《神经网络与深度学习》

# 卷积神经网络

https://nndl.github.io/

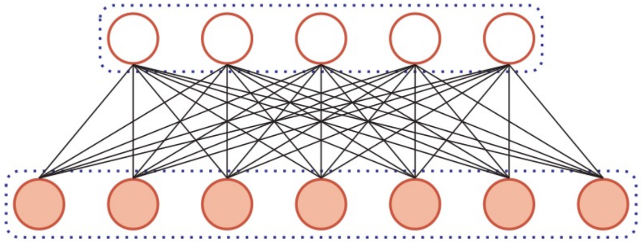
## 内容

- 卷积神经网络
  - 卷积
  - 卷积神经网络
  - 卷积神经网络的简单实现
  - 其他种类的卷积
- 卷积神经网络的扩展
  - 典型的卷积神经网络(*LeNet, AlexNet, VGG, NiN, ...*)
  - 卷积神经网络的应用

# 卷积

# 全连接前馈神经网络

- 权重矩阵的参数非常多

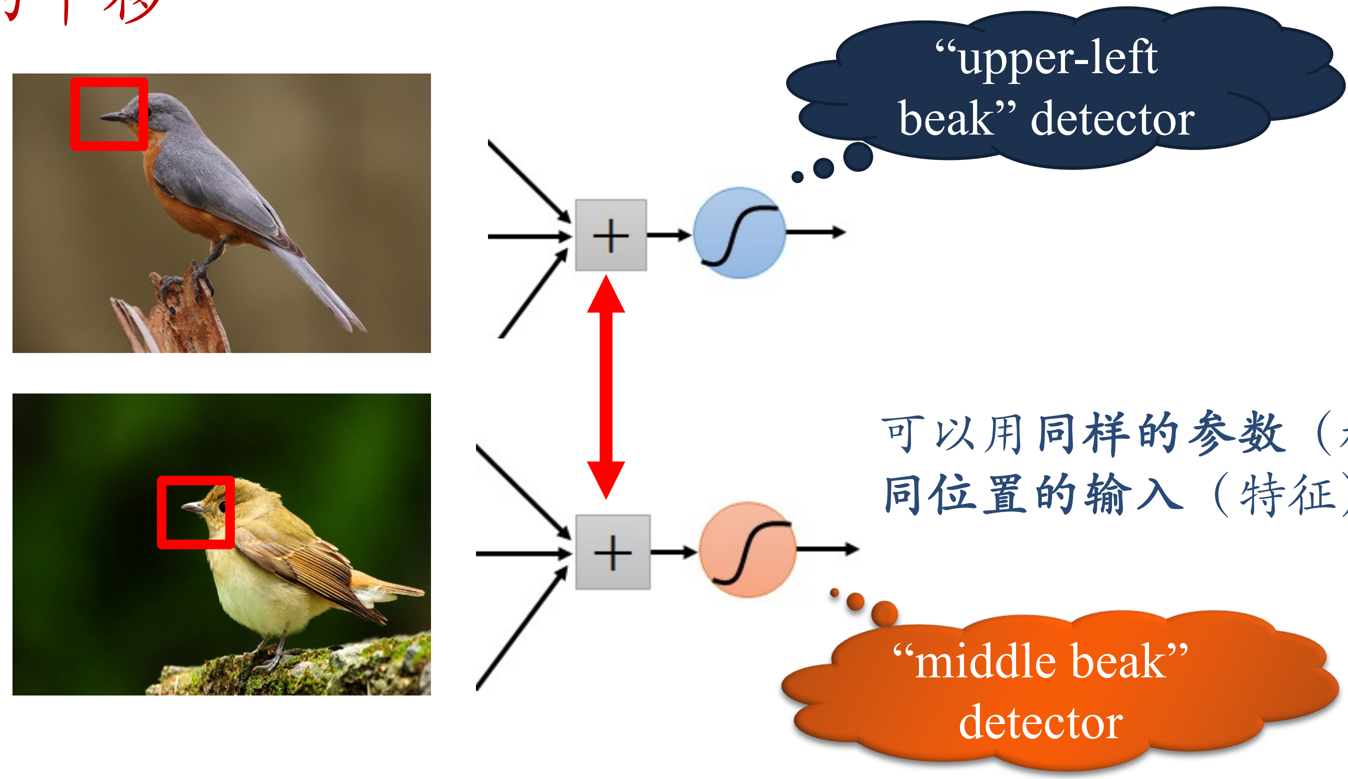


# 动机

- 局部不变性特征
  - 自然图像中的物体都具有局部不变性特征
    - 尺度缩放、平移、旋转等操作不影响其语义信息。
  - 全连接前馈网络很难提取这些局部不变特征

# 动机

- 特征的平移




可以用**同样的参数**（权重）与不同位置的输入（特征）相乘

# 动机

- 特征向下/次采样(sub-sampling)

bird

缩小

bird

我们可以次采样像素点，用**更少的参数**（权重）去处理图像

# 卷积神经网络

- 卷积神经网络（Convolutional Neural Networks，CNN）
  - 一种前馈神经网络
  - 受生物学上感受野（Receptive Field）的机制而提出的
    - 在视觉神经系统中，一个神经元的感受野是指视网膜上的特定区域，只有这个区域内的刺激才能够激活该神经元。

- 卷积神经网络有三个结构上的特性：
  - 局部连接
  - 权重共享
  - 空间或时间上的次采样

# 卷积

- 卷积经常用在信号处理中，用于计算信号的延迟累积。
- 假设一个信号发生器每个时刻t产生一个信号$x_t$，其信息的衰减率为$w_k$，即在k−1个时间步长后，信息为原来的$w_k$倍
  - 假设$w_1 = 1, w_2 = 1/2, w_3 = 1/4$
- 时刻t收到的信号$y_t$为当前时刻产生的信息和以前时刻延迟信息的叠加。

# 卷积

- 卷积经常用在信号处理中，用于计算信号的延迟累积。
- 假设一个信号发生器每个时刻t产生一个信号$x_t$，其信息的衰减率为$w_k$，即在k−1个时间步长后，信息为原来的$w_k$倍
  - 假设$w_1 = 1, w_2 = 1/2, w_3 = 1/4$
- 时刻t收到的信号$y_t$为当前时刻产生的信息和以前时刻延迟信息的叠加。

$$
\begin{aligned}
y_t &= 1 \times x_t + 1/2 \times x_{t-1} + 1/4 \times x_{t-2} \\
&= w_1 \times x_t + w_2 \times x_{t-1} + w_3 \times x_{t-2} \\
&= \sum_{k=1}^{3} w_k \cdot x_{t-k+1}.
\end{aligned}
$$

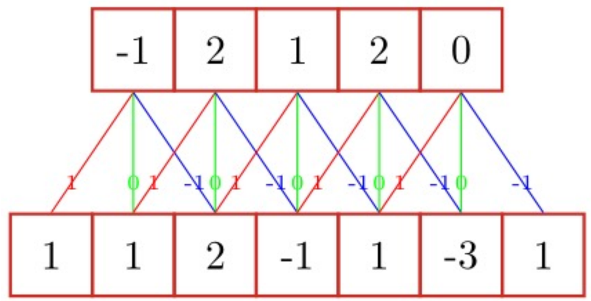
滤波器（filter）或卷积核（convolution kernel）

# 卷积

- 给定一个输入信号序列$x$和滤波器$w$,卷积的输出为:

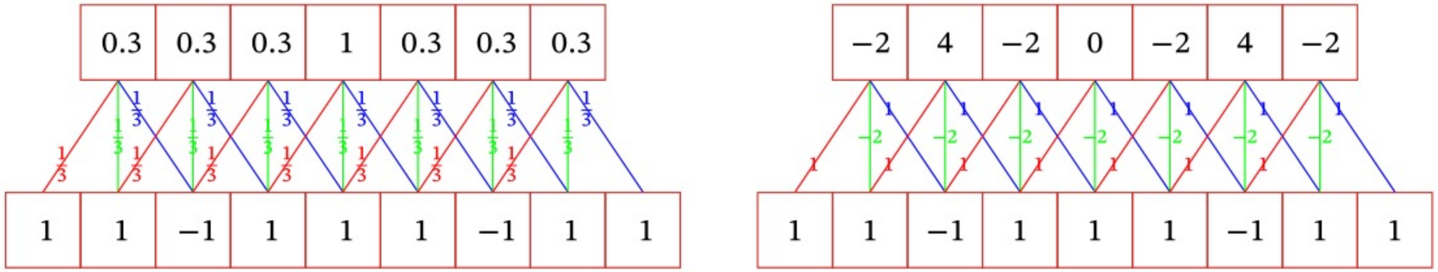
$$y_t = \sum_{k=1}^{K} w_k x_{t-k+1}$$

Filter: [-1,0,1]



# 卷积

- 不同的滤波器来提取信号序列中的不同特征




(a) 滤波器 [1/3, 1/3, 1/3]

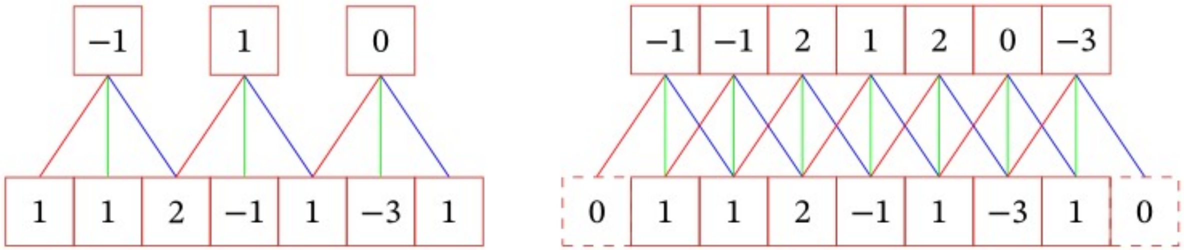
低频信息

(b) 滤波器 [1, −2, 1]

高频信息

# 卷积扩展

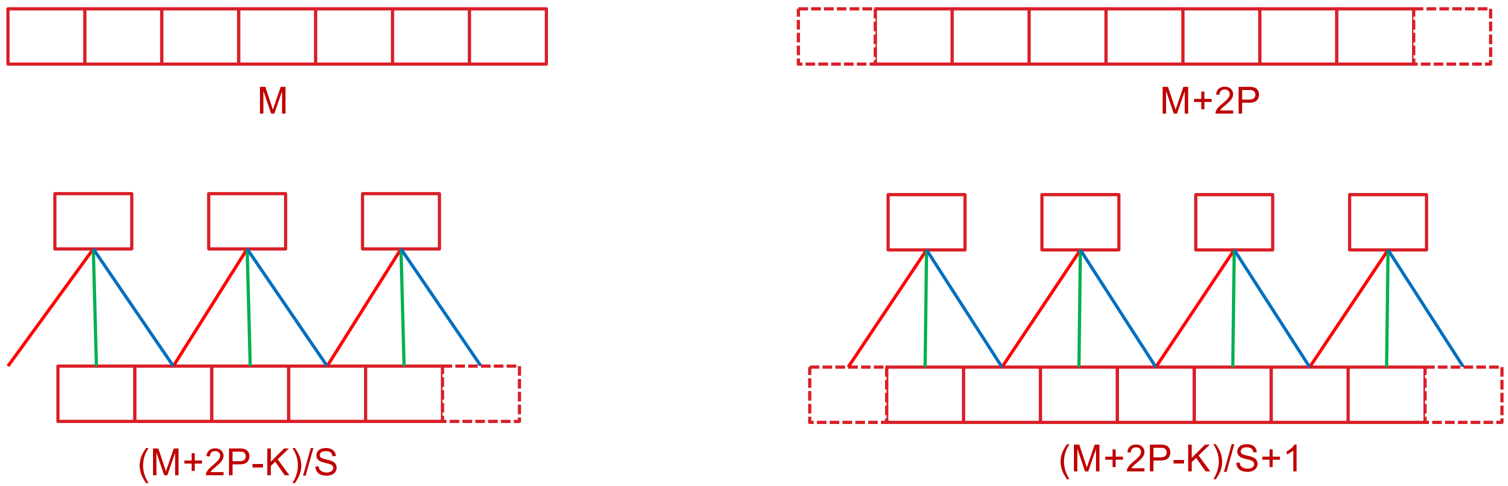
- 引入滤波器的滑动步长$S$和零填充$P$




(a) 步长 $S = 2$　　(b) 零填充 $P = 1$

# 卷积输出长度

- 假设卷积的输入长度为M，卷积大小为K，步长为S，再输入两端各填补P个0，那么该卷积输出长度为(M−K+2P)/S+1




选择合适的卷积大小使得(M−K+2P)/S+1为整数

# 卷积类型

- 卷积的结果按输出长度不同可以分为三类： (M+2P-K)/S+1
  - 窄卷积：步长 S = 1，两端不补零 $P = 0$，卷积后输出长度为 $M - K + 1$
  - 宽卷积：步长 S = 1，两端补零 $P = K - 1$，卷积后输出长度 $M + K - 1$
  - 等宽卷积：步长 S = 1，两端补零 $P = (K - 1)/2$，卷积后输出长度 $M$

- 在早期的文献中，卷积一般默认为窄卷积。
- 而目前的文献中，卷积一般默认为等宽卷积。

# 二维卷积

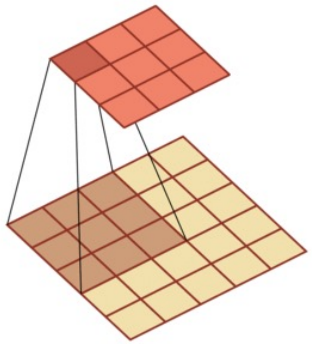
- 在图像处理中，图像是以二维矩阵的形式输入到神经网络中，因此我们需要二维卷积。

一个输入信息 $\boldsymbol{X}$ 和滤波器 $\boldsymbol{W}$ 的二维卷积定义为

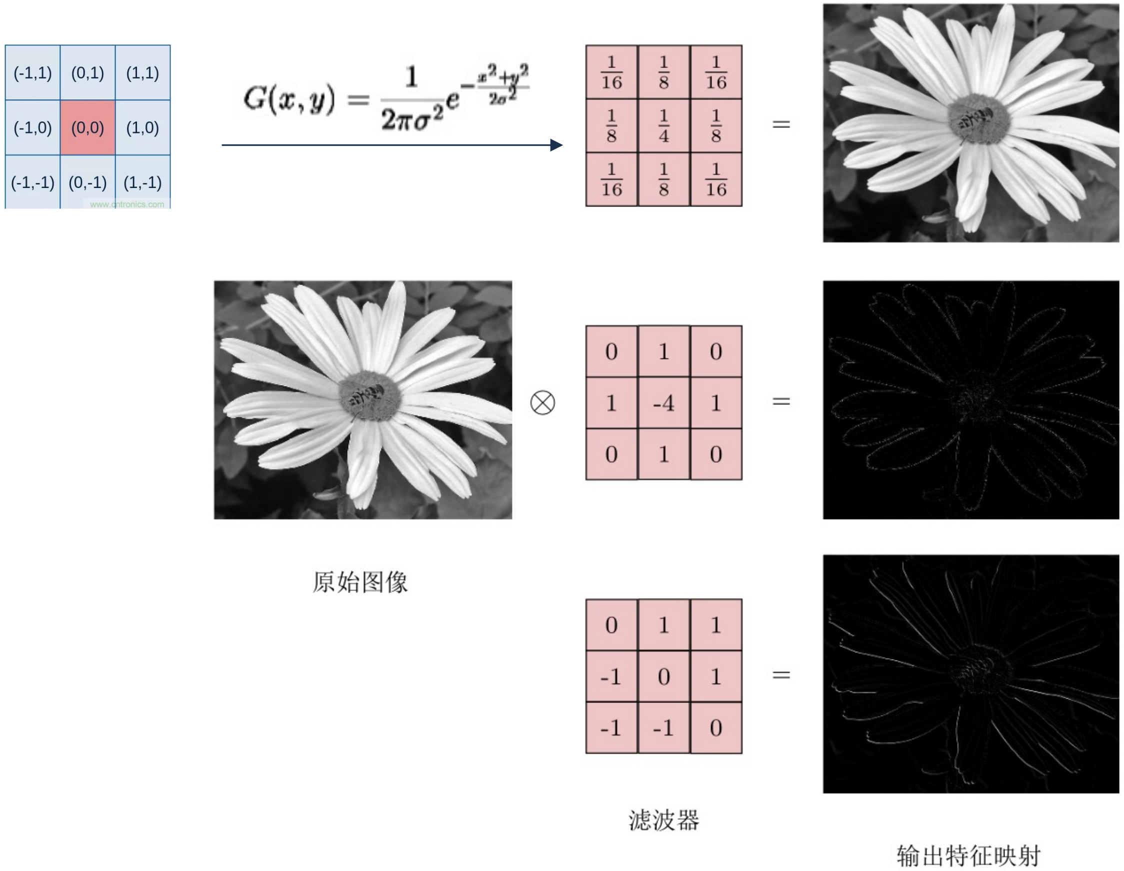
$$\boldsymbol{Y} = \boldsymbol{W} * \boldsymbol{X},$$

$$y_{ij} = \sum_{u=1}^{U} \sum_{v=1}^{V} w_{uv} x_{i-u+1,j-v+1}.$$

$$\begin{bmatrix} 1 & 1 & 1_{\times -1} & 1_{\times 0} & 1_{\times 0} \\ -1 & 0 & -3_{\times 0} & 0_{\times 0} & 1_{\times 0} \\ 2 & 1 & 1_{\times 0} & -1_{\times 0} & 0_{\times 1} \\ 0 & -1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & \boxed{-1} \\ 2 & 2 & 4 \\ -1 & 0 & 0 \end{bmatrix}$$

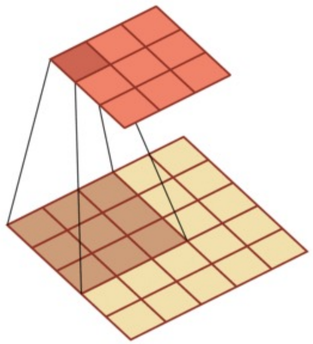


# 卷积作为特征提取器

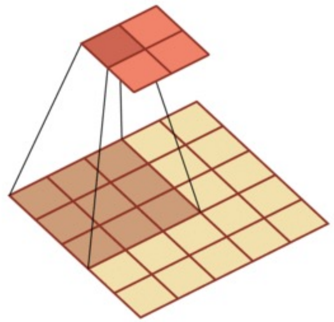


(-1,1) (0,1) (1,1)
(-1,0) (0,0) (1,0)
(-1,-1) (0,-1) (1,-1)
$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$
$\begin{bmatrix}\frac{1}{16} & \frac{1}{8} & \frac{1}{16}\\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8}\\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16}\end{bmatrix}$ =
$\otimes$
$\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix}$ =
$\begin{bmatrix}0 & 1 & 1\\ -1 & 0 & 1\\ -1 & -1 & 0\end{bmatrix}$ =
原始图像
滤波器
输出特征映射

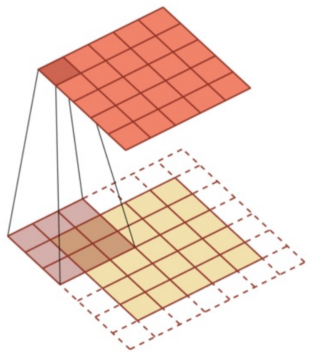
# 二维卷积



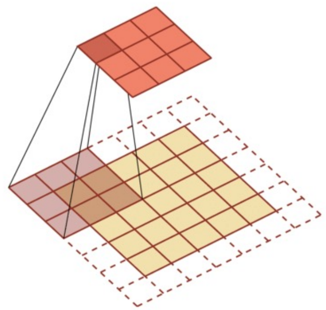
步长1，零填充0



步长2，零填充0



步长1，零填充1

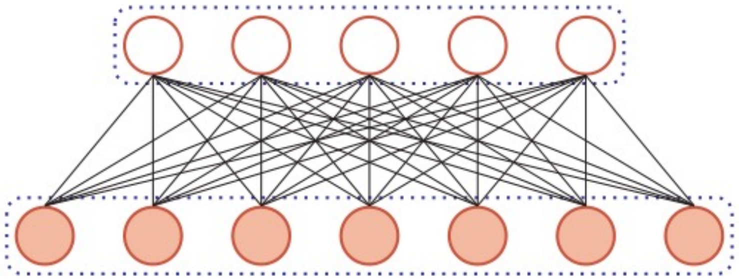


步长2，零填充1

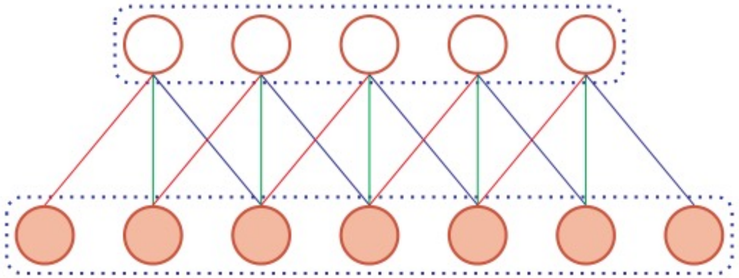
# 卷积神经网络

# 卷积神经网络

- 用卷积层代替全连接层



(a) 全连接层



(b) 卷积层

# 互相关

- 计算卷积需要进行卷积核翻转。
- 在神经网络中卷积操作的目标：提取特征，因此

翻转是不必要的！

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & -3 & 0 & 1 \\ 2 & 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & -1 \\ 2 & 2 & 4 \\ -1 & 0 & 0 \end{bmatrix}$$

- 互相关

$$y_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{n} w_{uv} \cdot x_{i+u-1,j+v-1}$$

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{W} \otimes \boldsymbol{X} \\ &= \mathrm{rot180}(\boldsymbol{W}) * \boldsymbol{X}, \end{aligned}$$

除非特别声明，卷积一般指“互相关”。

# 互相关

- 计算卷积需要进行卷积核翻转。
- 在神经网络中卷积操作的目标：提取特征，因此

翻转是不必要的！

- 互相关

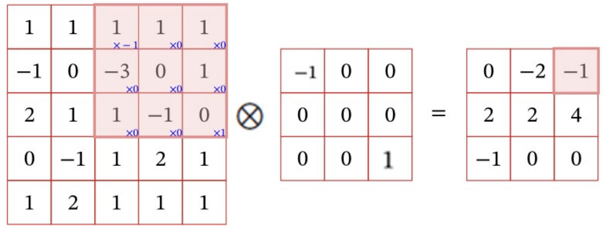
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & -3 & 0 & 1 \\ 2 & 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & -1 \\ 2 & 2 & 4 \\ -1 & 0 & 0 \end{bmatrix}$$

$$y_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{n} w_{uv} \cdot x_{i+u-1,j+v-1}$$

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{W} \otimes \boldsymbol{X} \\ &= \mathrm{rot180}(\boldsymbol{W}) * \boldsymbol{X}, \end{aligned}$$

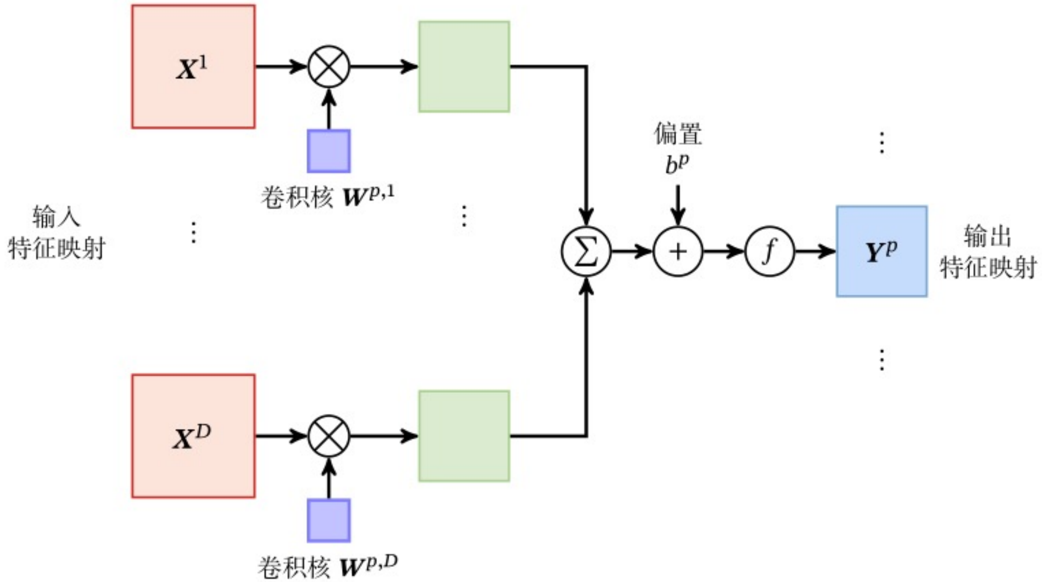
除非特别声明，卷积一般指“互相关”。

# 多个卷积核



- 特征映射（Feature Map）
  - 图像经过卷积后得到的特征。卷积核可看成一个特征提取器
- 感受野(Receptive Field)
  - 影响元素$x$(某数组或矩阵中的成员)的前向计算的所有可能的输入区域叫做$x$的感受野
- 卷积层
  - 输入：D个特征映射 $M \times N \times D$
  - 输出：P个特征映射 $M' \times N' \times P$
  - 大小为D的这一维称为输入通道(in channel)维，大小为P的这一维称为输出通道(out channel)维

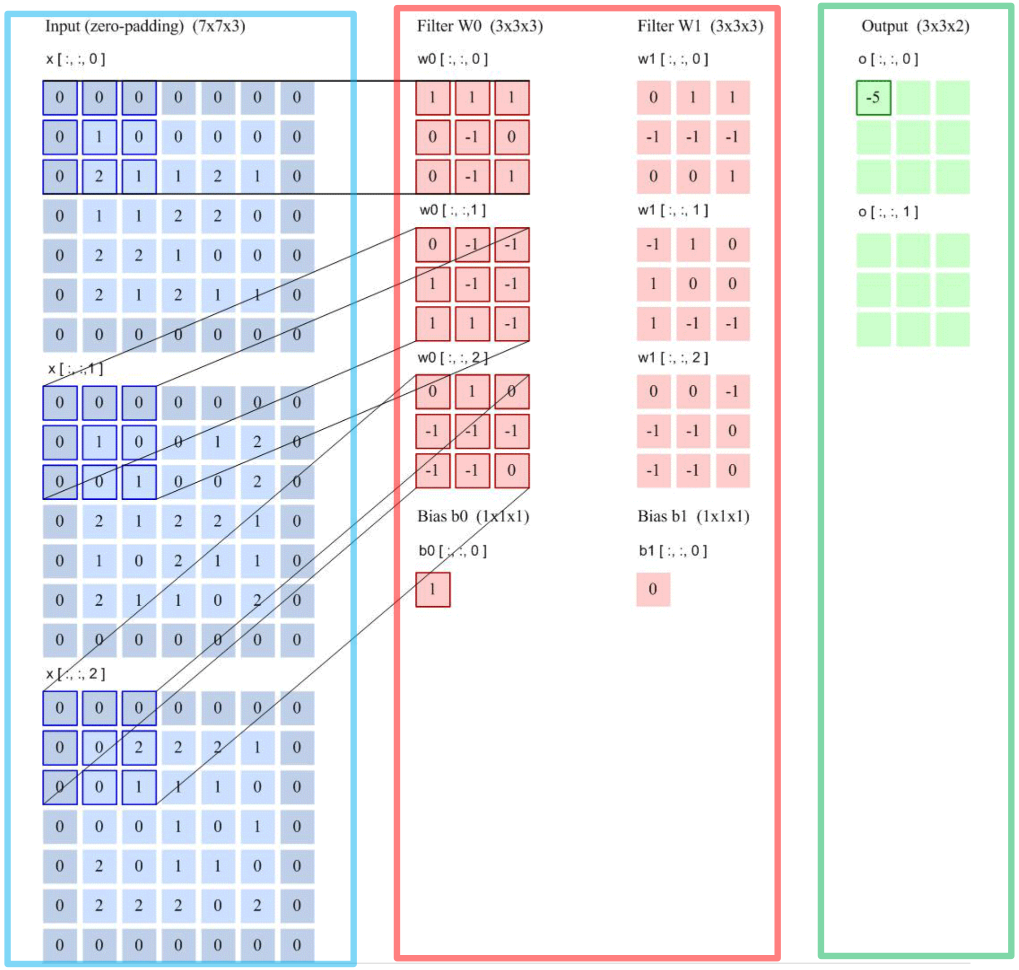
# 卷积层的映射关系




$$\boldsymbol{Z}^p = \boldsymbol{W}^p \otimes \boldsymbol{X} + b^p = \sum_{d=1}^{D} \boldsymbol{W}^{p,d} \otimes \boldsymbol{X}^d + b^p,$$

$$\boldsymbol{Y}^p = f(\boldsymbol{Z}^p).$$

https://cs231n.github.io/convolutional-networks/#conv




卷积核(convolutional kernel)

三/二维卷积核

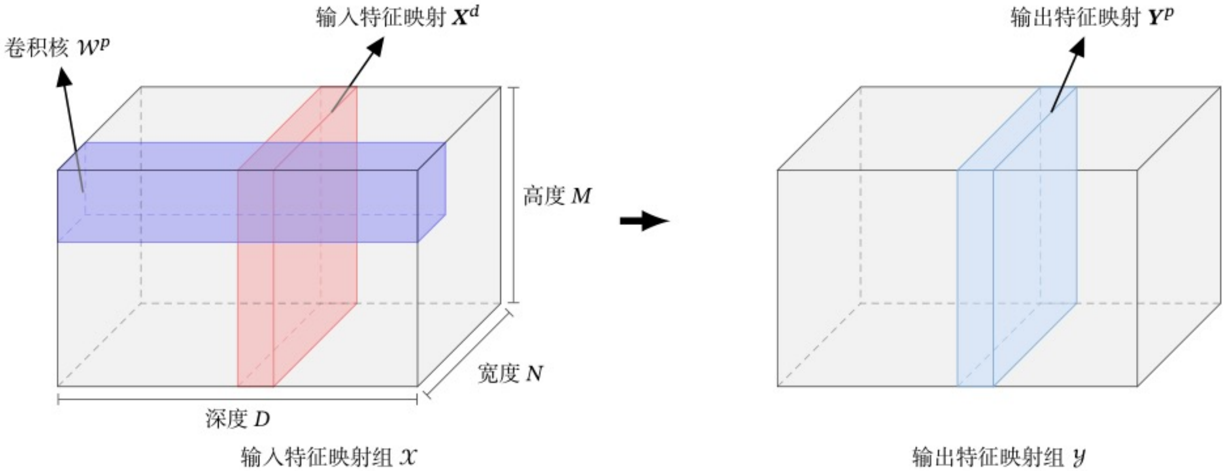
滤波器(filter)

核(kernel)

步长2
filter 3*3*3
filter个数~~6~~ 2
零填充 1

# 卷积层

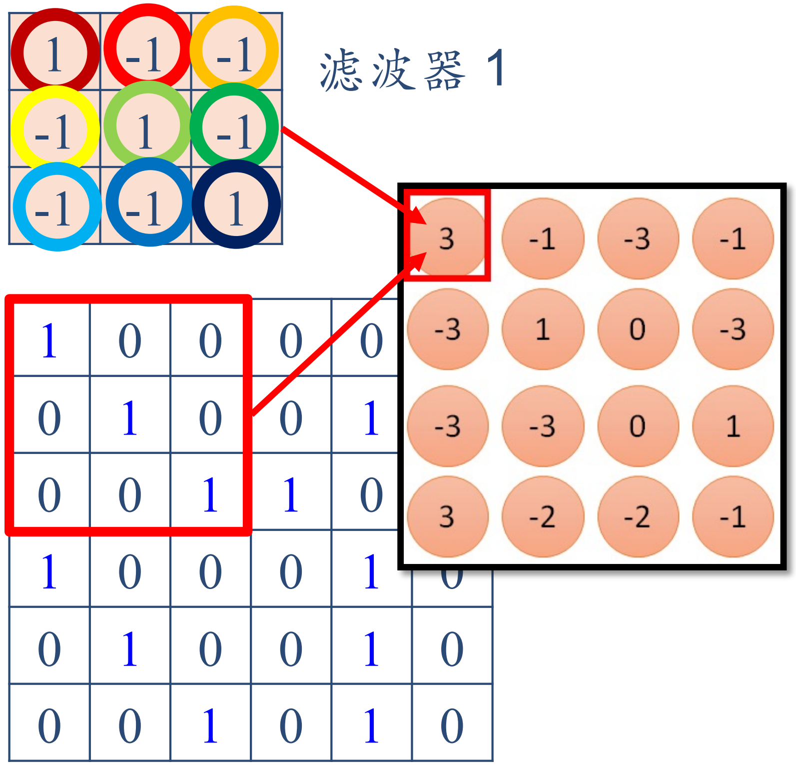
- 典型的卷积层为3维结构




$$\boldsymbol{Z}^p = \boldsymbol{W}^p \otimes \boldsymbol{X} + b^p = \sum_{d=1}^{D} \boldsymbol{W}^{p,d} \otimes \boldsymbol{X}^d + b^p,$$

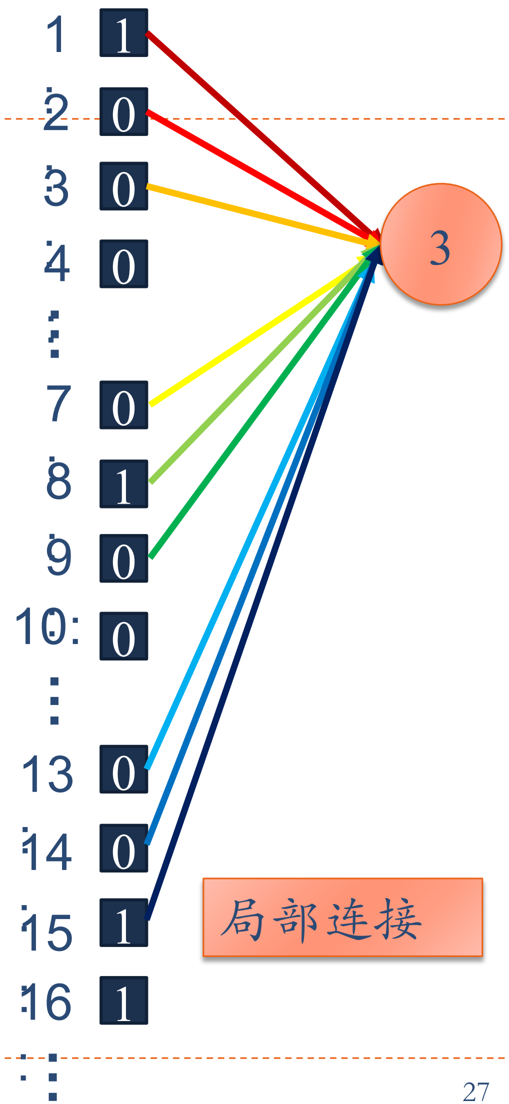
$$\boldsymbol{Y}^p = f(\boldsymbol{Z}^p).$$

# 卷积层 v.s. 全连接层



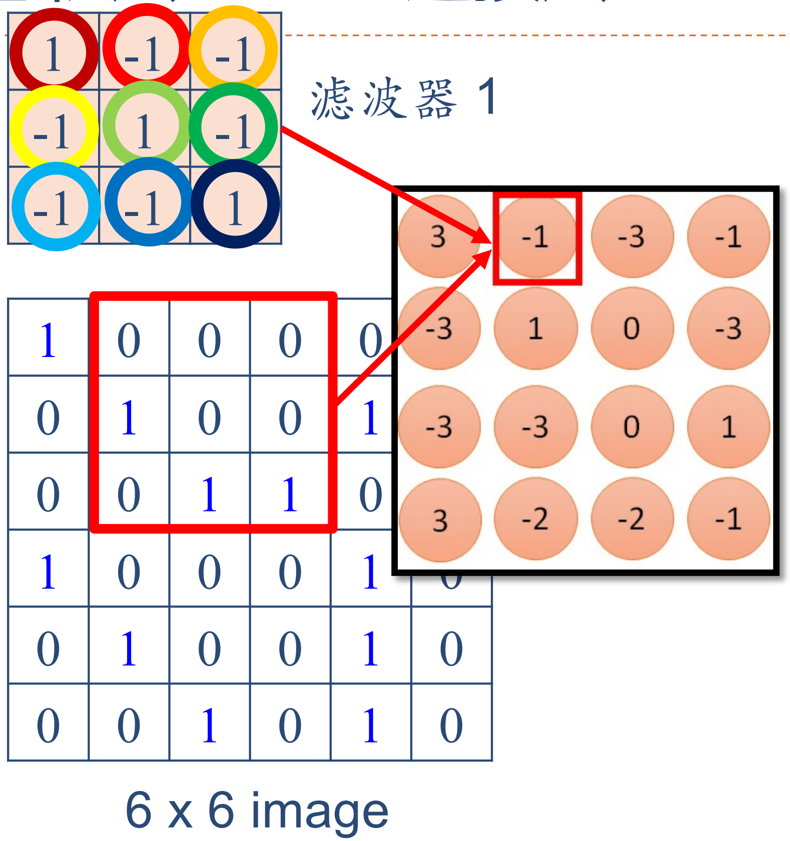

6 x 6 image

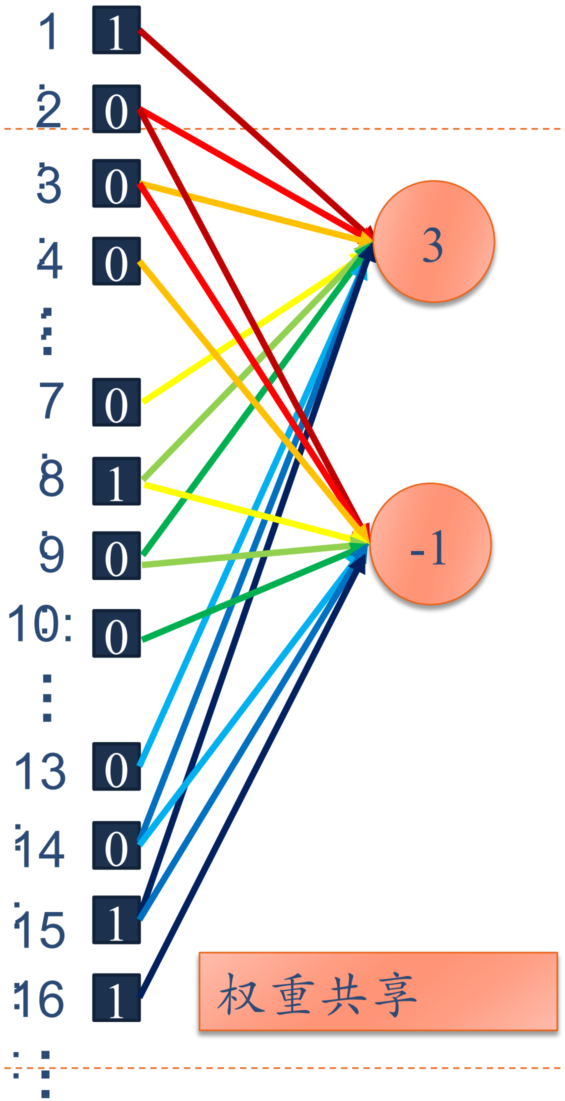
减少参数!




局部连接

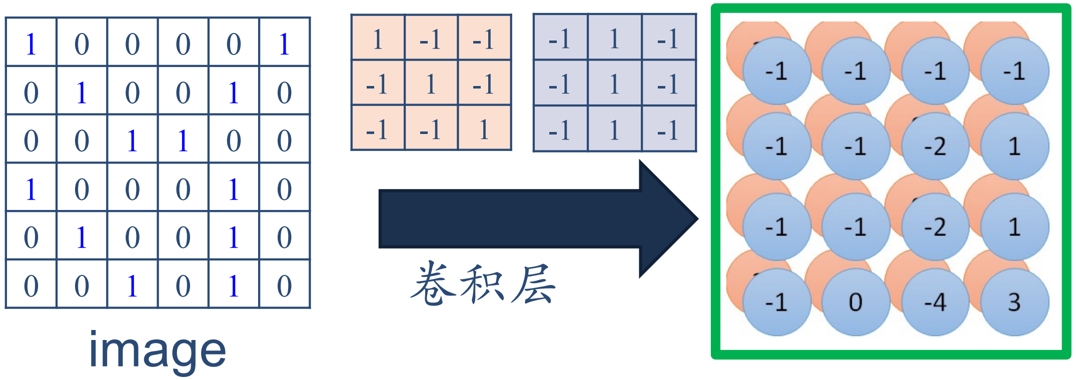
# 卷积层 v.s. 全连接层




**进一步减少参数!**

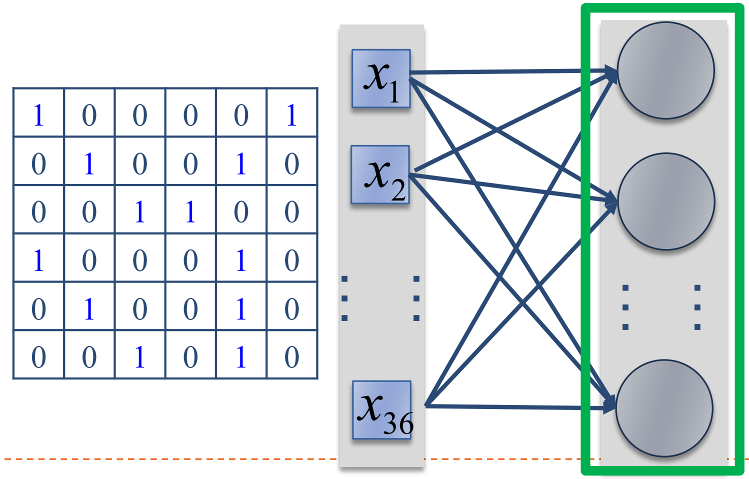
# 卷积层 v.s. 全连接层




想象输入图像大小为6*6

带两个滤波器的卷积层参数个数
（3*3+1）*2

全连接层




全连接层两个神经元参数个数
（6*6+1）*2

# 卷积的导数

- $Y = W \otimes X$，其中$X \in R^{M\times N}$, W$\in R^{U\times V}$, Y$\in R^{(M-U+1)\times(N-V+1)}$。函数$f(Y) \in R$为一个标量函数

- $f(Y)$关于$W$的偏导数为$X$和$\frac{\partial f(Y)}{\partial Y}$的卷积

$$
\begin{aligned}
\frac{\partial f(\mathbf{Y})}{\partial w_{uv}} &= \sum_{i=1}^{M-U+1}\sum_{j=1}^{N-V+1} \frac{\partial y_{ij}}{\partial w_{uv}} \frac{\partial f(\mathbf{Y})}{\partial y_{ij}} \\
&= \sum_{i=1}^{M-U+1}\sum_{j=1}^{N-V+1} x_{i+u-1,j+v-1} \frac{\partial f(\mathbf{Y})}{\partial y_{ij}} \\
&= \sum_{i=1}^{M-U+1}\sum_{j=1}^{N-V+1} \frac{\partial f(\mathbf{Y})}{\partial y_{ij}} x_{u+i-1,v+j-1}.
\end{aligned}
$$

$$y_{ij}=\sum_{u,v} w_{uv}x_{i+u-1,j+v-1}.$$

$$\frac{\partial f(\mathbf{Y})}{\partial \boldsymbol{W}} = \frac{\partial f(\mathbf{Y})}{\partial \boldsymbol{Y}} \otimes \boldsymbol{X}.$$

# 卷积的导数

$f(Y)$关于$X$的偏导数为$W$和$\frac{\partial f(Y)}{\partial Y}$的真实卷积(非自相关)

$$\frac{\partial f(\mathbf{Y})}{\partial x_{st}} = \sum_{i=1}^{M-U+1}\sum_{j=1}^{N-V+1} \frac{\partial y_{ij}}{\partial x_{st}} \frac{\partial f(\mathbf{Y})}{\partial y_{ij}}$$

$$y_{ij} = \sum_{s,t} x_{s,t} w_{s-i+1,t-j+1}$$

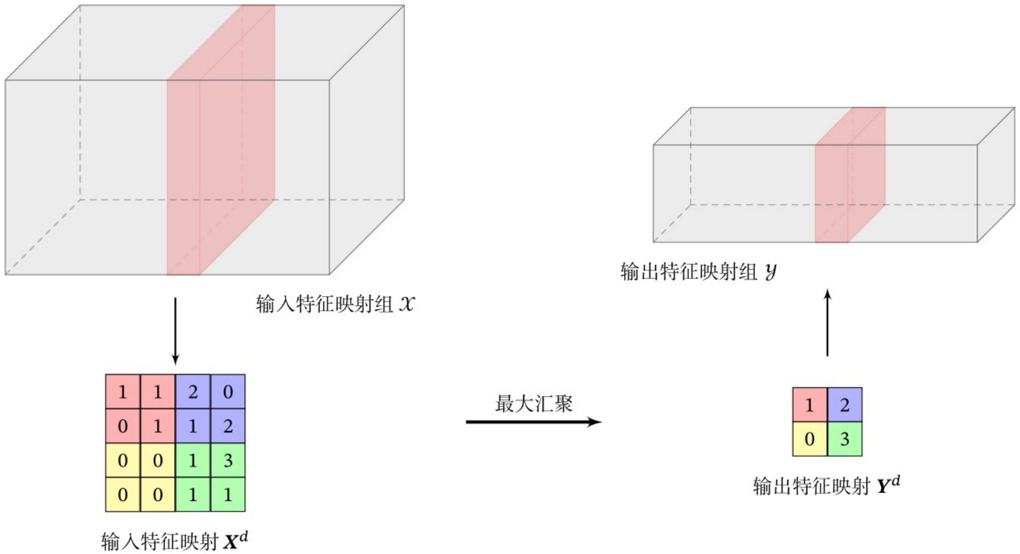
$$= \sum_{i=1}^{M-U+1}\sum_{j=1}^{N-V+1} w_{s-i+1,t-j+1} \frac{\partial f(\mathbf{Y})}{\partial y_{ij}},$$

$$\frac{\partial f(Y)}{X} = \mathrm{W} * \frac{\partial f(Y)}{Y}$$

# 汇聚层

- 卷积层虽然可以显著减少连接的个数，但是每一个特征映射的神经元个数并没有显著减少
- 为了对特征进一步进行选择，降低特征数量，减少参数数量，引入汇聚层对特征进行次采样
  - 最大汇聚(max-pooling) $y_{m,n}^d = \max_{i \in R_{m,n}^d} x_i$
  - 平均汇聚(mean-pooling) $y_{m,n}^d = \frac{1}{|R_{m,n}^d|} \sum_{i \in R_{m,n}^d} x_i$
- 一般情况下，汇聚层不含任何参数

# 汇聚层




- 汇聚层也可以看作一个特殊的卷积层，卷积核大小为 $K \times K$，卷积核为max函数或mean函数

## Mean-pooling的导数

- $Y = \mathrm{Mean-pooling}(X)$,其中$X \in R^{M\times N}$, $\mathrm{Y} \in R^{(M-U+1)\times(N-V+1)}$
- 可看作向上采样(up sampling)


$\frac{\partial f(\mathbf{Y})}{\partial \mathbf{Y}}$ $\frac{\partial f(Y)}{X}$

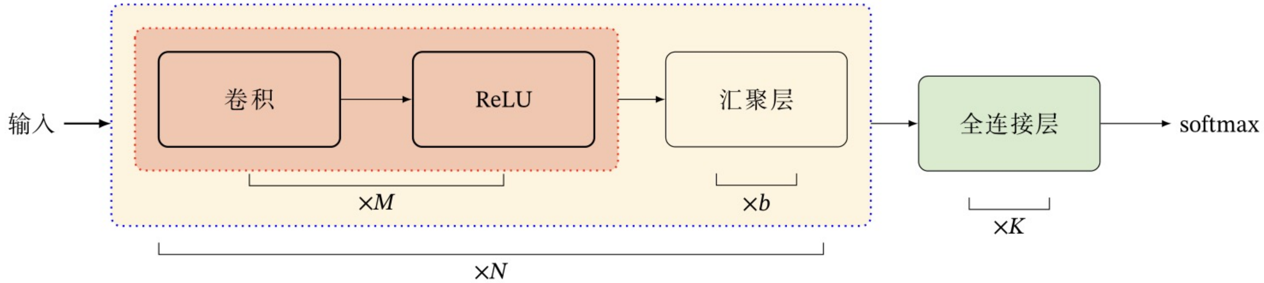
# Max-pooling的导数

- $Y = \mathrm{Max-pooling}(X)$,其中$X \in R^{M\times N}$, $\mathrm{Y} \in R^{(M-U+1)\times(N-V+1)}$


$\frac{\partial f(\mathbf{Y})}{\partial \mathbf{Y}}$ $\frac{\partial f(Y)}{X}$

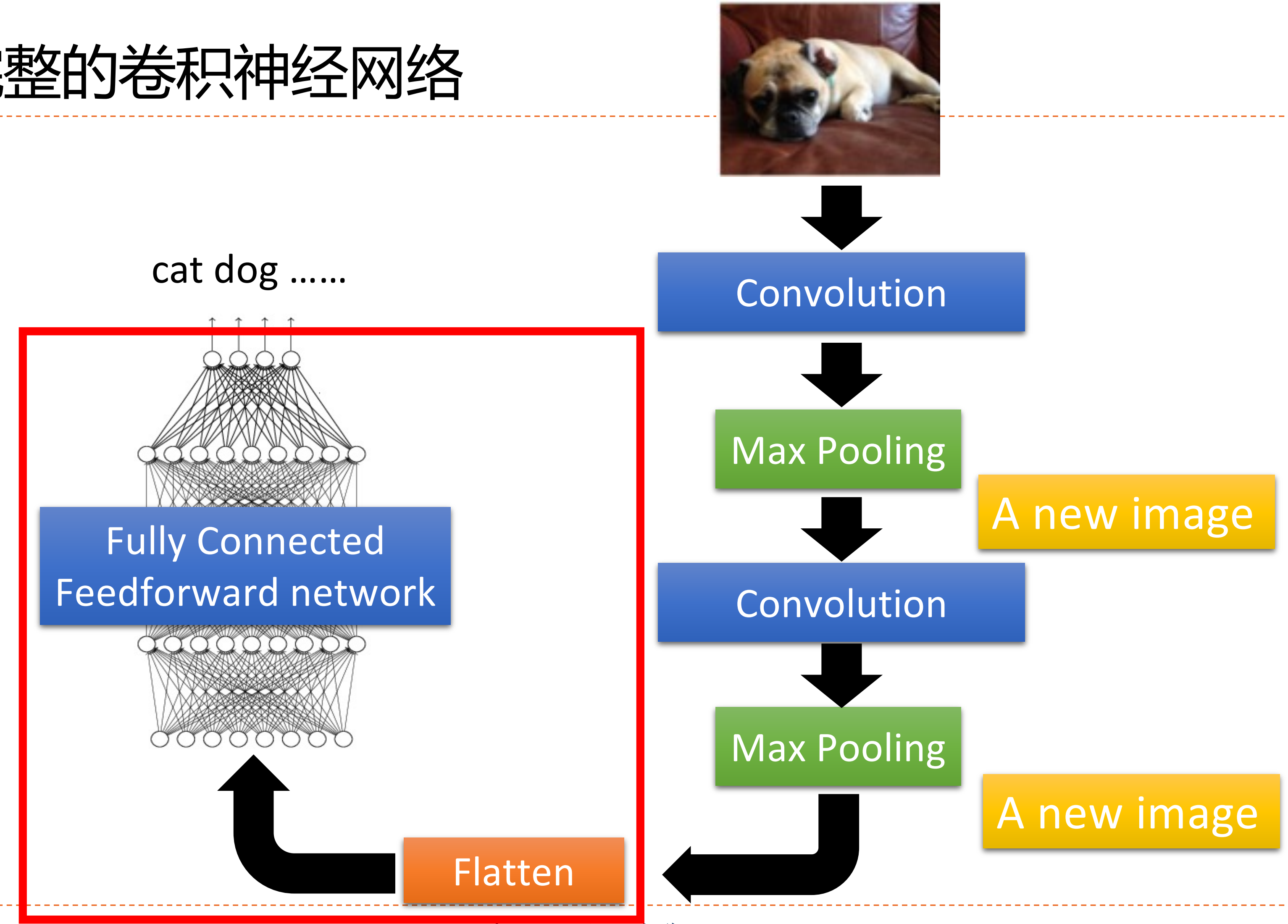
# 卷积网络结构

- 卷积网络是由卷积层、汇聚层、全连接层交叉堆叠而成。
  - 趋向于小卷积、大深度
  - 汇聚层比例逐渐降低，趋向于全卷积
- 典型结构



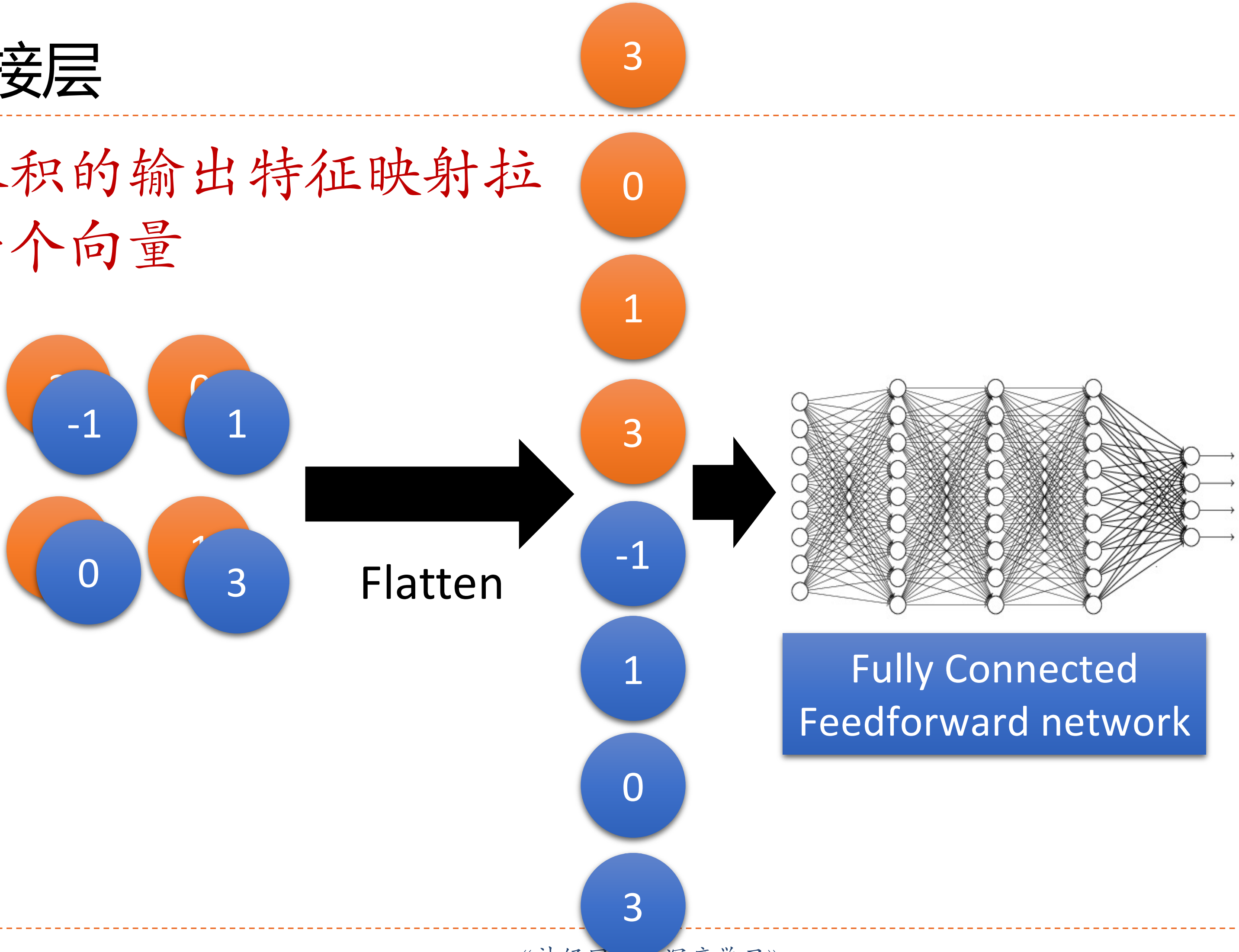

  - 一个卷积块为连续M 个卷积层和b个汇聚层（M通常设置为2～5，b为0或1）。一个卷积网络中可以堆叠N 个连续的卷积块，然后在接着K 个全连接层（N 的取值区间比较大，比如1～100或者更大；K一般为0～2）。

# 一个完整的卷积神经网络

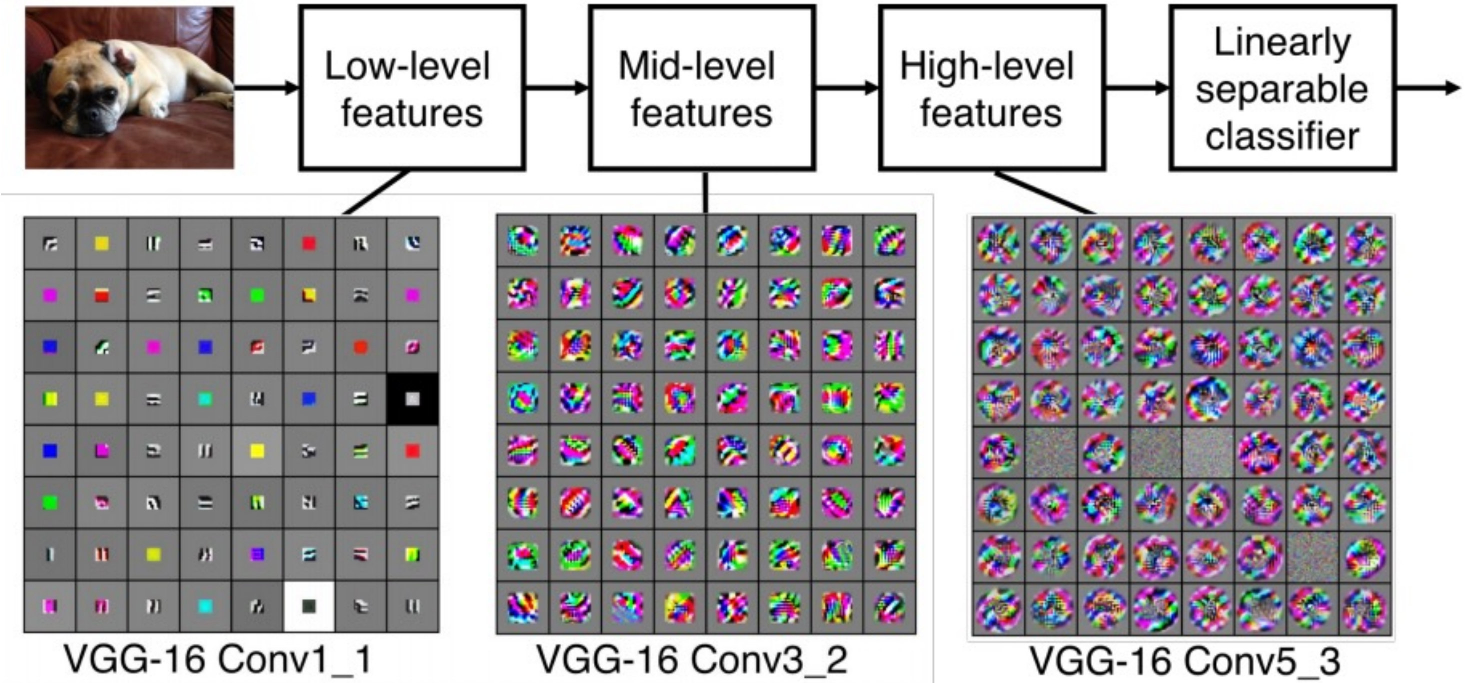


# 全连接层

- 将卷积的输出特征映射拉成一个向量

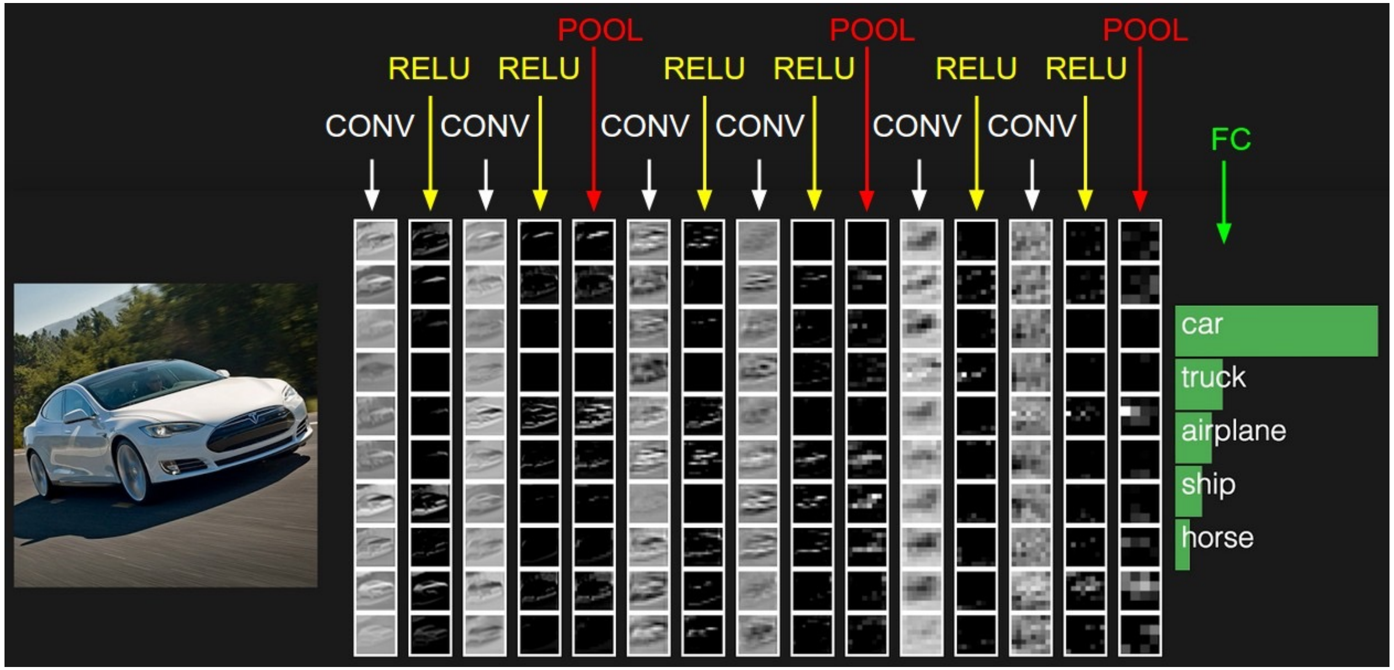


# 表示学习



Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3

# 表示学习


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

# 卷积神经网络的参数学习

$$Z^{(l,p)} = \sum_{d=1}^{D} W^{(l,p,d)} \otimes X^{(l-1,d)} + b^{(l,p)}$$

- 损失函数关于第$l$层的卷积核$W^{(l,p,d)}$和偏置$b^{(l,p)}$的偏导数

$$\frac{\partial \mathcal{L}}{\partial W^{(l,p,d)}} = \frac{\partial \mathcal{L}}{\partial Z^{(l,p)}} \otimes X^{(l-1,d)} = \delta^{(l,p)} \otimes X^{(l-1,d)},$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l,p)}} = \sum_{i,j} [\delta^{(l,p)}]_{i,j}.$$

- 反向传播算法误差项
  - 第$l$+1层为卷积层

$$\delta^{(l,d)} = f_l'(Z^{(l,d)}) \odot \sum_{P=1}^{P} \left( W^{(l+1,p,d)} * \delta^{(l+1,p)} \right),$$

  - 第$l+1$层为汇聚层

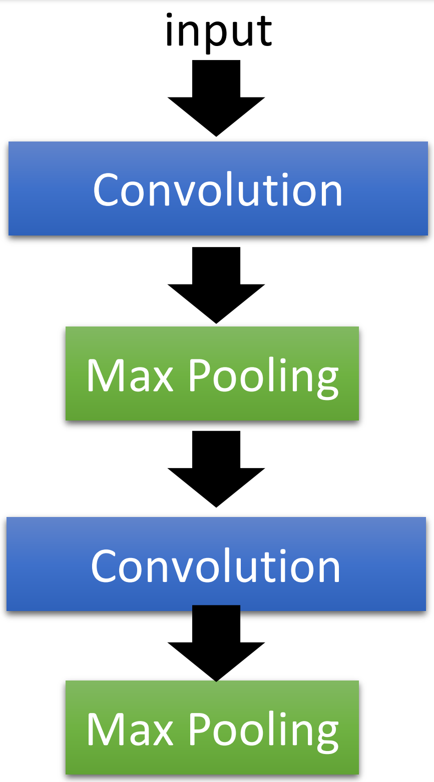
$$\delta^{(l,p)} = f_l'(Z^{(l,p)}) \odot \mathrm{up}(\delta^{(l+1,p)}),$$

# 卷积神经网络的简单实现

## ***CNN via Pytorch***

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

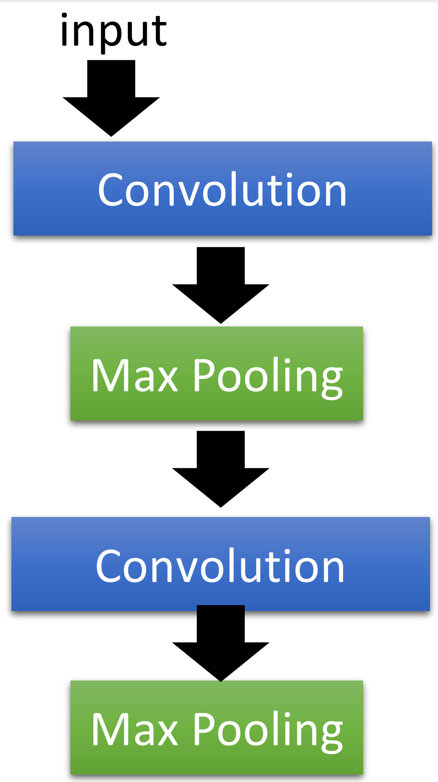
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential( #input shape (1,28,28)
            nn.Conv2d(in_channels=1, #input height
                      out_channels=16, #n_filter
                     kernel_size=5, #filter size
                     stride=1, #filter step
                     padding=2 #con2d出来的图片大小不变
                     ), #output shape (16,28,28)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2) #2x2采样，output shape (

        )
        self.conv2 = nn.Sequential(nn.Conv2d(16, 32, 5, 1, 2),
                                   nn.ReLU(),
                                   nn.MaxPool2d(2))
        self.out = nn.Linear(32*7*7,10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1) #flat (batch_size, 32*7*7)
        output = self.out(x)
        return output
```



## ***CNN via Pytorch***

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

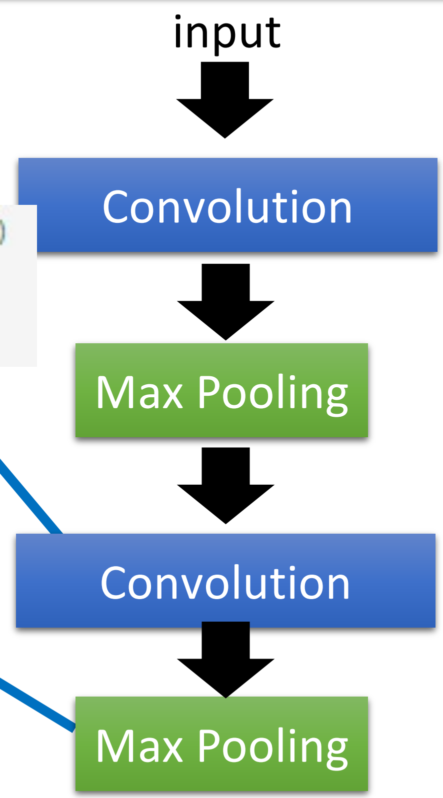
```
self.conv1 = nn.Sequential( #input shape (1,28,28)
    nn.Conv2d(in_channels=1, #input height
              out_channels=16, #n_filter
             kernel_size=5, #filter size
             stride=1, #filter step
             padding=2 #con2d出来的图片大小不变
             ), #output shape (16,28,28)
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2) #2x2采样，output shape (16,14,14)

)
```



## ***CNN via Pytorch***

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
self.conv2 = nn.Sequential(nn.Conv2d(16, 32, 5, 1, 2), #output shape (32,7,7)
                           nn.ReLU(),
                           nn.MaxPool2d(2))
```



## ***CNN via Pytorch***

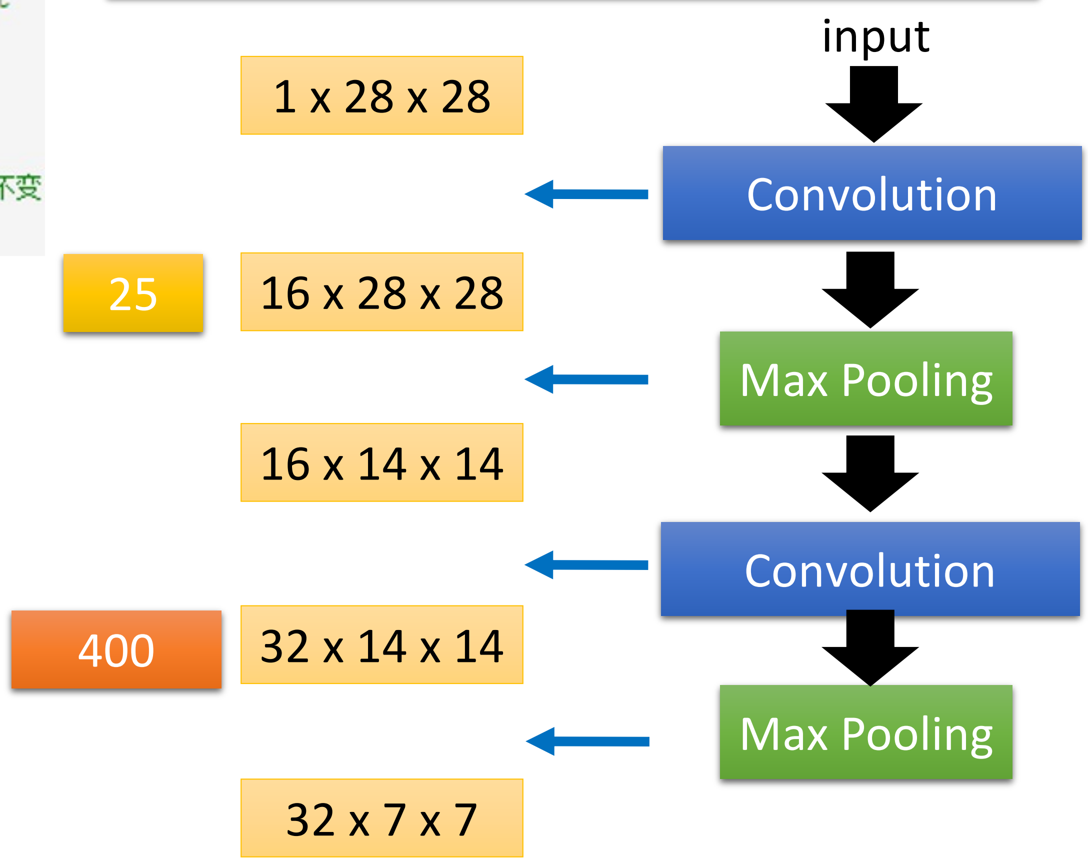
Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
nn.Conv2d(in_channels=1, #input height
          out_channels=16, #n_filter
         kernel_size=5, #filter size
         stride=1, #filter step
         padding=2 #con2d出来的图片大小不变
         ), #output shape (16,28,28)
```

How many parameters for each filter?

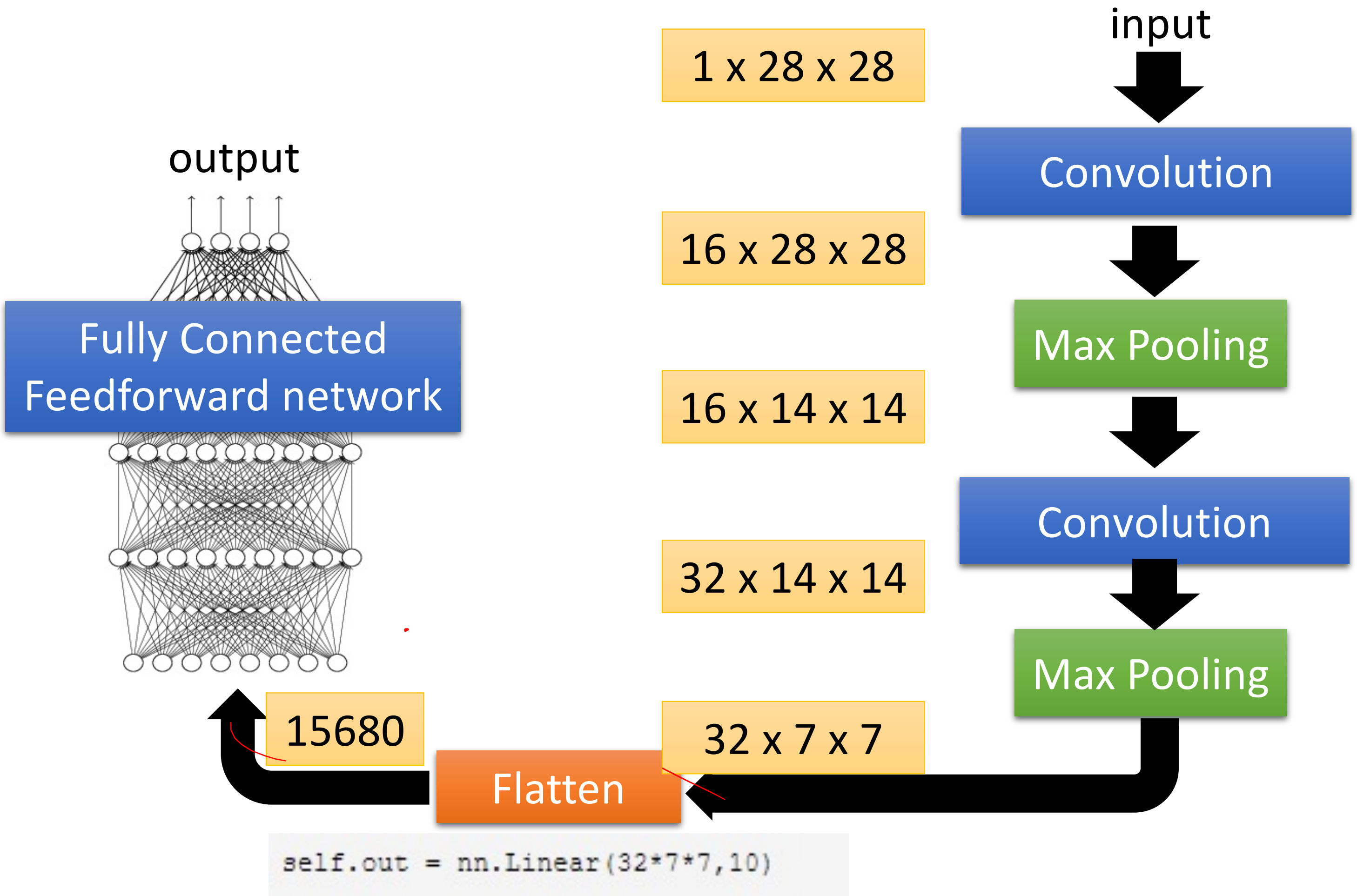
```
(nn.Conv2d(16, 32, 5, 1, 2),
```

How many parameters for each filter?



***CNN via Pytorch***

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



# 卷积神经网络的时间复杂度

https://zhuanlan.zhihu.com/p/31575074

```
def conv2d(img, kernel):
    height, width, in_channels = img.shape
    kernel_height, kernel_width, in_channels, out_channels = kernel.shape
    out_height = height - kernel_height + 1
    out_width = width - kernel_width + 1
    feature_maps = np.zeros(shape=(out_height, out_width, out_channels))
    for oc in range(out_channels):              # Iterate out_channels (# of kernels)
        for h in range(out_height):             # Iterate out_height
            for w in range(out_width):          # Iterate out_width
                for ic in range(in_channels):   # Iterate in_channels
                    patch = img[h: h + kernel_height, w: w + kernel_width, ic]
                    feature_maps[h, w, oc] += np.sum(patch * kernel[:, :, ic, oc])

    return feature_maps
```

# 卷积神经网络的时间复杂度

$$\textbf{Time} \sim O\left(\sum_{l=1}^{L} M_l^2 \cdot K_l^2 \cdot C_{l-1} \cdot C_l\right)$$

- $L$ 神经网络所具有的卷积层数，也即**网络的深度。**
- $l$ 神经网络第 $l$ 个卷积层
- $C_l$ 神经网络第 $l$ 个卷积层的输出通道数 $C_{out}$
- 对于第 $l$ 个卷积层而言，其输入通道数 $C_{in}$ 就是第 $(l-1)$ 个卷积层的输出通道数。

# 卷积神经网络的空间复杂度

空间复杂度（访存量），严格来讲包括两部分：总参数量 + 各层输出特征图。

- **参数量**：模型所有带参数的层的权重参数总量（即**模型体积**，下式第一个求和表达式）
- **特征图**：模型在实时运行过程中每层所计算出的输出特征图大小（下式第二个求和表达式）

$$\mathbf{Space} \sim O\left(\sum_{l=1}^{L} K_l^2 \cdot C_{l-1} \cdot C_l + \sum_{l=1}^{L} M^2 \cdot C_l\right)$$

- 总参数量只与卷积核的尺寸 $K$ 、通道数 $C$ 、层数 $L$ 相关，而**与输入数据的大小无关**。
- 输出特征图的空间占用比较容易，就是其空间尺寸 $M^2$ 和通道数 $C$ 的连乘。

# 思考

对于一个输入为$100 \times 100 \times 256$ 的特征映射组，使用$3 \times 3$ 的卷积核，输出为$100 \times 100 \times 256$ 的特征映射组的卷积层，求其时间和空间复杂度．如果引入一个$1 \times 1$ 卷积核，先得到$100 \times 100 \times 64$ 的特征映射，再进行$3 \times 3$ 的卷积，得到$100 \times 100 \times 256$ 的特征映射组，求其时间和空间复杂度．

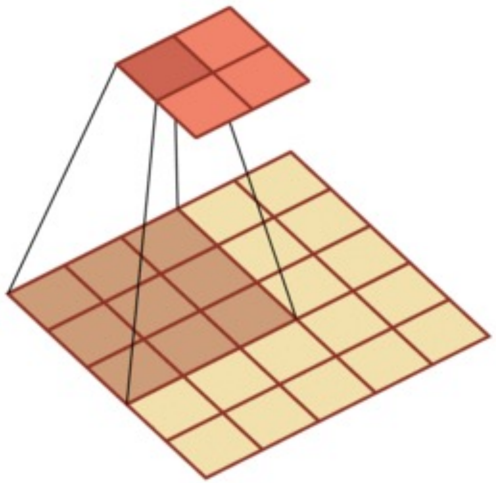
时间复杂度：100*100*256*3*3*256 =
空间复杂度：256*3*3*256 + 100*100*256 =

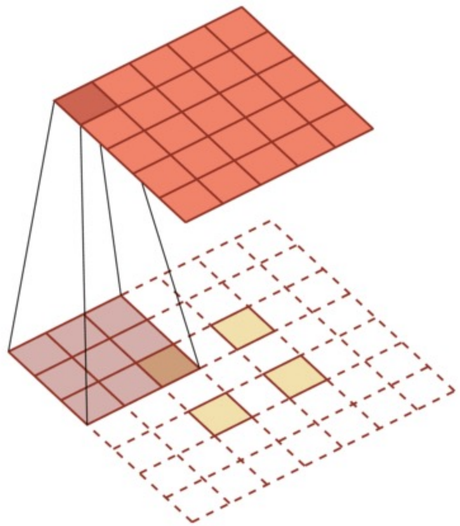
时间复杂度：100*100*256*64 + 100*100*64*3*3*256=
空间复杂度：1*1*64*256 + 100*100*64 + 3*3*64*256 + 100*100*256 =

# 其它卷积种类

# 转置卷积(Transposed Convolution)/微步卷积(Fractionally-Strided Convolution)

- 低维特征映射到高维特征

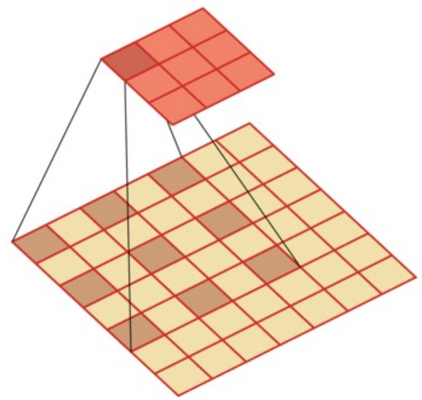


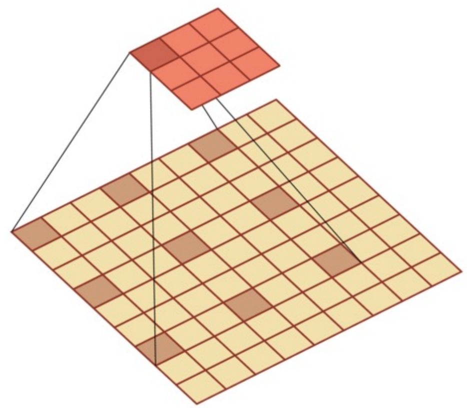


# 空洞卷积(Atrous Convolution)

- 如何增加输出单元的感受野
  - 增加卷积核的大小
  - 增加层数来实现
  - 在卷积之前进行汇聚操作

- 空洞卷积
  - 通过给卷积核插入“空洞”来变相地增加其大小。





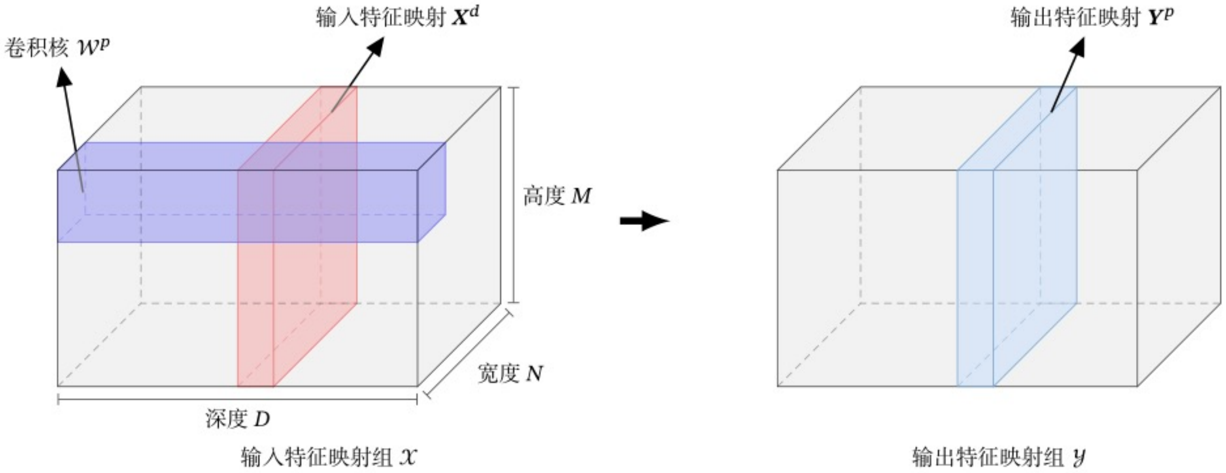
# 内容

- 卷积神经网络
  - 卷积
  - 卷积神经网络
  - 卷积神经网络的简单实现
  - 其他种类的卷积
- 卷积神经网络的扩展
  - 典型的卷积神经网络(*LeNet, AlexNet, VGG, NiN, ...*)
  - 卷积神经网络的应用
- 深度学习计算硬件

# 典型的卷积网络

# 卷积层

- 典型的卷积层为3维结构

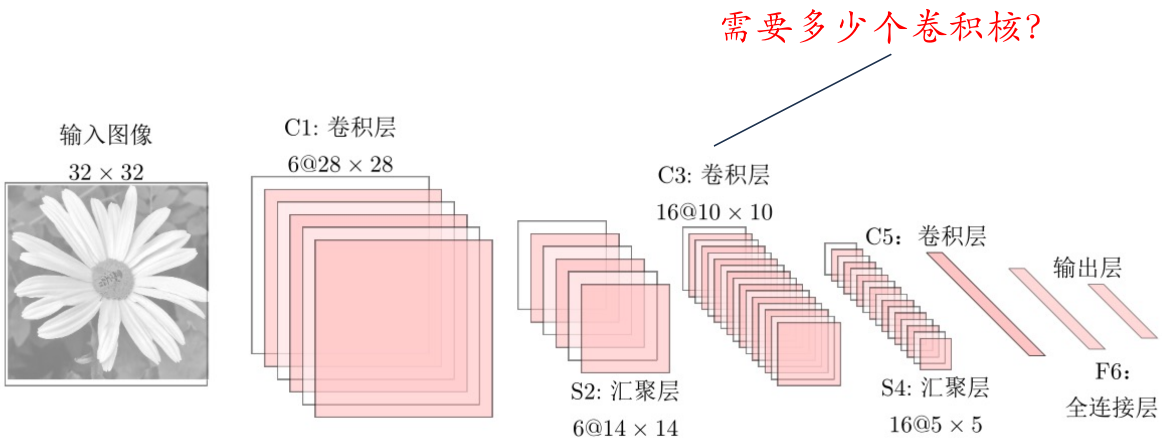



$$\boldsymbol{Z}^p = \boldsymbol{W}^p \otimes \boldsymbol{X} + b^p = \sum_{d=1}^{D} \boldsymbol{W}^{p,d} \otimes \boldsymbol{X}^d + b^p,$$

$$\boldsymbol{Y}^p = f(\boldsymbol{Z}^p).$$

# LeNet-5

- LeNet-5 是一个非常成功的神经网络模型。
  - 基于 LeNet-5 的手写数字识别系统在 90 年代被美国很多银行使用，用来识别支票上面的手写数字。
  - LeNet-5 共有 7 层。





[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

# LeNet的简单实现

```python
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 6, 5), # in_channels, out_channels, kernel_size
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2), # kernel_size, stride
            nn.Conv2d(6, 16, 5),
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2)
        )
        self.fc = nn.Sequential(
            nn.Linear(16*4*4, 120),
            nn.Sigmoid(),
            nn.Linear(120, 84),
            nn.Sigmoid(),
            nn.Linear(84, 10)
        )

    def forward(self, img):
        feature = self.conv(img)
        output = self.fc(feature.view(img.shape[0], -1))
        return output
```

# 连接表

| | 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 |

图 5.11　LeNet-5 中 C3 层的连接表（图片来源：[LeCun et al., 1998]）

- 如果第$p$个输出特征映射依赖于第$d$个输入特征映射，则$T_{p,d}=1$，否则为0，$Y^p$为：

$$\boldsymbol{Y}^p = f\left( \sum_{\substack{d,\\ T_{p,d}=1}} \boldsymbol{W}^{p,d} \otimes \boldsymbol{X}^d + b^p \right),$$

假设连接表的非零个数为$K$，每个卷积核的大小为$U\times V$，那么共需要$K\times U\times V+P$参数

# Large Scale Visual Recognition Challenge

| 2012 Teams | %error |
| --- | --- |
| Supervision (Toronto) | 15.3 |
| ISI (Tokyo) | 26.1 |
| VGG (Oxford) | 26.9 |
| XRCE/INRIA | 27.0 |
| UvA (Amsterdam) | 29.6 |
| INRIA/LEAR | 33.4 |

| 2013 Teams | %error |
| --- | --- |
| Clarifai (NYU spinoff) | 11.7 |
| NUS (singapore) | 12.9 |
| Zeiler-Fergus (NYU) | 13.5 |
| A. Howard | 13.5 |
| OverFeat (NYU) | 14.1 |
| UvA (Amsterdam) | 14.2 |
| Adobe | 15.2 |
| VGG (Oxford) | 15.2 |
| VGG (Oxford) | 23.0 |

| 2014 Teams | %error |
| --- | --- |
| GoogLeNet | 6.6 |
| VGG (Oxford) | 7.3 |
| MSRA | 8.0 |
| A. Howard | 8.1 |
| DeeperVision | 9.5 |
| NUS-BST | 9.7 |
| TTIC-ECP | 10.2 |
| XYZ | 11.2 |
| UvA | 12.1 |

# AlexNet

[1] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

- 2012 ILSVRC winner
  - （top 5 error of 16% compared to runner-up with 26% error）
  - 第一个现代深度卷积网络模型
    - 首次使用了很多现代深度卷积网络的一些技术方法
  - 5个卷积层、3个汇聚层和3个全连接层

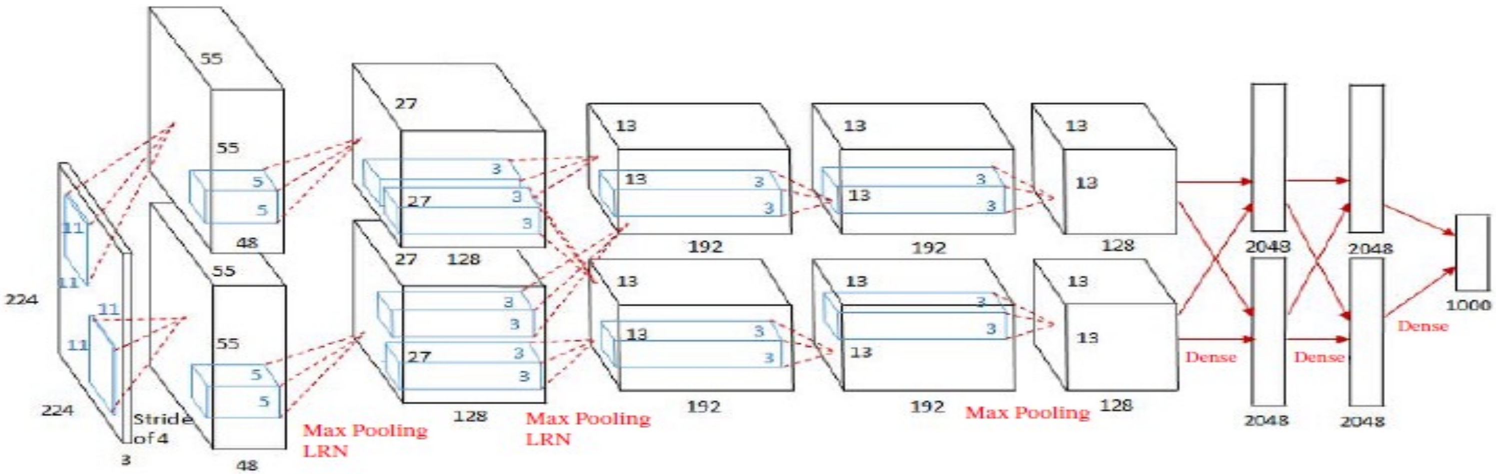



图 5.12 AlexNet 网络结构[1]

## AlexNet v.s. LeNet

- AlexNet是更深的神经网络模型
- AlexNet中卷积通道数比LeNet大， AlexNet中全连接层带来近1GB的模型参数
- AlexNet将sigmoid激活函数改成了ReLU激活函数
- AlexNet通过dropout，图像增广(翻转、裁剪和颜色变化)的手段缓解过拟合问题

# AlexNet的简单实现

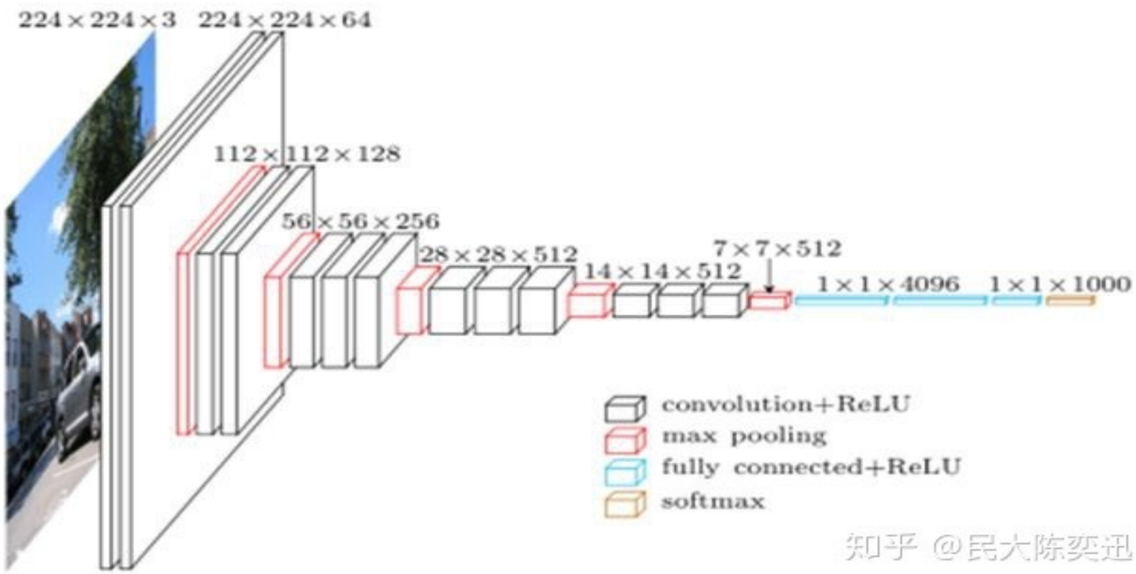
```
class AlexNet(nn.Module):
    def __init__(self):
        super(AlexNet, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 96, 11, 4), # in_channels, out_channels, kernel_size, stride, padding
            nn.ReLU(),
            nn.MaxPool2d(3, 2), # kernel_size, stride
            # 减小卷积窗口，使用填充为2来使得输入与输出的高和宽一致，且增大输出通道数
            nn.Conv2d(96, 256, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(3, 2),
            # 连续3个卷积层，且使用更小的卷积窗口。除了最后的卷积层外，进一步增大了输出通道数。
            # 前两个卷积层后不使用池化层来减小输入的高和宽
            nn.Conv2d(256, 384, 3, 1, 1),
            nn.ReLU(),
            nn.Conv2d(384, 384, 3, 1, 1),
            nn.ReLU(),
            nn.Conv2d(384, 256, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(3, 2)
        )
```

```
 # 这里全连接层的输出个数比LeNet中的大数倍。使用丢弃层来缓解过拟合
self.fc = nn.Sequential(
    nn.Linear(256*5*5, 4096),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(4096, 4096),
    nn.ReLU(),
    nn.Dropout(0.5),
    # 输出层。由于这里使用Fashion-MNIST，所以用类别数为10，而非论文中的1000
    nn.Linear(4096, 10),
)
```

# VGG

[1] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

- ILSVRC 2014 runner-up
  - Top 5 error of 7.3%
  - 使用重复元素的网络
  - 展示了网络的深度是算法优良性能的关键部分



# VGG

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

# VGG的简单实现

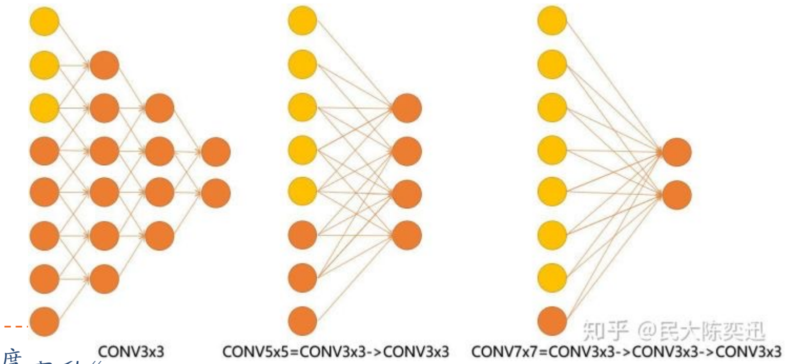
```python
def vgg_block(num_convs, in_channels, out_channels):
    blk = []
    for i in range(num_convs):
        if i == 0:
            blk.append(nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1))
        else:
            blk.append(nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1))
        blk.append(nn.ReLU())
    blk.append(nn.MaxPool2d(kernel_size=2, stride=2)) # 这里会使宽高减半
    return nn.Sequential(*blk)
```

```python
conv_arch = ((1, 1, 64), (1, 64, 128), (2, 128, 256), (2, 256, 512), (2, 512, 512))
# 经过5个vgg_block, 宽高会减半5次, 变成 224/32 = 7
fc_features = 512 * 7 * 7 # c * w * h
fc_hidden_units = 4096 # 任意
```

```python
def vgg(conv_arch, fc_features, fc_hidden_units=4096):
    net = nn.Sequential()
    # 卷积层部分
    for i, (num_convs, in_channels, out_channels) in enumerate(conv_arch):
        # 每经过一个vgg_block都会使宽高减半
        net.add_module("vgg_block_" + str(i+1), vgg_block(num_convs, in_channels, out_channels))
    # 全连接层部分
    net.add_module("fc", nn.Sequential(d2l.FlattenLayer(),
                                 nn.Linear(fc_features, fc_hidden_units),
                                 nn.ReLU(),
                                 nn.Dropout(0.5),
                                 nn.Linear(fc_hidden_units, fc_hidden_units),
                                 nn.ReLU(),
                                 nn.Dropout(0.5),
                                 nn.Linear(fc_hidden_units, 10)
                                ))
    return net
```

# VGG的优点

- 小卷积核
  - 多个小卷积堆叠在分类精度上比单个大卷积要好
- 小汇聚核
  - 相比AlexNet的3*3的汇聚核，VGG全部为2*2的汇聚核
- 层数更深
  - 作者通过6个实验证明，最后两个实验层数最深，效果也最好
- 卷积核堆叠的感受野
  - 两个3*3的卷积堆叠在一起获得的感受野相当于一个5*5卷积；3个3x3卷积的堆叠获取到的感受野相当于一个7x7的卷积



# NiN

[1] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

- 网络中的网络(Network in Network)
  - 使用1×1卷积核和全局平均汇聚层代替全连接层

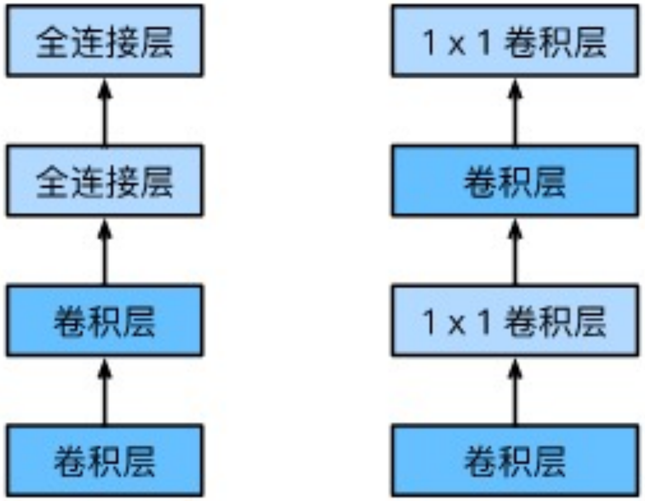
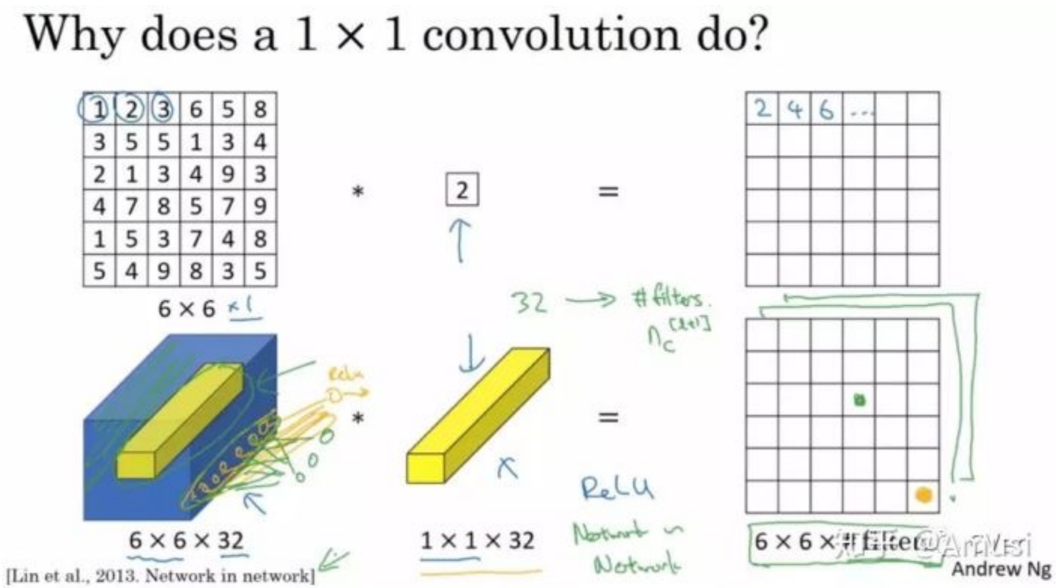



图5.7 左图是AlexNet和VGG的网络结构局部，右图是NiN的网络结构局部

# 1×1卷积核

- 在不改变输入特征映射的长宽的情况下，对其进行降维或者升维
- 输出中每个元素来自输入中在高和宽相同位置的元素在不同输入通道之间的按权重累加
- 将输入通道维当作特征维，将高和宽维度上的元素当成数据样本，作用与全连接等价



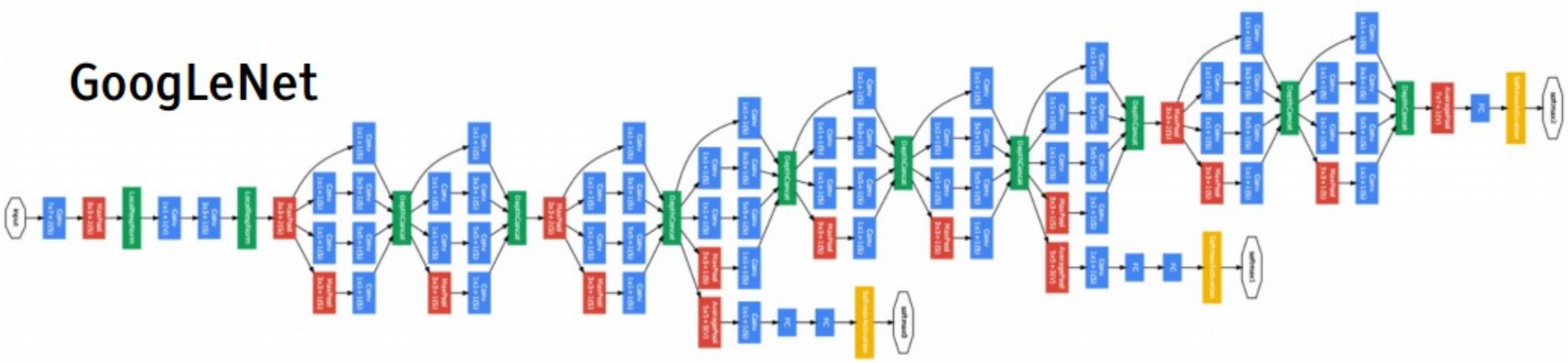
# NiN的简单实现

```
net = nn.Sequential(
    nin_block(1, 96, kernel_size=11, stride=4, padding=0),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nin_block(96, 256, kernel_size=5, stride=1, padding=2),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nin_block(256, 384, kernel_size=3, stride=1, padding=1),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Dropout(0.5),
    # 标签类别数是10
    nin_block(384, 10, kernel_size=3, stride=1, padding=1),
    GlobalAvgPool2d(),
    # 将四维的输出转成二维的输出，其形状为(批量大小, 10)
    d2l.FlattenLayer())
```

```
6 output shape:  torch.Size([1, 384, 5, 5])
7 output shape:  torch.Size([1, 10, 5, 5])
8 output shape:  torch.Size([1, 10, 1, 1])
```

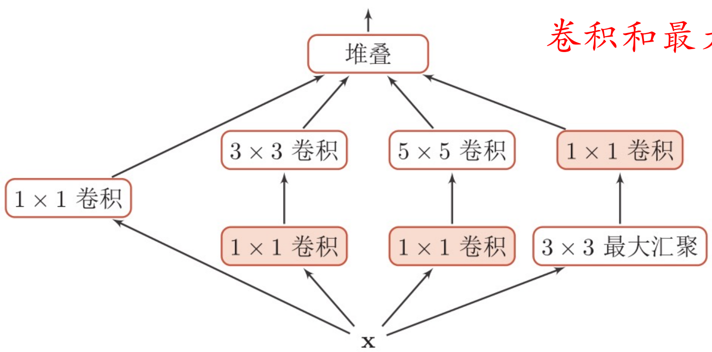
# Inception网络

- 2014 ILSVRC winner （22层）
  - 参数：GoogLeNet：4M VS AlexNet：60M
  - 错误率：6.7%
  - 含并行结构
  - Inception网络是由多个inception模块和少量的汇聚层堆叠而成。



# Inception模块 v1

- 在卷积网络中，如何设置卷积层的卷积核大小是一个十分关键的问题。
  - 在Inception网络中，一个卷积层包含多个不同大小的卷积操作，称为Inception模块。
  - Inception模块同时使用$1 \times 1$、$3 \times 3$、$5 \times 5$等不同大小的卷积核，并将得到的特征映射在深度上拼接（堆叠）起来作为输出特征映射。




卷积和最大汇聚都是等宽的。

# Inception模块 v3

- 用多层**小卷积核**替换**大卷积核**，以减少计算量和参数量。
  - 使用两层3x3的卷积来替换v1中的5x5的卷积
  - 使用连续的nx1和1xn来替换nxn的卷积。

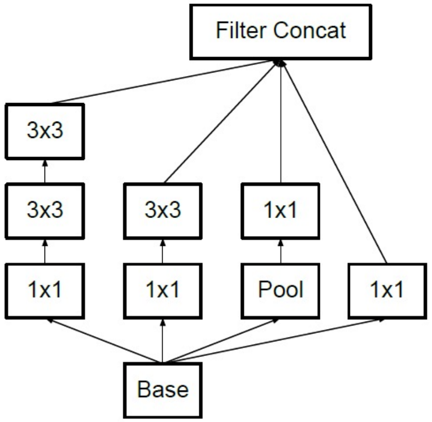



Figure 5. Inception modules where each 5 × 5 convolution is replaced by two 3 × 3 convolution, as suggested by principle 3 of Section 2.

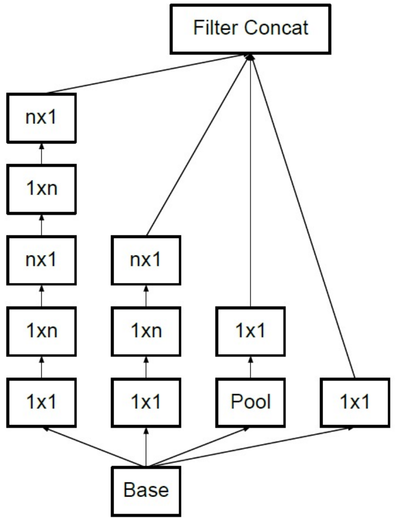



Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the $17 \times 17$ grid. (The filter sizes are picked using principle 3)

# Inception的简单实现

```python
class Inception(nn.Module):
    # c1 - c4为每条线路里的层的输出通道数
    def __init__(self, in_c, c1, c2, c3, c4):
        super(Inception, self).__init__()
        # 线路1，单1 x 1卷积层
        self.p1_1 = nn.Conv2d(in_c, c1, kernel_size=1)
        # 线路2，1 x 1卷积层后接3 x 3卷积层
        self.p2_1 = nn.Conv2d(in_c, c2[0], kernel_size=1)
        self.p2_2 = nn.Conv2d(c2[0], c2[1], kernel_size=3, padding=1)
        # 线路3，1 x 1卷积层后接5 x 5卷积层
        self.p3_1 = nn.Conv2d(in_c, c3[0], kernel_size=1)
        self.p3_2 = nn.Conv2d(c3[0], c3[1], kernel_size=5, padding=2)
        # 线路4，3 x 3最大池化层后接1 x 1卷积层
        self.p4_1 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
        self.p4_2 = nn.Conv2d(in_c, c4, kernel_size=1)

    def forward(self, x):
        p1 = F.relu(self.p1_1(x))
        p2 = F.relu(self.p2_2(F.relu(self.p2_1(x))))
        p3 = F.relu(self.p3_2(F.relu(self.p3_1(x))))
        p4 = F.relu(self.p4_2(self.p4_1(x)))
        return torch.cat((p1, p2, p3, p4), dim=1)  # 在通道维上连结输出
```

# 残差网络

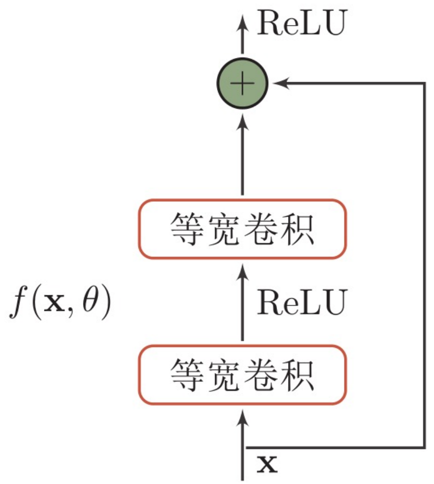
[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778)

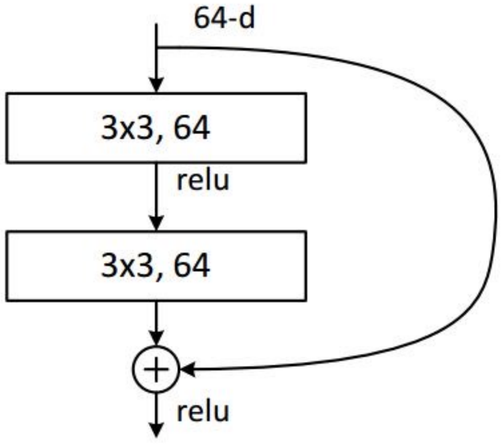
- 残差网络（Residual Network，ResNet）是通过给非线性的卷积层增加**直连边**的方式来提高信息的传播效率。
  - 假设在一个深度网络中，我们期望一个非线性单元（可以为一层或多层的卷积层）f(x,θ)去逼近一个目标函数为h(x)。
  - 将目标函数拆分成两部分：恒等函数和残差函数

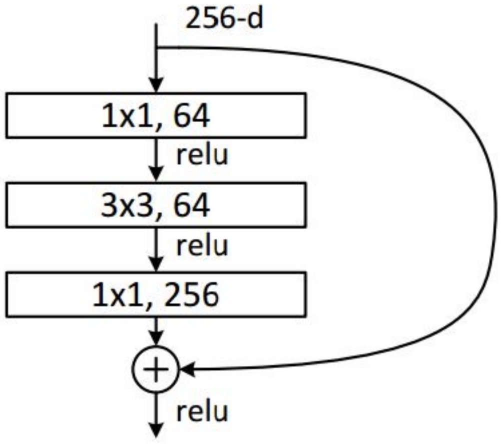
$$h(\mathbf{x}) = \underbrace{\mathbf{x}}_{\text{恒等函数}} + \underbrace{\big(h(\mathbf{x}) - \mathbf{x}\big)}_{\text{残差函数}}$$

$$\text{残差函数} \rightarrow f(\mathbf{x}, \theta)$$

# 残差单元

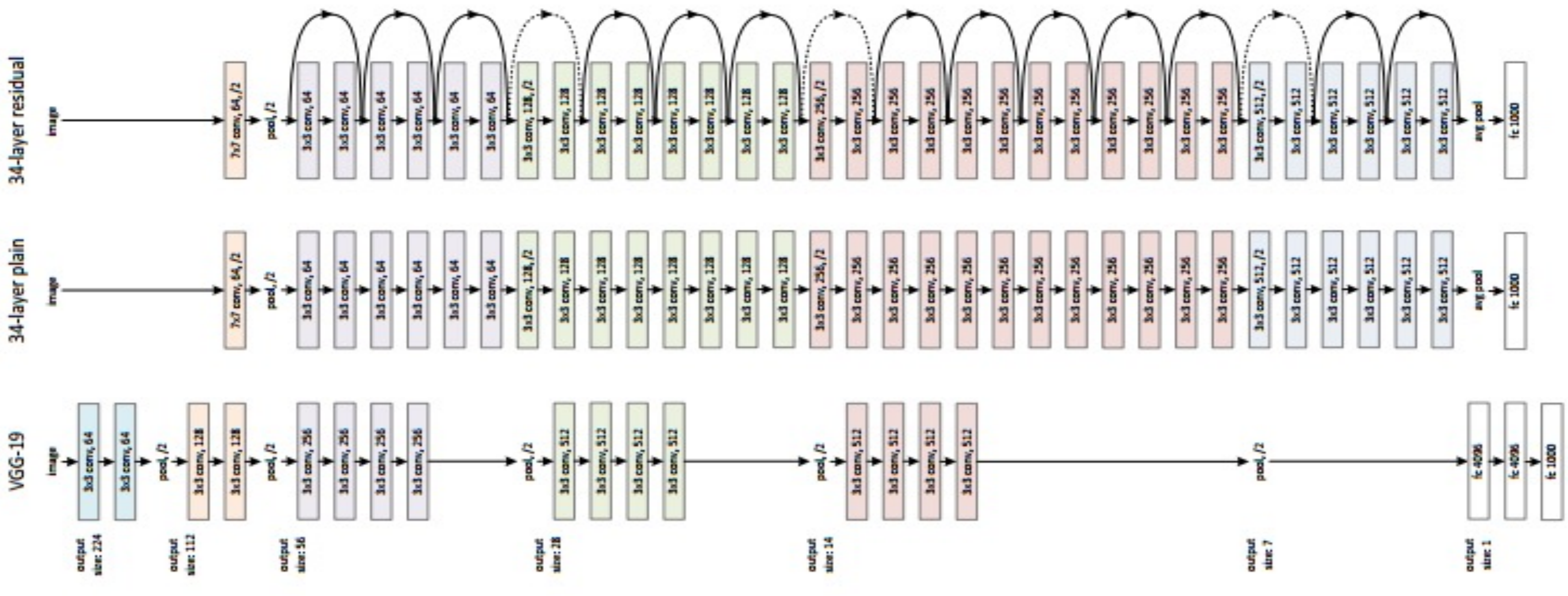

ReLU
等宽卷积
ReLU
$f(\mathbf{x}, \theta)$
等宽卷积
$\mathbf{x}$


64-d
3x3, 64
relu
3x3, 64
relu


256-d
1x1, 64
relu
3x3, 64
relu
1x1, 256
relu

# ResNet

- 2015 ILSVRC winner （152层）
  - 错误率：3.57%

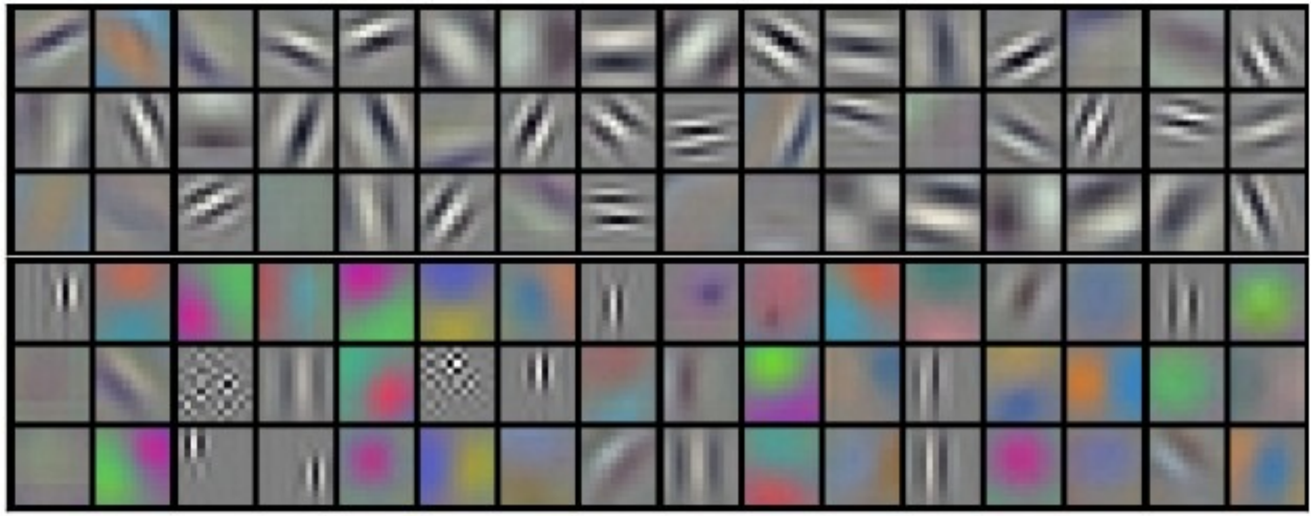


# ResNet的简单实现

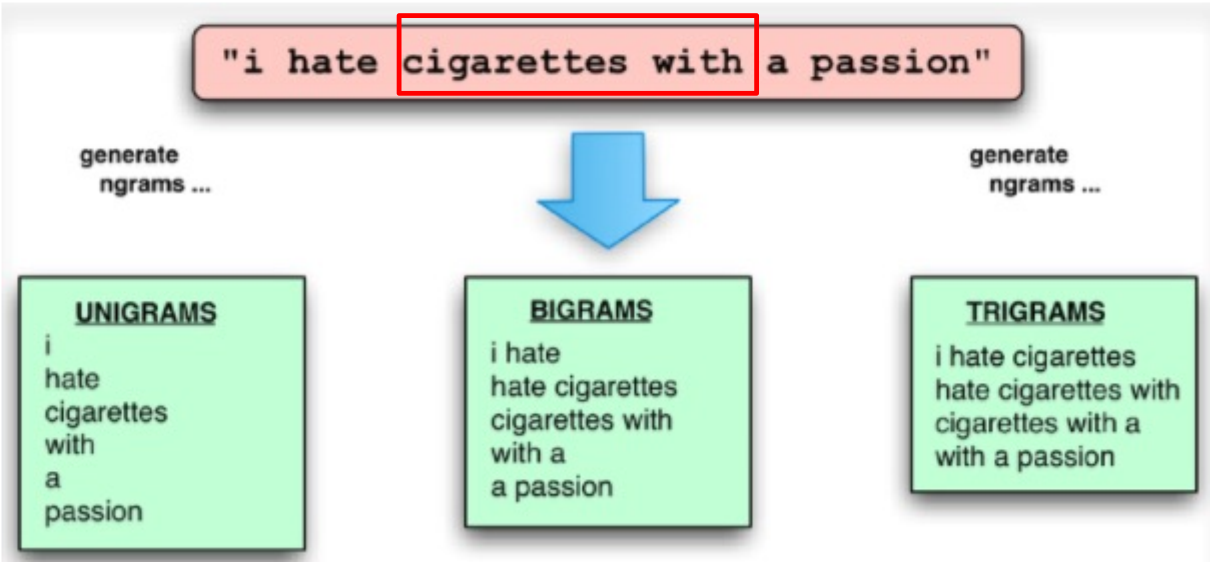
```python
class Residual(nn.Module):  # 本类已保存在d2lzh_pytorch包中方便以后使用
    def __init__(self, in_channels, out_channels, use_1x1conv=False, stride=1):
        super(Residual, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1, stride=stride)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride)
        else:
            self.conv3 = None
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        return F.relu(Y + X)
```

# CNN 可视化：滤波器

- AlexNet中的滤波器（96 filters [11x11x3]）

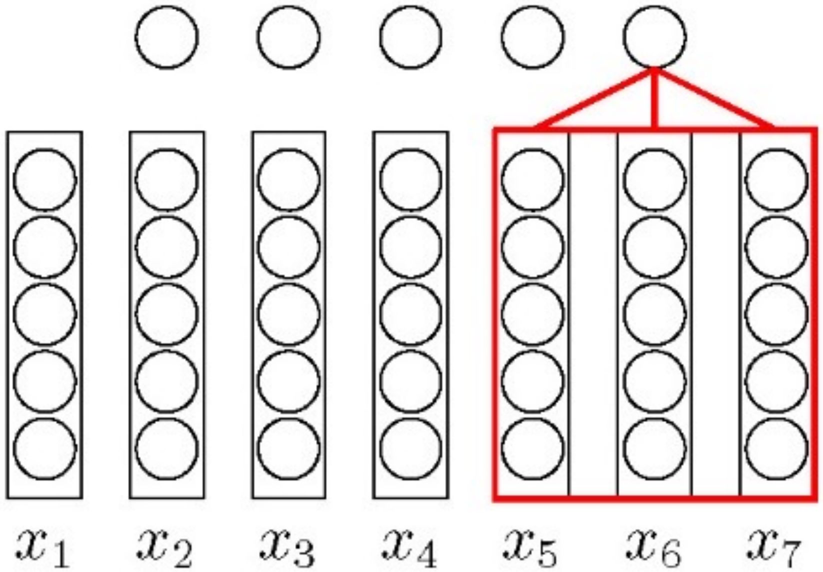


# Ngram特征与卷积

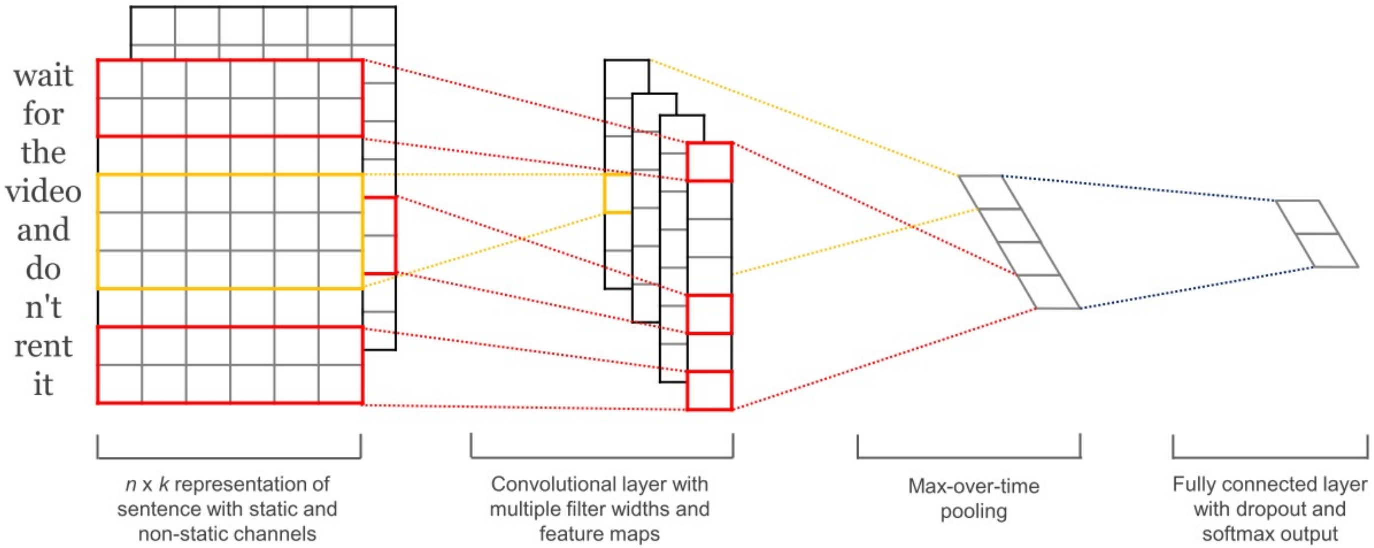



如何用卷积操作来实现?

# 文本序列的卷积

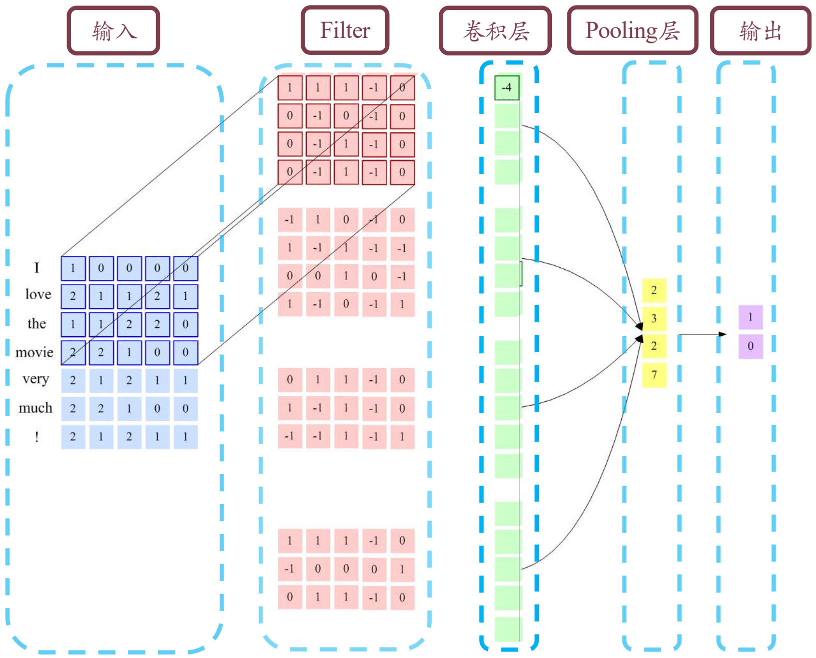


# 基于卷积模型的句子表示

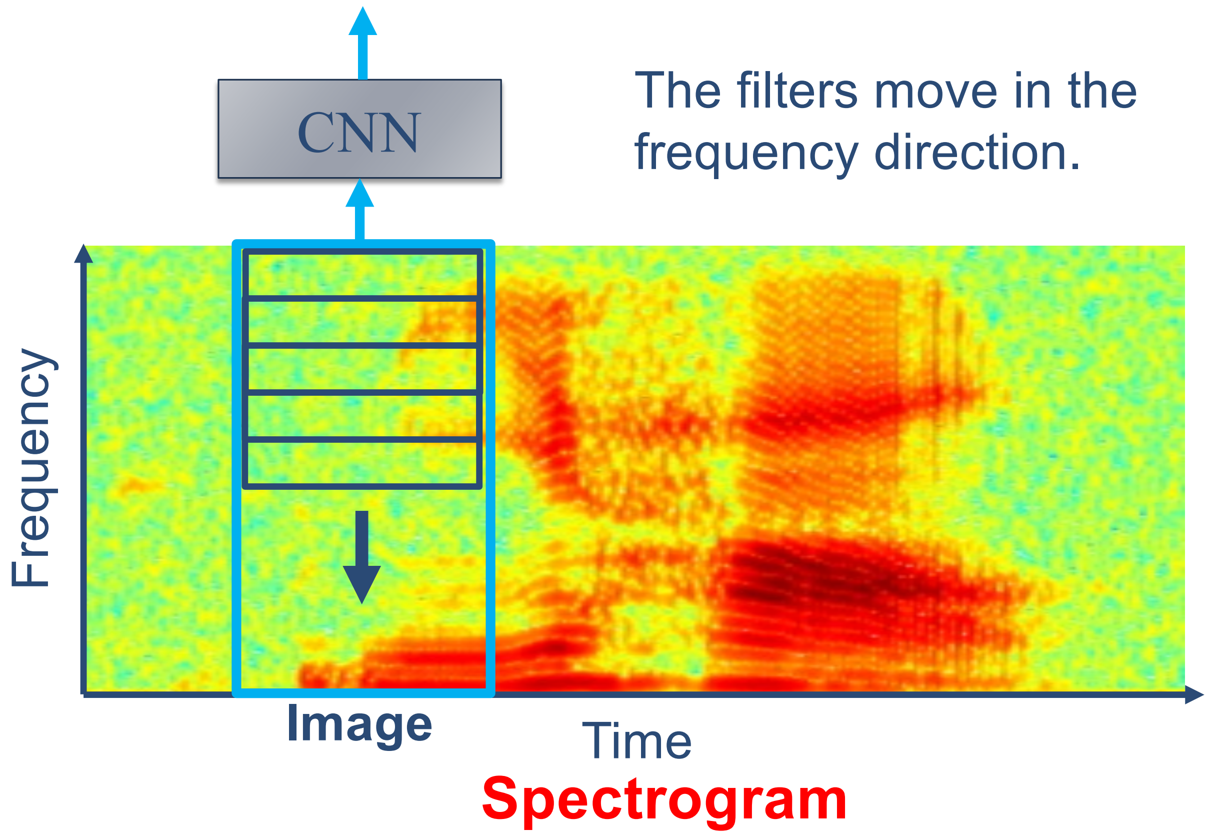



Y. Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

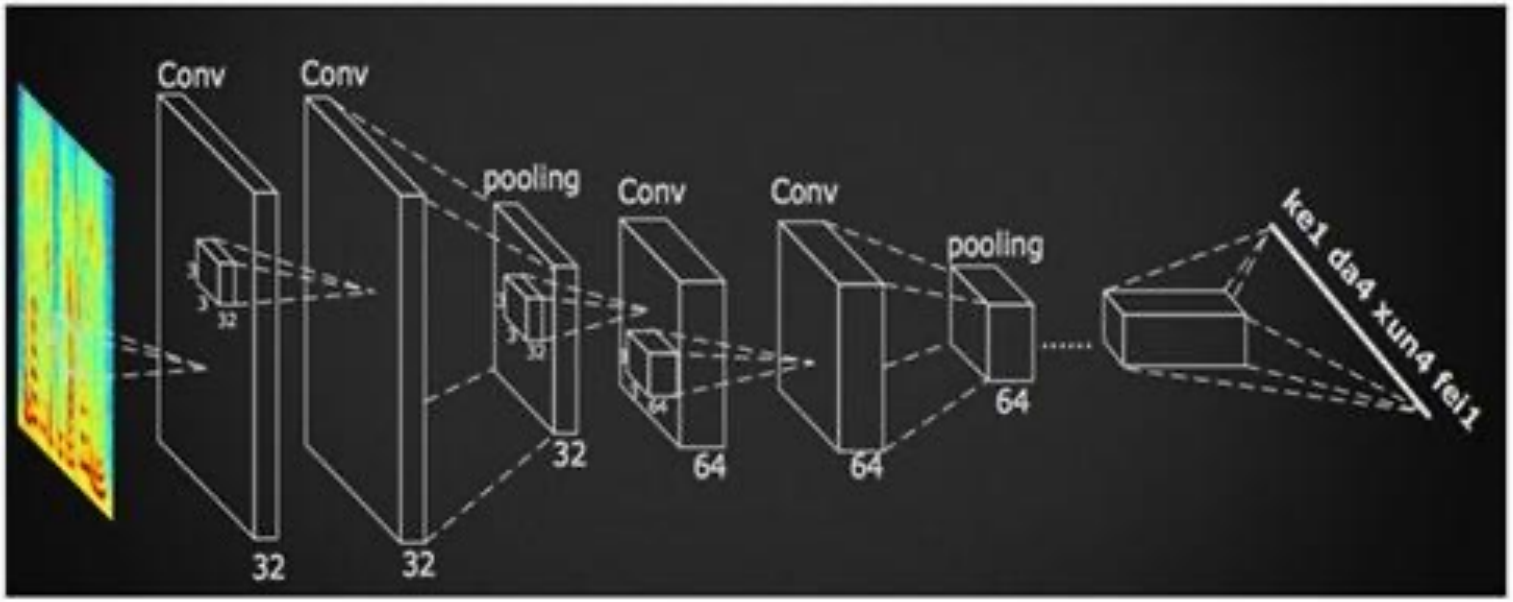
# 文本序列的卷积模型



# 基于卷积模型的语音表示

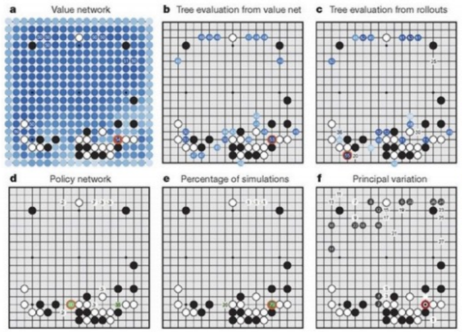


# 语音识别的DFCNN框架



# 卷积的应用

# AlphaGo





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)*

分布式系统：1202 个CPU 和176 块GPU
单机版：48 个CPU 和8 块GPU
走子速度：3 毫秒-2 微秒

# Playing Go

Training: record of previous plays 黑: 5之五 → 白: 天元 → 黑: 五之5

…


CNN

Target:
"天元" = 1
else = 0

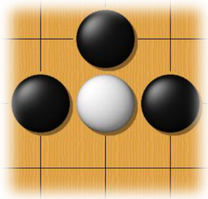

CNN

Target:
"五之5" = 1
else = 0

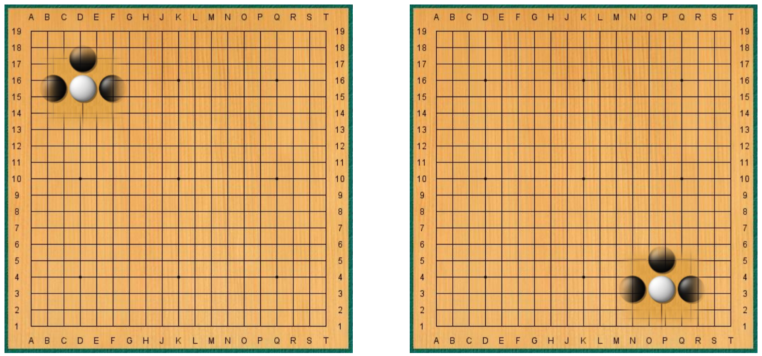
# Why CNN for playing Go?

- Some patterns are much smaller than the whole image

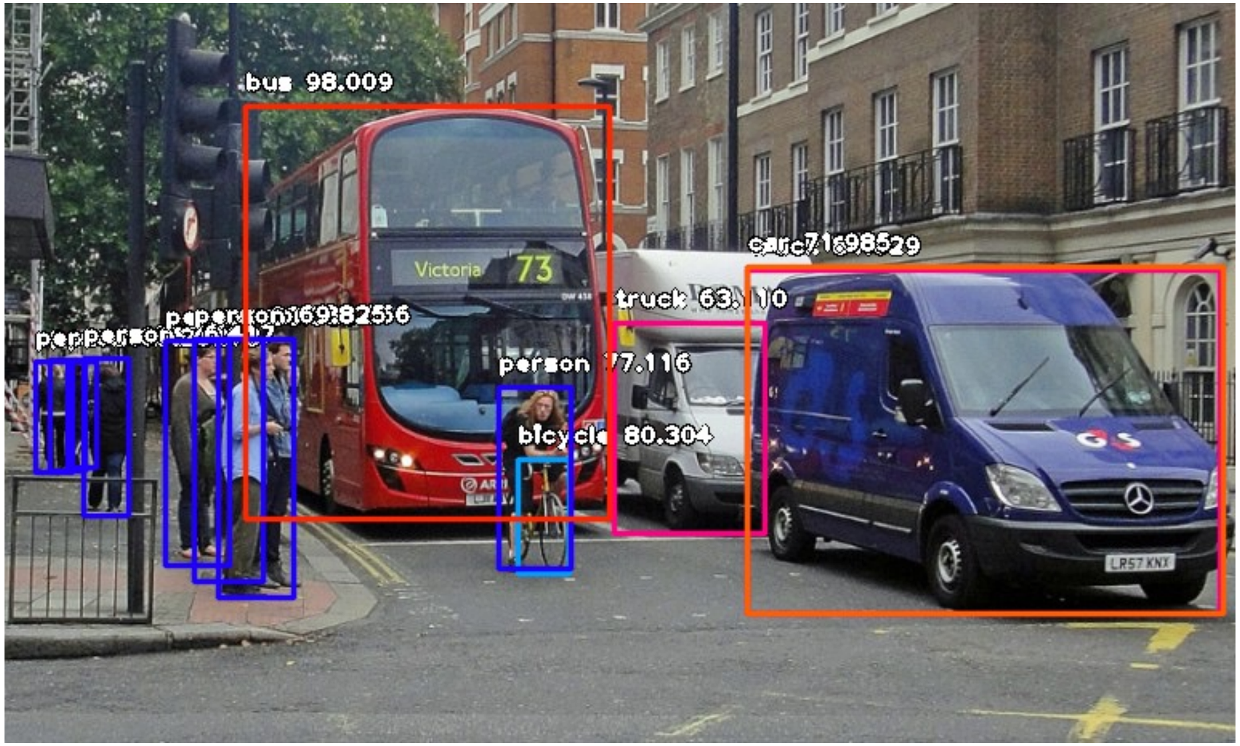
Alpha Go uses 5 x 5 for first layer



- The same patterns appear in different regions.



# 目标检测（Object Detection）

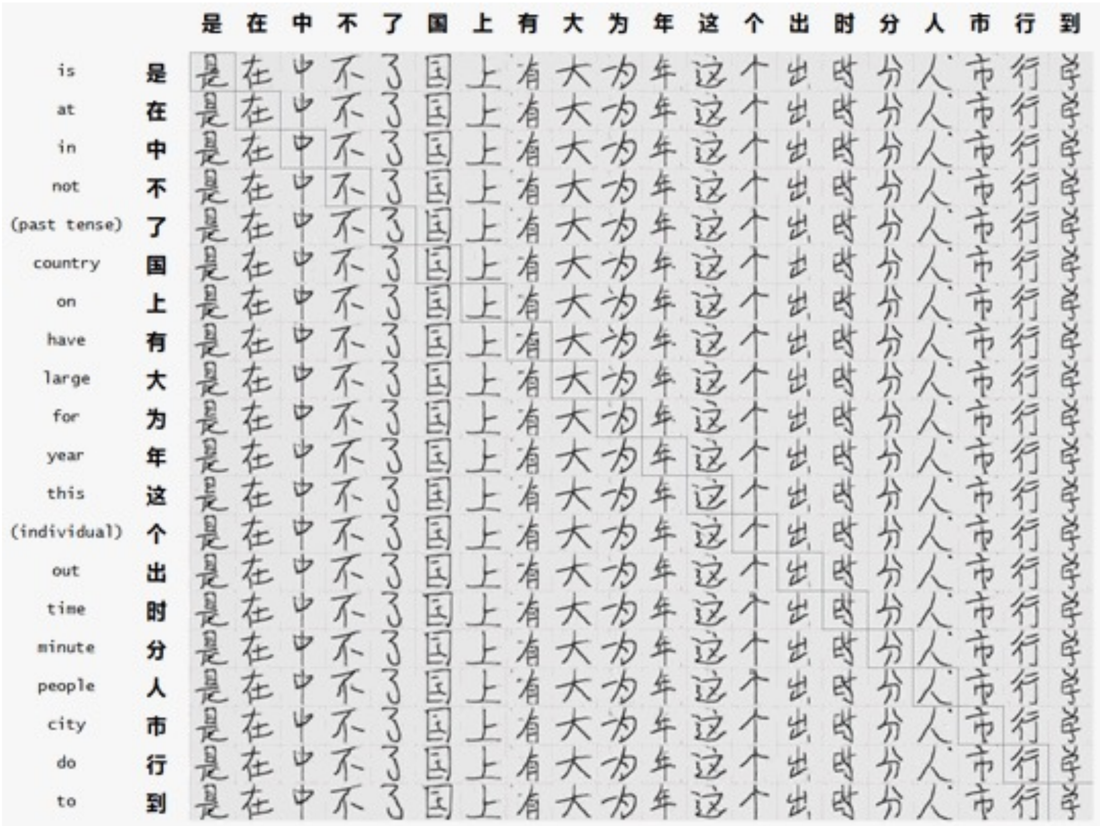


# 语义分割(Semantic Segmentation)

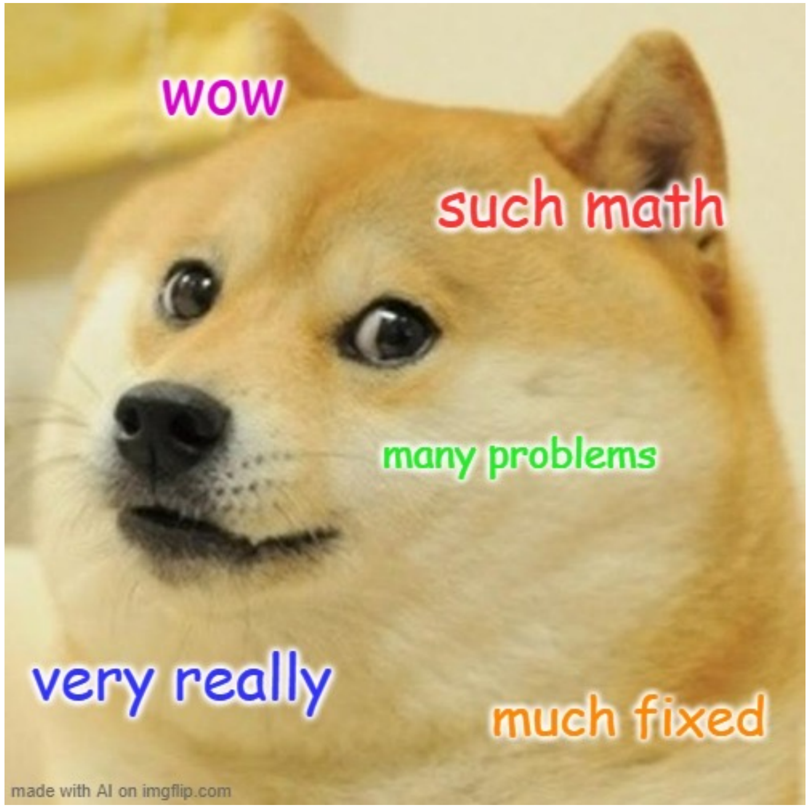
Figure 4. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

# OCR(optical character recognition)

# 图像生成(image generation)

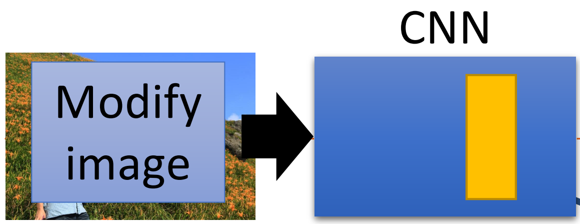


# 表情包生成(meme generation)

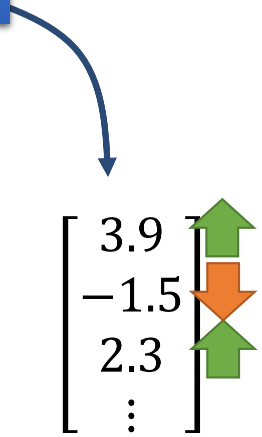



https://imgflip.com/ai-meme

# Deep Dream




- Given a photo, machine adds what it sees ……





http://deepdreamgenerator.com/

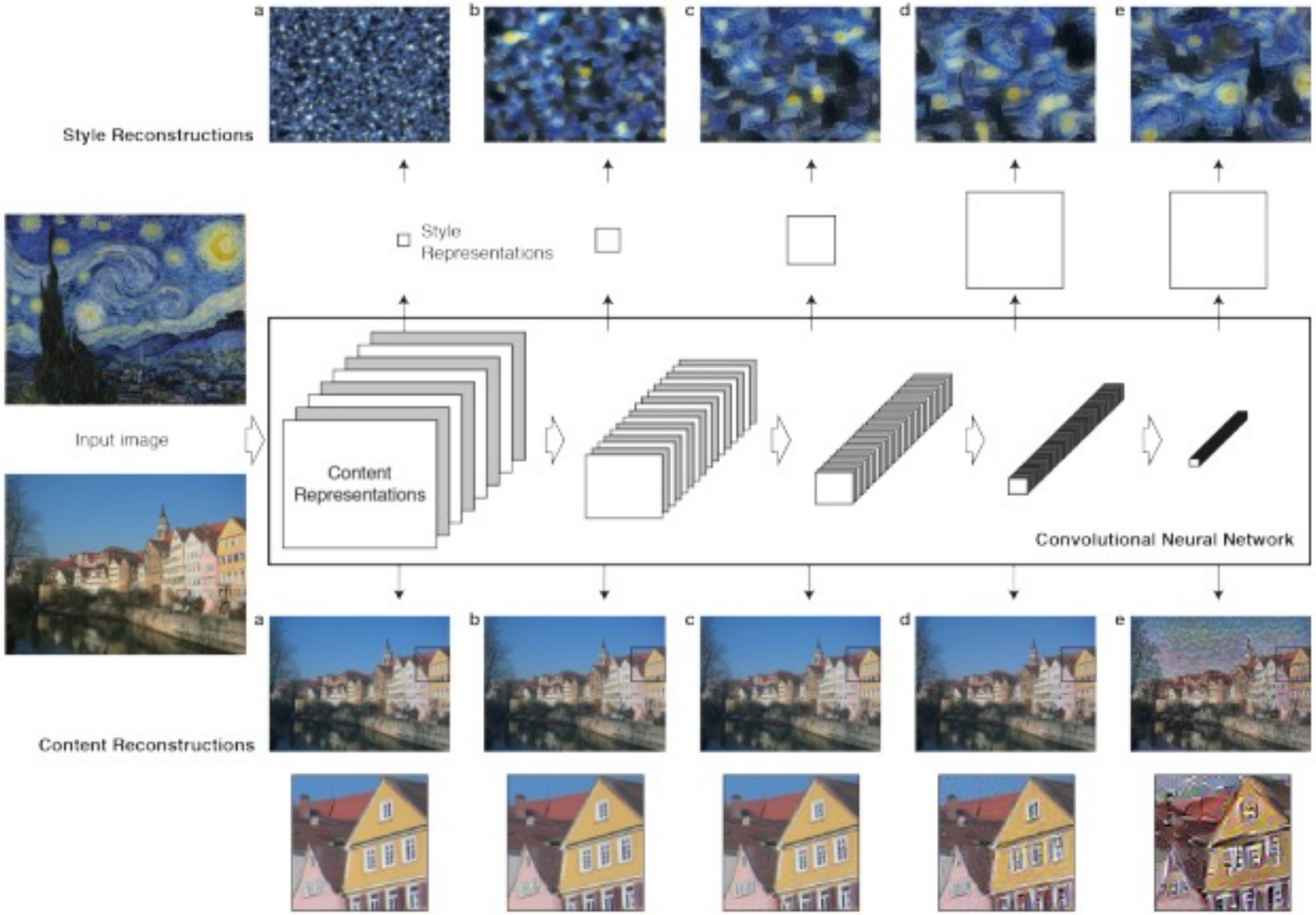
# Deep Dream

- Given a photo, machine adds what it sees ……

http://deepdreamgenerator.com/

# 画风迁移(image style transfer)

# 画风迁移


a
b
c
d
e
Style Reconstructions
Style
Representations
Input image
Content
Representations
Convolutional Neural Network
a
b
c
d
e
Content Reconstructions