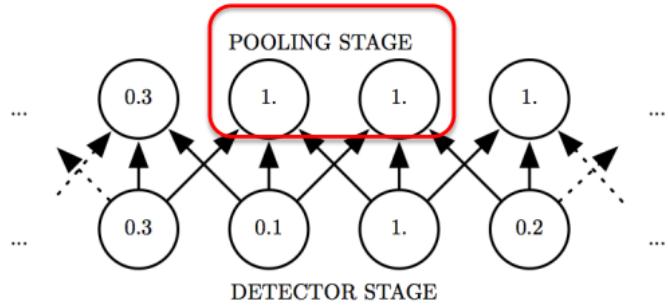


- $w_{out} = \frac{w - \text{kernel size}_w + 2 \times \text{padding}_w}{\text{stride}_w} + 1$
- $h_{out} = \frac{h - \text{kernel size}_h + 2 \times \text{padding}_h}{\text{stride}_h} + 1$
- $c_{out} = c_{in}$ (通道个数不变)

池化的鲁棒性



- 这是一个卷积层的中间部分
- 下面的节点的值是从非线性层输出的结果
- 上面的节点的值是最大池化输出的结果
- 步长 stride = 1, 池化核大小为 3 x 3



- 当下面的节点的输入右移了一个像素单位的时候，下面的节点的值每一个都移动了
- 但是上面的节点的值只有一半改变了
- 因为最大池化单元只对邻域的最大值敏感，而不是某个节点的确切位置

一定需要池化吗？

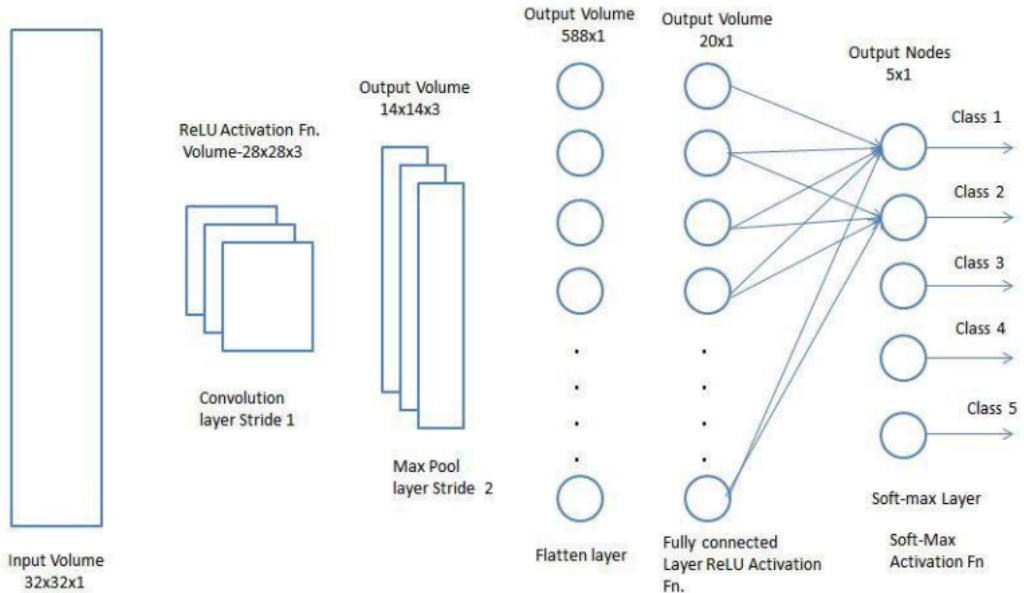
Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32×32 RGB image		
3×3 conv. 96 ReLU	3×3 conv. 96 ReLU	3×3 conv. 96 ReLU
3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU	3×3 conv. 96 ReLU
	3×3 max-pooling stride 2	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU	3×3 conv. 192 ReLU	3×3 conv. 192 ReLU
3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU	3×3 conv. 192 ReLU
	3×3 max-pooling stride 2	3×3 conv. 192 ReLU with stride $r = 2$

CIFAR-10 classification error

Model	Error (%)	# parameters
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	9.08%	≈ 1.4 M

- 我们知道，池化可以降维，增加卷积核的步长大小也会减少图像的大小
- 所以为什么不这样做而不是使用池化呢？
- 事实上，也可以！

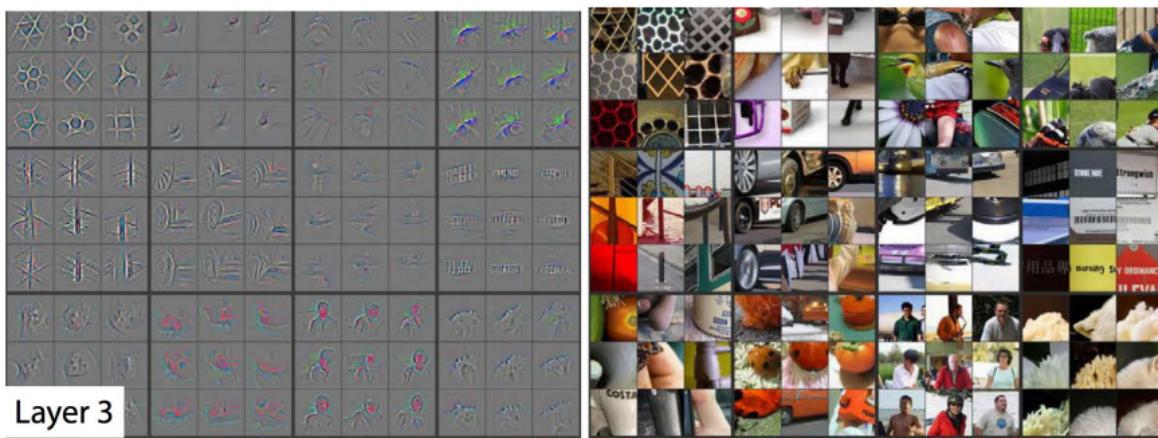
全连接层 -- 分类



- 展平最终输出，并将其输入到一个常规的用于分类的神经网络
- 使用 Softmax 分类技术对它们进行分类

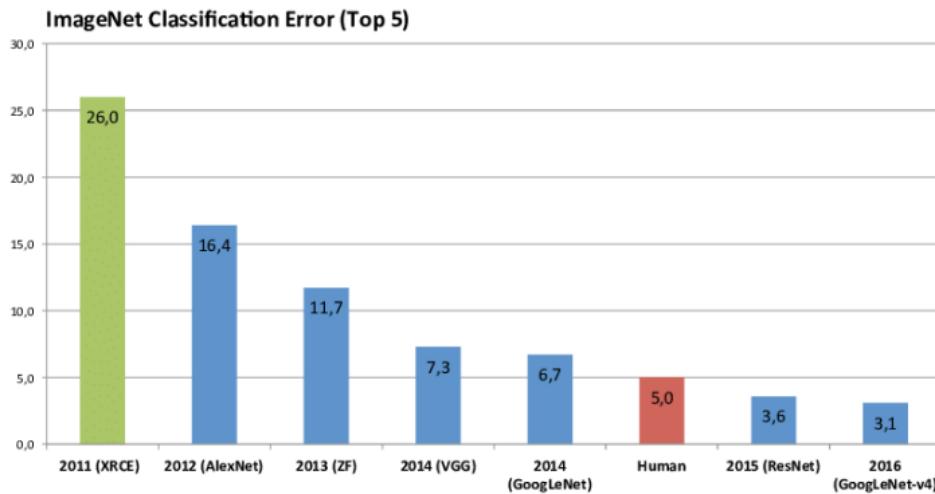
3. CNN 的相关介绍

CNN 的可视化



- 每一层的投影显示了网络中特征的分层性质
 - Layer2：响应角和其他边缘/颜色的结合
 - Layer3：更复杂的不变性，捕捉相似的纹理
 - Layer4：表现出显著的差异，并且更具有类型特异性
 - Layer5：显示具有显著姿态变化的整个物体

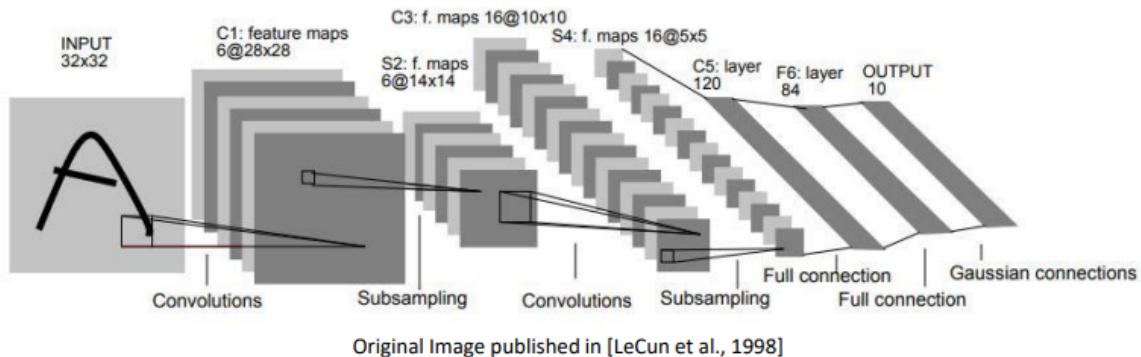
ImageNet 的挑战



- ImageNet 项目是一个大型的可视化数据库，设计用于可视化对象识别软件研究
- ImageNet 项目每年都会举办一项软件竞赛，即 ImageNet 大型视觉识别挑战赛 (ILSVRC)
- 规则：对 1000 个事物进行分类，只要分类的名称出现在最相似的 TOP-5，即算分类正确

LeNet-5

<https://engmrk.com/lenet-5-a-classic-cnn-architecture/>



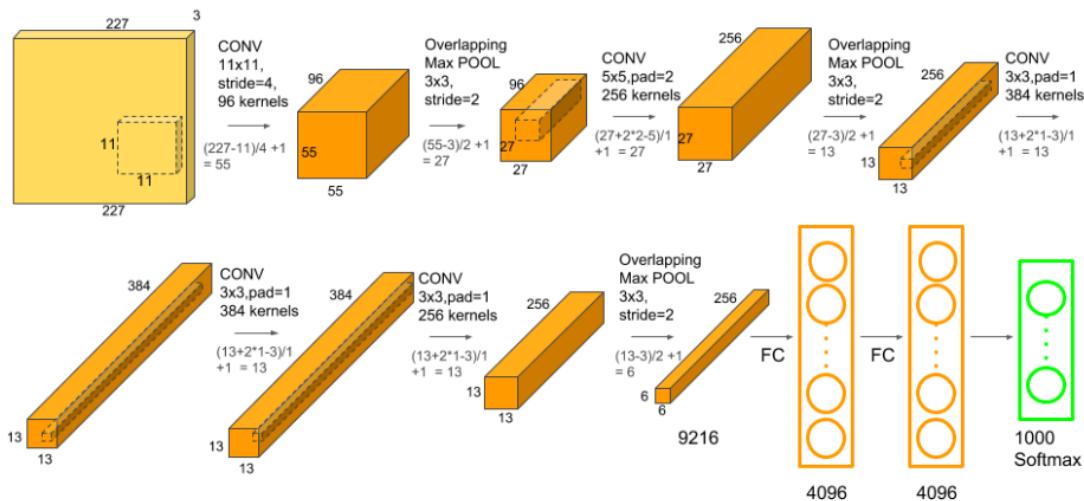
- 由 LeCun 等人在 1998 年提出
- 是 CNN 的鼻祖
- 用于识别手写支票上的数字(32 x 32)，但由于计算上的限制，无法放大成更大的图像

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

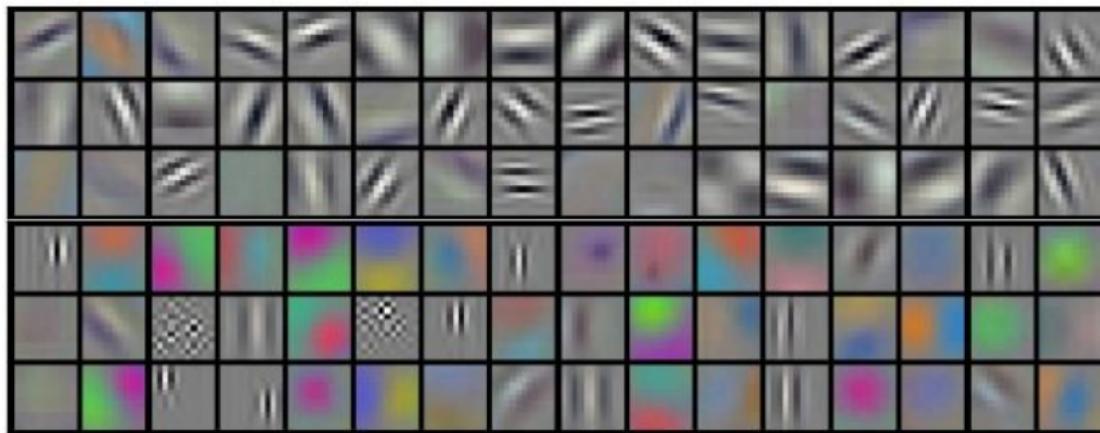
- 大约有 60,000 个参数
-

2012 - AlexNet

<https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>



- 超过所有的竞争者，并通过将 2011 年 TOP5 的错误从 26% 减少到 16.4% 赢得挑战
- 大约 6000 万个参数
- 使用了 ReLU 作为激活函数，而不是 LeNet-5 的 tanh
- 5 个卷积层（广义）+ 3 个全连接层
- 在两个 Nvidia Geforce GTX 580 GPU 上同时训练了6天：第 2 层、第 4 层和第 5 层的 kernel 只连接到同一 GPU 上的前一层的卷积核映射，第三层的内核连接到第二层的所有卷积映射



- 第一层 96 个卷积核的可视化
 - 上面 48 个在 GPU1 学习
 - 下面 48 个在 GPU2 学习

模型介绍

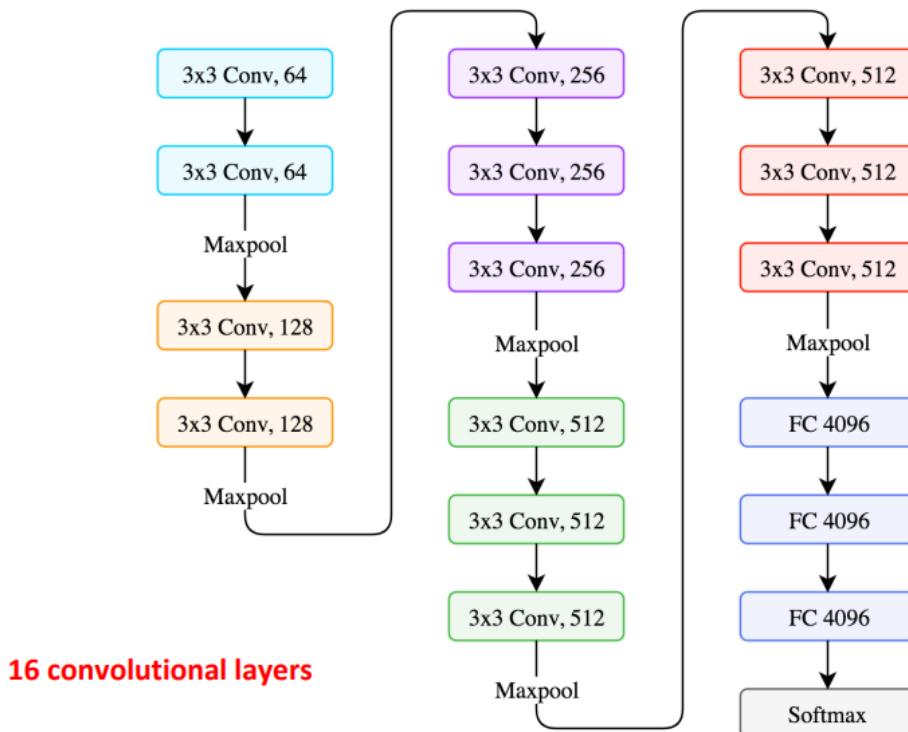
- 输入：227 x 227 x 3 （上图存在问题）
- 第一层卷积核：11 x 11 x 3 （一共 96 个） stride = 4
- 第一层卷积层输出：55 x 55 x 96
- 第一层池化核：3 x 3 x 96 stride = 2

- 第一层输出: $27 \times 27 \times 96$
- 第二层卷积核: $5 \times 5 \times 96$ (一共 256 个) padding = 2
- 第二层卷积层输出: $27 \times 27 \times 256$
- 第二层池化核: $3 \times 3 \times 265$ stride = 2
- 第二层输出: $13 \times 13 \times 265$
- 第三层卷积核: $3 \times 3 \times 265$ (一共 384 个) padding = 1
- 第三层卷积层输出: $13 \times 13 \times 384$
- 第四层卷积核: $3 \times 3 \times 384$ (一共 384 个) padding = 1
- 第四层卷积层输出: $13 \times 13 \times 384$
- 第五层卷积核: $3 \times 3 \times 384$ (一共 256 个) padding = 1
- 第五次卷积层输出: $13 \times 13 \times 256$
- 第五层池化核: $3 \times 3 \times 256$ stride = 2
- 第五层输出: $6 \times 6 \times 256 = 9216$ 个节点

2014 - VGG

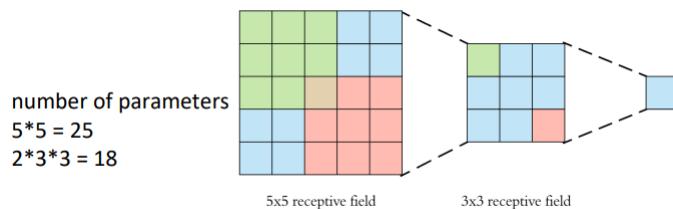
<https://arxiv.org/abs/1409.1556>

<https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>



- 2014 年的 ILSVRC 亚军，错误率为 7.3%，由牛津大学视觉几何组开发
- 16 个卷积层（广义）（大量的 3x3 滤波器）
- 大概 1.38 亿个参数
- 在 4 张 GPU 上训练了 3 周
- 简单但是有效（核心是网络深度的增加）

- 两个 3×3 的卷积实际上模仿了一个 5×5 的卷积的行为，但是要学习的参数更少了！



- 更多的 ReLU 操作产生更多的非线性，这意味着更强大的模型

2014 - GoogleNet

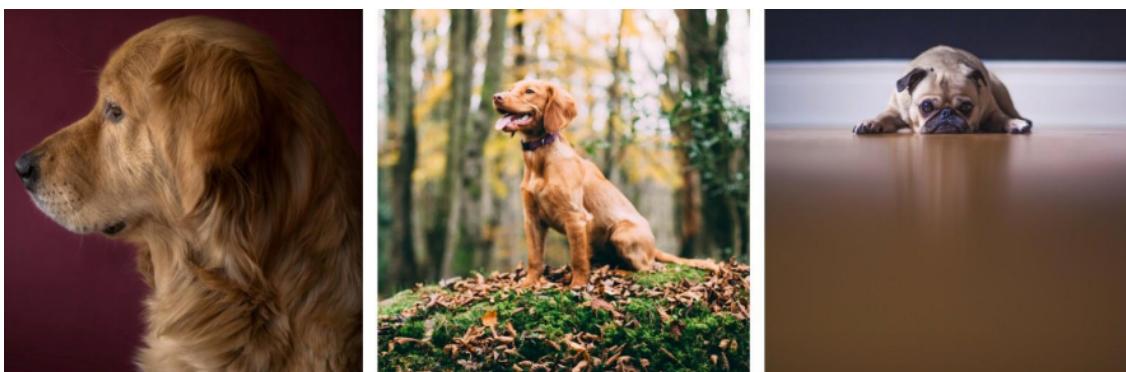
<https://arxiv.org/abs/1409.4842>

<https://arxiv.org/abs/1512.00567>

- 2014 年 ILSVRC 的获胜者实际上是 GoogleNet (由谷歌开发)，围绕 Inception 单元构建，错误率为 6.7%
- 非常接近人类水平(5%)
- 主要功能：Inception Unit, batch normalization, image distortions, RMSprop
- 参数：500 万 (初代版本) , 2300万 (三代版本)

Inception 单元想要解决的问题

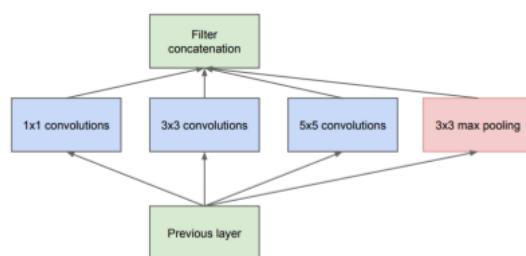
- 超深网络由于**参数过多**，容易出现**过拟合**现象，在整个网络中传递梯度更新也很困难
- 简单地叠加大的卷积运算在**计算上是昂贵的**：如果两个卷积层是链式的，那么它们的滤波器数量的任何均匀增加都会导致计算量的二次增加
- 图像中的**突出的，用于识别判断的部分在大小上有很大的变化**



上面三张图都是狗狗，但是表示狗狗的范围不一样

解决方案

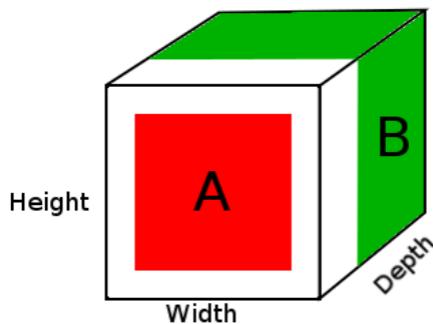
- 为什么不让**多个大小不一样的卷积核**在同一层上运行呢？
 - 使网络更宽，而不是更深，有多个大小的卷积核在同一水平上运行



(a) Inception module, naïve version

- 如何保证不同大小的卷积核卷出来的输出维度相同呢？

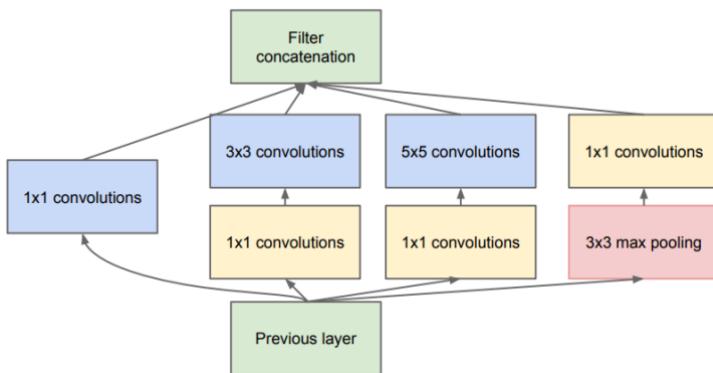
- 使用 padding 以确保 kernel 输出具有相同的大小，堆叠形成下一层



```
A = tensor of size (14, 14, 2)
B = tensor of size (16, 16, 3)
result = DepthConcat([A, B])
```

- 如何解决参数过多的问题？

- 使用 1×1 的卷积核来降通道数



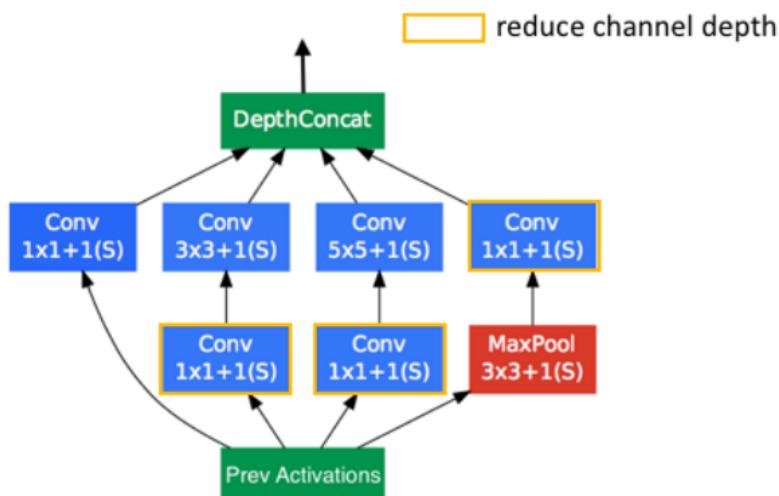
- 层数过深存在梯度消失的问题？

- 使用中间分类器

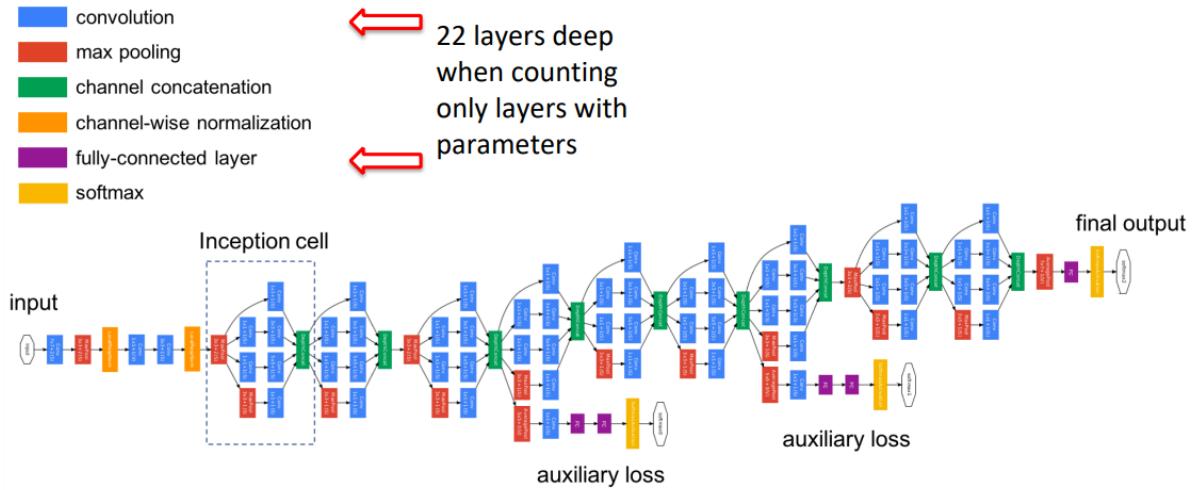
```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

模型介绍

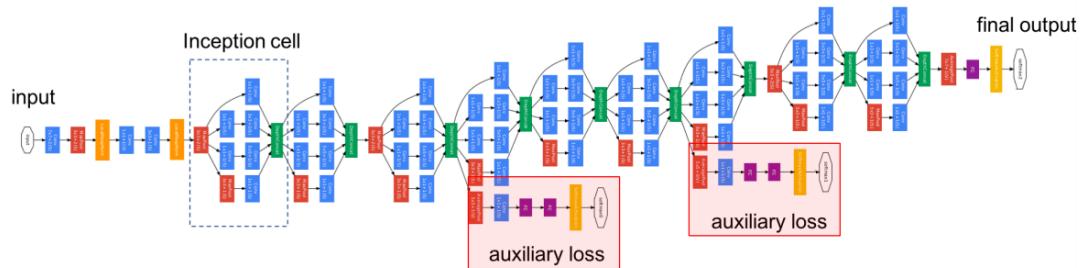
<https://www.jeremyjordan.me/convnet-architectures/>



- 这是一个广义卷积层的大体架构



- 这是模型架构
- 22 层深 (VGG为16) , 但只有 500 万个参数 (VGG为 1.38 亿个参数)
- 但是, 太深了, 无法避免消失的梯度问题
 - 许多层的参数 < 1 导致第一层的梯度非常小
 - 使用中间分类器

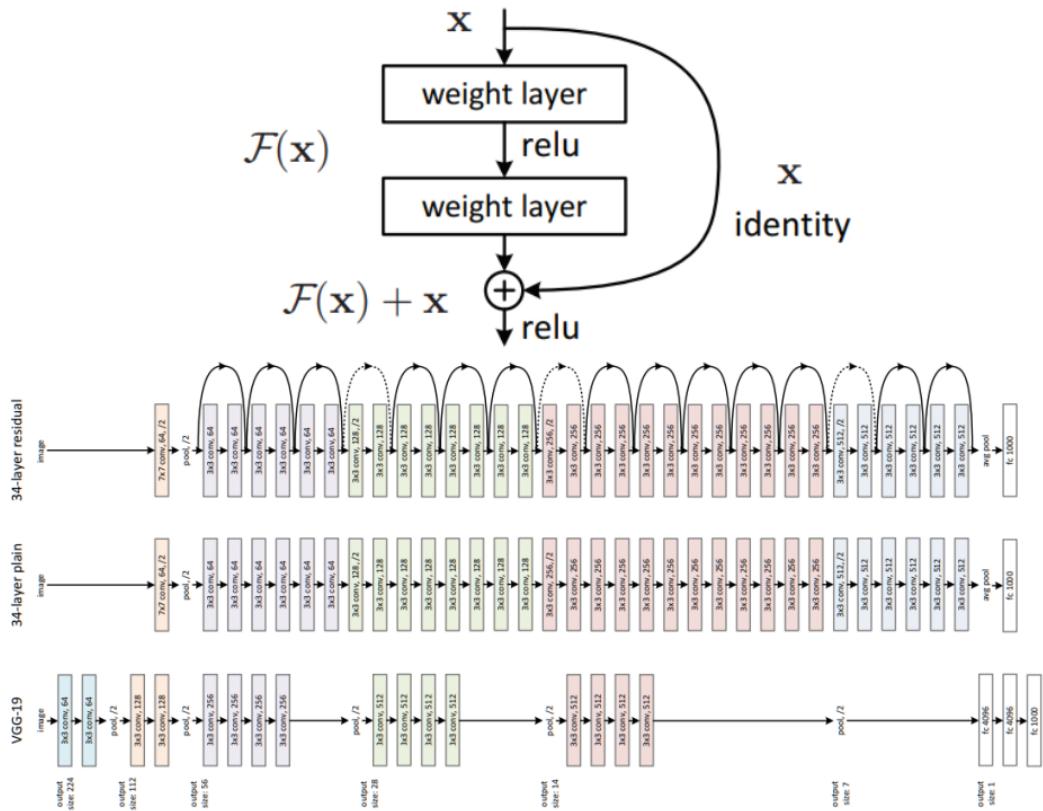


```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

type	patch size/stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

ResNet



神经网络在追求卓越性能的竞赛中已经走得更远，但随之而来的紧迫问题是消失和爆发梯度

- 梯度消失 Vanishing gradient
 - 梯度变得非常小，没有更新的权值，NN不学习
- 梯度爆炸 Exploding gradient
 - 权重的更新太过频繁，导致它们的价爆炸，变得比需要的大得多，导致训练失败

ResNet 在跨层之后，会使用 $F(x) + x$ 作为后面某一层的输入，以包含原来的输入，用来抵抗梯度爆炸和梯度消失