3 卷积神经网络

Convolutional Neural Networks

教学内容

1. Convolutional Nueral Networks
2. Convolutional layer
3. Pooling layer
4. CNN Architectures
5. 经典深度神经网络

1. 卷积神经网络

- 卷积神经网络（CNNs）来自对大脑视觉皮层的研究，在图像识别领域有很好的表现。
- CNN已经成功地在一些复杂的视觉任务上实现了超人的表现。它们为图像搜索服务，自动驾驶汽车，自动视频分类系统等提供动力。
- 他们也在其他任务中取得成功，例如语音识别或自然语言处理（NLP）。

本节中我们将讨论：

- CNN来自哪里？
- 它们的构建块是什么样的？
- 以及如何使用TensorFlow实现它们？
- 一些常见的CNN架构。

深度前馈网络

在前馈神经网络中，不同的神经元属于不同的层，每一层的神经元可以接受到前一层的神经元信号，并产生信号输出到下一层。

第0层叫做输入层，最后一层叫做输出层，中间的叫做隐藏层，整个网络中无反馈，信号从输入层到输出层单向传播，可用一个有向无环图表示。

前馈网络的目标是近似某个函数 f^* 。例如，对于分类器， $y = f^*(\mathrm{x})$

将输入 x 映射到一个类别 y。前馈网络定义了一个映射 $y = f(\mathrm{x};\theta)$ ，并且学习参数 θ 的值使它能够得到最佳的函数近似。

前馈型神经网络的学习主要采用误差修正法（如BP算法），计算过程一般比较慢，收敛速度也比较慢；

而反馈型神经网络主要采用Hebb学习规则，一般情况下计算的收敛速度很快。

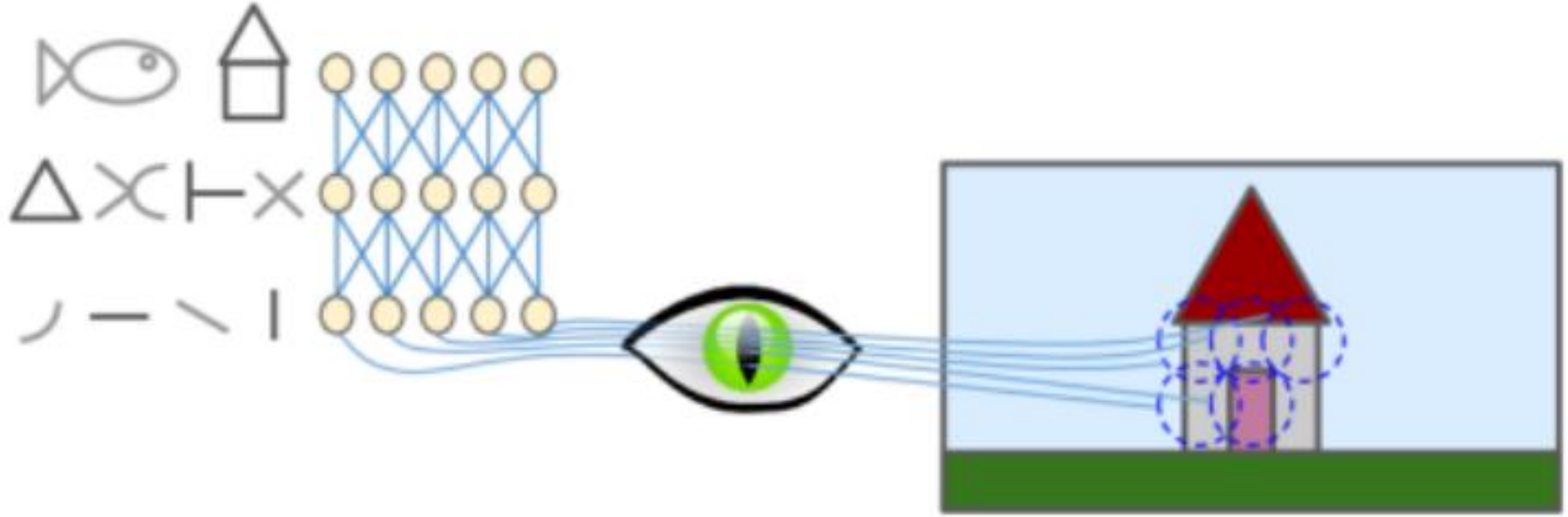
视觉皮层结构

- David H.Hubel 和 Torsten Wiesel 在 1958 年和 1959 年对动物的一系列实验，揭示了视觉皮层的结构（1981 年作者因此获得了诺贝尔生理和医学奖）.

- 视皮层中的许多神经元有一个小的局部感受区，它们只对位于视野中部分区域的视觉刺激起作用。

- 不同神经元的感受区可能重叠，并且它们一起平铺了整个视野。
- 此外，一些神经元只对水平线方向的图像作出反应，而另一些神经元只对其它方向的线作出反应。
- 他们还注意到一些神经元具有较大的感受区，并且它们对较复杂的模式作出反应，这些模式是较低层模式的组合。

这些观察结果让我们想到：更高级别的神经元是基于相邻低级神经元的输出。这个强大的结构能够检测视野中任何区域的各种复杂图案。

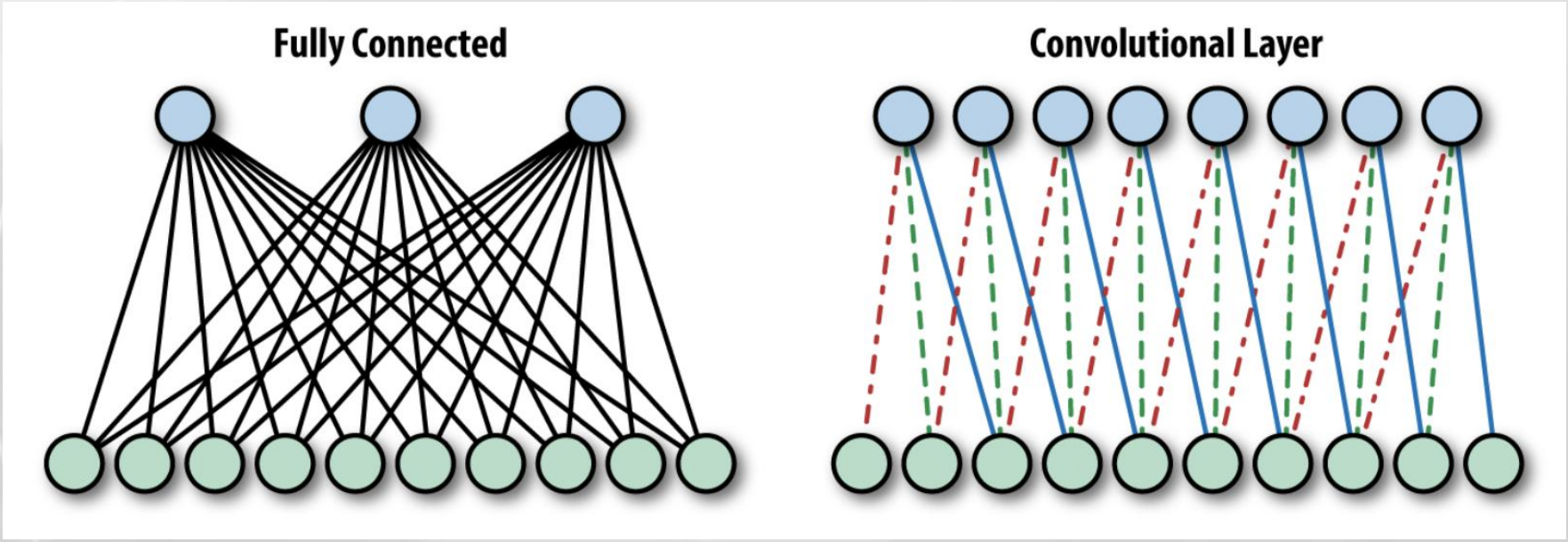


卷积神经网络

- 这些对视觉皮层的研究启发了 1980 年推出的新认知机（Neocognitron），后者逐渐演变为卷积神经网络（Convolutional Nueral Networks,CNN）。
- CNN的重要里程碑是 Yann LeCun等人于 1998 年发表的一篇论文，该论文引入了著名的 LeNet-5 架构，后来广泛用于识别手写支票号码。这个架构包括全连接层和 Sigmoid 激活函数，还引入了两个新的构建块：卷积层和池化层。

- 为什么不使用具有完全连接层的常规深度神经网络来进行图像识别？
- 图像需要大量参数，会因参数过多而使系统崩溃。例如，100×100图像有10,000像素，如果第一层只有1000个神经元，就会产生1000万个连接。
- CNN使用部分连接层解决了这个问题。

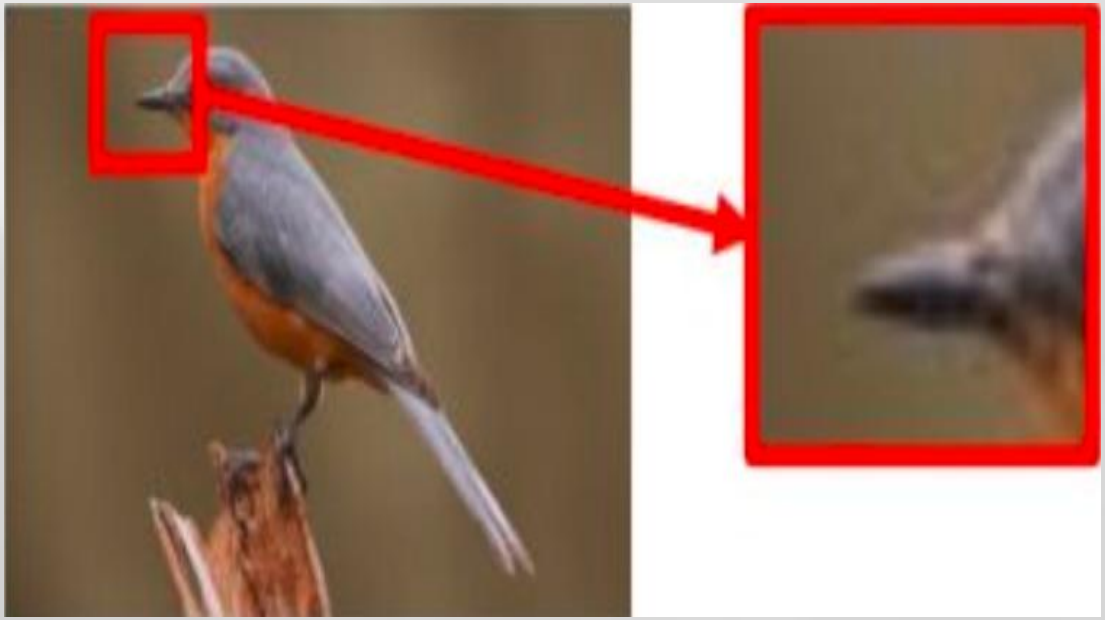
全连接神经网络与卷积神经网络



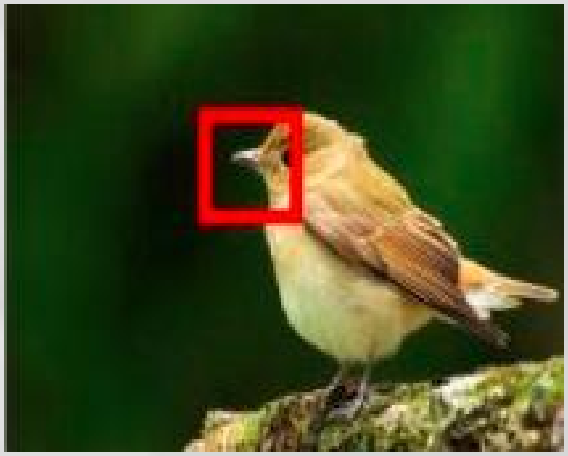
卷积神经网络原理

卷积神经网络的三个思想根源如下：

1、局部性



2、相同性

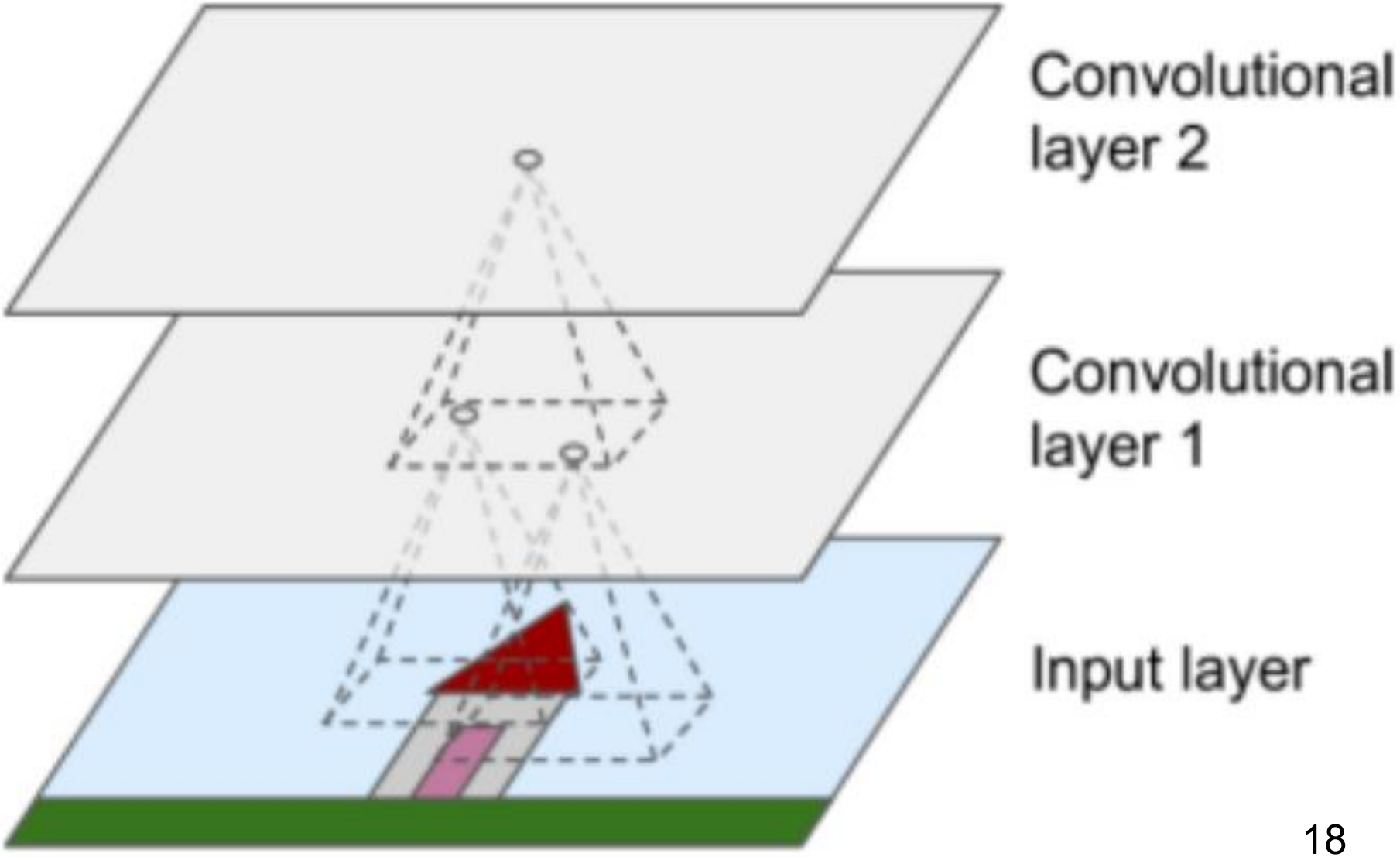


3、不变性

2. 卷积层

卷积层是CNN最重要的组成部分，其基本结构如下：

- 第一层中的神经元没有连接输入图像中的所有像素，而只是连接到它的感知区域中的像素。
- 第二层中的神经元仅连接到位于第一层中的感知区域内的神经元。


Convolutional layer 2
Convolutional layer 1
Input layer

- 第一个卷积层专注图像的的低级特征，然后将它们组装成下一个卷积层中的更高级别特征，依此类推。
- 这种分层结构在现实世界的图像中很常见，这也是CNN在图像识别方面表现良好的原因之一。

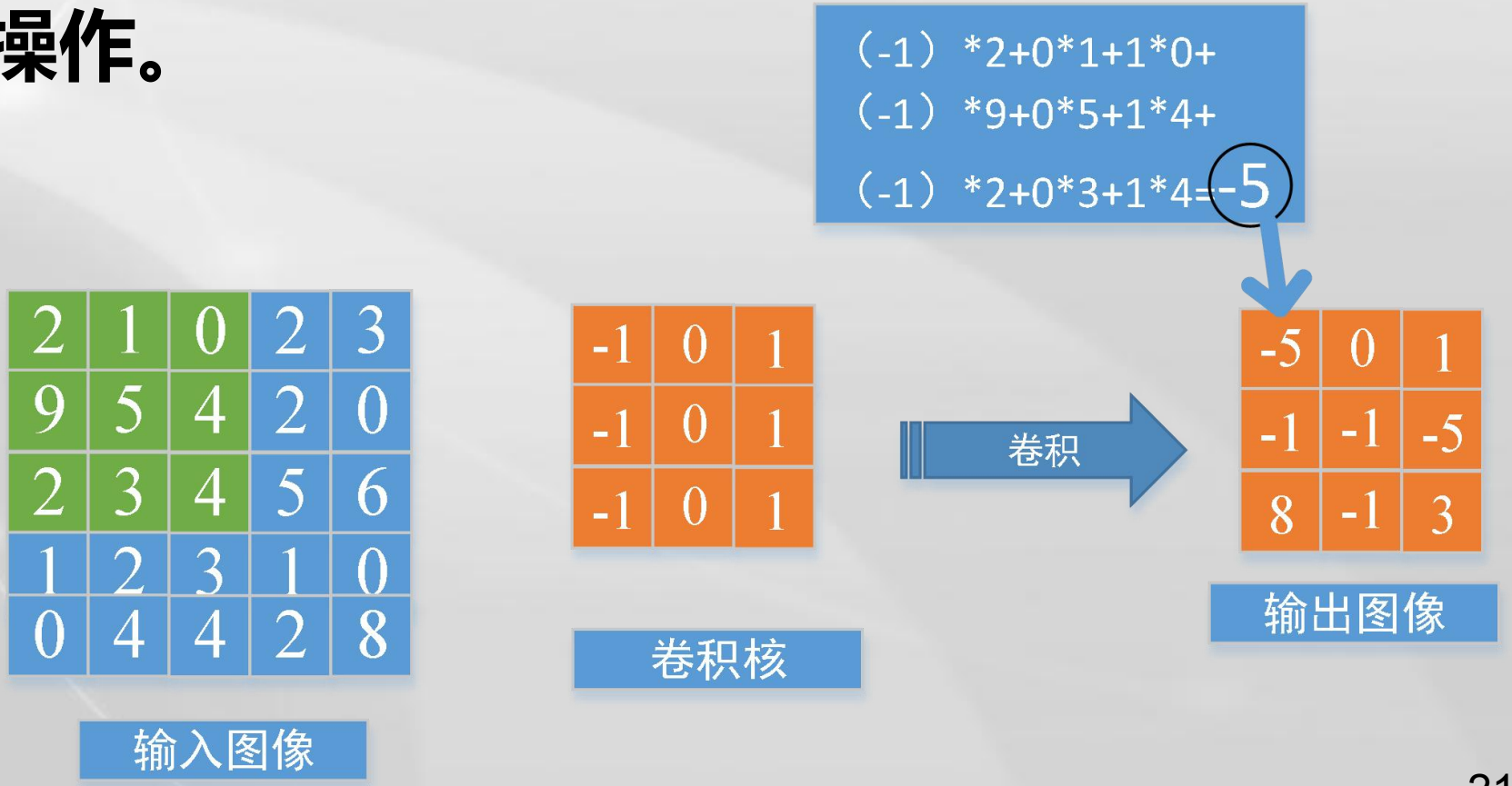
过滤器（Filters、卷积核）

当给定一张新图时，CNN并不能准确地知道这些特征到底要匹配原图的哪些部分，所以它会在原图中把每一个可能的位置都进行尝试，相当于把这个特征变成了一个过滤器（卷积核）。

这个用来匹配的过程就被称为卷积操作，这也是卷积神经网络名字的由来。

卷积操作

卷积就是做滤波器和输入图像的矩阵内积操作。



与全连接不同的是，每个神经元只与输入数据的一个局部区域连接，因此滤波器提取到的是图像的局部特征。

与神经元连接的区域大小，即**感受视野**的大小，或**滤波器**的宽和高，是需要通过人工来设置。

在卷积中要设定的参数：

1.滤波器的长宽高

2.步长

3.边界填充

滤波器长宽计算

输入图像尺寸为：$w_i \times h_i \times d_i$ ，步长设为s，边界填充的大小为p，滤波器尺寸为$f \times f$，输出图像尺寸为：$w_o \times h_o \times d_o$，其计算公式如下：

$$w_o = \frac{w_i - f + 2 \times p}{s} + 1$$

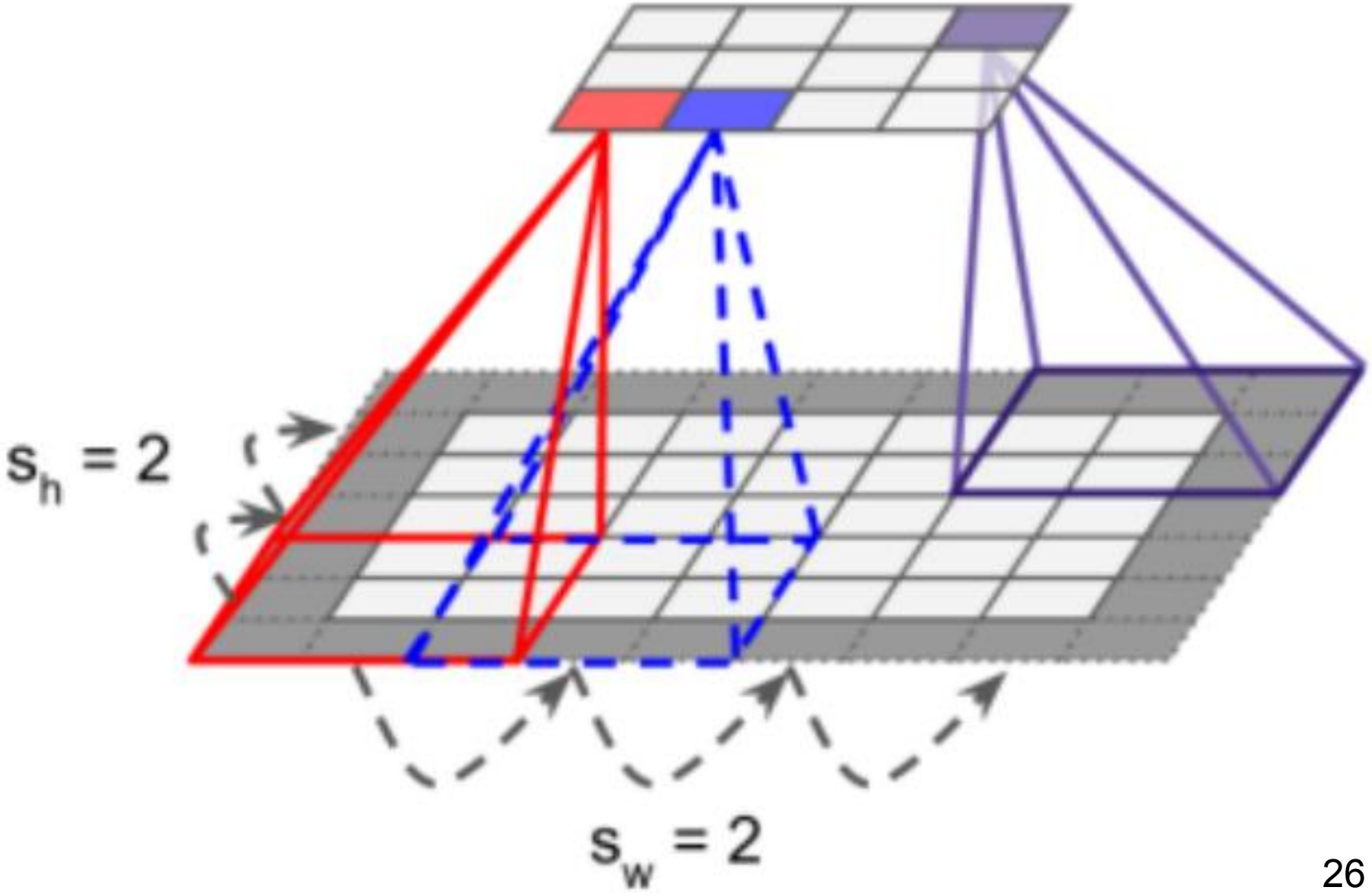
$$h_o = \frac{h_i - f + 2 \times p}{s} + 1$$

步长

- 为了减小输出层的大小，可以将感受区按大于1的距离隔开。两个连续的感受区之间的距离被称为步长（Stride）。

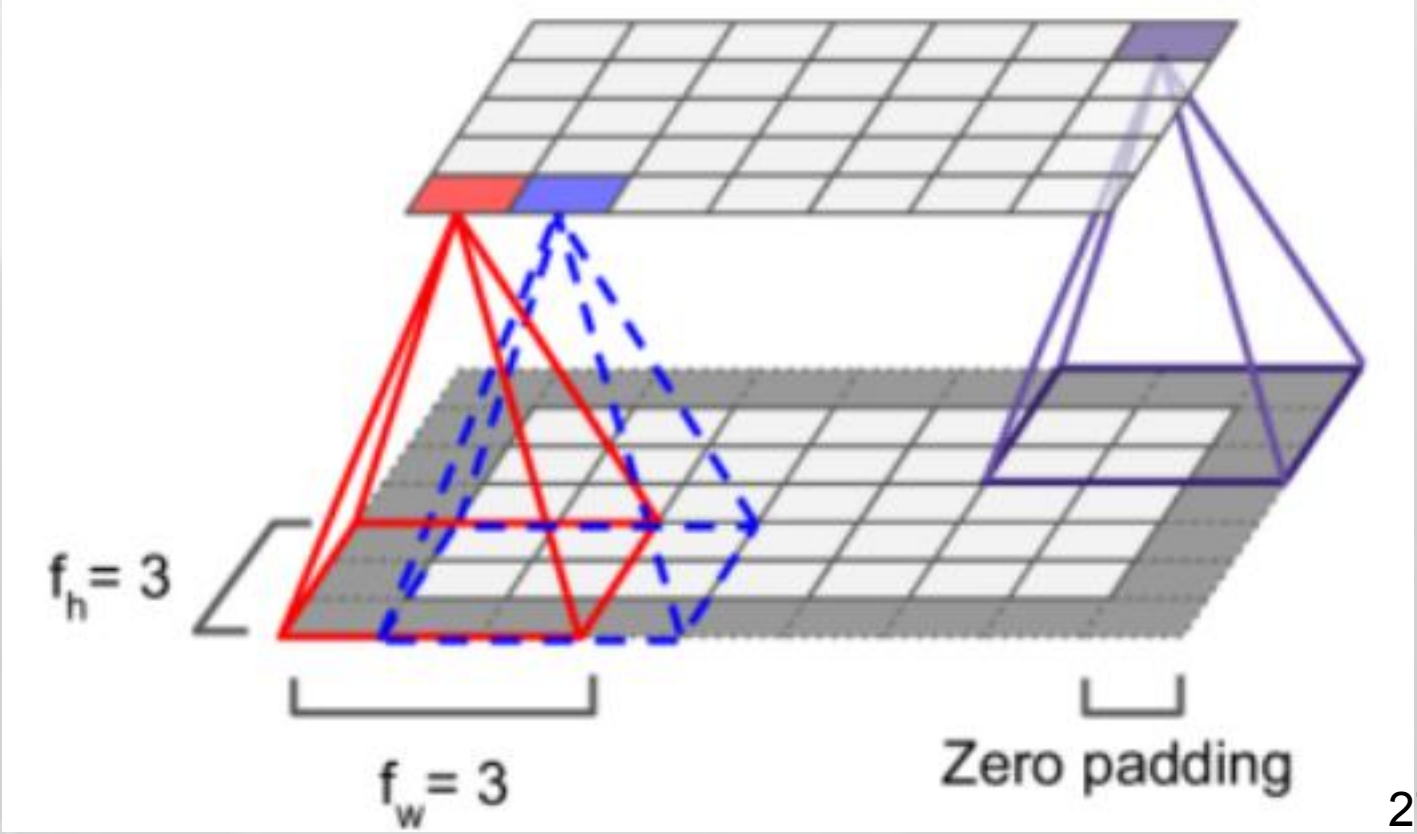
- 下图中，使用3×3滤波器，步长2，将一个5×7的输入层（需加零填充）连接到一个3×4的输出层。其中Sh和Sw是垂直和水平的步长。

3×3滤波器示例

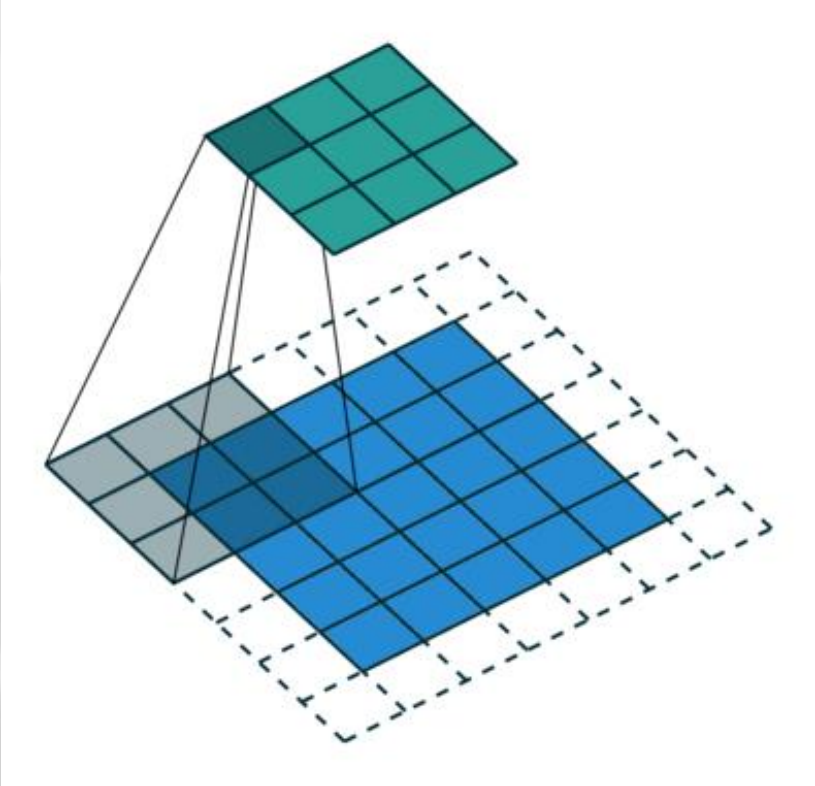


边界填充（零填充）

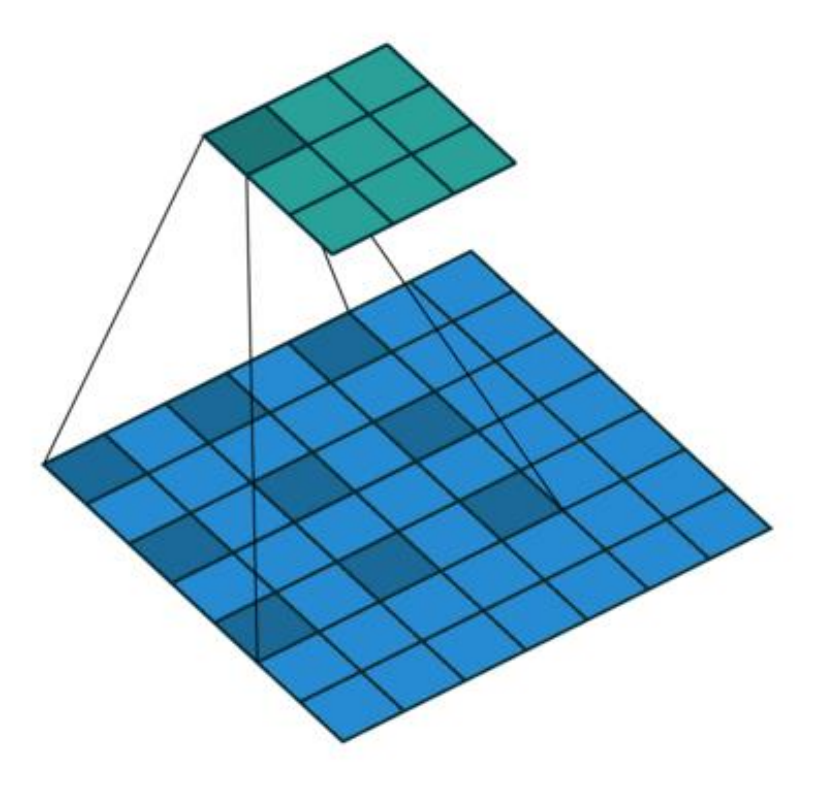
- 为了使输出图层具有特定的高度和宽度，通常在输入周围添加零，称为**边界填充**或**零填充**。



空洞卷积（Dilated Convolution）



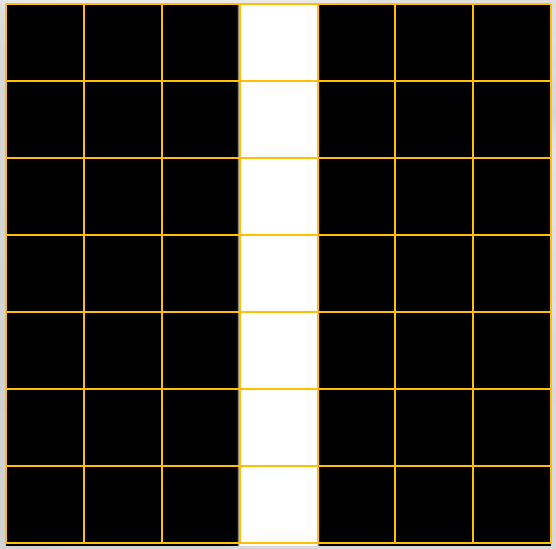
Standard Convolution

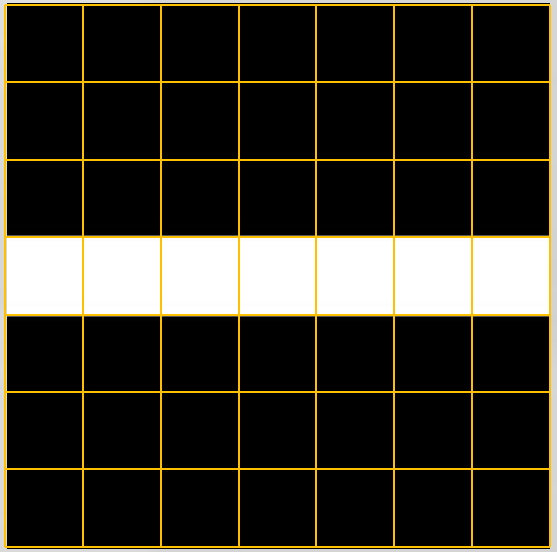


Dilated Convolution

过滤器示例

- 神经元的权重可以用大小为感受区的图像来表示，我们把它称为过滤器（或卷积内核）。例如，下图显示了两组不同的过滤器。
- 第一个是中间有一条垂直白线的黑色正方形，大小为7×7，黑色为0，白色为1。只感知中心白色垂直线。
- 第二个是中间有一条水平白线的黑色正方形。只感知中心白色水平线。



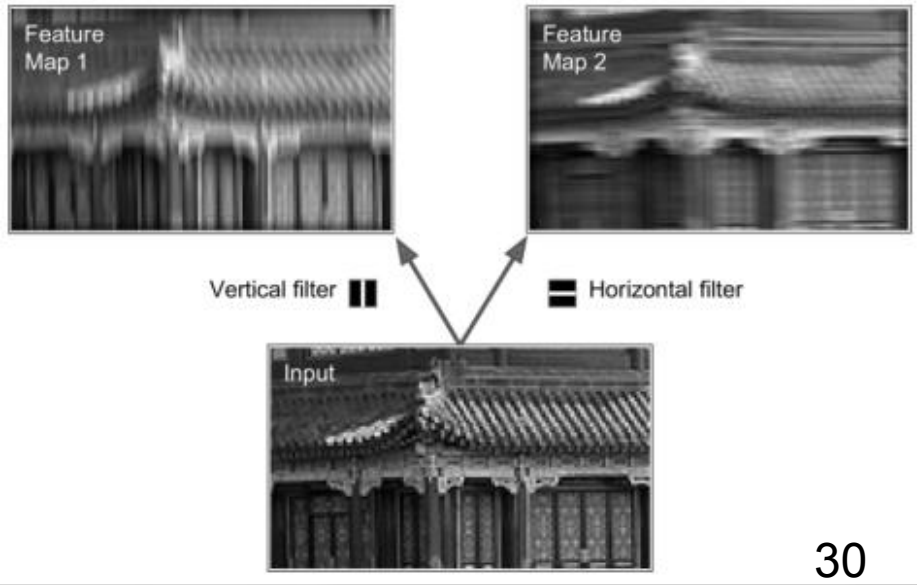


特征图

现在，将上面2个过滤器应用到同一个图像。

- 垂直线过滤器会使垂直白线会增强，而其余部分会变得模糊。
- 水平线过滤器使水平白线增强而其余部分模糊。

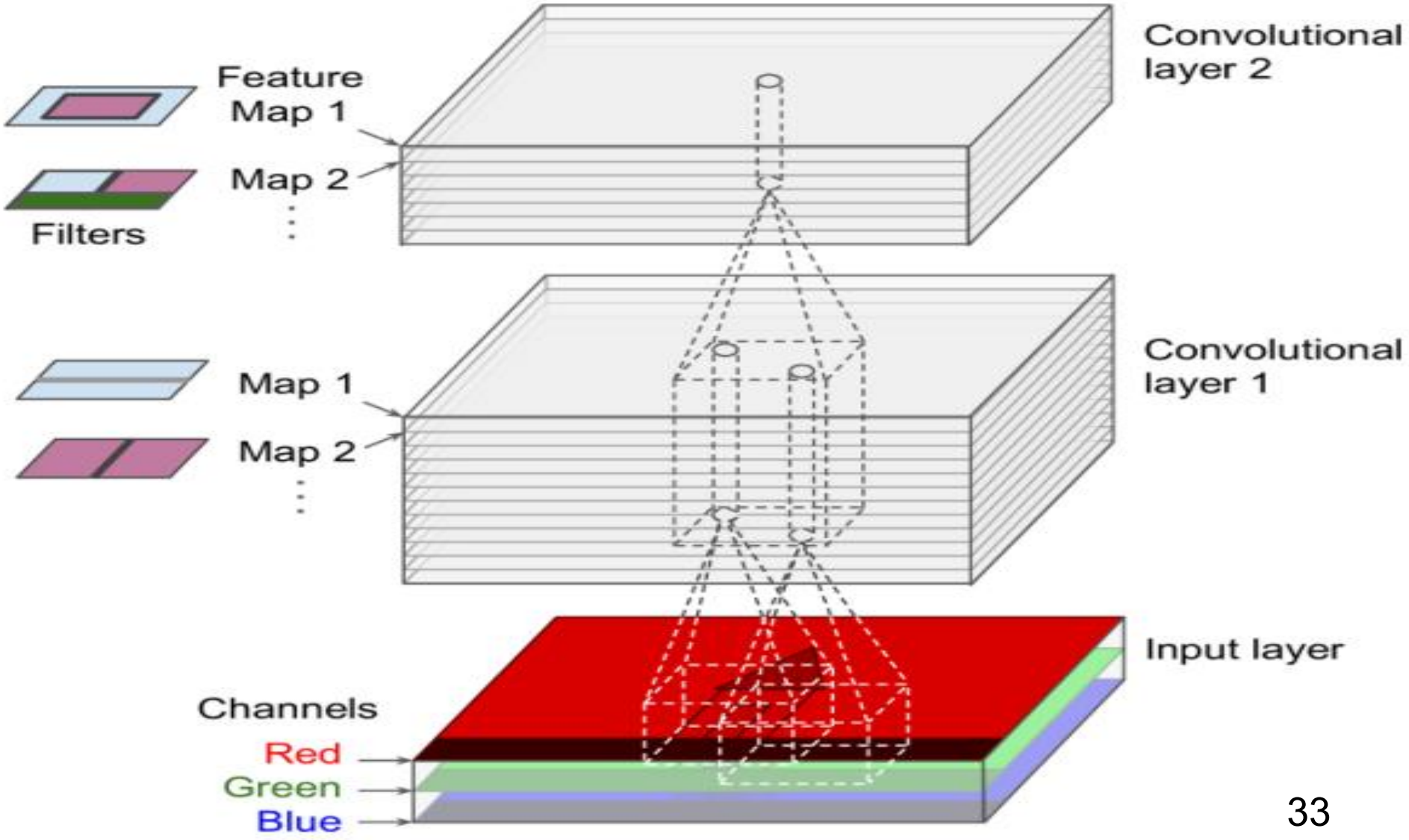
经过过滤器处理后的图像称为特征图（Feature Map）。



- 在一个特征图中，所有神经元共享相同的参数（权重和偏差项），但不同的特征图具有不同的参数。
- 特征图中的所有神经元共享相同的参数，这大大减少了模型中的参数数量。
- 一旦CNN学会识别一个位置的模式，它就可以在任何其他位置识别它。
- 常规DNN学会识别一个位置中的模式，它只可以在该特定位置识别它。

输入图像也由多个子层组成：每个颜色通道一个。

- 彩色图像有三层：红色，绿色和蓝色（RGB）
- 灰度图像只有一个通道：亮度（灰度）
- 但有些图像可能有更多层。例如，多光谱的卫星图像（如红外线）


Feature
Map 1
Map 2
Filters
Convolutional
layer 2
Map 1
Map 2
Convolutional
layer 1
Input layer
Channels
Red
Green
Blue

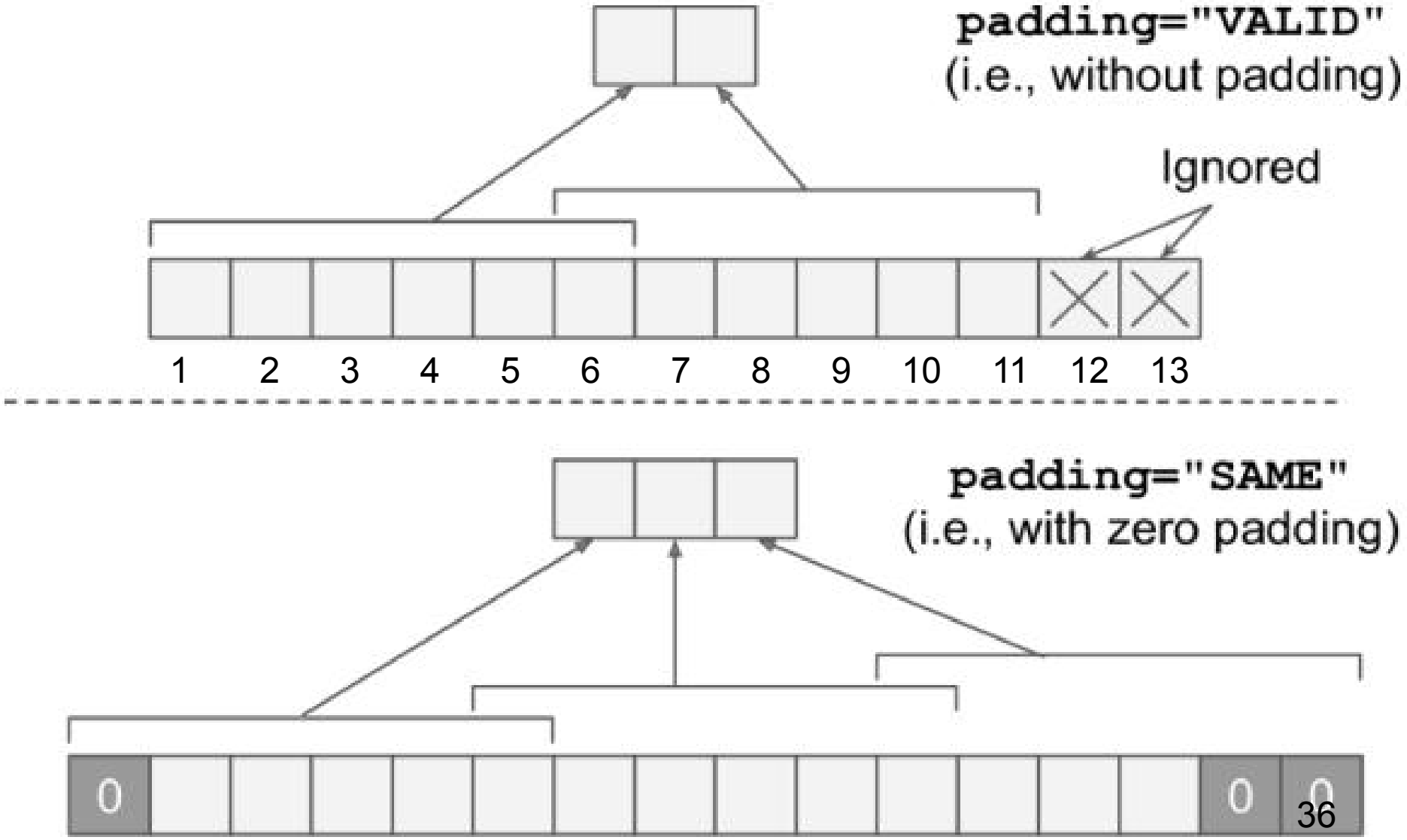
TensorFlow 实现

- 每个输入图像通常用[高度，宽度，通道]形状的3D张量表示。
- 加上批量以后变成[批量数，高度，宽度，通道]形状的4D张量。
- 卷积层过滤器用[fh,fw,fn′,fn]形状的4D张量表示。
  - fh为当前层的过滤器高度　fw为当前层的过滤器宽度
  - fn为当前层的过滤器数量　fn'为上一层的特征图数量
- 卷积层的偏置项用[fn]的1D张量表示.

Example 1

```
input = tf.random.normal([1,13,5,5])
filter = tf.random.normal([6,3,5,2])
op1 = tf.nn.conv2d( input, filter,
strides=[1, 5, 2, 1], padding='SAME')
print(op1.shape)
op2 = tf.nn.conv2d( input, filter,
strides=[1, 5, 2, 1], padding='VALID')
print(op2.shape)
(1, 3, 3, 2)
(1, 2, 2, 2)
```

填充方式


padding="VALID"
(i.e., without padding)
Ignored
1 2 3 4 5 6 7 8 9 10 11 12 13
padding="SAME"
(i.e., with zero padding)
0
0 0

Example 2

- 使用Scikit-Learn的load_sample_images（ ）加载两个样本图像。
- 然后创建两个7×7过滤器（一个在中间有一条垂直的白线，另一个有一条水平的白线），
- 使用TensorFlow的conv2d（ ）函数构建的卷积层将它们应用于两个图像（零填充、步长为2)。
- 绘制结果特征图。

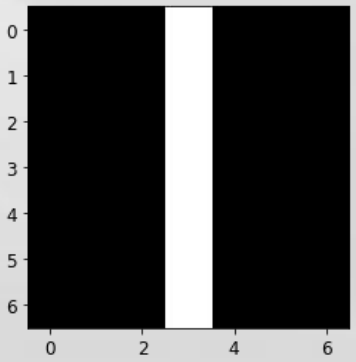
1）加载图像

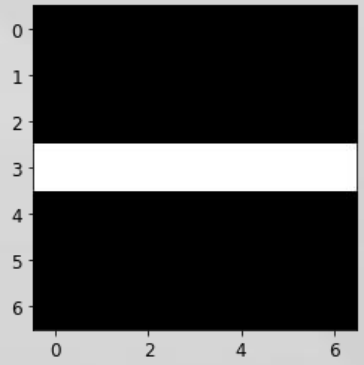
```
from sklearn.datasets import load_sample_image

china = load_sample_image("china.jpg") / 255
flower = load_sample_image("flower.jpg") / 255

images = np.array([china, flower])

batch_size, height, width, channels = images.shape
```

2）构造filters

```
filters = np.zeros(
shape=(7, 7, channels, 2),
dtype=np.float32
)

filters[:, 3, :, 0] = 1 # vertical line
filters[3, :, :, 1] = 1 # horizontal line
```

- plot_image(fmap[:, :, 0, 0])
- plt.show()
- plot_image(fmap[:, :, 0, 1])
- plt.show()

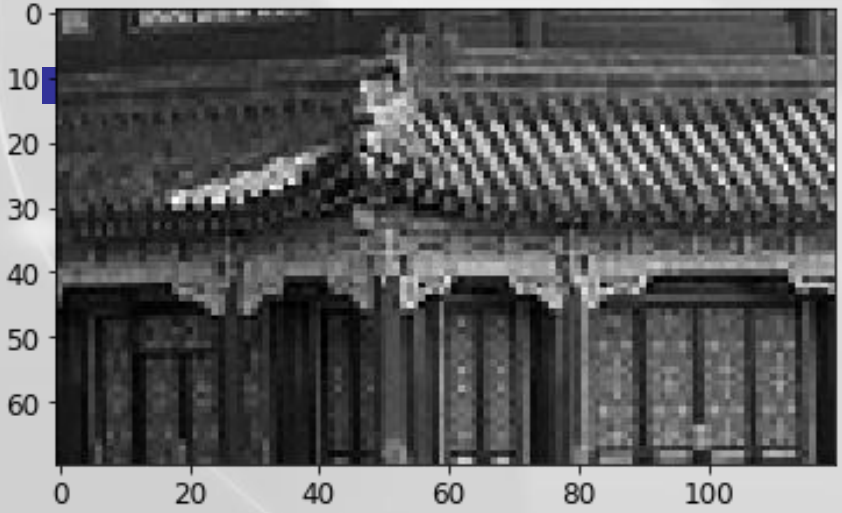


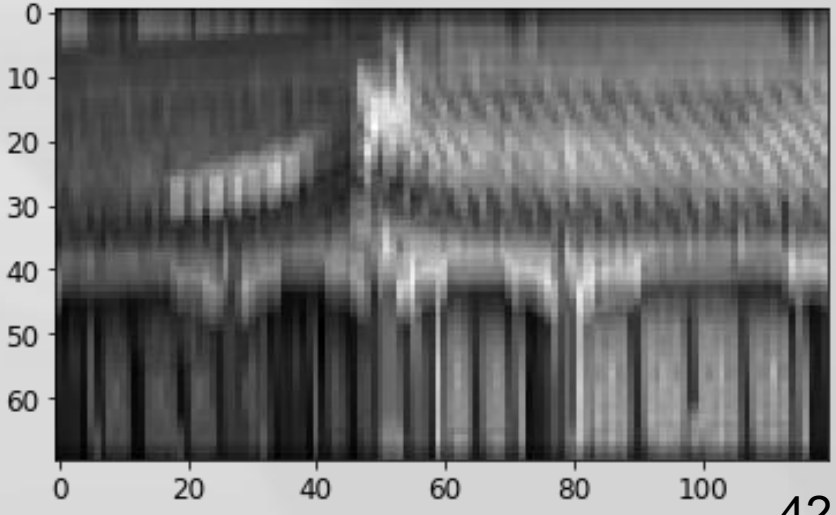


3) 二维卷积

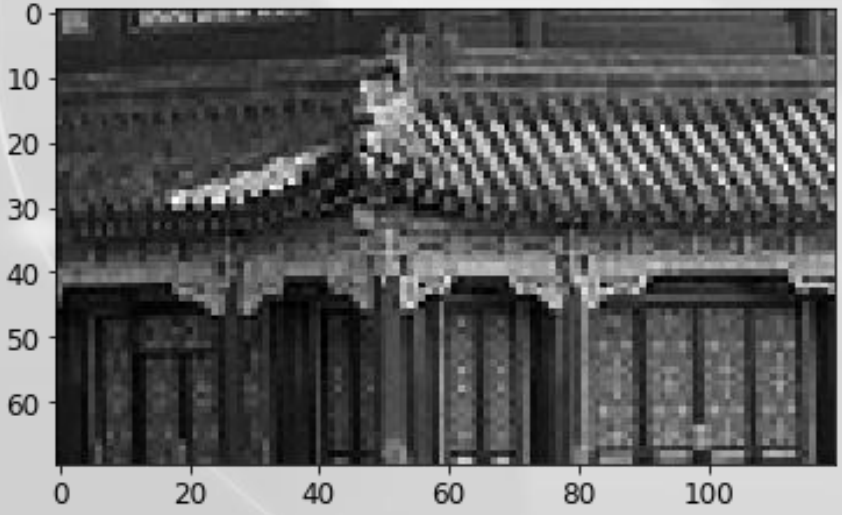
```
outputs = tf.nn.conv2d(
images,
filters,
strides=1,
padding="SAME"
)
```

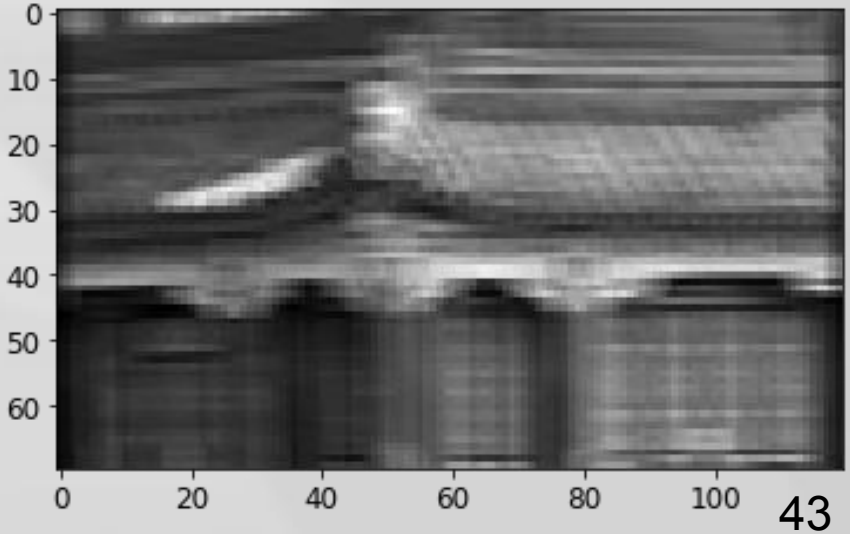
- plot_image(images[0, :, :, 0])
- plt.show()
- plot_image(outputs[0, :, :, 0])





- plot_image(outputs[0, :, :, 1])
- plt.show()





TensorFlow实现2维卷积

```
tf.nn.conv2d(
    input,
    filters,
    strides,
    padding,
    data_format='NHWC',
    dilations=None,
    name=None )
```

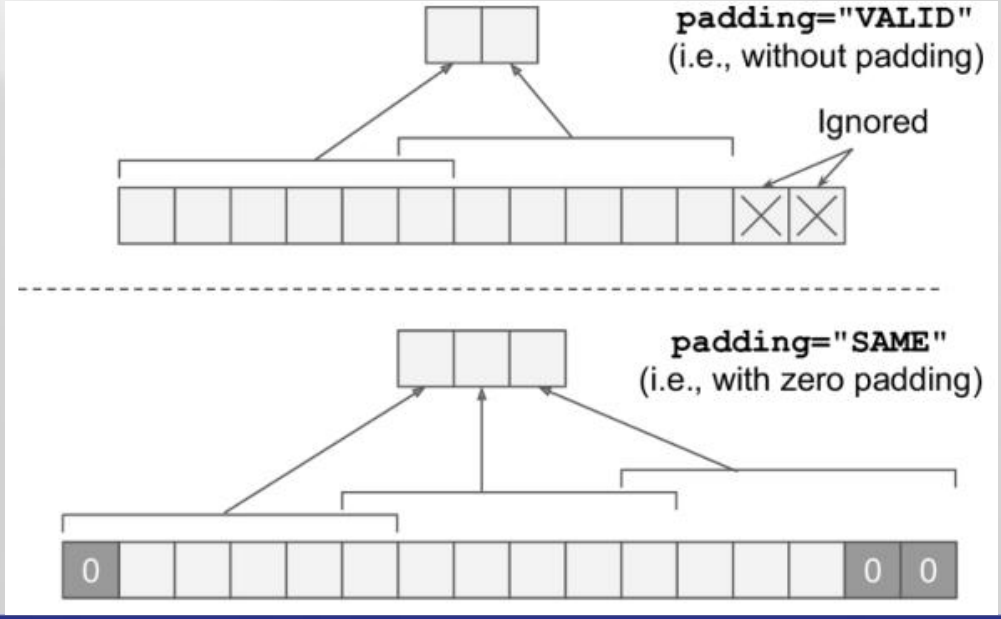
- input : [batch, height, width, in_channels]
- filter: [filter_height, filter_width, in_channels, out_channels]
- strides:
  - 1个整数
  - 包含1, 2 or 4 个整数的list

- data_format: "NHWC" or "NCHW"
- padding:
  - "SAME"卷积层在必要时使用零填充
  - "VALID"卷积层不使用零填充

output shape :

[batch, out_h, out_w, out_channels]

- out_h: 由输入高度、filter高度、stride高度决定
- out_w: 由输入宽度、filter宽度、stride宽度决定



Keras实现2维卷积

```
keras.layers.Conv2D(
    filters,# 滤波器的数量
    kernel_size,# kernel的高度和宽度
                # 1个或2个整数（tuple、list）
    strides=(1,1),
    padding='valid',
    data_format=None,
    …
)
```

- Input shape
  - data_format = ‘channels_first’

[batch, channels, height, width]

  - data_format = ‘channels_last’

[batch, height, width, channels]

- Output shape
  - data_format = ‘channels_first’

[batch, filters, new_height, new_width]

  - data_format = ‘channels_last’

[batch, new_height, new_width, filters]

Memory Requirements

CNN卷积层需要大量的RAM，特别是在训练期间，因为反向传播的反向传递需要在前向传递期间计算的所有中间值。

例如，一个具有5×5过滤器的卷积层，输出尺寸为150×100的200个特征图，带有步幅1和SAME填充。

如果输入是150×100 RGB图像（三个通道），那么参数的数量是(5×5×3+1)×200=15,200。其中+1对应于偏置项。

然而，200个特征图中的每一个都包含150×100个神经元，并且这些神经元中的每一个都需要计算其5×5×3=75个输入的加权和，总共有2.25亿次浮点乘法。

此外，如果使用32位浮点数表示特征图，那么卷积层的输出将占用200×150×100×32=96百万位（约11.4 MB）的内存。

如果训练批包含100个实例，则该层将占用超过1 GB的RAM！

在推理期间（即，在对新实例进行预测时），一旦计算出下一层，就可以释放一层占用的RAM，因此你只需要两个连续层所需的RAM。

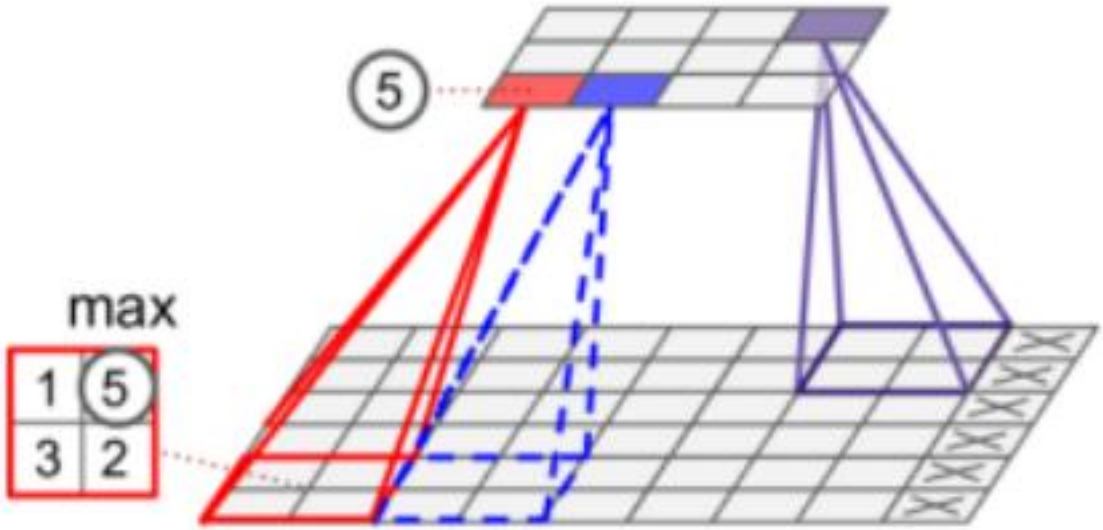
3. 池化层

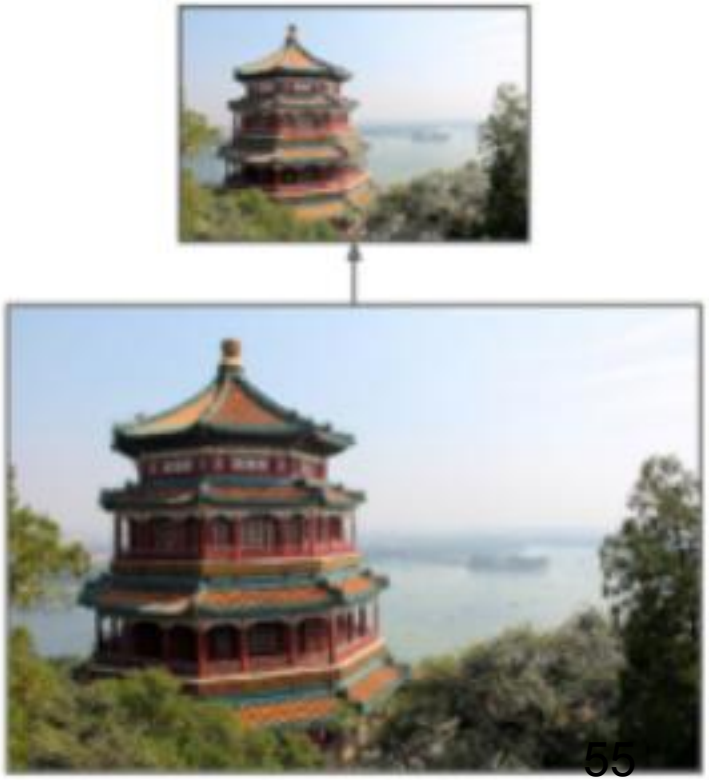
池化层的目的是对输入图像进行二次采样（即缩小），以减少计算量、内存占用和参数数量（减少过拟合风险）。

- 和卷积层一样，池化层中的每个神经元都和前一层神经元的输出相连，位于一个小的矩形感受区内。
- 然而，池化神经元没有权重，它所做的就是使用聚合函数（如max_pool()或avg_pool()）聚合输入。

Max池化层示例

这显然是一种非常具有破坏性的层：即使使用2×2内核和步幅 2 ，两个方向的输出也会小两倍（因此它的面积将减小四倍）。





- 池化层通常独立地在每个输入通道（层）上工作，因此输出深度（层数）与输入深度（层数）相同。你也可以在深度维度上进行池化，图像的空间维度（高度和宽度）保持不变，但通道数量会减少。

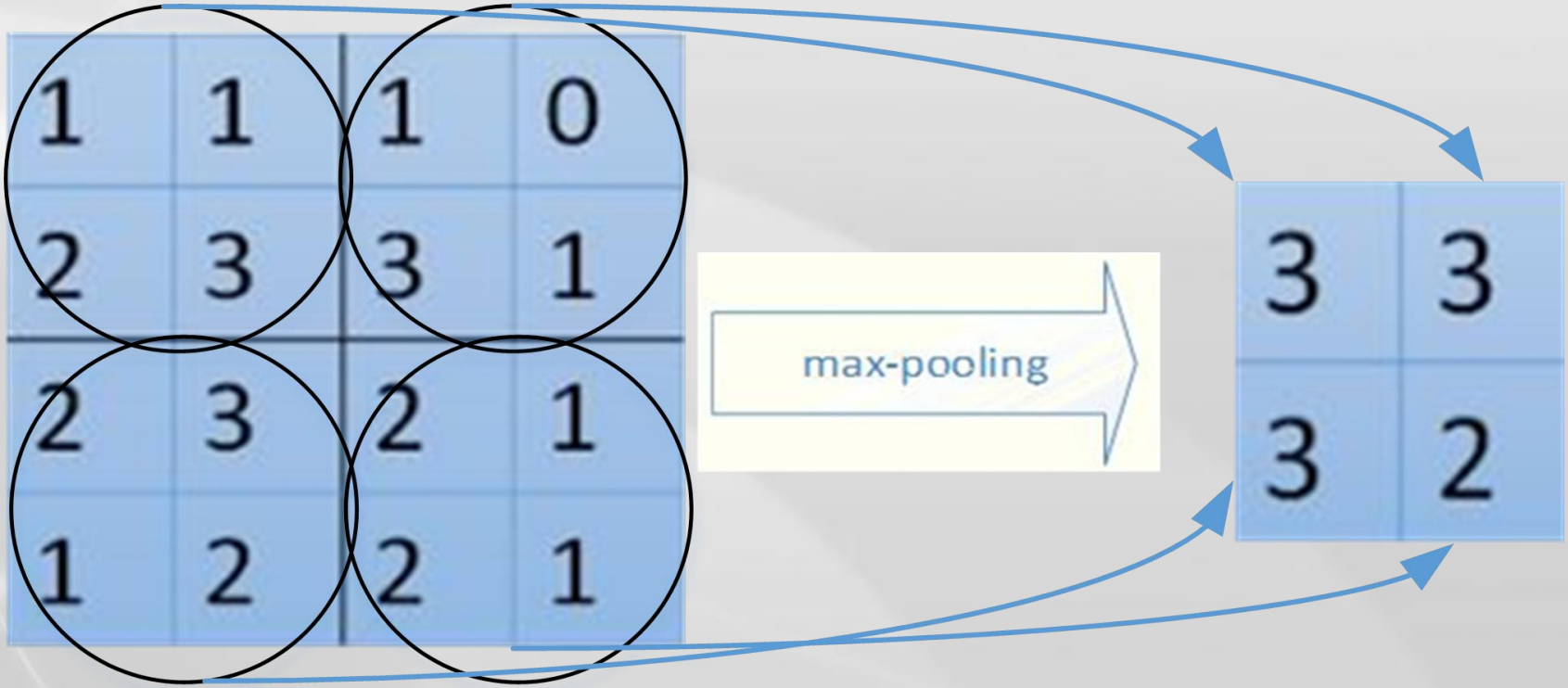
池化层的作用

1.特征降维，避免过拟合

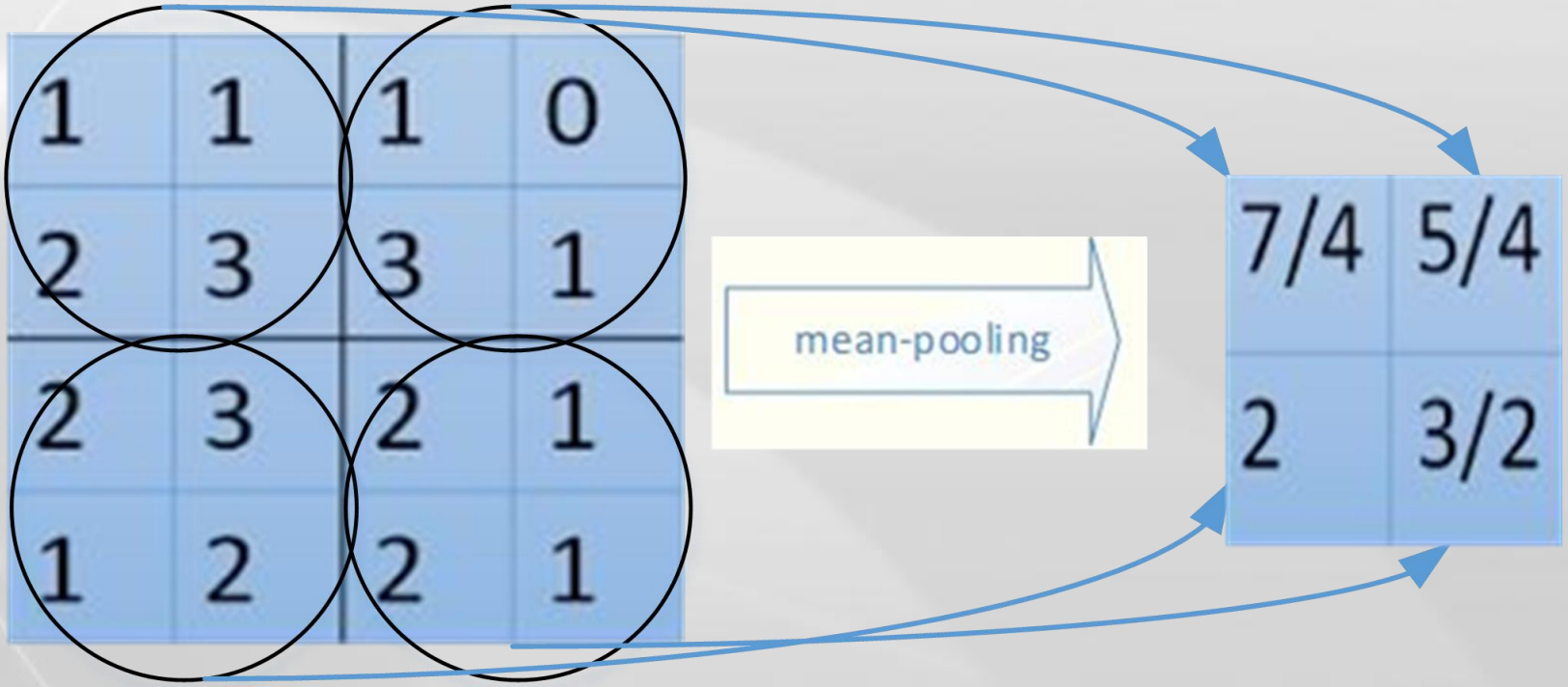
2.空间不变性

3.减少参数，降低训练难度

最大池化



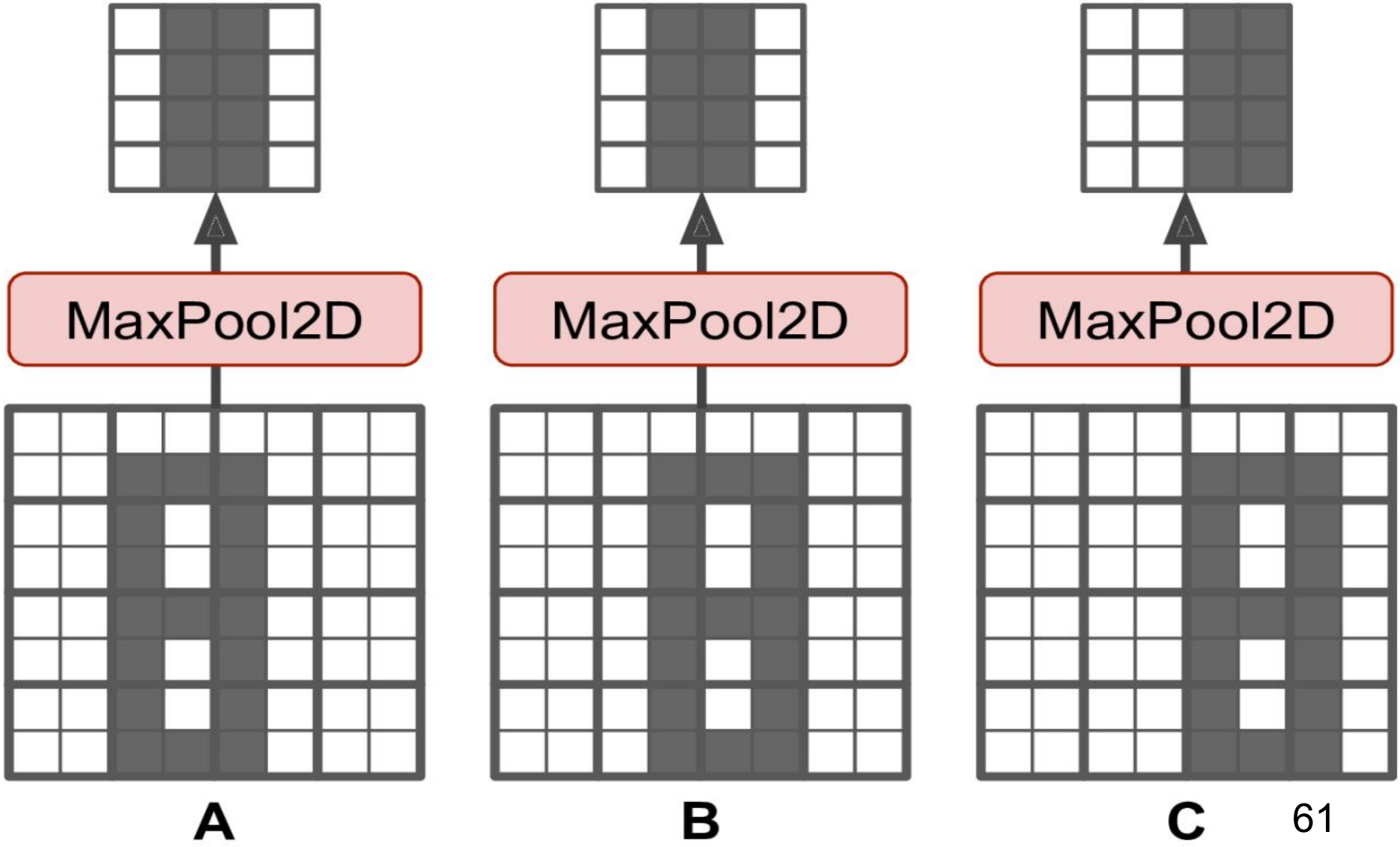
平均池化



池化操作示例

```
batch_size, height, width, channels = dataset.shape
filters = np.zeros(shape=(7, 7, channels, 2),
                   dtype=np.float32)
filters[:, 3, :, 0] = 1  # vertical line
filters[3, :, :, 1] = 1  # horizontal line
max_pool=tf.nn.max_pool(dataset,ksize=[1,2,2,1],
strides=[1,2,2,1],padding="VALID")
plt.imshow(tf.cast(max_pool[0],np.uint8))
plt.show()
```

最大池化的不变性



Keras实现

- 最大池化

keras.layers.MaxPooling2D

keras.layers.MaxPool2D

- keras.layers.MaxPooling2D(

pool_size=(2,2),

strides=None, #默认值和pool_size相等

padding='valid', #'valid' or 'same'

data_format=None)

平均池化

keras.layers.AveragePooling2D
keras.layers.AvgPool2D

```
keras.layers.AveragePooling2D(
pool_size=(2,2),#int or (height, width)
strides= None, # 默认值和pool_size相等
padding='valid',# 'valid' or 'same'
data_format=None)
```

全局均值池化

GlobalAveragePooling2D
GlobalAvgPool2D

keras.layers.GlobalAveragePooling2D(
data_format=None)

- input shape: 4D tensor
- output shape: (batch, channels)

TensorFlow实现池化操作

tf.nn.max_pool2d(

- input,
- ksize,#an int or list of 1,2,or4
- strides,
- padding,
- data_format='NHWC',
- name=None)

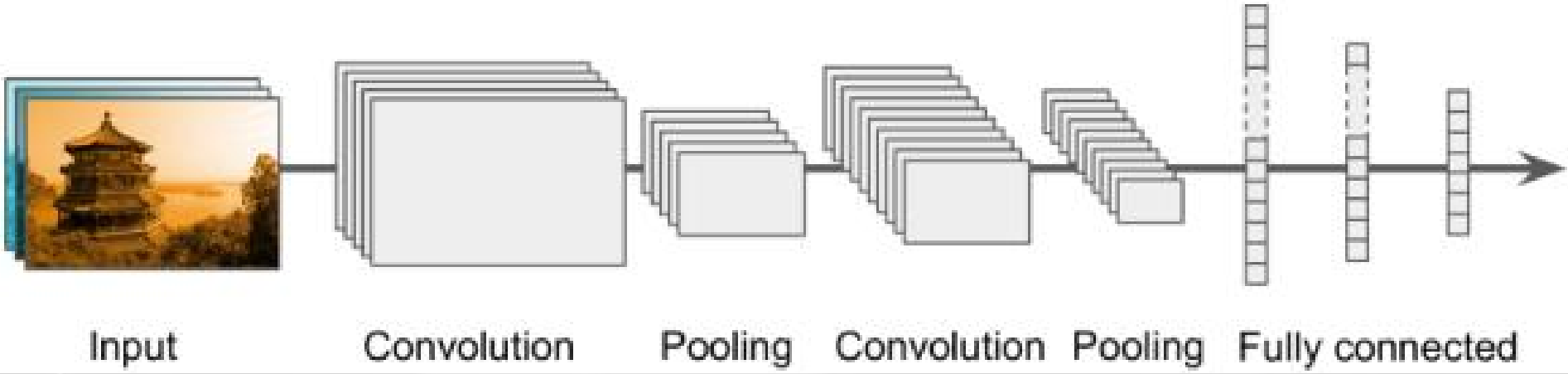
avg_pool2d(

- input,
- ksize,#an int or list of 1,2,or4
- strides,
- padding,
- data_format='NHWC',
- name=None)

4. CNN Architectures

典型的CNN架构

- 卷积层+ReLU层
- 池化层
- 卷积层+ ReLU层
- 池化层
- …
- 若干全连接层
- 输出层


Input
Convolution
Pooling
Convolution
Pooling
Fully connected

练习

设计一个对MNIST处理的CNN网络，网络结构和参数要求如下表，其中每一次MaxPooling后都添加系数为0.25的Dropout.

Layer	Type	Maps	Size	Kernal Size	Stride	Activation
Out	Fully connected	-	10	-	-	softmax
F9	Fully connected	-	128	-	-	relu
S8	Max pooling	128	1x1	2x2	2	
C7	Convolution	128	3x3	5x5	1	relu
S6	Max pooling	64	3x3	2x2	2	
C5	Convolution	64	7x7	5x5	1	re;u
S4	Max pooling	32	7x7	2x2	2	
C3	Convolution	32	14x14	2x2	1	relu
S2	Max pooling	16	14x14	2x2	2	
C1	Convolution	16	28x28	5x5	1	relu
In	Input	1	28x28	-	-	-

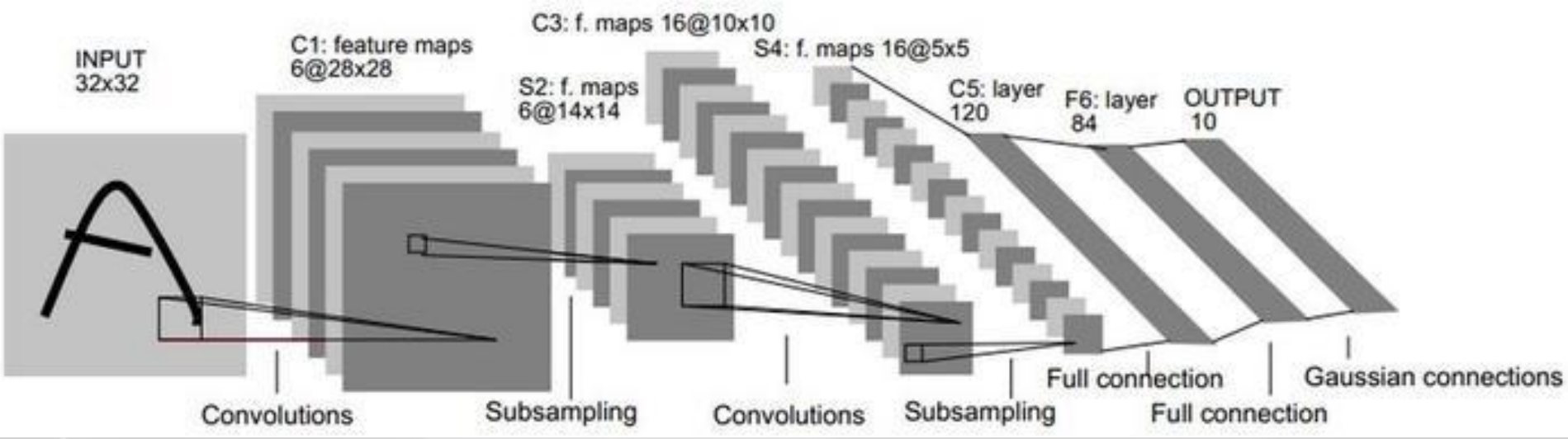
5 经典深度神经网络

- LeNet-5（1998）
- AlexNet（2012）
- GoogLeNet（2014）
- VGGNet (2014)
- ResNet（2015）
- Xception (2016)
- SENet (2017)

5.1 LeNet-5

LeNet-5 架构也许是最广为人知的 CNN 架构。 它是由 Yann LeCun 于 1998 年创建的，广泛用于手写数字识别（MNIST）。

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

LeNet-5特点

1. MNIST图像是28×28像素，但是它们被零填充到32×32像素，在送入网络之前进行标准化。网络的其余部分不使用任何填充，随着图像通过网络进展而尺寸不断缩小。
2. 平均池化层比平常更复杂：每个神经元计算其输入的平均值，然后将结果乘以可学习的**系数**（每个图一个）并添加可学习的**偏差**项（每个图一个），然后最终应用激活函数。

3. C3图中的大多数神经元仅与三个或四个S2图中的神经元相连。

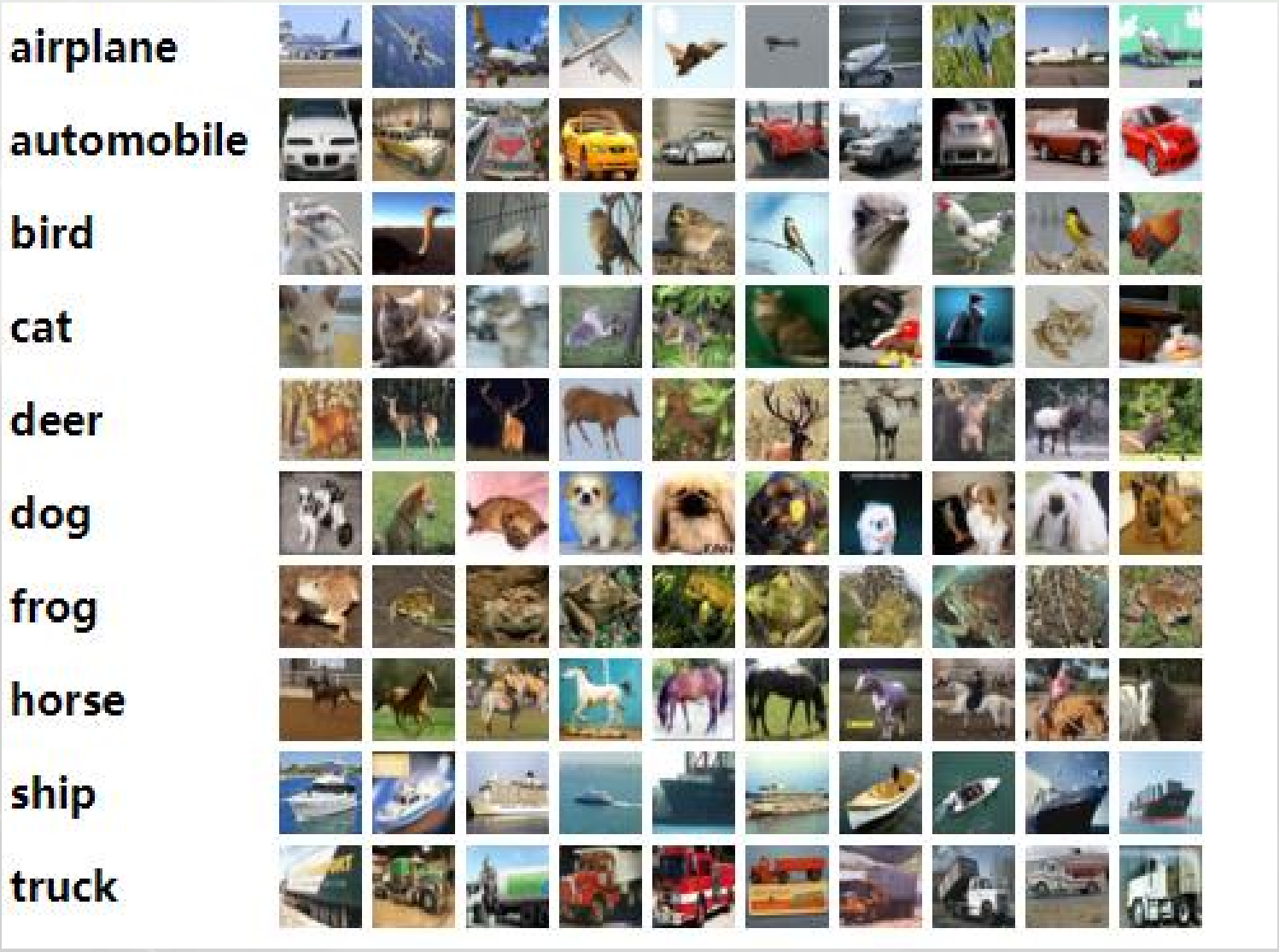
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

4. 输出层：不是计算输入和加权矢量的点积，而是每个神经元输出其输入矢量与其权重矢量之间的欧几里德距离的平方，每个输出表示测量图像属于特定数字类的程度。使用交叉熵成本（cross-entropy cost）函数，它可以更好地处理不良预测，产生更大的梯度，从而更快地收敛。

CIFAR 10数据集

CIFAR 10 数据集一共有 50000 张训练集，10000 张测试集，两个数据集里面的图片都是 png 彩色图片，图片大小是 32×32×3，一共是 10 分类问题，分别为飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船和卡车。

CIFAR 10数据集


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

5.2 AlexNet

AlexNet由Alex Krizhevsky，Ilya Sutskever和Geoffrey Hinton共同开发。它与LeNet-5非常相似，只是更大更深，它是第一个将卷积层直接堆叠在彼此之上，而不是在每个卷积层顶部堆叠池化层。

AlexNet

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

为了减少过拟合，作者使用了两种正则化技术：

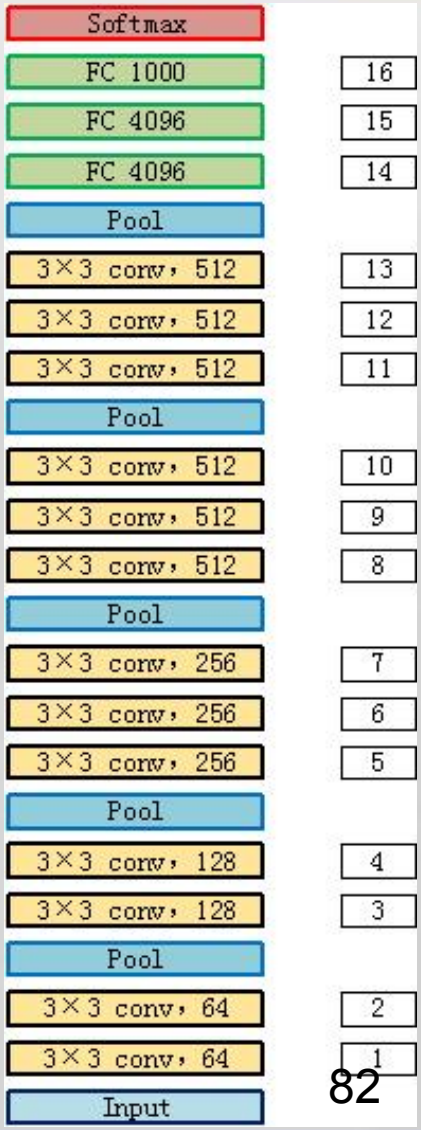
- 1. 在训练期间将DropOut（50%的丢失率）应用于层F8和F9的输出。
- 2. 通过对训练图像进行偏移随机移动、水平翻转、变照明条件来进行数据增强。

AlexNet在层C1和C3的ReLU步骤之后使用局部响应标准化(LRN)。

5.3 VGGNet

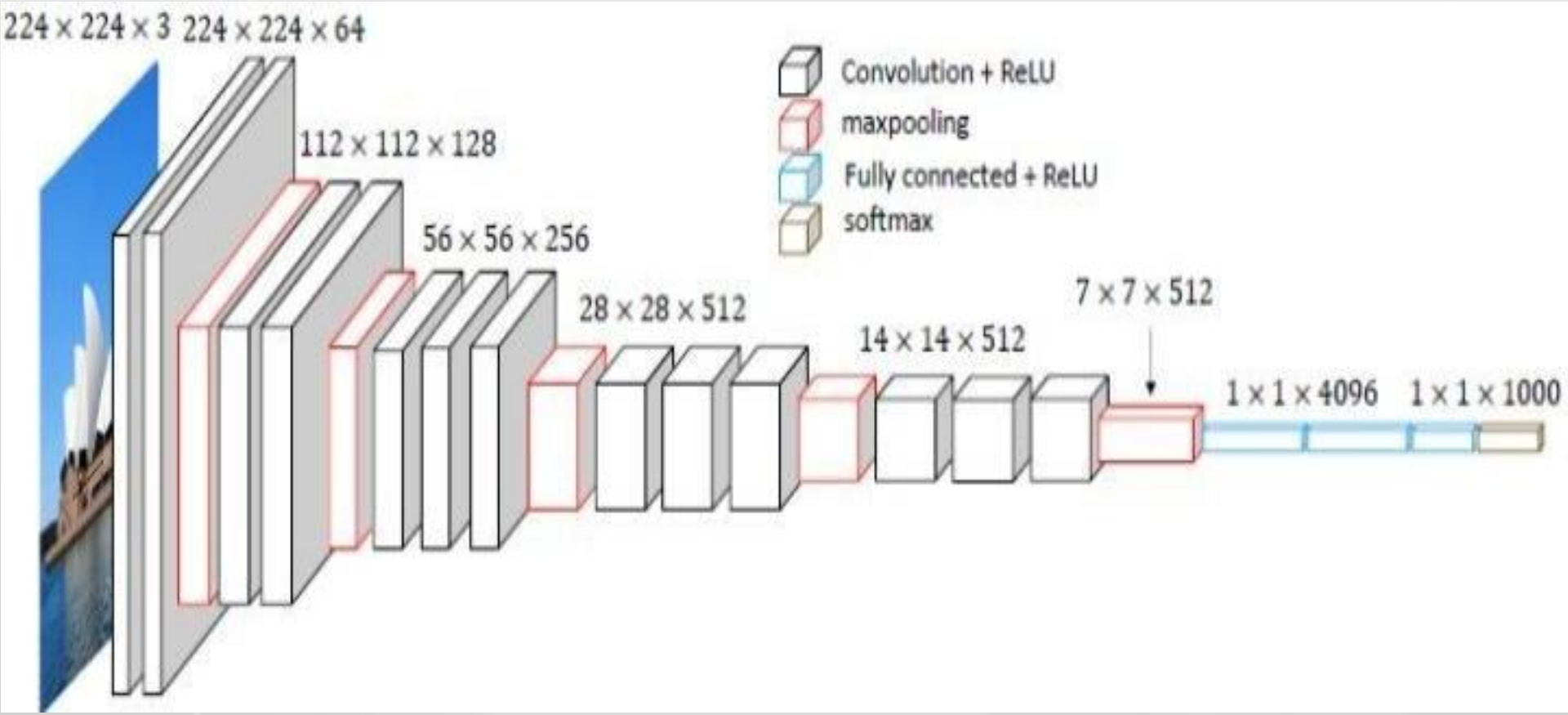
使用3x3卷积核，模型达到16或19层，16层的被称为VGG16，19层的被称为VGG19。

使用Single-Scale和Multi-Scale 训练和评估模型。



VGGNet全部使用3*3的卷积核和2*2的池化核。

同时，两个3*3卷积层的串联相当于1个5*5的卷积层，3个3*3的卷积层串联相当于1个7*7的卷积层。但是3个3*3的卷积层参数量只有7*7的一半左右，同时前者可以有3个非线性操作，而后者只有1个非线性操作，这样使得前者对于特征的学习能力更强。


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
Convolution + ReLU
maxpooling
Fully connected + ReLU
softmax

5.4 GoogLeNet

GoogLeNet架构由谷歌的Christian Szegedy等人开发。

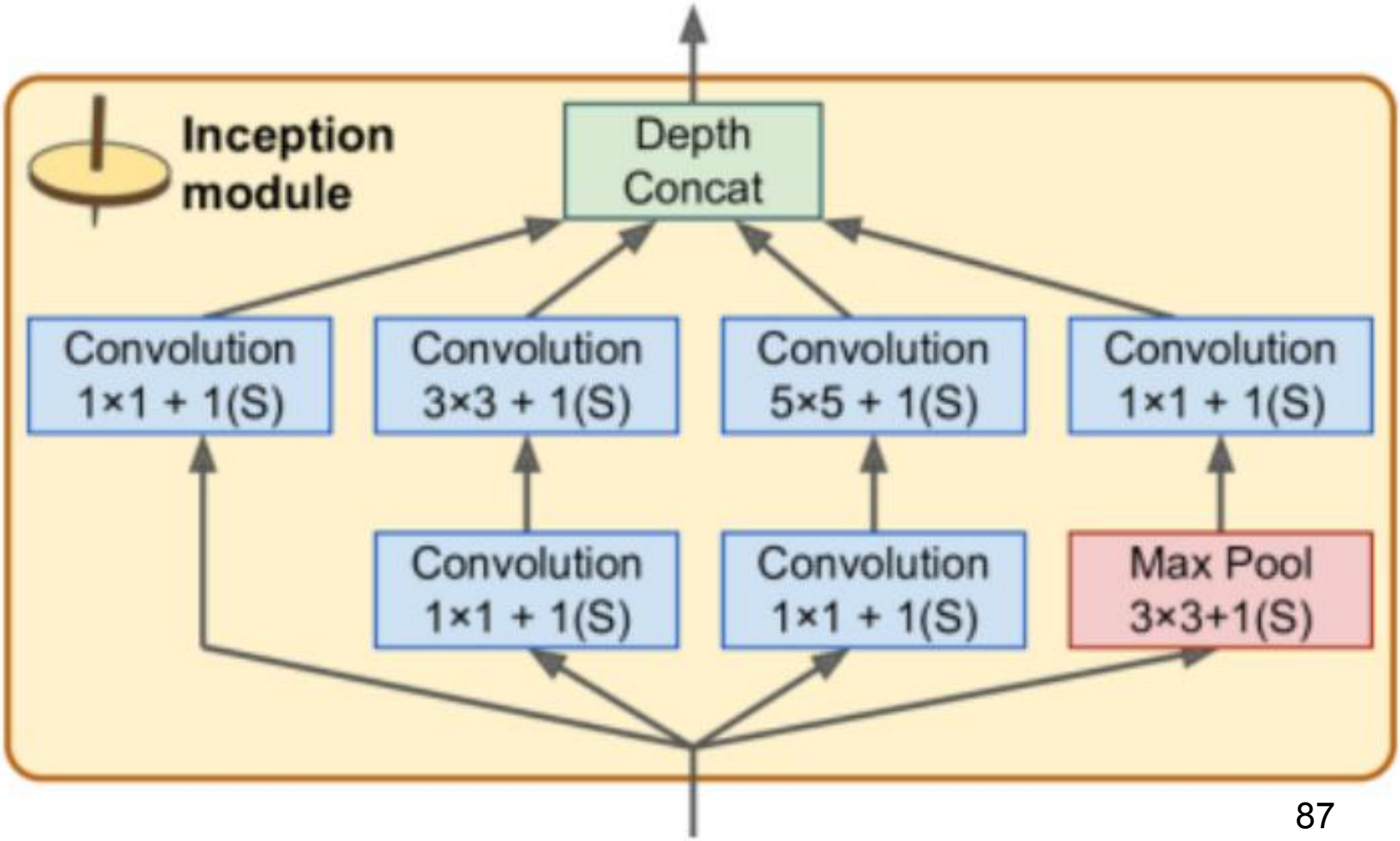
通过引入称为Inception module的子网络实现了更深的网络结构，使GoogLeNet比以前的架构更有效地使用参数。

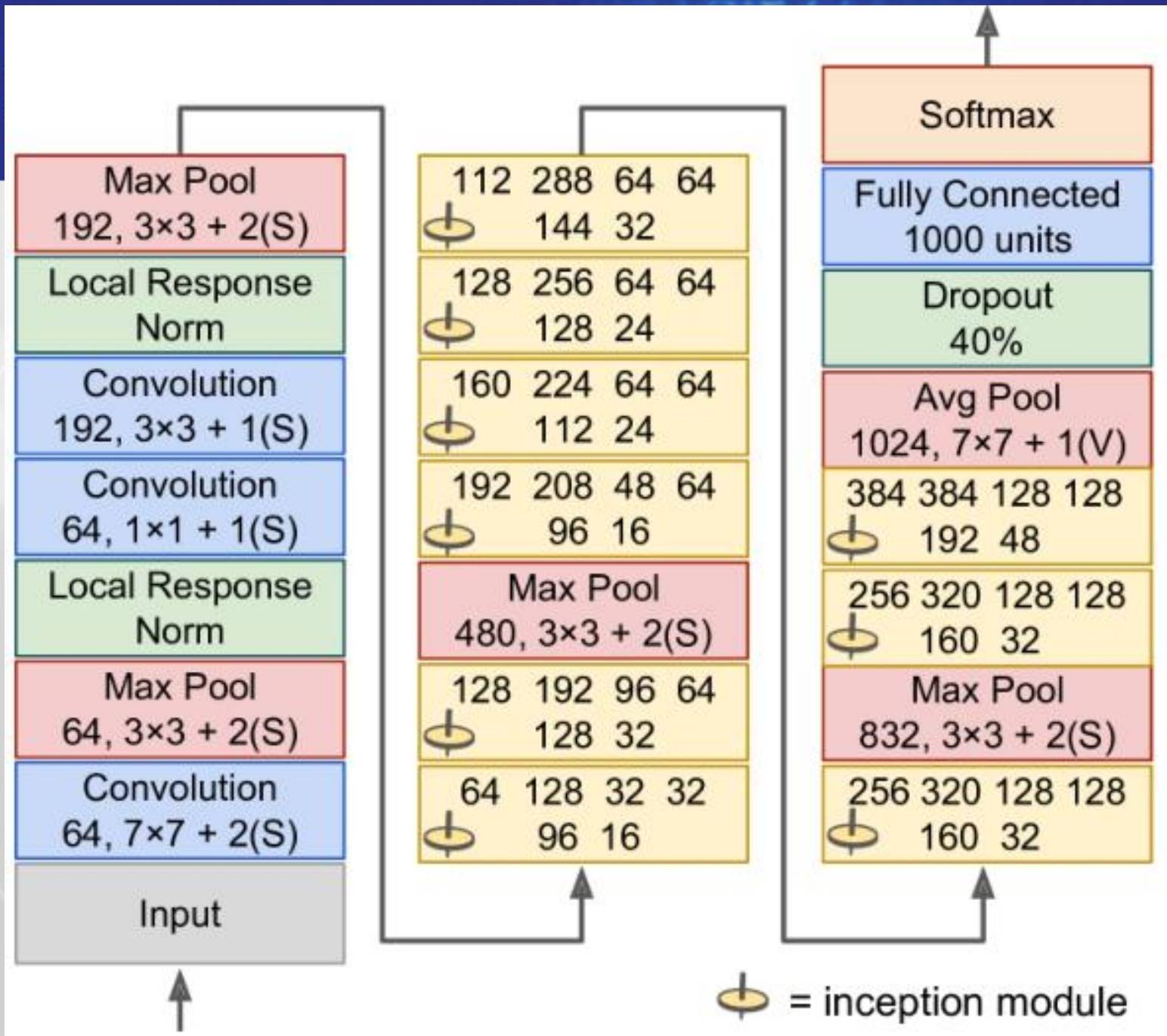
GoogLeNet的参数比AlexNet少10倍（大约600万）。

Inception module

- 符号“3×3+2（S）”表示该层使用3×3内核、步幅2、SAME填充。
- 复制输入信号并将其馈送到四个不同的层
- 所有卷积层都使用ReLU激活功能
- 所有层都使用步幅 1 和SAME填充（包括最大池化层）

Inception module




Softmax
Fully Connected
1000 units
Dropout
40%
Avg Pool
1024, 7×7 + 1(V)
384 384 128 128
192 48
256 320 128 128
160 32
Max Pool
832, 3×3 + 2(S)
256 320 128 128
160 32
112 288 64 64
144 32
128 256 64 64
128 24
160 224 64 64
112 24
192 208 48 64
96 16
Max Pool
480, 3×3 + 2(S)
128 192 96 64
128 32
64 128 32 32
96 16
Max Pool
192, 3×3 + 2(S)
Local Response
Norm
Convolution
192, 3×3 + 1(S)
Convolution
64, 1×1 + 1(S)
Local Response
Norm
Max Pool
64, 3×3 + 2(S)
Convolution
64, 7×7 + 2(S)
Input
= inception module

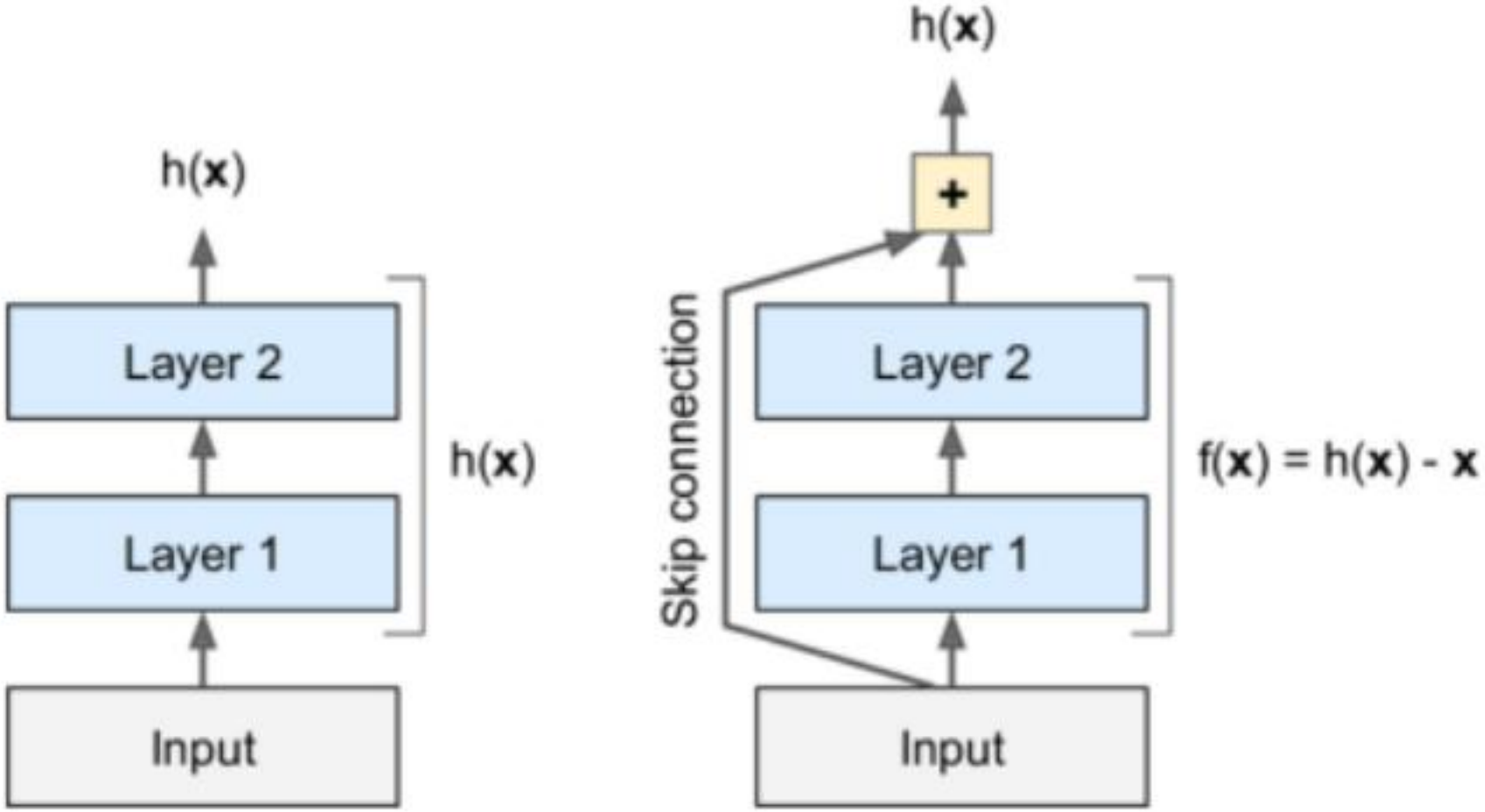
5.5 ResNet

2015年ILSVRC挑战赛的获胜者是由Kaiming He等人开发的残差网络（或ResNet），它使用由152层组成的极深CNN，使得前5个错误率低于3.6%。

能够训练这种深度网络的关键是使用跳跃连接（skip connections）：进入每一层的信号同时也被添加到位于堆栈上方稍高的层的输出。

Residual learning(残差学习)

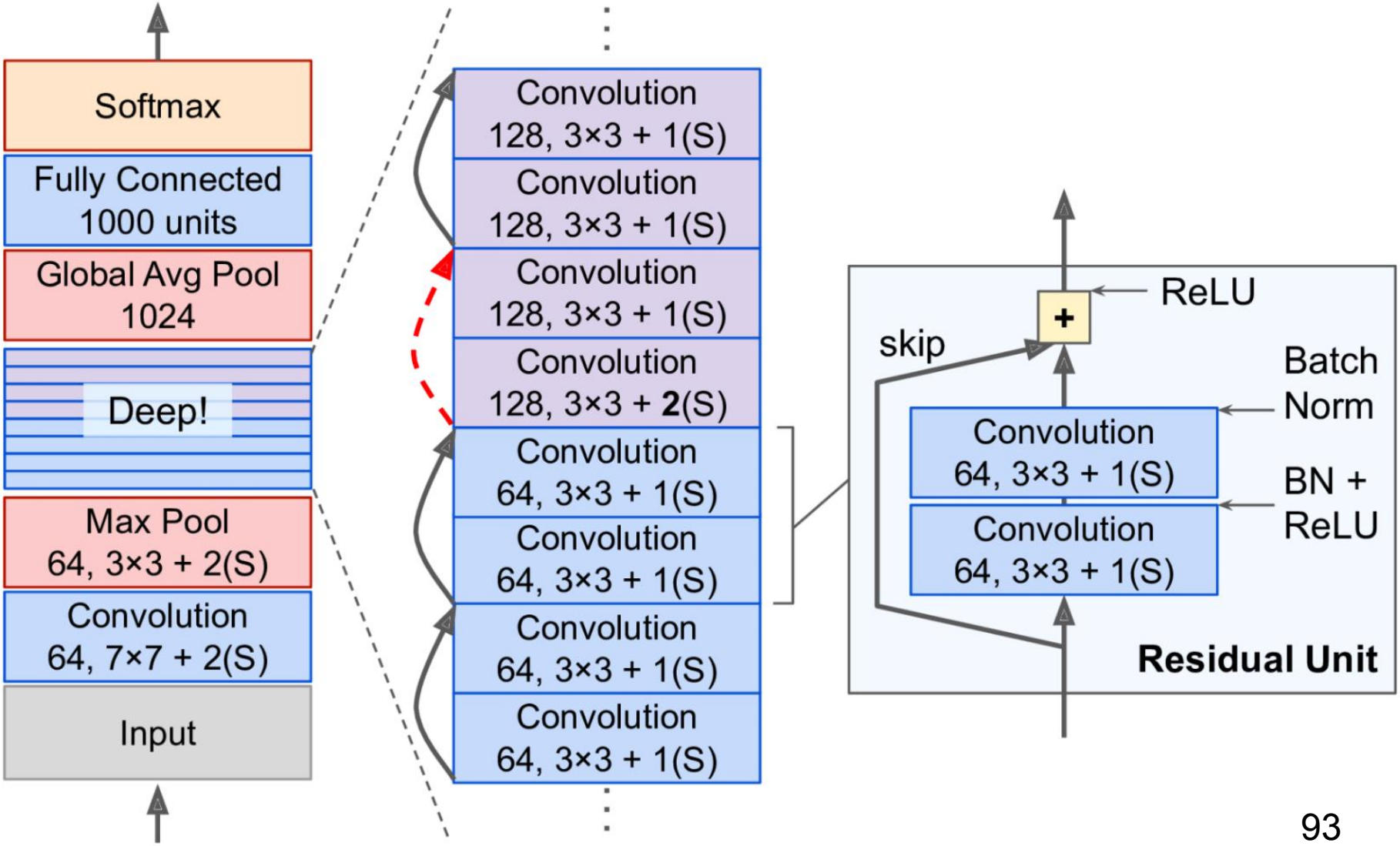
假定某段神经网络的输入是x，期望输出是H(x)，如果直接把输入x传到输出作为初始结果，那么此时需要学习的目标就是F(x) = H(x) - x。这就是一个ResNet的残差学习单元，ResNet相当于将学习目标改变了，不再是学习一个完整的输出H(x)，只是输出和输入的差别H(x)-x，即残差。


h(x)
Layer 2
Layer 1
Input
h(x)
h(x)
+
Skip connection
Layer 2
Layer 1
Input
f(x) = h(x) - x

ResNet的结构

- 与GoogLeNet非常相似，除了
  - 1）没有DropOut层
  - 2）加入了一个非常深的简单残差单元堆栈
- 每个残差单元由两个卷积层组成，用批量归一化（BN）和ReLU激活，使用3×3内核并保留空间维度（步幅1，SAME填充）。

ResNet结构


Softmax
Fully Connected
1000 units
Global Avg Pool
1024
Deep!
Max Pool
64, 3×3 + 2(S)
Convolution
64, 7×7 + 2(S)
Input
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 2(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
+
ReLU
skip
Batch
Norm
Convolution
64, 3×3 + 1(S)
BN +
ReLU
Convolution
64, 3×3 + 1(S)
Residual Unit

ResNet-34

ResNet-34是34层的ResNet，其中:

- 3个输出64个特征映射的残差单元（Residual Units）
- 4个具有128个映射的RU
- 6个具有256个映射的RU
- 3个具有512个映射的RU

*仅对卷积层和完全连接层进行计数

更深层次的ResNets，例如ResNet-152，使用稍微不同的残差单元。他们使用三个卷积层而不是带有256个特征映射的两个3x3卷积层：

- 首先是一个1×1的卷积层，只有64个特征图（少4倍），它作为一个瓶颈层（如前所述），然后是一个带有64个特征图的3×3层，
- 最后是另一个1×1卷积层，具有256个特征图（4倍64），可恢复原始深度。

ResNet-152包含三个这样的RU，它们输出256个映射，然后是8个具有512个映射的RU，一个高达36个RU和1,024个映射，最后是3个具有2,048个映射的RU。

作业1

用TensorFlow或Keras编写代码，完成PPT中第70页的练习。

作业2

Consider a CNN composed of three convolutional layers, each with 3 × 3 kernels, a stride of 2, and SAME padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200 × 300 pixels.

1) What is the total number of parameters in the CNN?

2) If we are using 32-bit floats, at least how much RAM will this network require when making a prediction for a single instance? What about when training on a mini-batch of 50 images?