ThoughtWorks®

StuQ

佟达

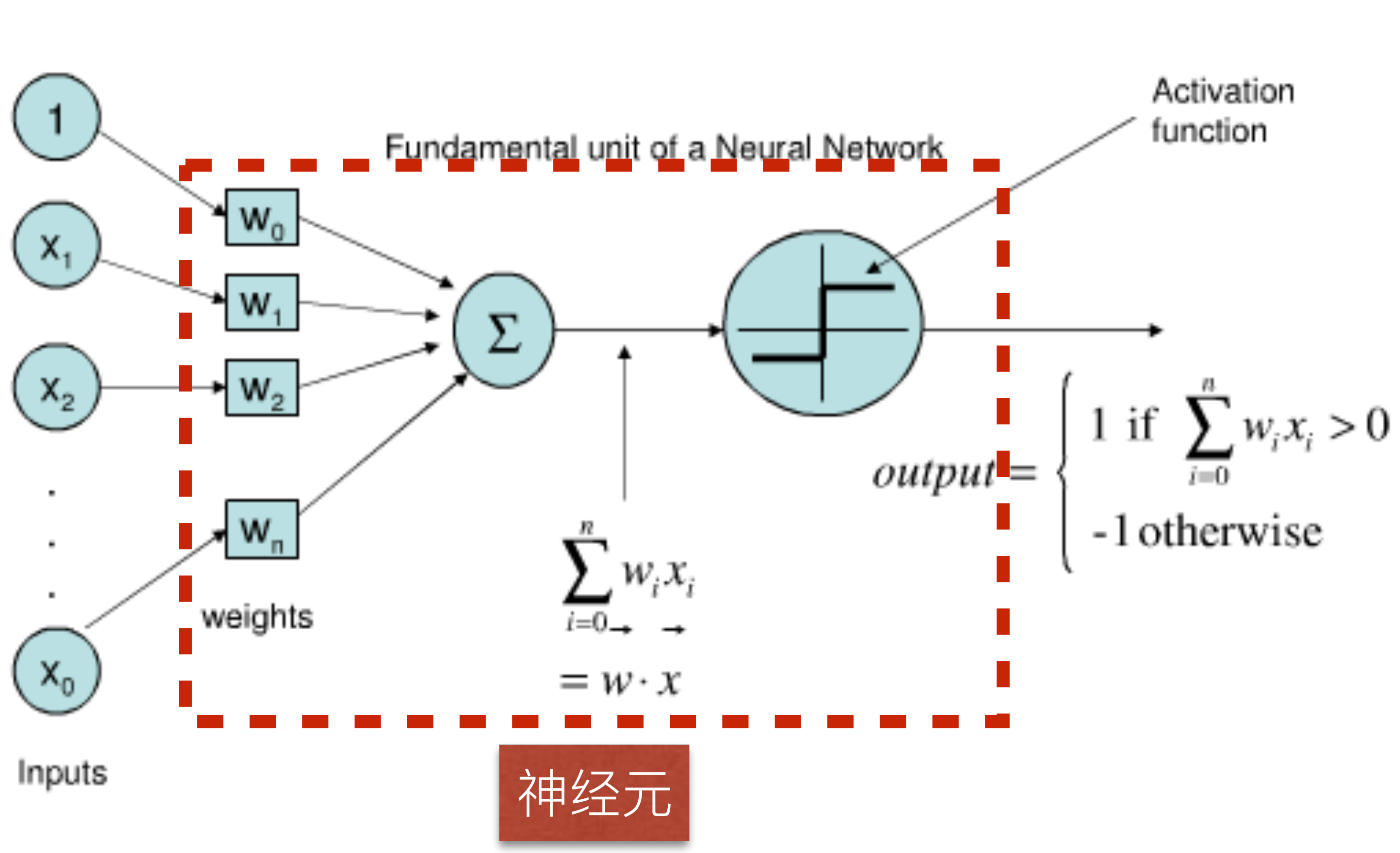
TENSORFLOW实现卷积神经网络

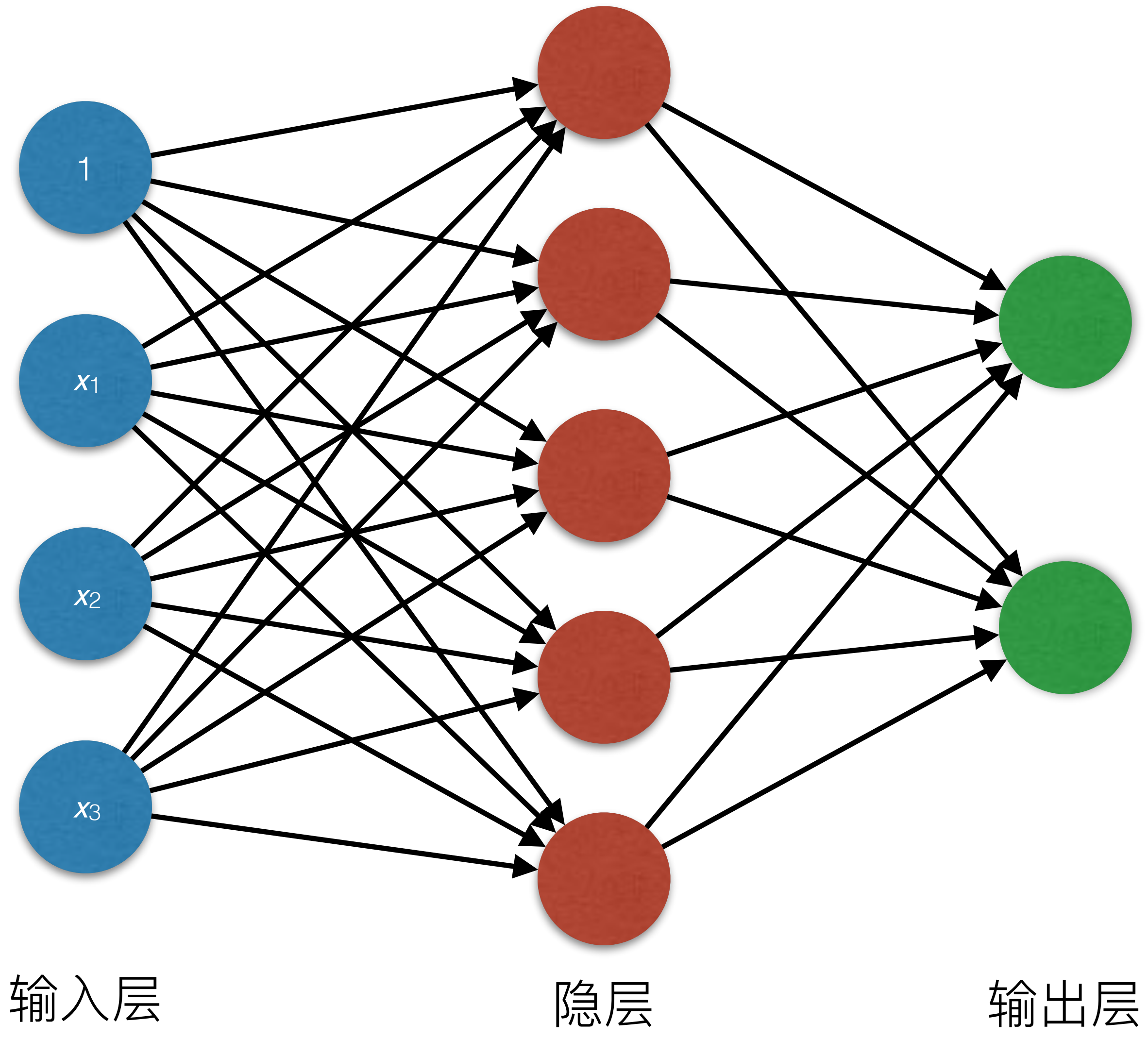
2016.10.28

回顾 - 神经网络

详细请看《程序员眼中的深度学习》

多层感知机——任务定义





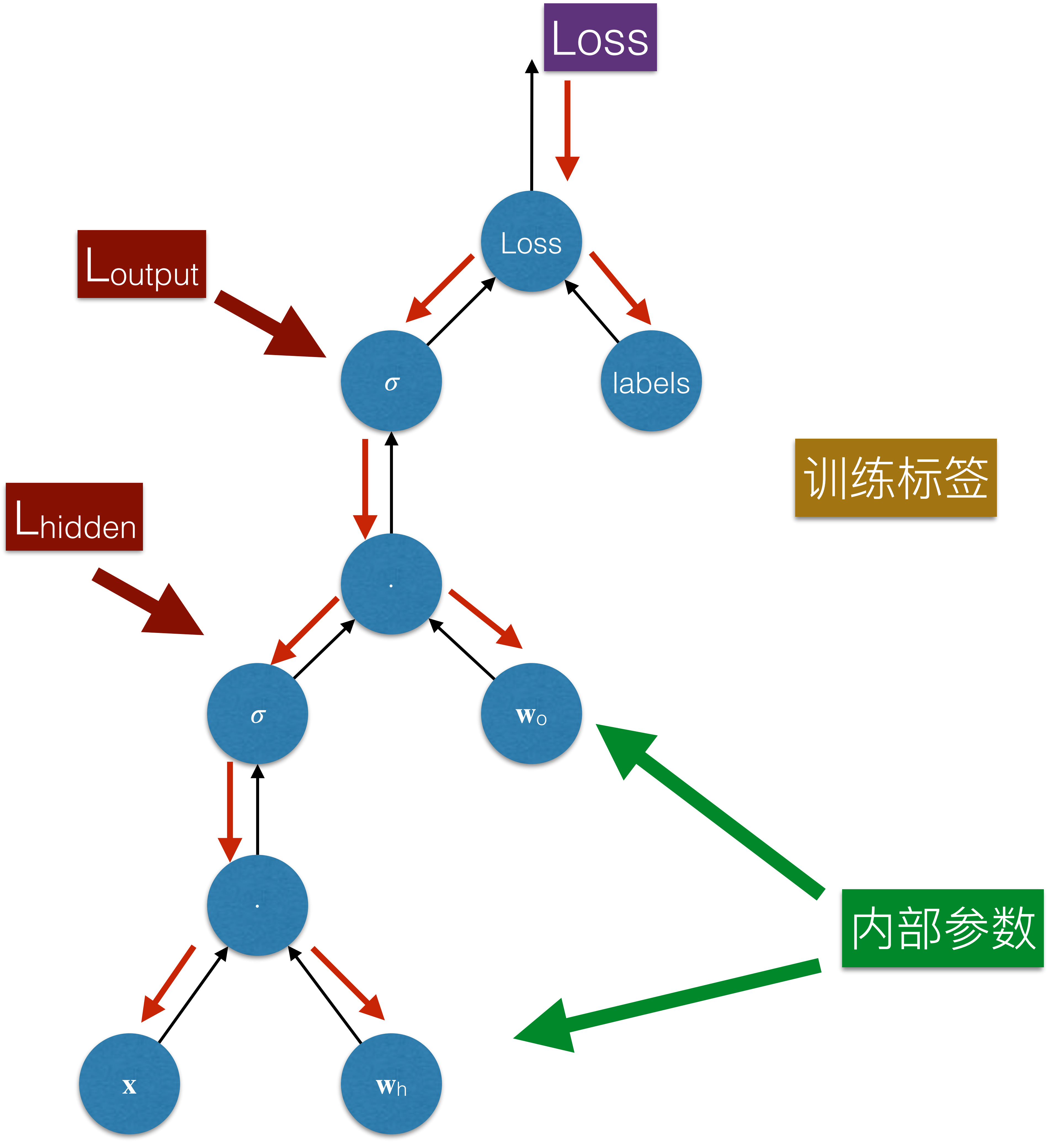
$$L_{hidden} = sigmoid(\boldsymbol{x} \cdot \boldsymbol{W}_{hidden} + \boldsymbol{b}_{hidden})$$

$$L_{output} = \boldsymbol{x} \cdot \boldsymbol{W}_{output} + \boldsymbol{b}_{output}$$

$$Loss = cross\_entropy(L_{output}, labels)$$

通过大量的$\boldsymbol{x}$和labels，找到一组$\boldsymbol{W}_{hidden}$, $\boldsymbol{W}_{output}$, $\boldsymbol{b}_{hidden}$, $\boldsymbol{b}_{output}$，使得Loss最小。

反向传播



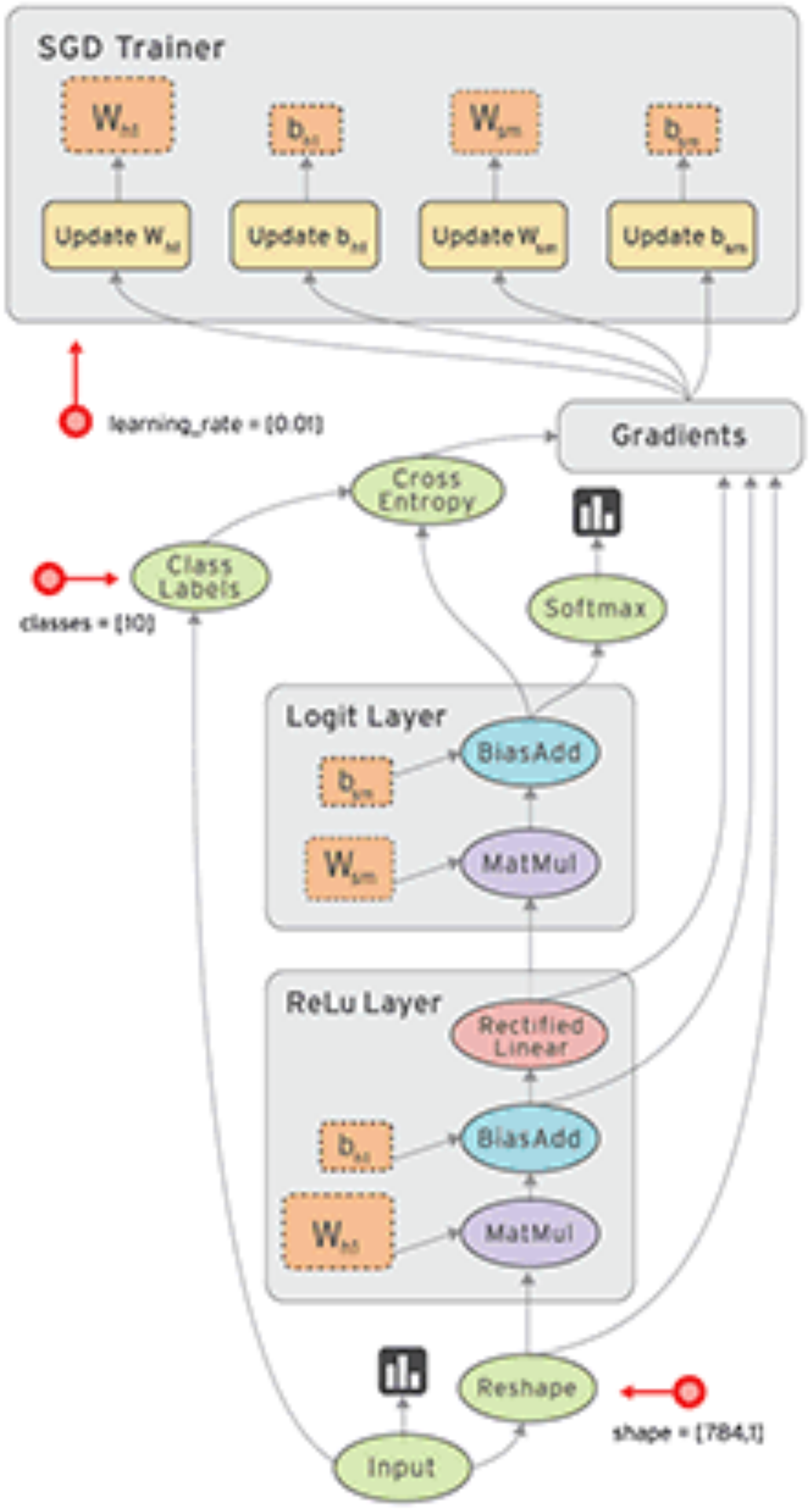
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

$$d\mathrm{Loss} / d\mathrm{w_o} = (d\mathrm{Loss}/d\mathrm{L_{output}})\cdot(d\mathrm{L_{output}}/d\mathrm{w_o})$$

$$d\mathrm{Loss} / d\mathrm{w_h} = (d\mathrm{Loss}/d\mathrm{L_{output}})\cdot(d\mathrm{L_{output}}/d\mathrm{L_{hidden}})\cdot(d\mathrm{L_{hidden}}/d\mathrm{w_h})$$

$$d\mathrm{Loss} / d\mathrm{w_{xxx}} = ?$$

- Google Brain开发维护
    - Jeff Dean (BigTable, MapReduce)
    - Yangqing Jia (Caffe)
    - Ian Goodfellow (Theano)
- 声明式定义计算图
- 自动求导
- 支持集群计算
    - 模型并行
    - 数据并行
- 工程支持
    - TensorBoard
    - TensorFlow Serving
    - Integrated with Kubernetes



训练数据

```
import tensorflow as tf

X = tf.constant([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]], dtype=tf.float32)
y = tf.constant([0, 1, 1, 0])

syn0 = tf.Variable(tf.random_uniform([3, 4], minval=-1, maxval=1, dtype=tf.float32))
syn1 = tf.Variable(tf.random_uniform([4, 2], minval=-1, maxval=1, dtype=tf.float32))

l1 = tf.sigmoid(tf.matmul(X, syn0))
l2 = tf.sigmoid(tf.matmul(l1, syn1))

loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(l2, y))
train_op = tf.train.GradientDescentOptimizer(0.1).minimize(loss_op)

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

for i in range(60000):
  _, loss = sess.run([train_op, loss_op])
```

声明计算逻辑

声明参数

声明反向计算逻辑和参数更新策略

执行计算

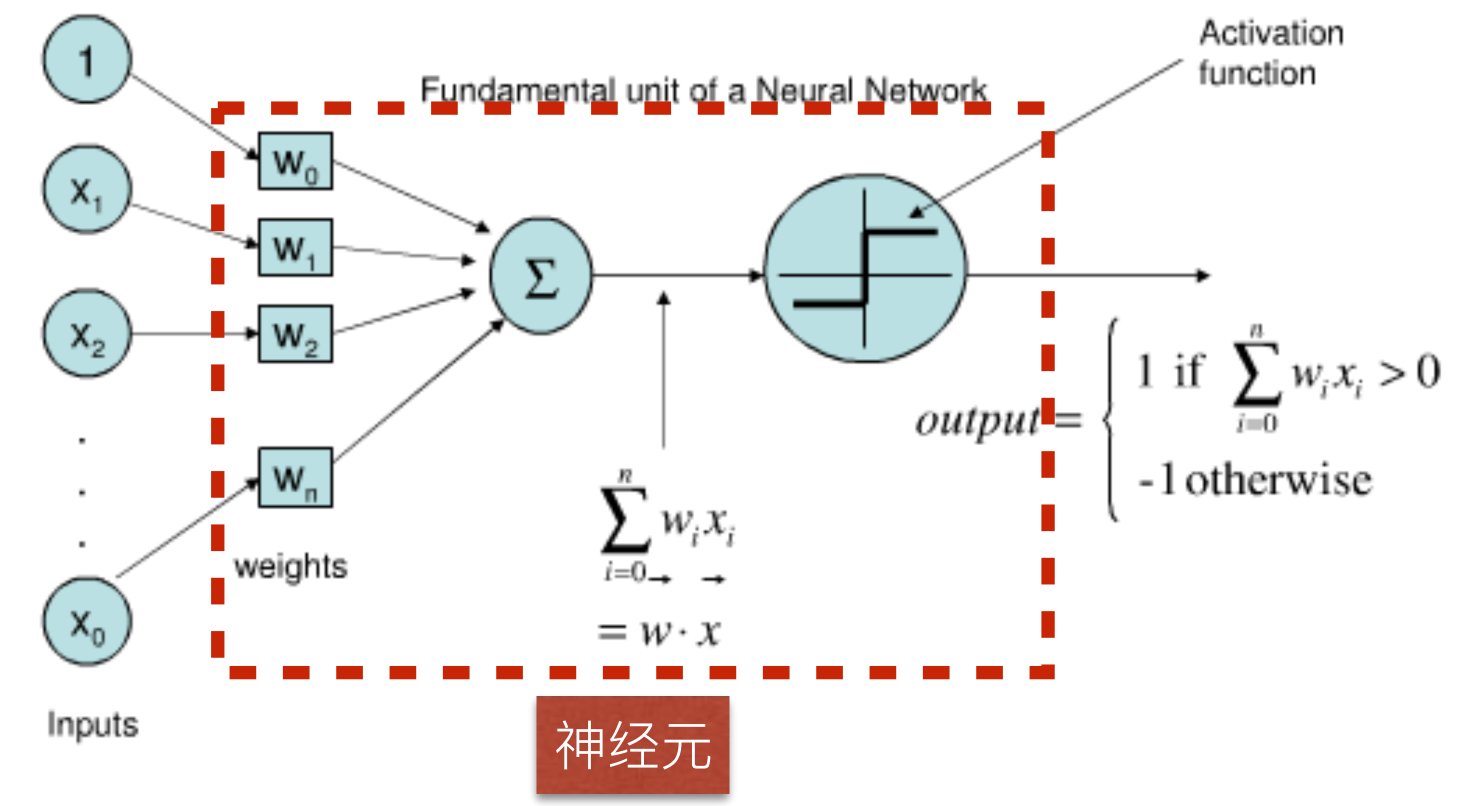
练习1 - 线性回归

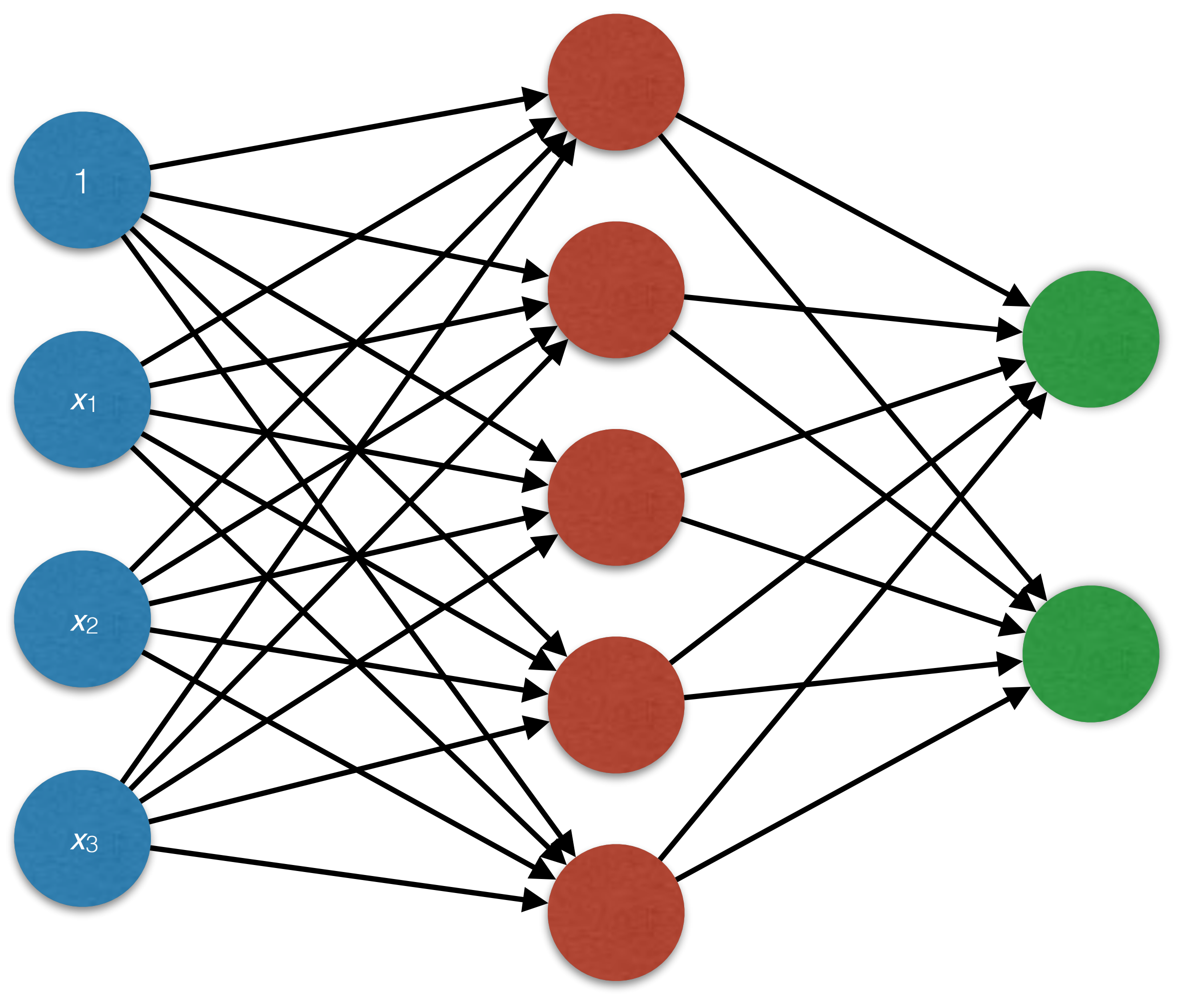
练习环境

- https://github.com/tongda/tf-tutorial
- Python
- TensorFlow
- ipython notebook
- matplotlib

练习2 - 神经网络

多层感知机——任务定义





$$L_{hidden} = sigmoid(\boldsymbol{x} \cdot \boldsymbol{W}_{hidden} + \boldsymbol{b}_{hidden})$$

$$L_{output} = \boldsymbol{x} \cdot \boldsymbol{W}_{output} + \boldsymbol{b}_{output}$$

$$Loss = cross\_entropy(L_{output},\ labels)$$

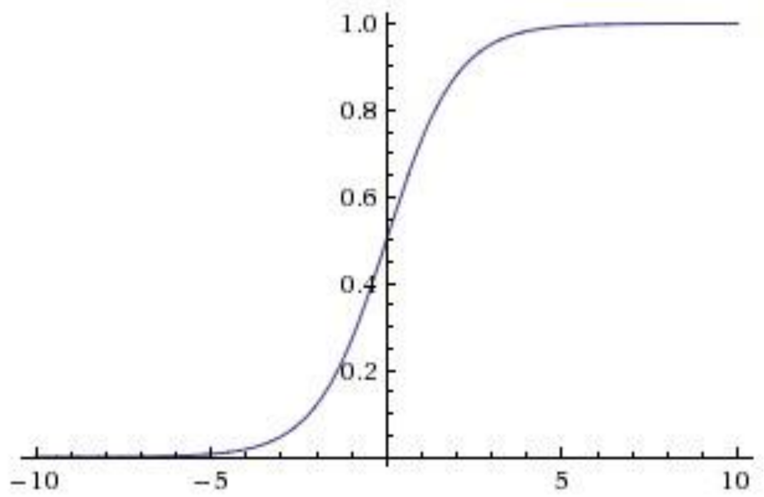
通过大量的$\boldsymbol{x}$和labels，找到一组$\boldsymbol{W}_{hidden}$, $\boldsymbol{W}_{output}$, $\boldsymbol{b}_{hidden}$, $\boldsymbol{b}_{output}$，使得Loss最小。

激活函数

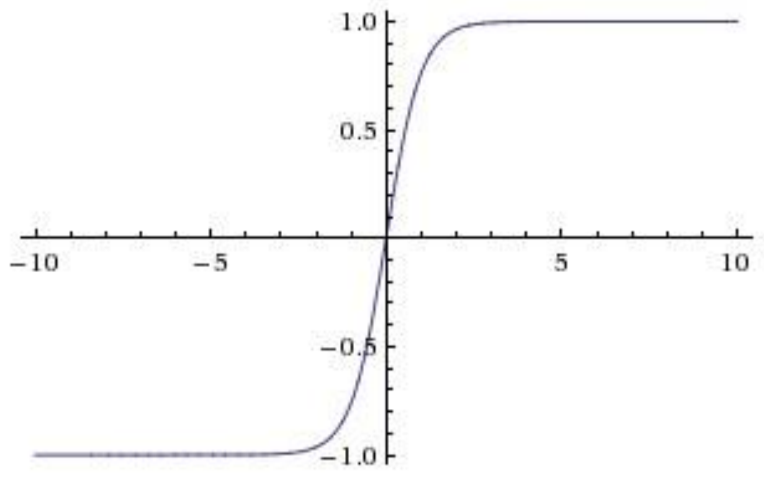
Activation Functions

Sigmoid

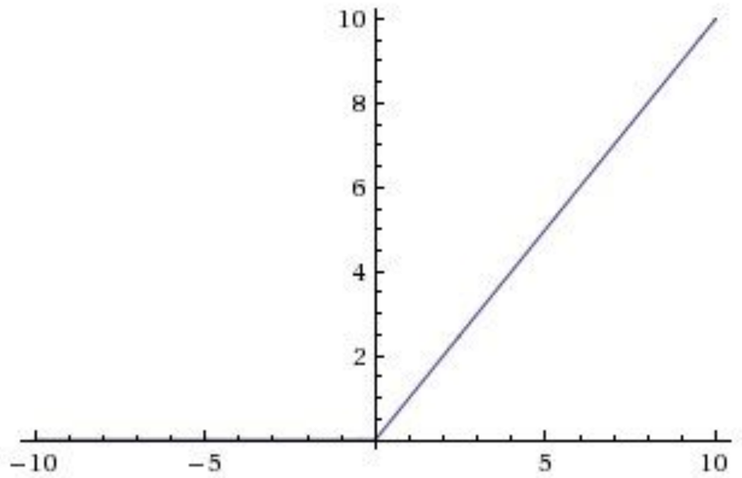
$$\sigma(x) = 1/(1+e^{-x})$$



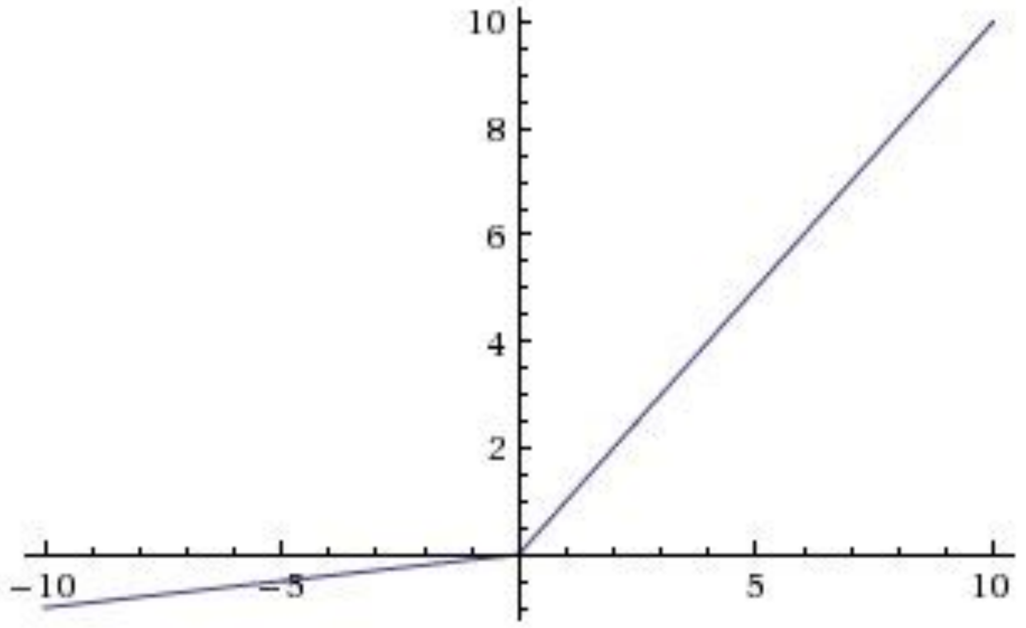
tanh tanh(x)



ReLU max(0,x)

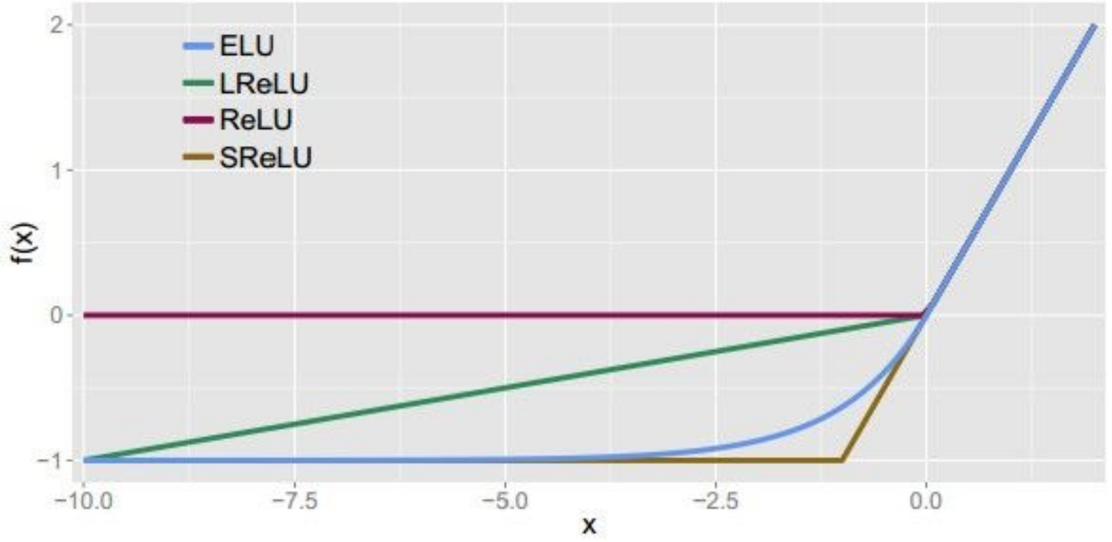


Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$

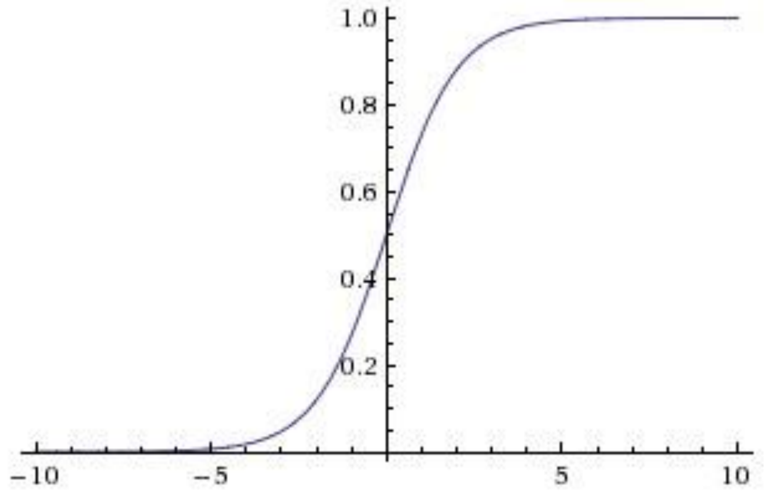


激活函数

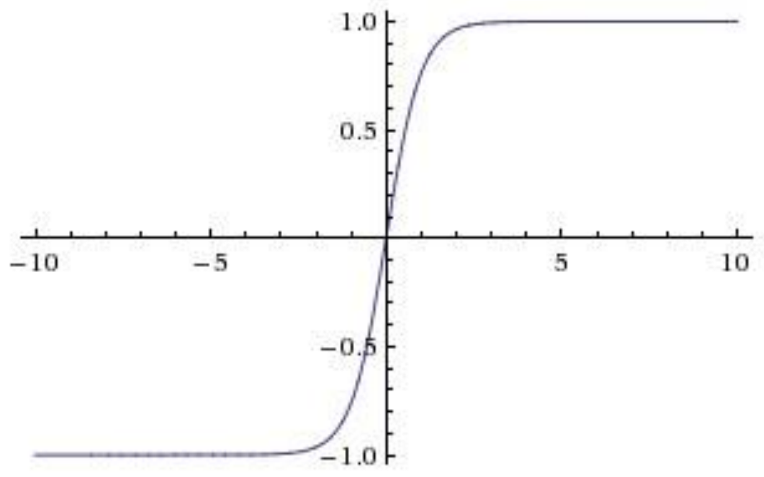
Activation Functions

Sigmoid

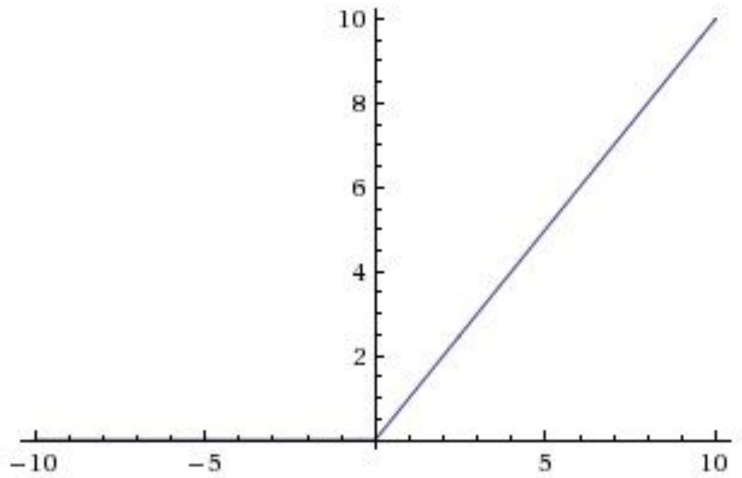
$$\sigma(x) = 1/(1+e^{-x})$$



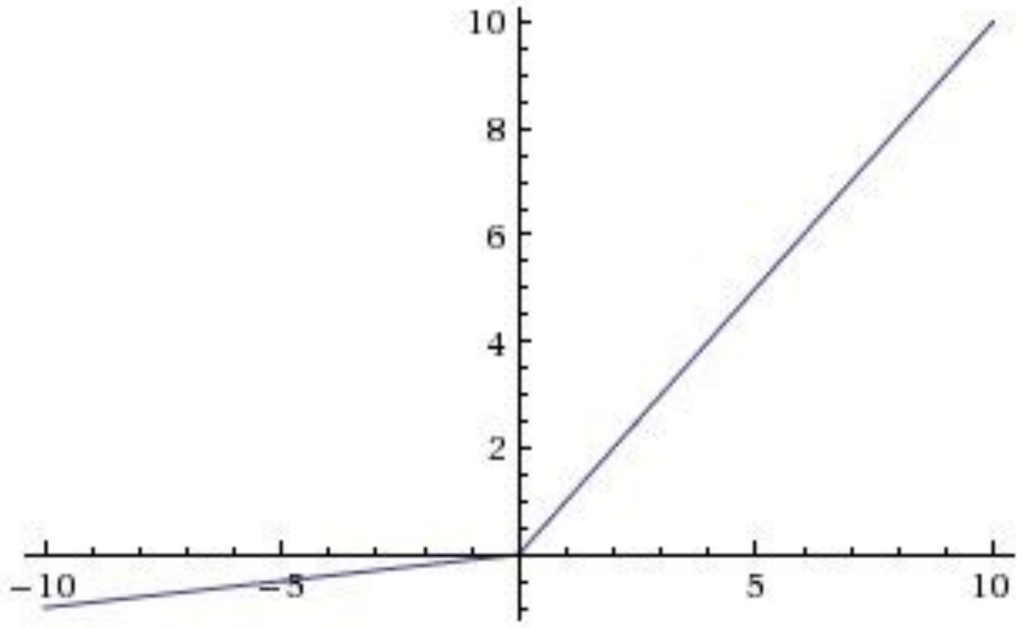
tanh tanh(x)



ReLU max(0,x)



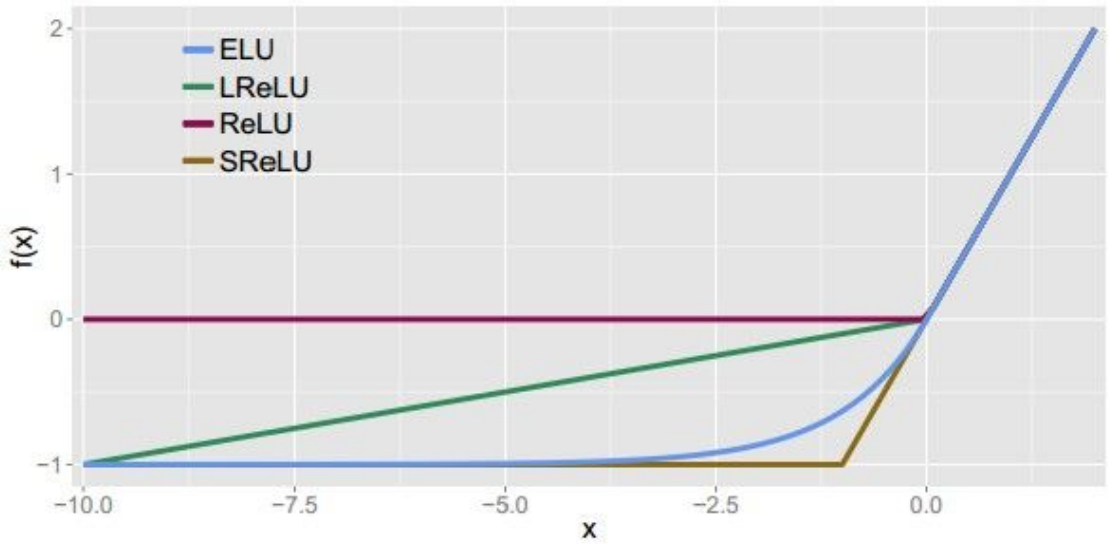
Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

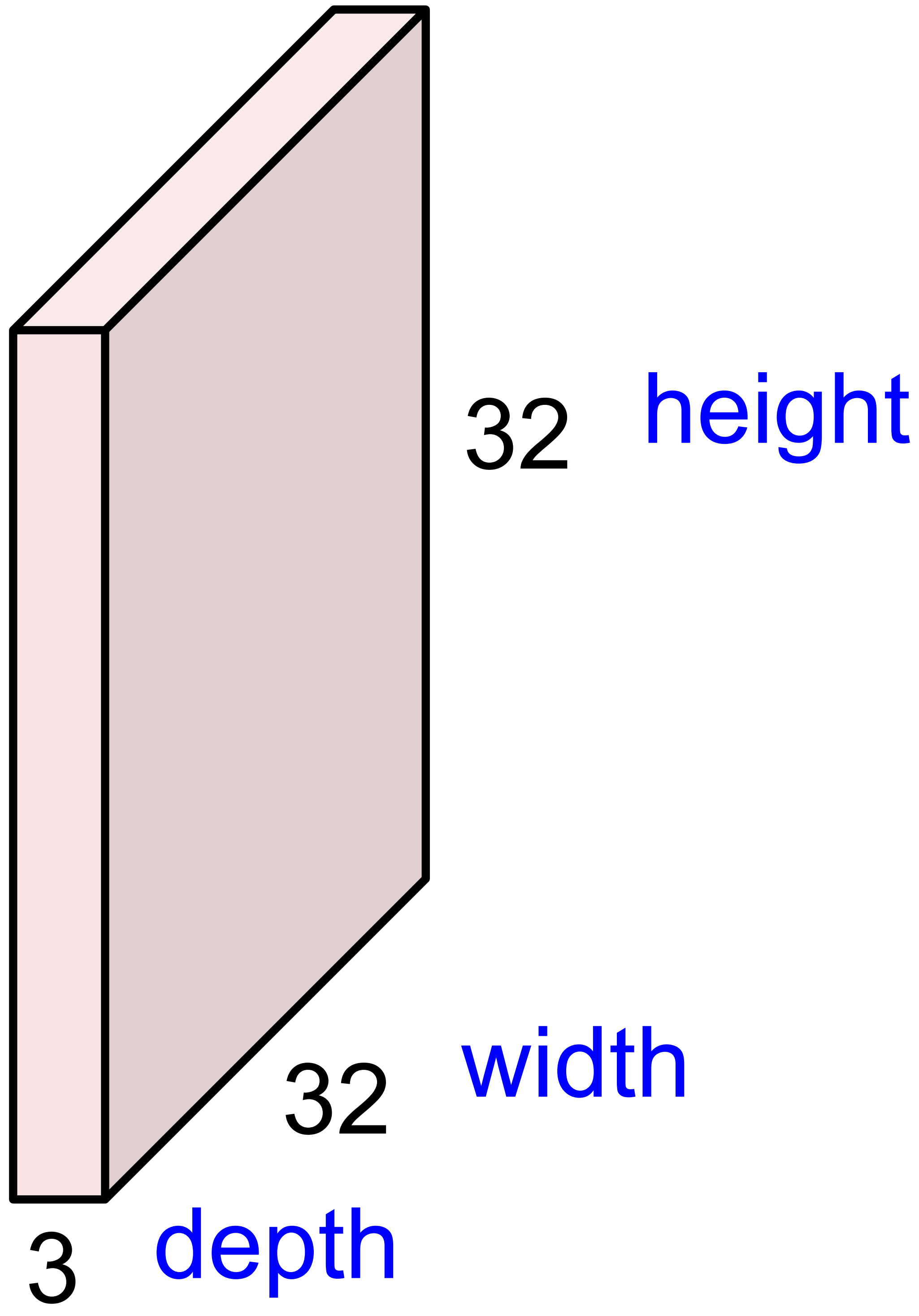
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



卷积神经网络

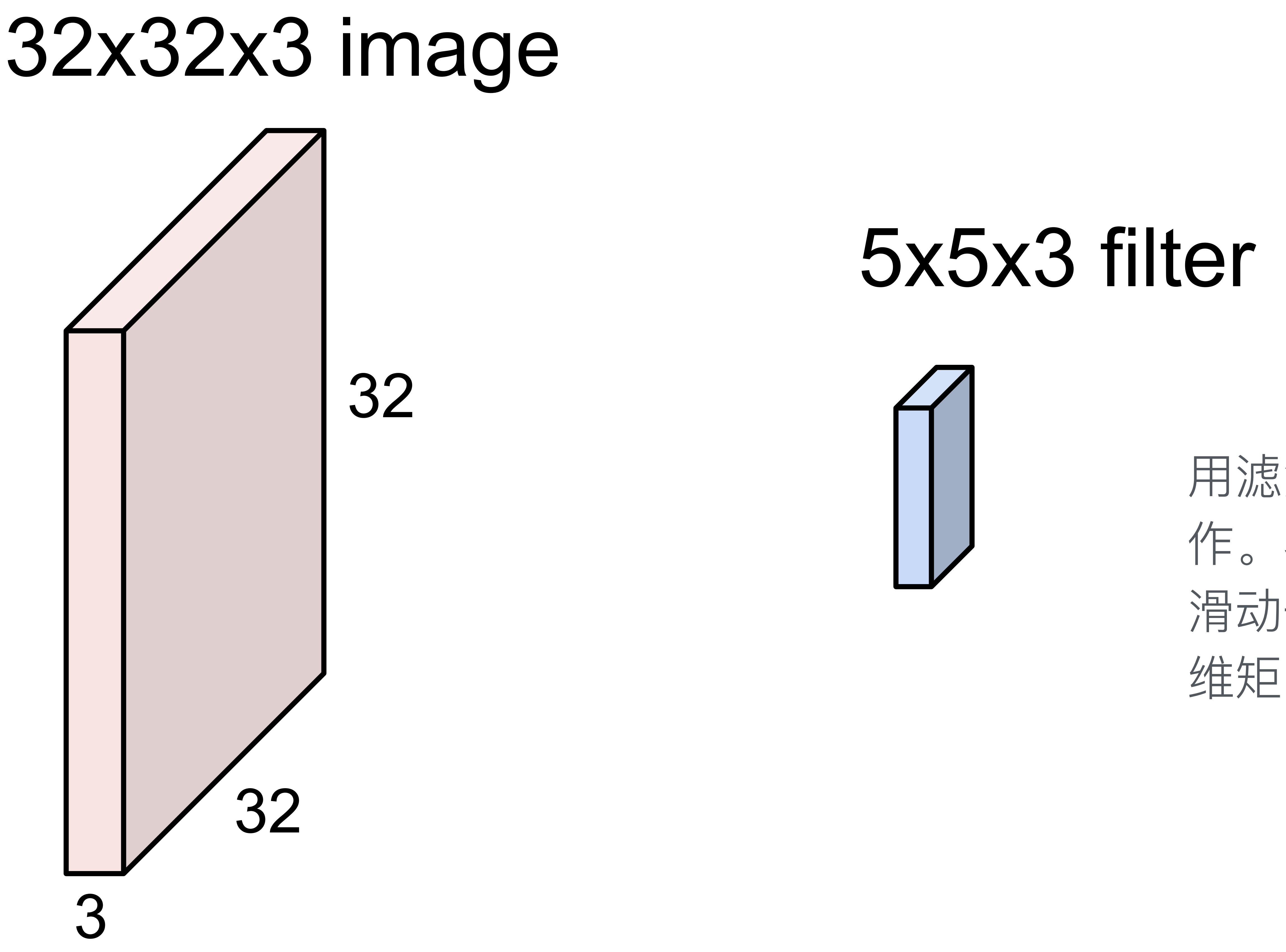
Convolution Layer

32x32x3 image



http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

Convolution Layer



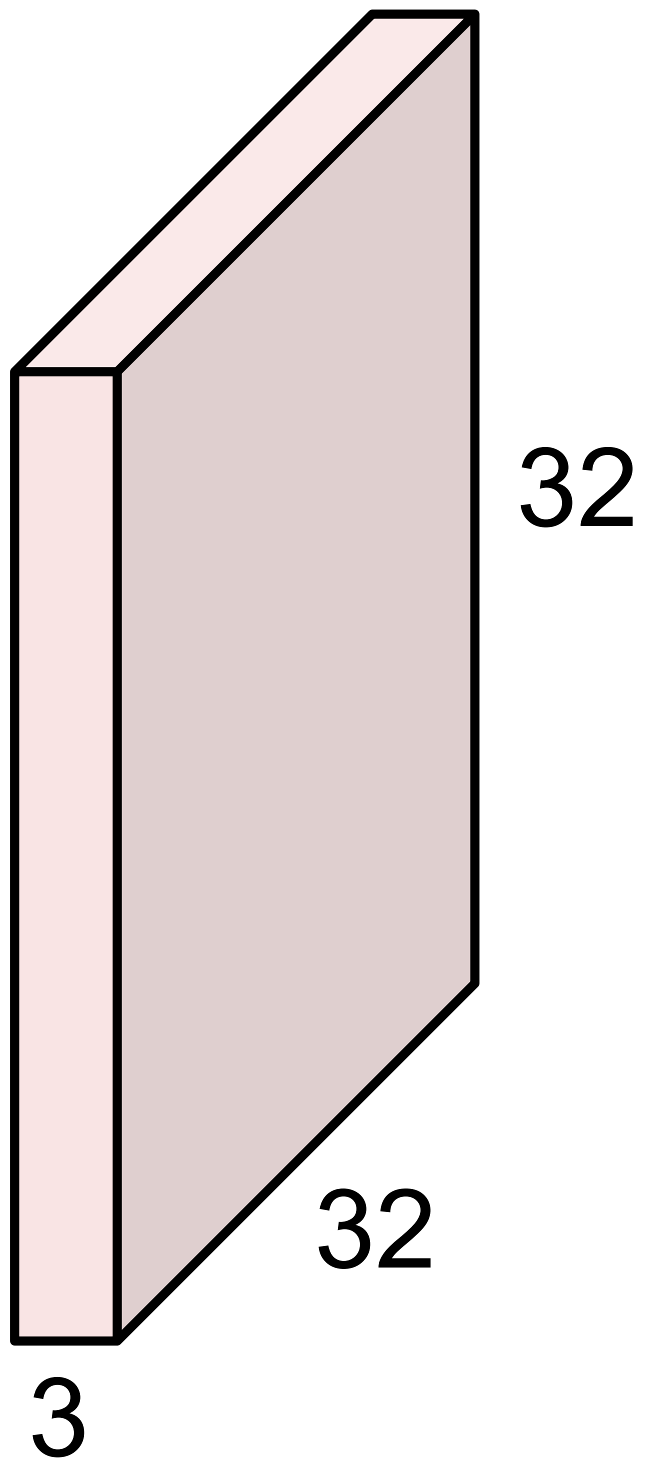
用滤波器/滤镜/过滤器（filter）对图片执行卷及操作。卷积操作是将滤波器在图片上进行滑动，每滑动一个位置，计算滤波器和图片上对应位置多维矩阵的点积。

http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

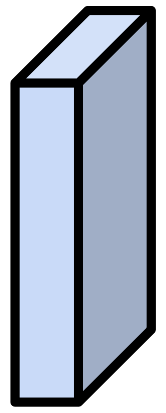
Convolution Layer

卷积核的深度（即图片的通道数目），和输入图片的深度保持一致。

32x32x3 image



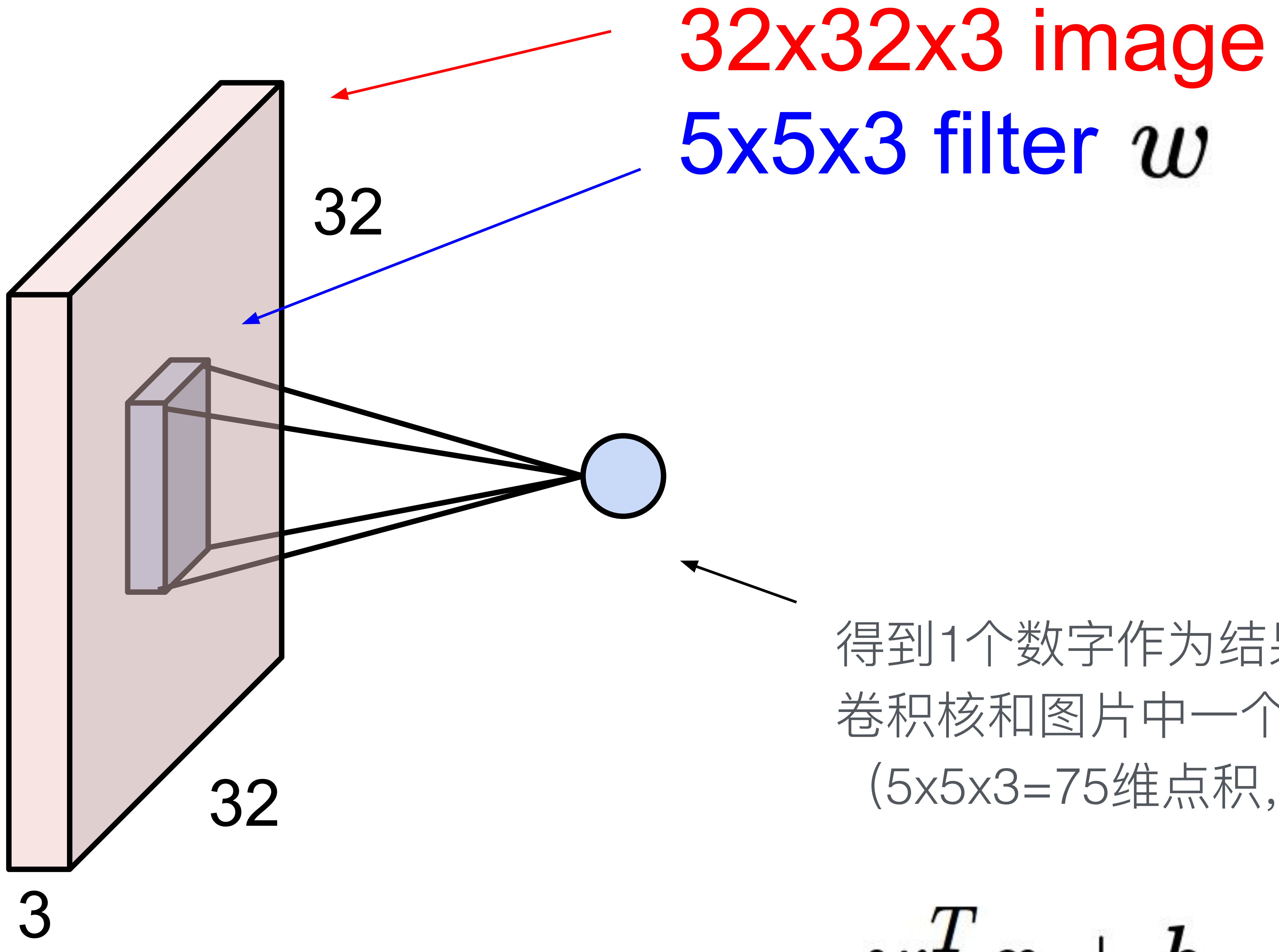
5x5x3 filter



用滤波器/滤镜/过滤器/卷积核（filter）对图片执行卷及操作。卷积操作是将卷积核在图片上进行滑动，每滑动一个位置，计算滤波器和图片上对应位置多维矩阵的点积。

http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

Convolution Layer



得到1个数字作为结果：
卷积核和图片中一个5x5x3数据块的点积结果。
（5x5x3=75维点积，加上偏置）

$$w^T x + b$$

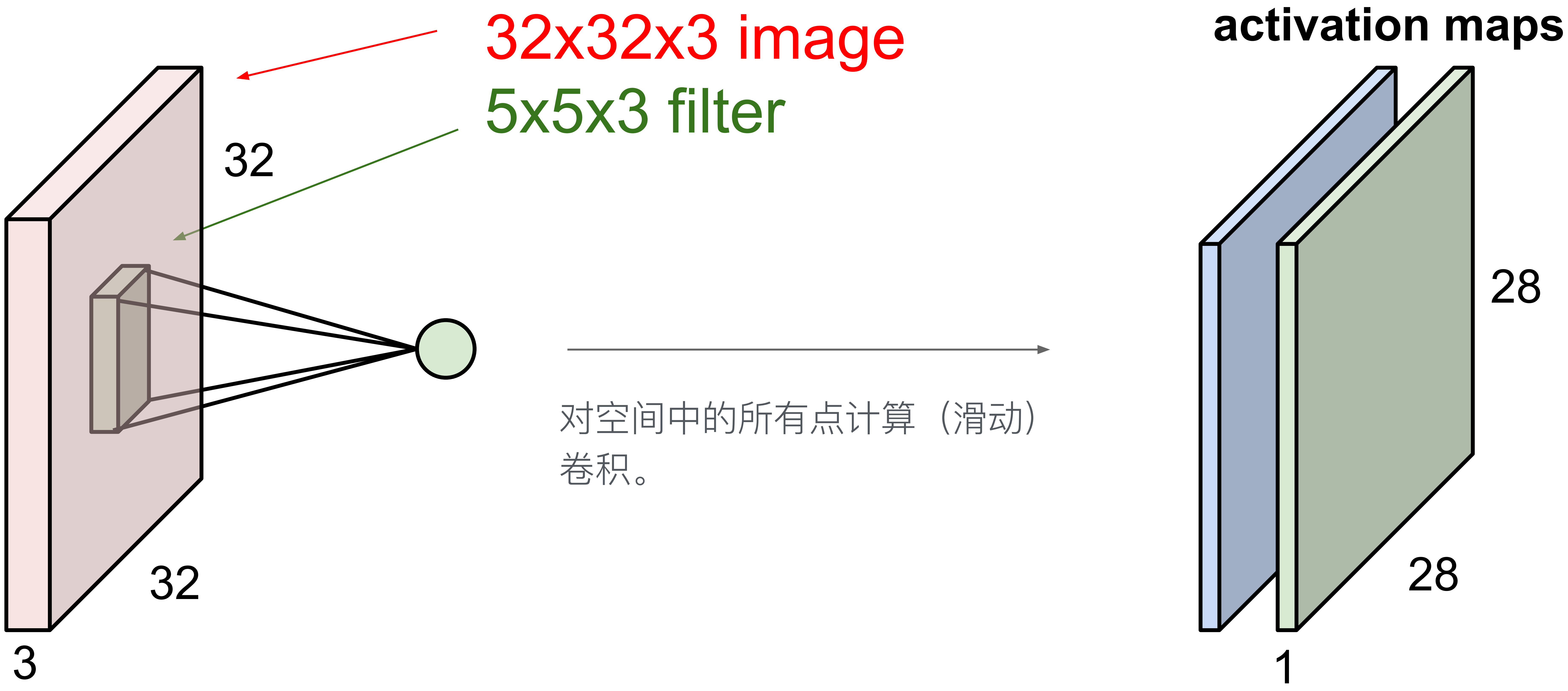
http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

Convolution Layer

http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

Convolution Layer

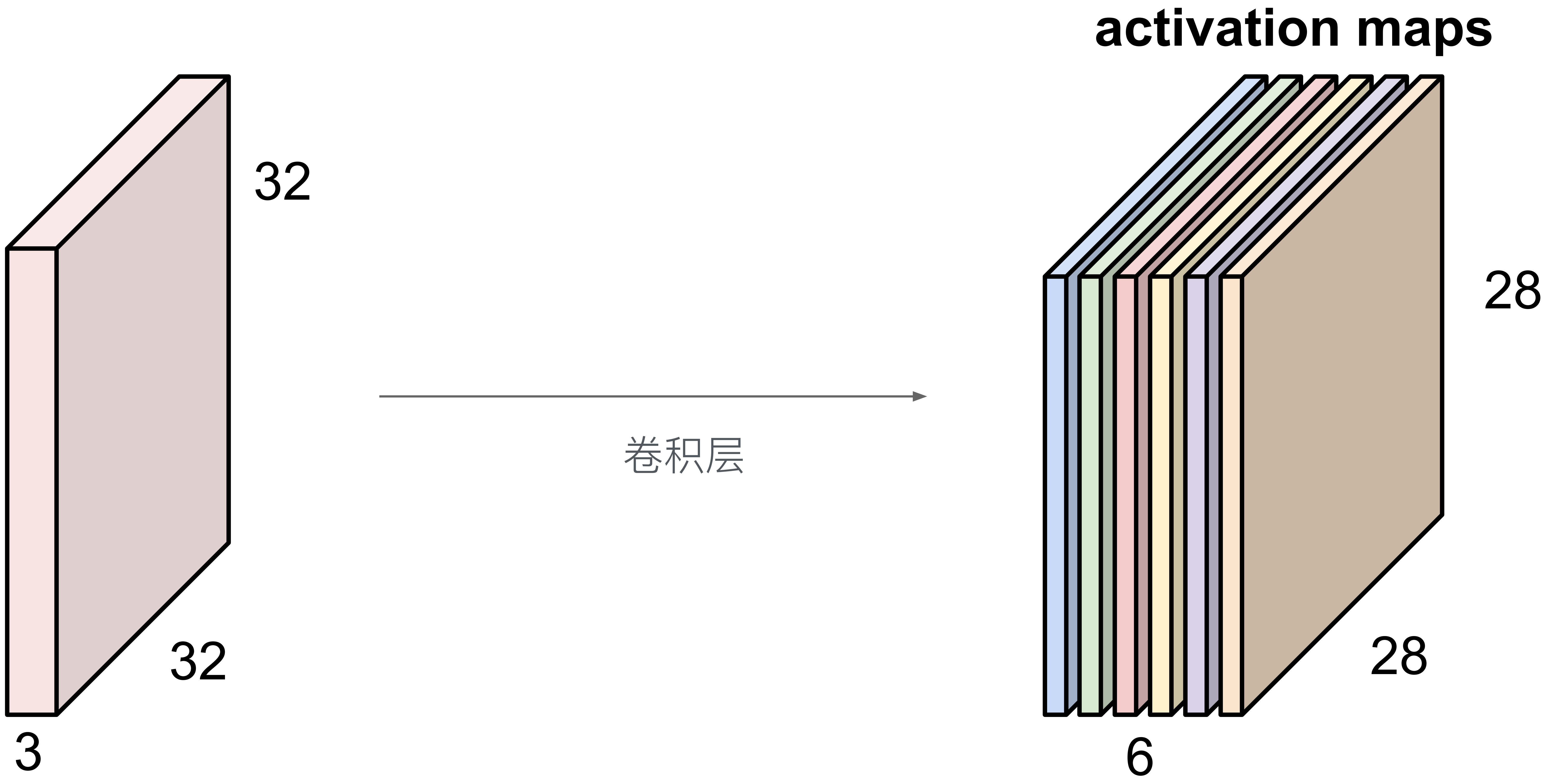
再添加另一个卷积核，见图中的绿色部分。



对空间中的所有点计算（滑动）卷积。

http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

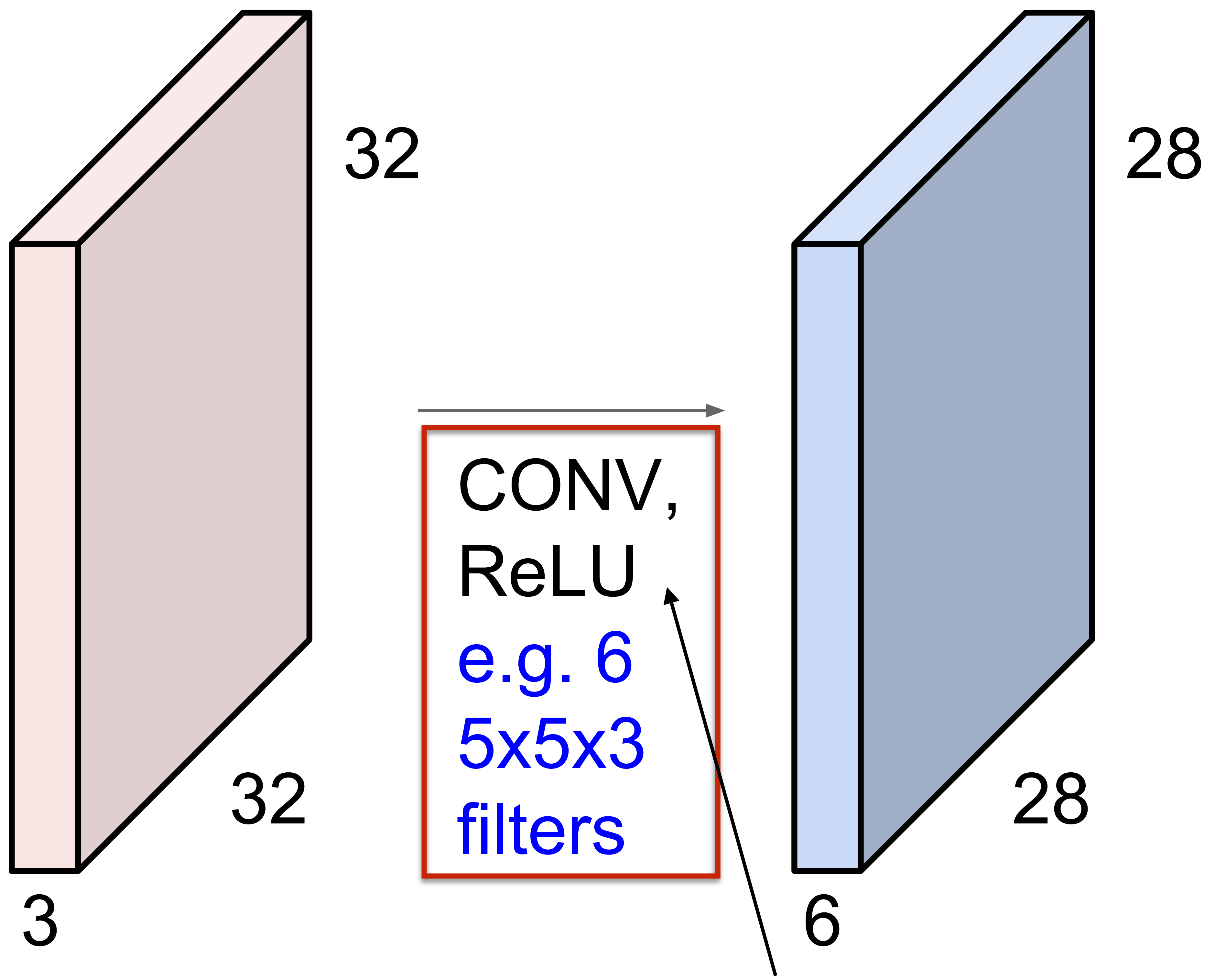
假如我们有6个5x5x3的卷积核，就会得到6个激活地图（activation map）。



将6个激活地图叠加在一起，就得到一个新的图片，尺寸是28x28x6。

http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

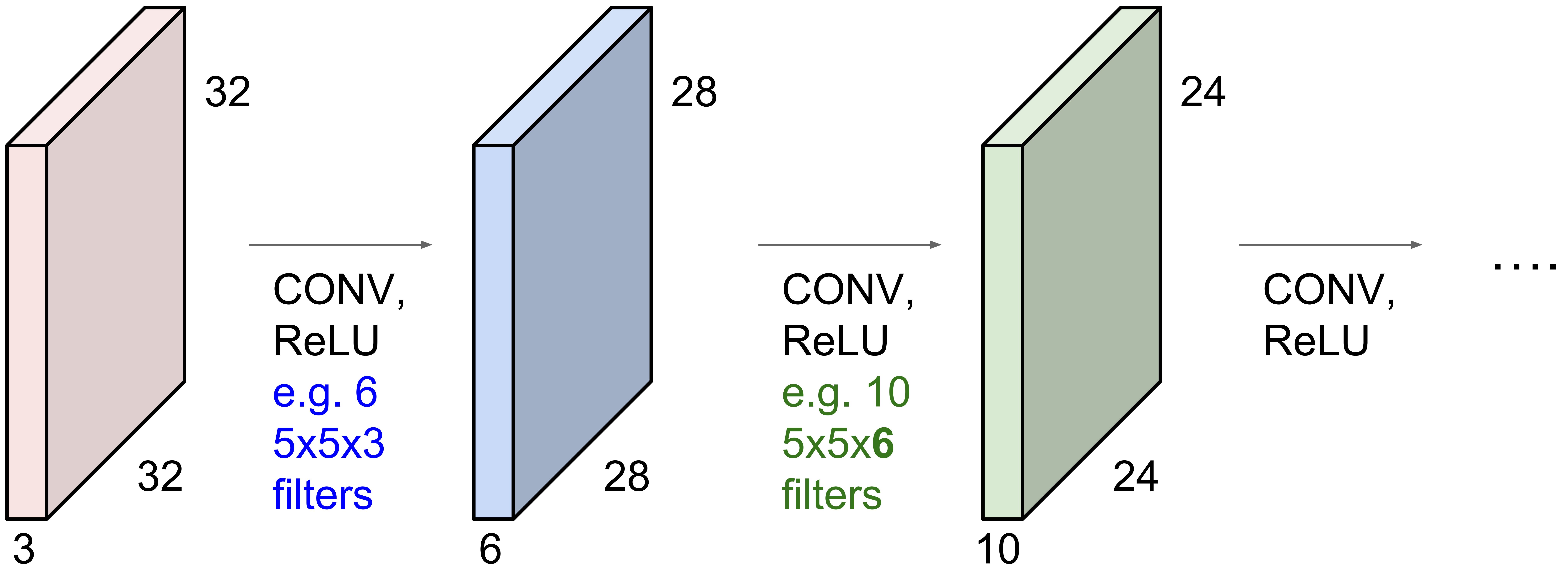
卷积网络（ConvNet）就是一系列卷积层，通过激活函数连接在一起。



最常见的激活函数ReLU

http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

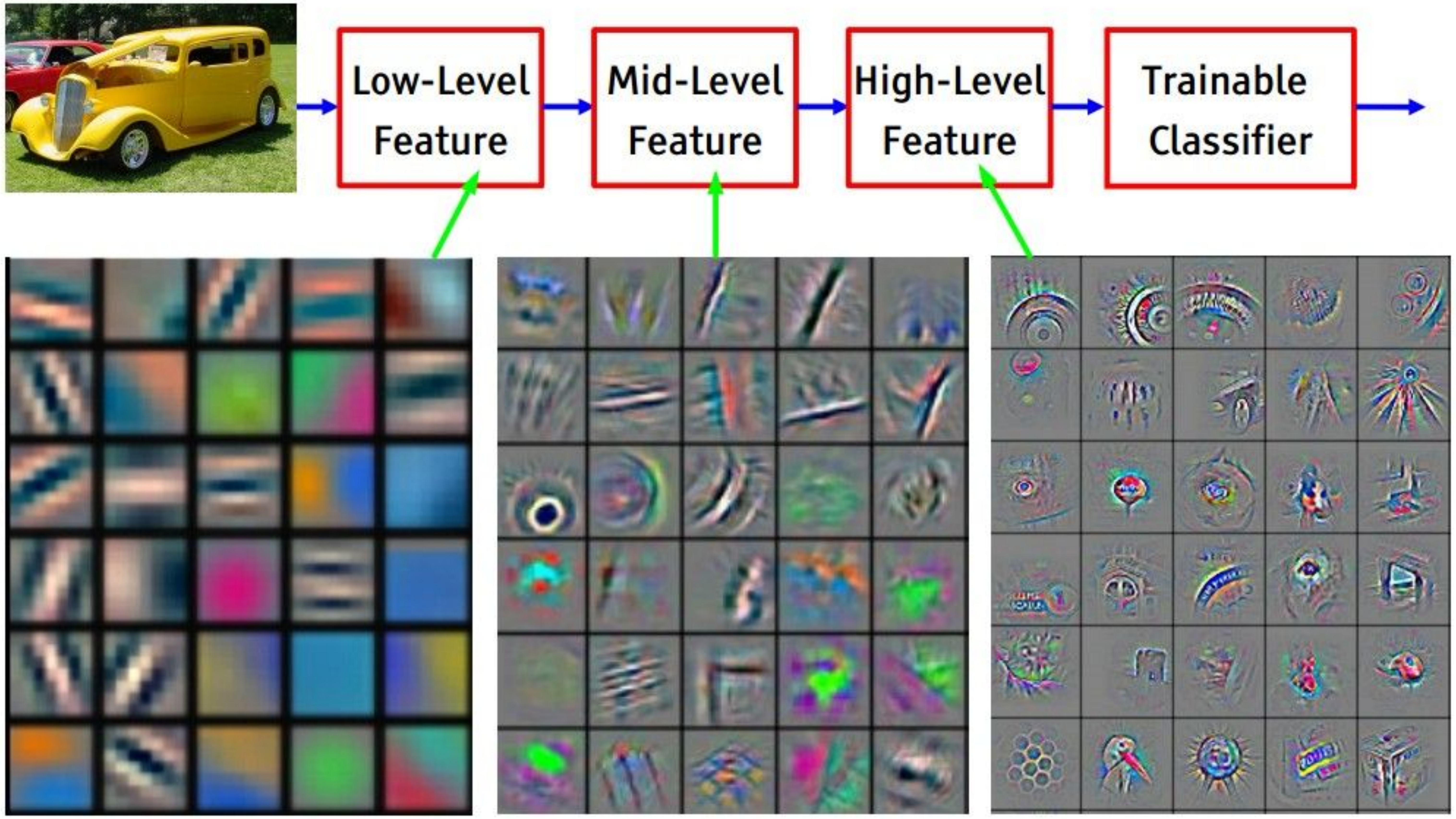
卷积网络（ConvNet）就是一系列卷积层，通过激活函数连接在一起。



http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

Preview

[From recent Yann LeCun slides]

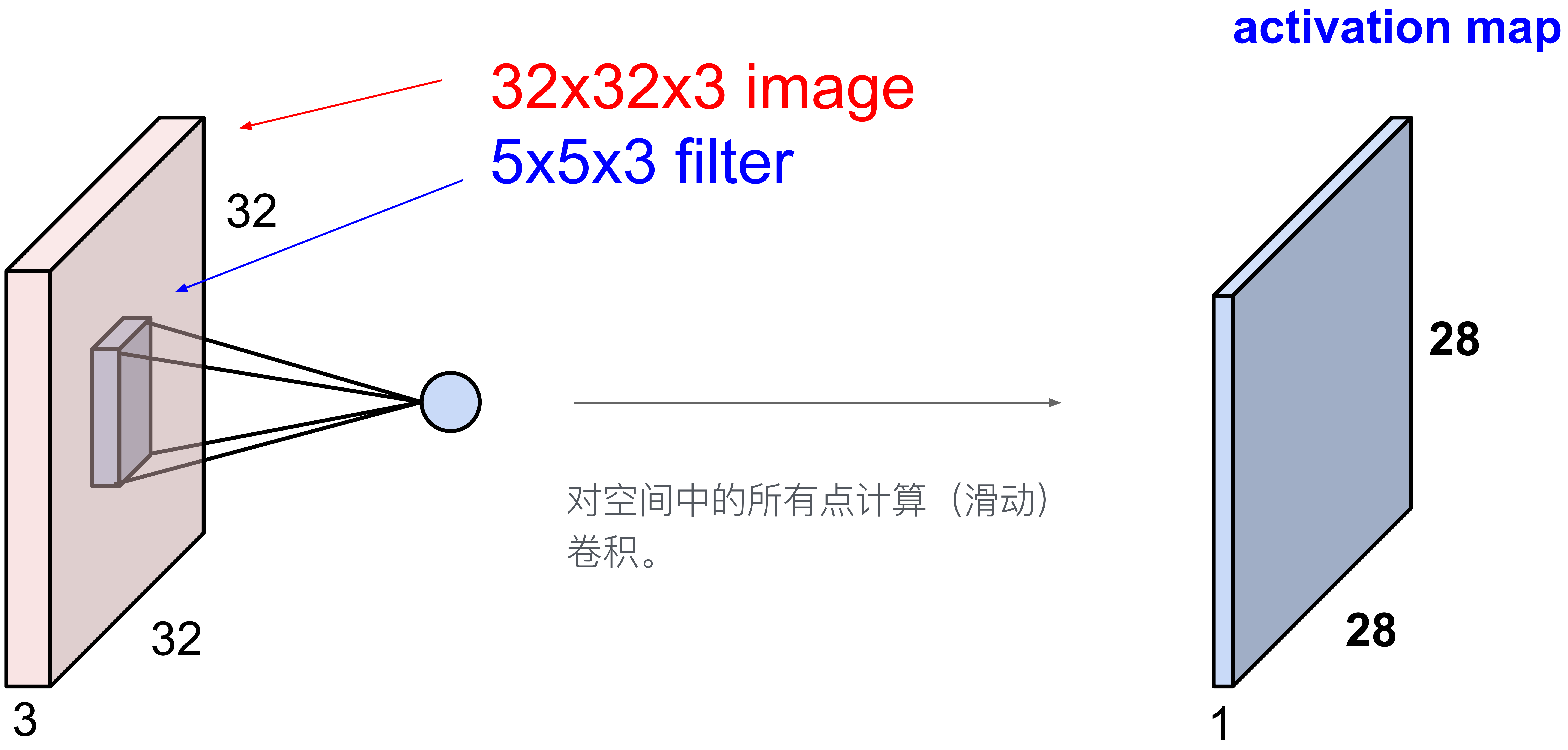


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

Stride & Padding

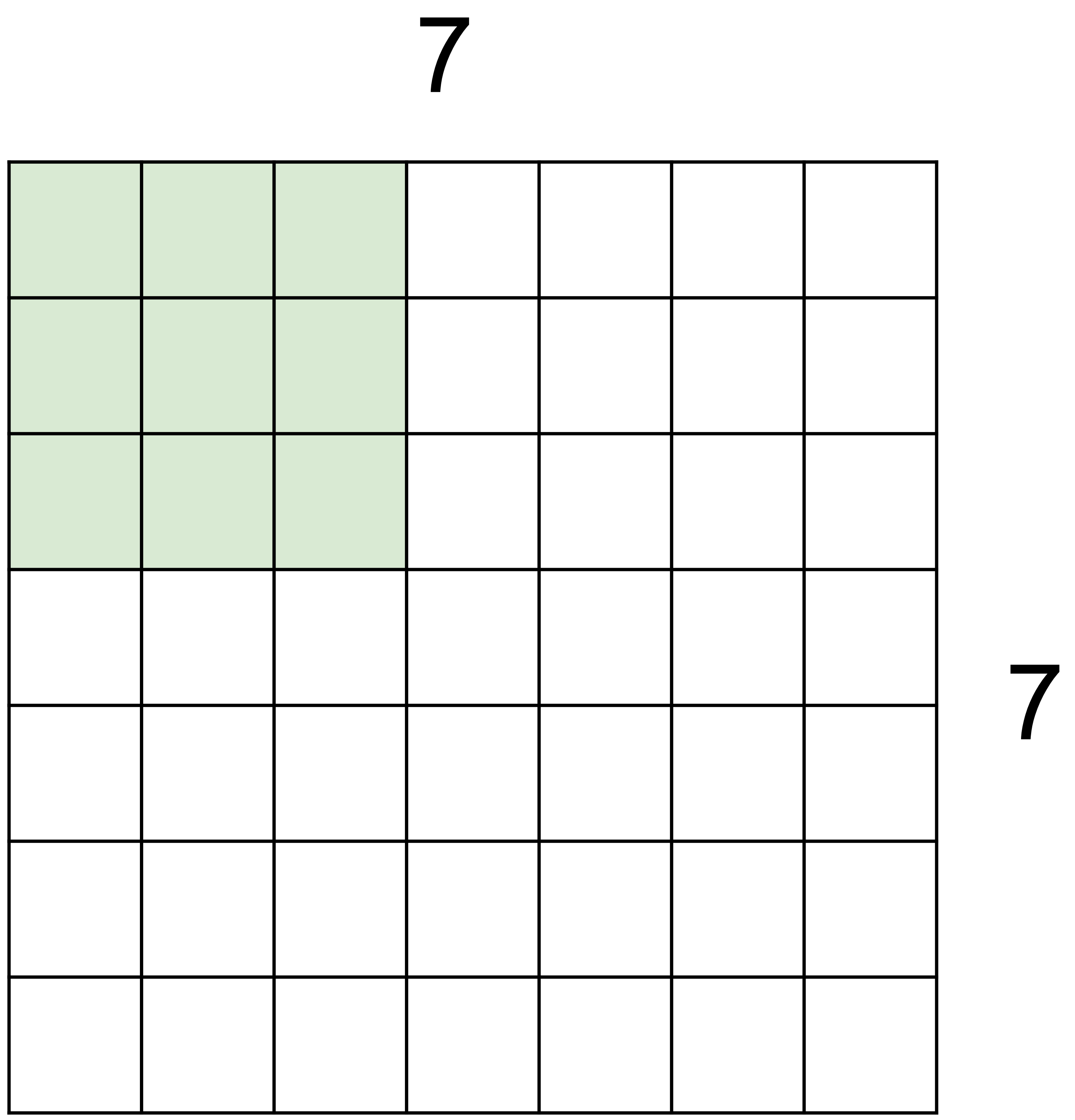
仔细看一下空间维度的变化。



对空间中的所有点计算（滑动）卷积。

Stride & Padding

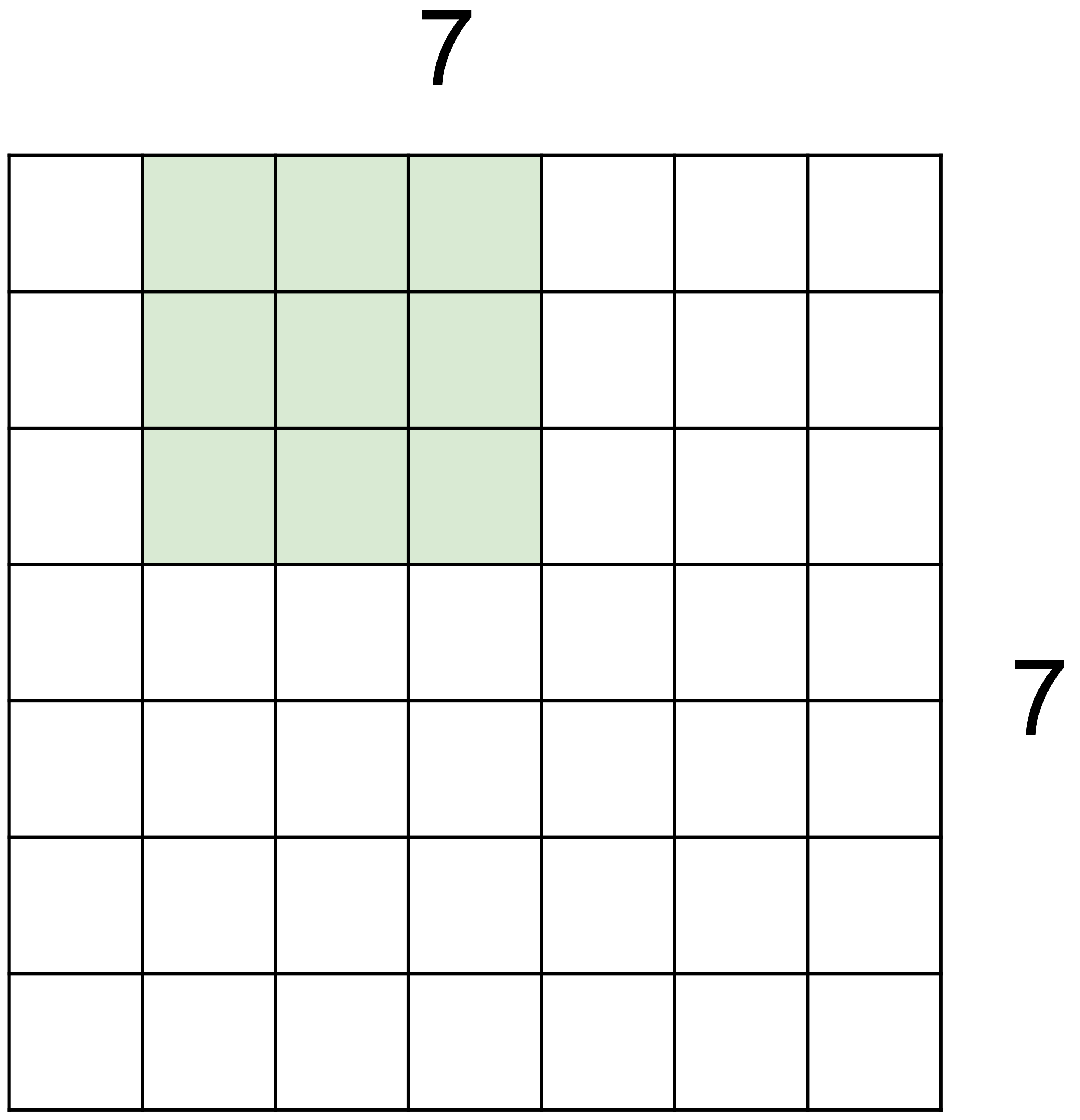
仔细看一下空间维度的变化。



7x7 input (spatially)
assume 3x3 filter

Stride & Padding

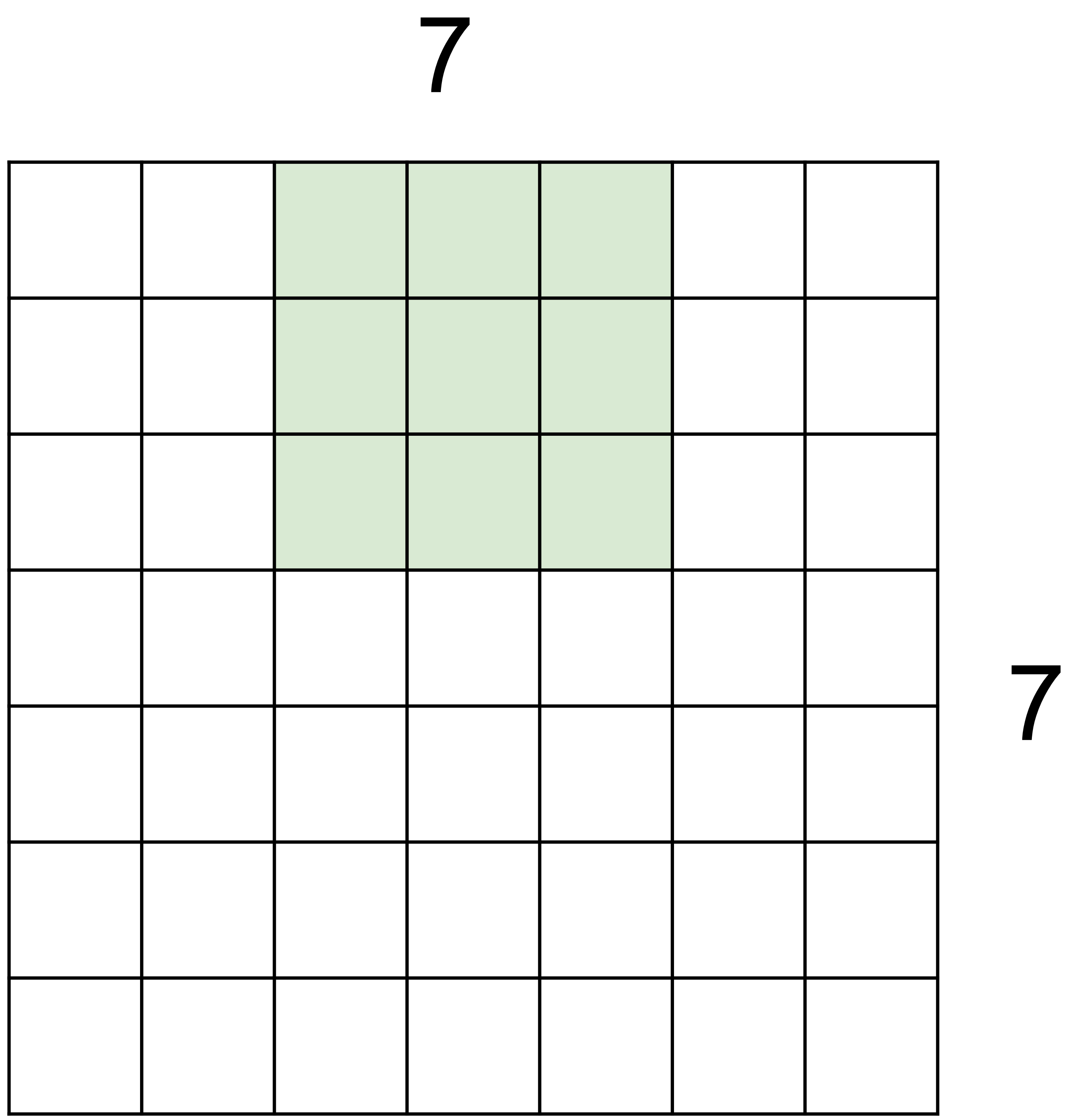
仔细看一下空间维度的变化。



7x7 input (spatially)
assume 3x3 filter

Stride & Padding

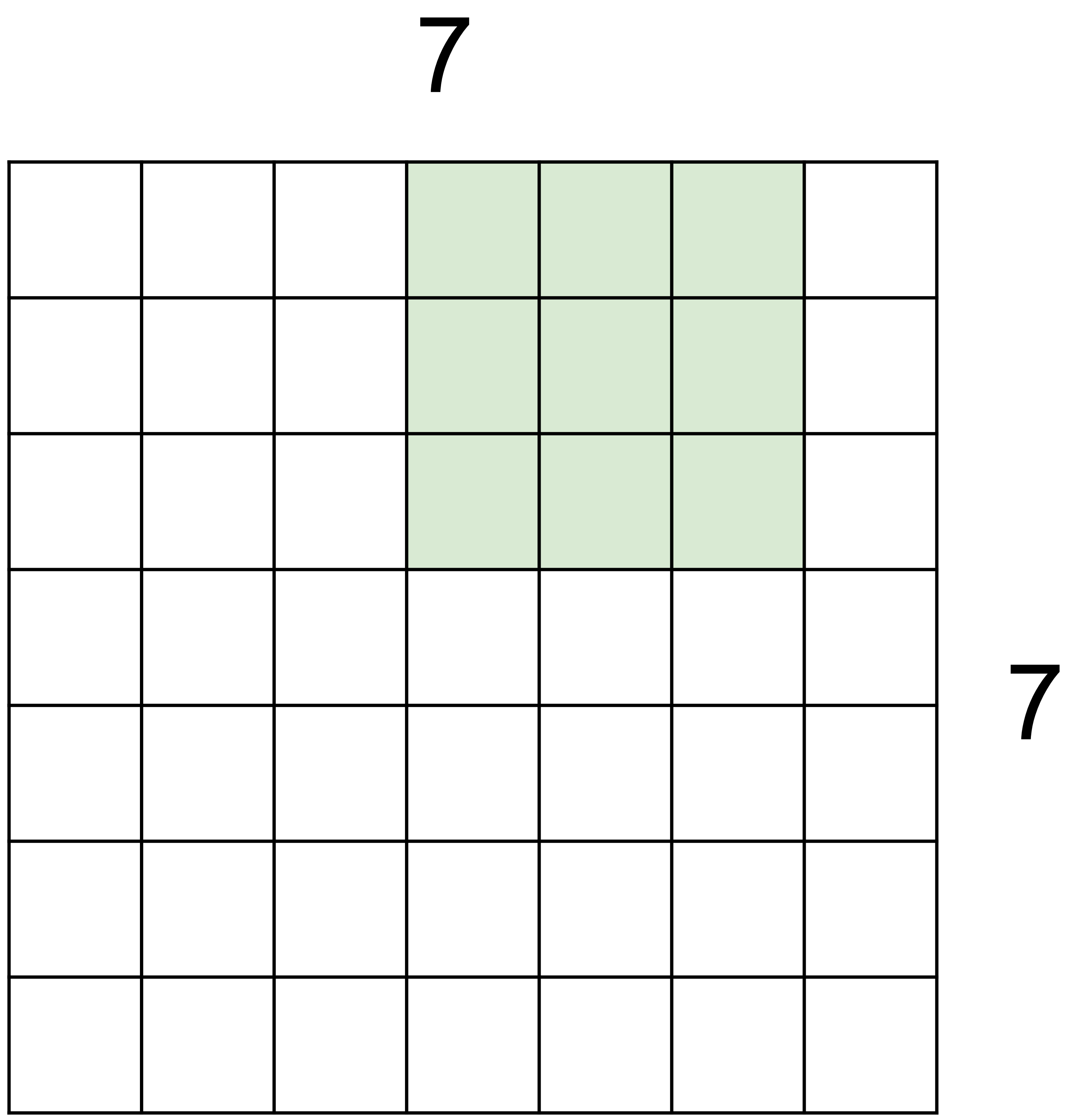
仔细看一下空间维度的变化。



7x7 input (spatially)
assume 3x3 filter

Stride & Padding

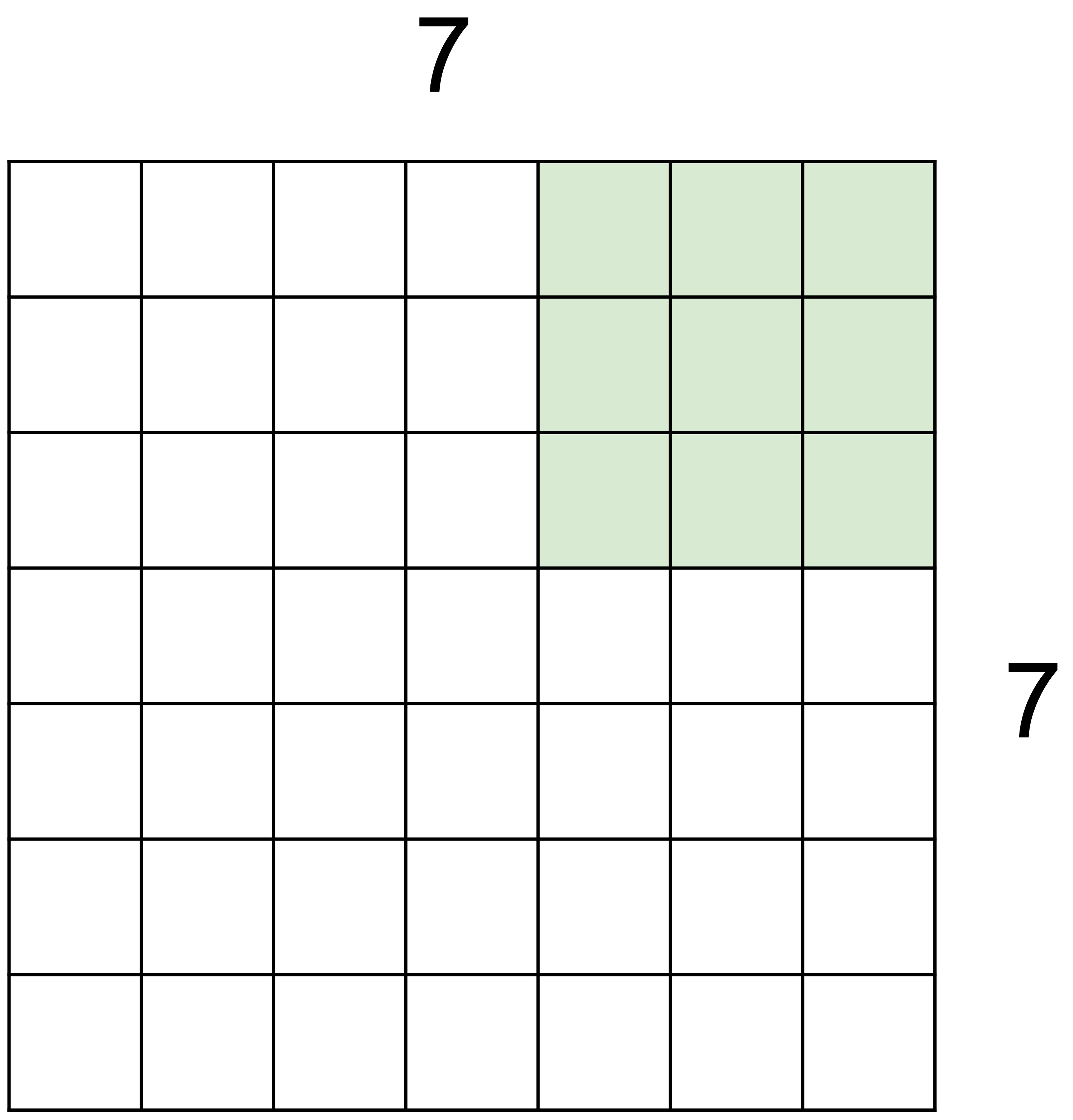
仔细看一下空间维度的变化。



7x7 input (spatially)
assume 3x3 filter

Stride & Padding

仔细看一下空间维度的变化。

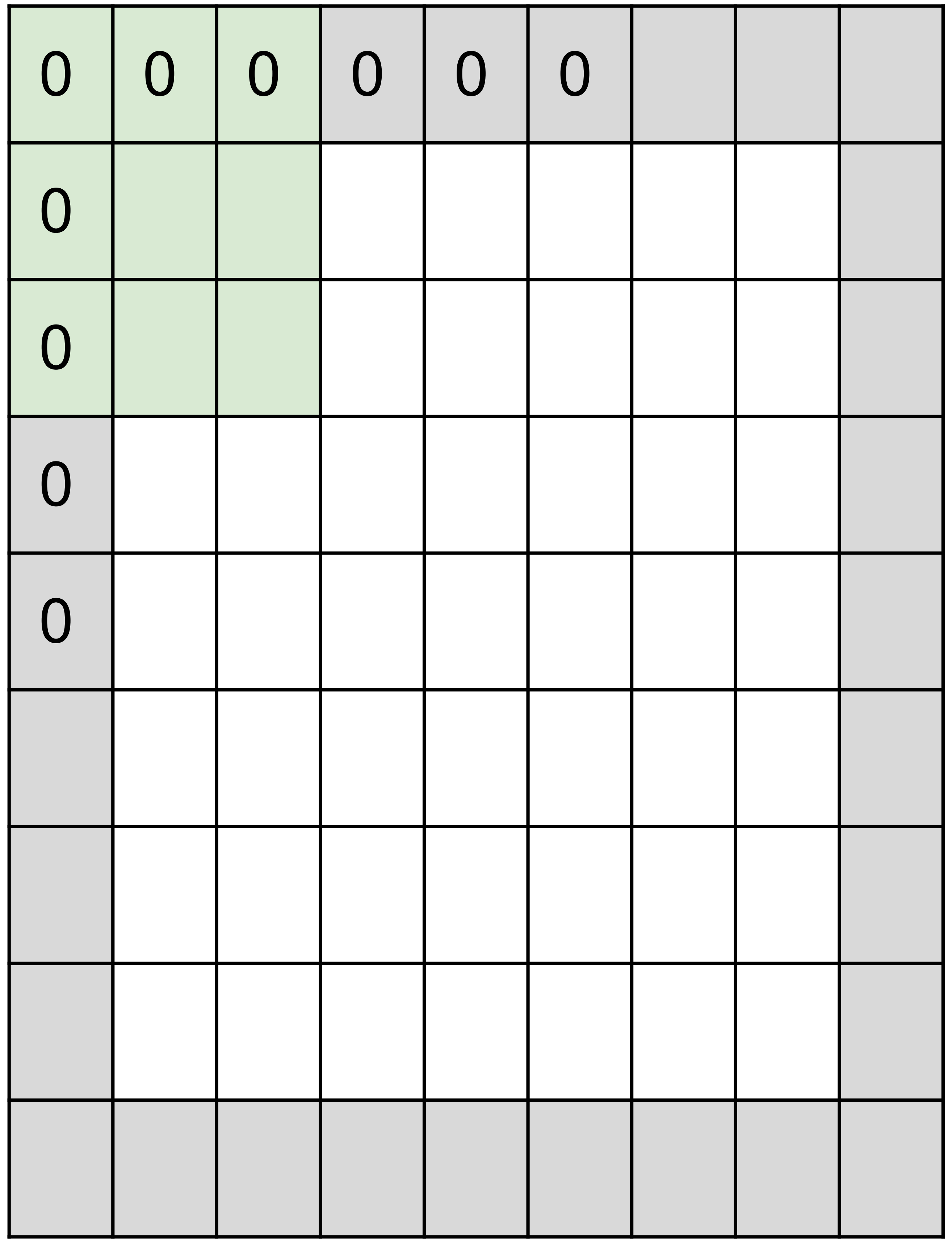


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Stride & Padding

实际使用中，通常用0来填充边缘。



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

常见的做法是步进为1的卷基层，卷积核为FxF，填充的0就是(F-1)/2，这样可以保证输出的图片大小不变。

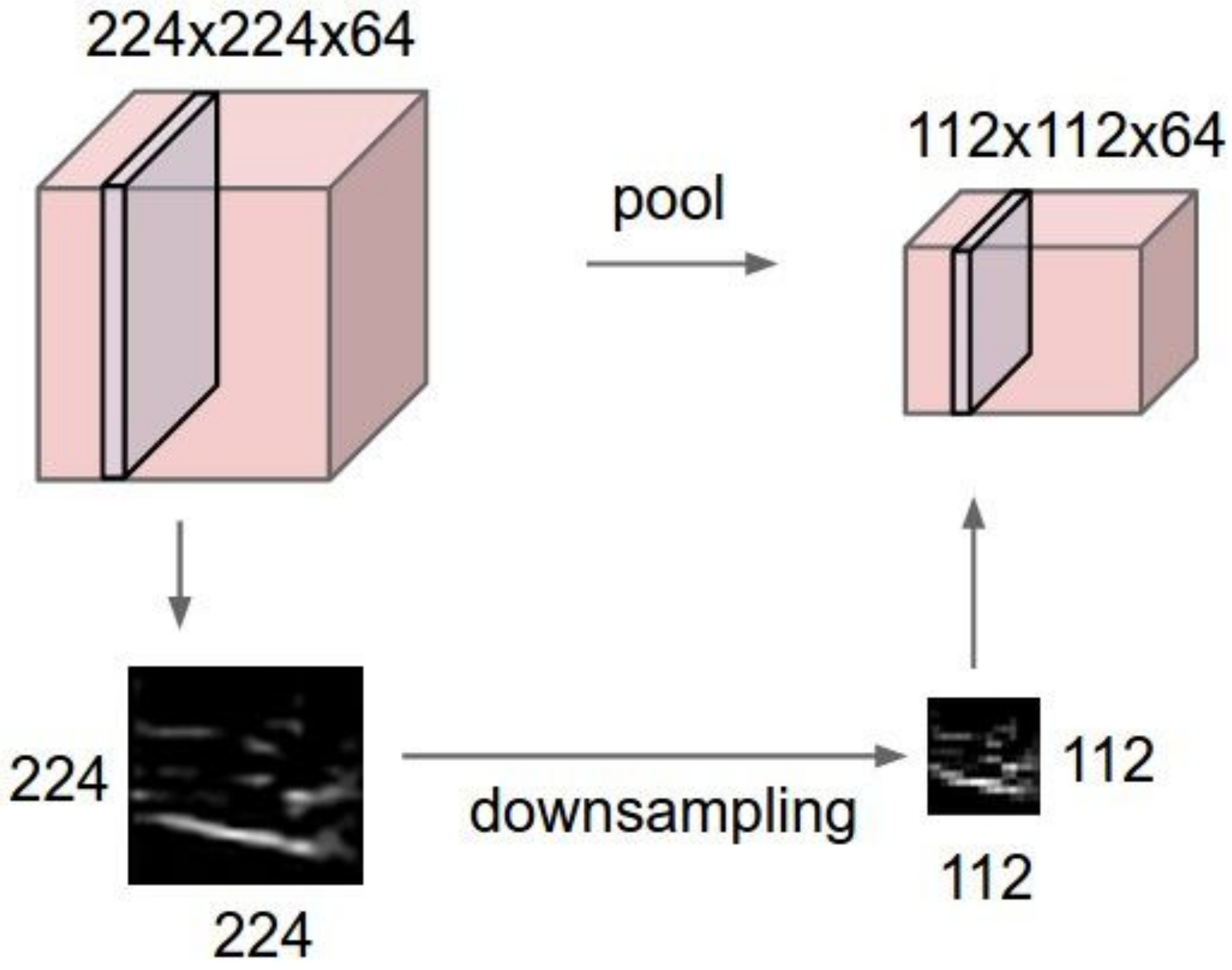
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

卷积神经网络 - 续

池化

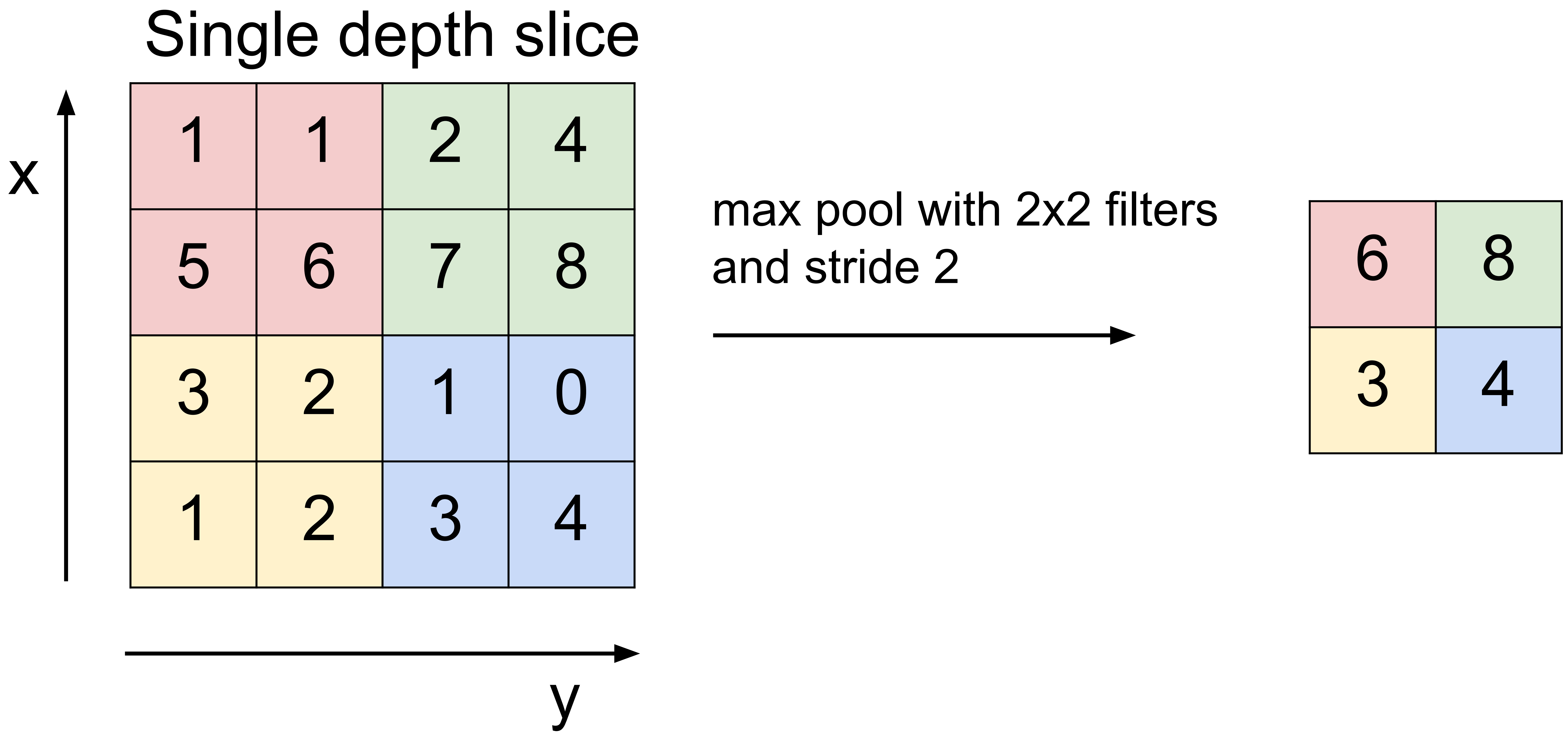
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



池化

MAX POOLING

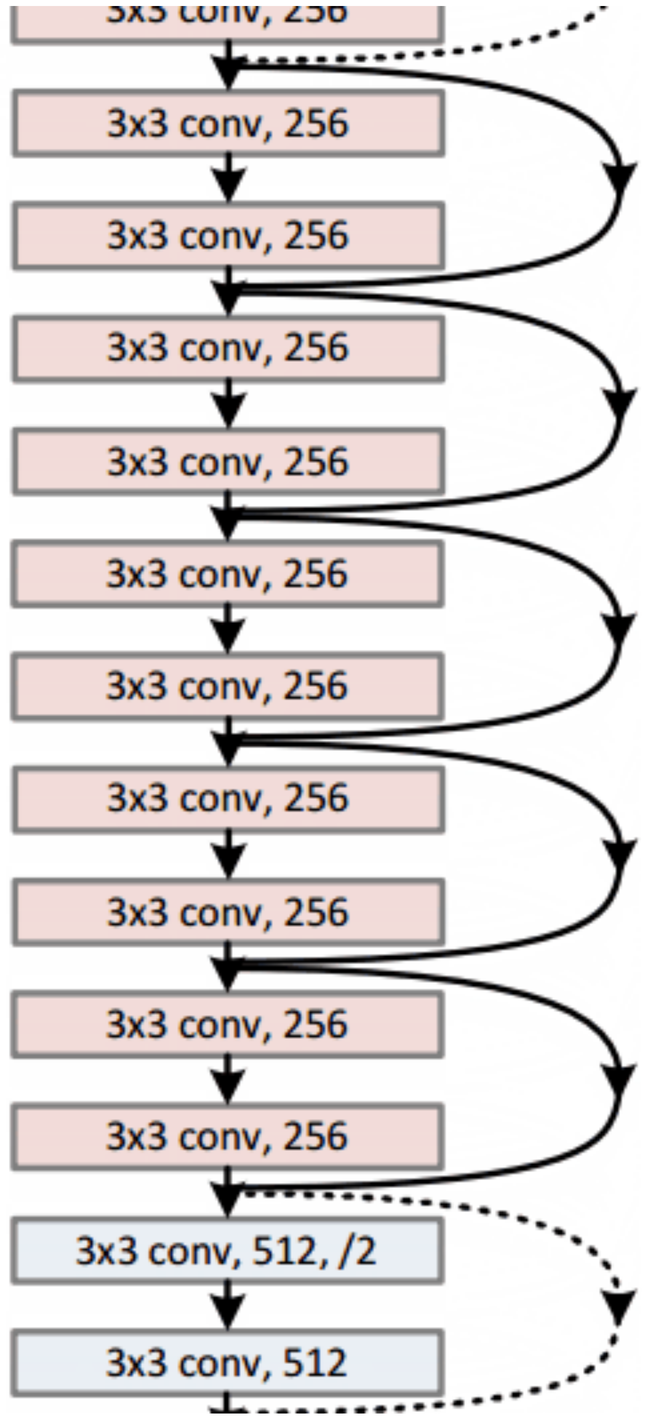


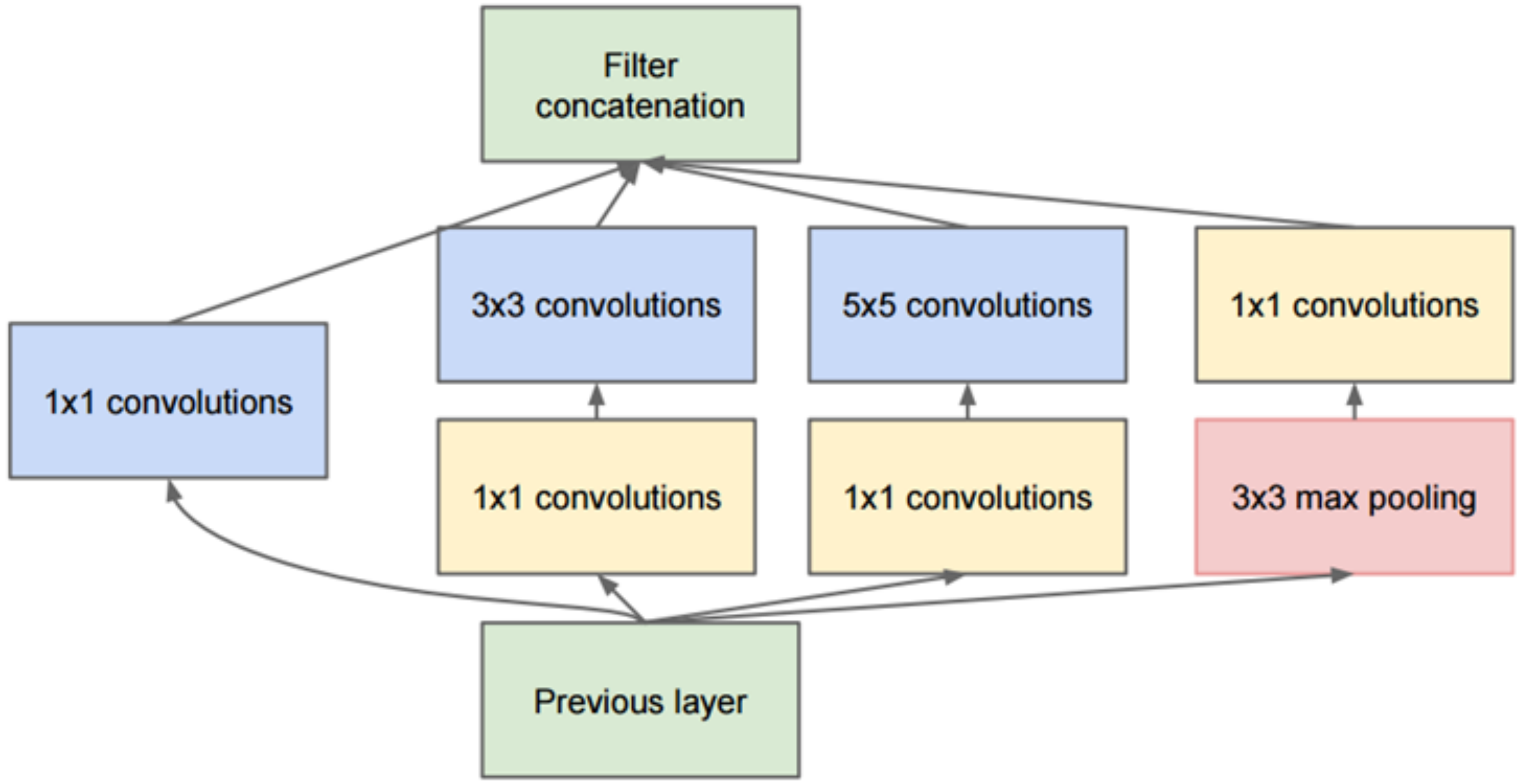
Regularization

- L2 Regularization
- Drop out
- Batch Norm

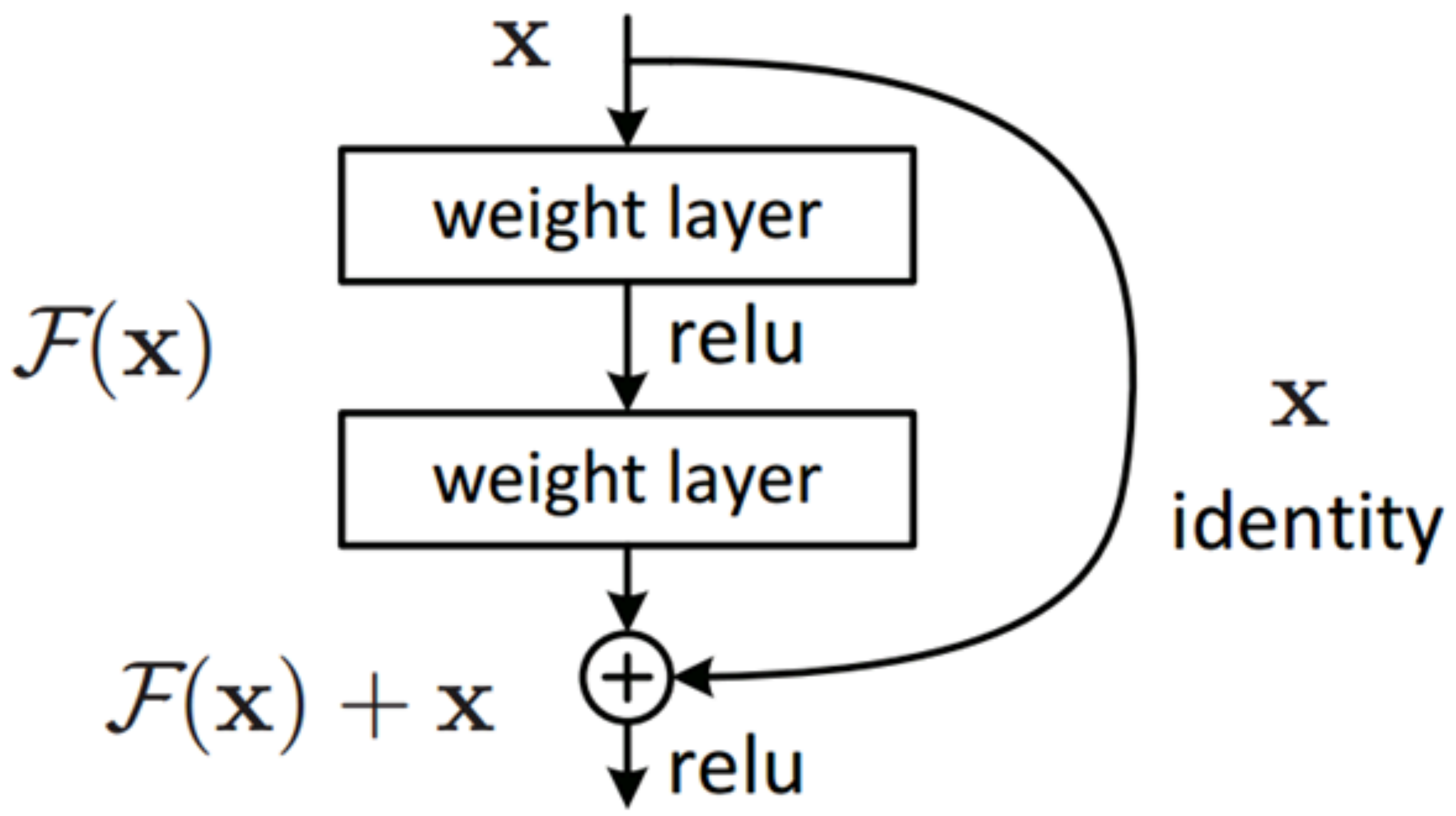
其他结构

- Inception
- ResNet





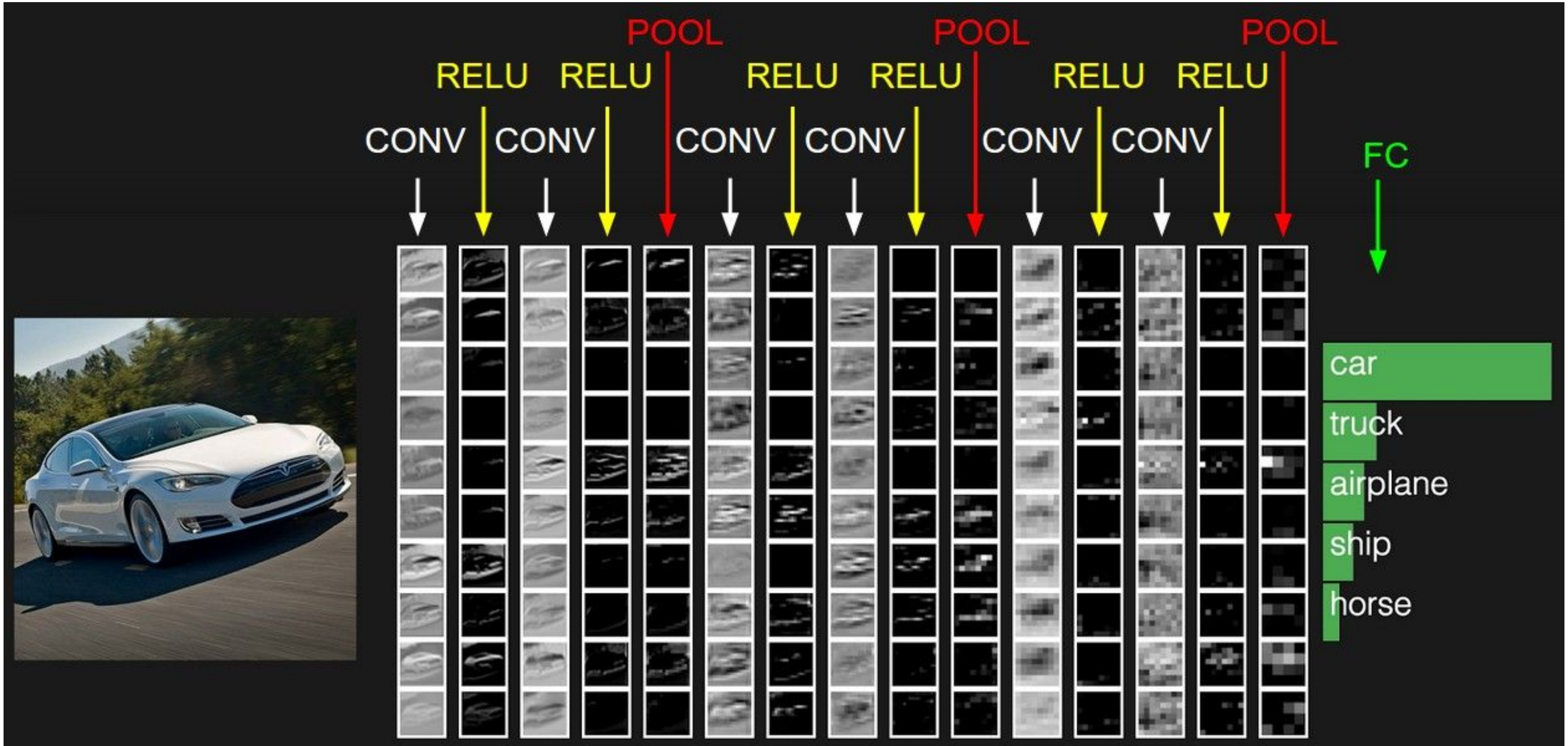
Full Inception module



A residual block

论文中的卷积网络

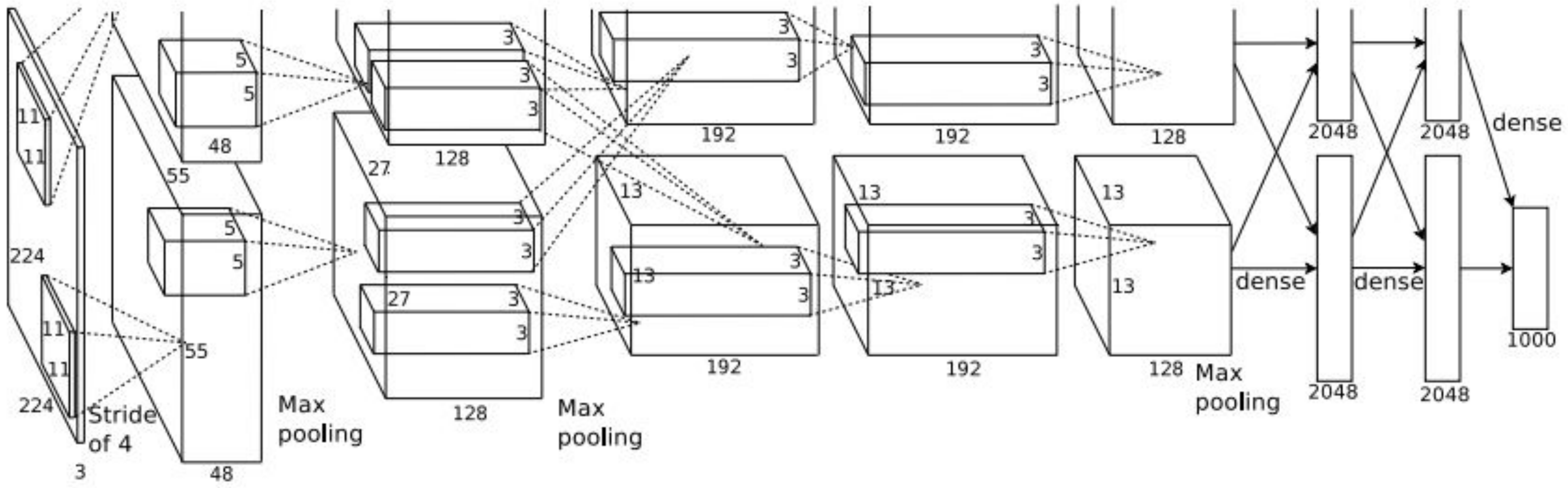
two more layers to go: POOL/FC



论文中的卷积网络

Case Study: AlexNet

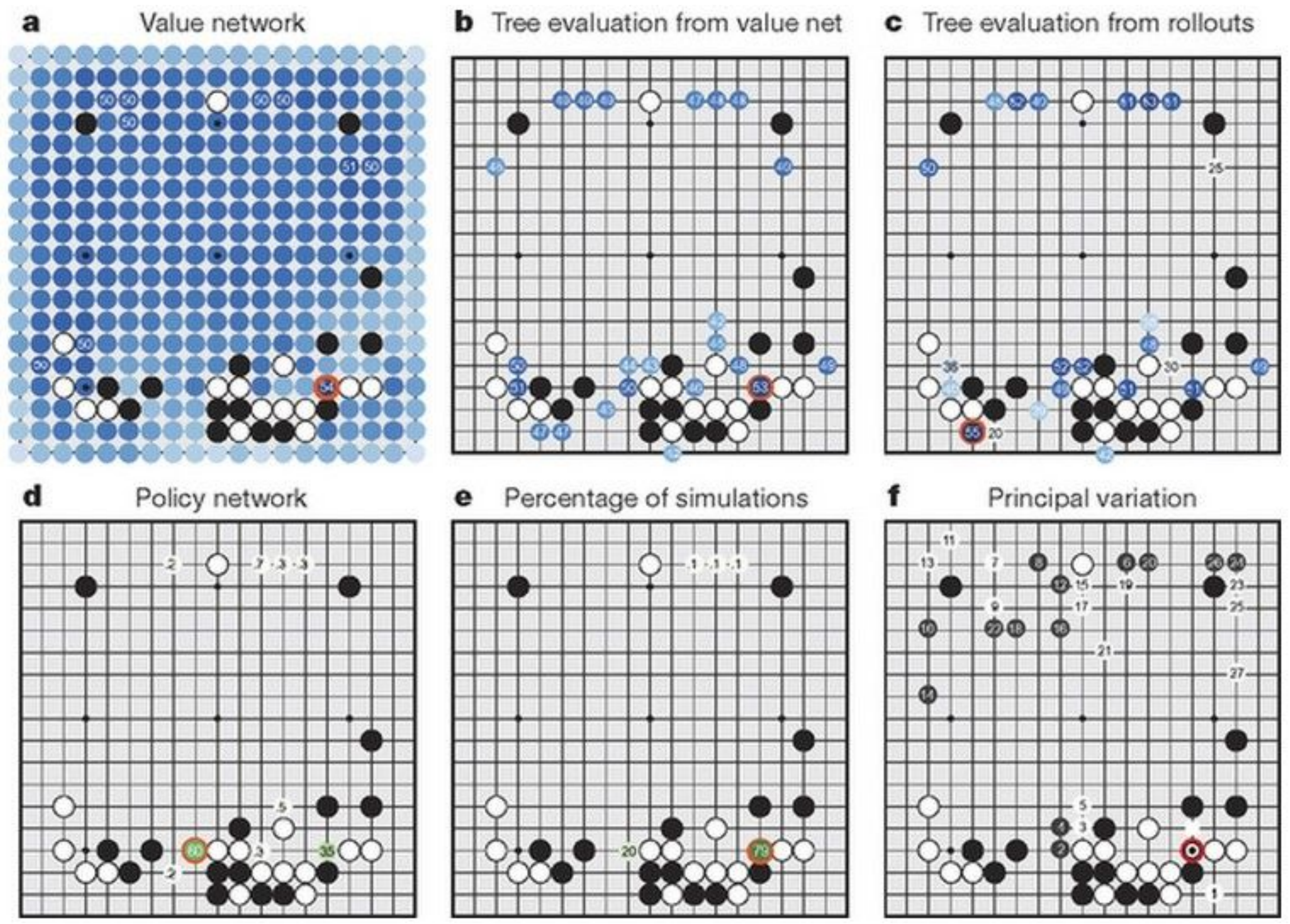
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

论文中的卷积网络

Case Study Bonus: DeepMind’s AlphaGo



论文中的卷积网络

The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

练习

对比不同模型性能

- Model Evaluation
- Cross Validation

对比不同模型性能

```
# Build graph to count correct answers.
def mnist_evaluation(logits, labels):
    """Evaluate the quality of the logits at predicting the label.
    Args:
    logits: Logits tensor, float - [BATCH_SIZE, NUM_CLASSES].
    labels: Labels tensor, int32 - [BATCH_SIZE], with values in the
    range [0, NUM_CLASSES).
    Returns:
    A scalar int32 tensor with the number of examples (out of batch_size)
    that were predicted correctly.
    """
    # For a classifier model, we can use the in_top_k Op.
    # It returns a bool tensor with shape [batch_size] that is true for
    # the examples where the label is in the top k (here k=1)
    # of all logits for that example.
    correct = tf.nn.in_top_k(logits, labels, 1)
    # Return the number of true entries.
    return tf.reduce_sum(tf.cast(correct, tf.int32))
```

对比不同模型性能

```
# Print out loss value.
    if step % 100 == 0:
        eval_value = sess.run(eval_op,
                              feed_dict={images_placeholder: images_feed,
                                         labels_placeholder: labels_feed})
        print('Step %d: loss = %.2f, eval = %.2f' % (step, loss_value, eval_value / BATCH_SIZE))
```