CNN

- Convolutional Neural Network

概要

- 卷积
  - 平移不变性和局部性
  - 卷积层
  - 填充和步幅
- 多个输入和输出通道
- 池化
- 批标准化
- 丢弃

分类图像中的狗和猫

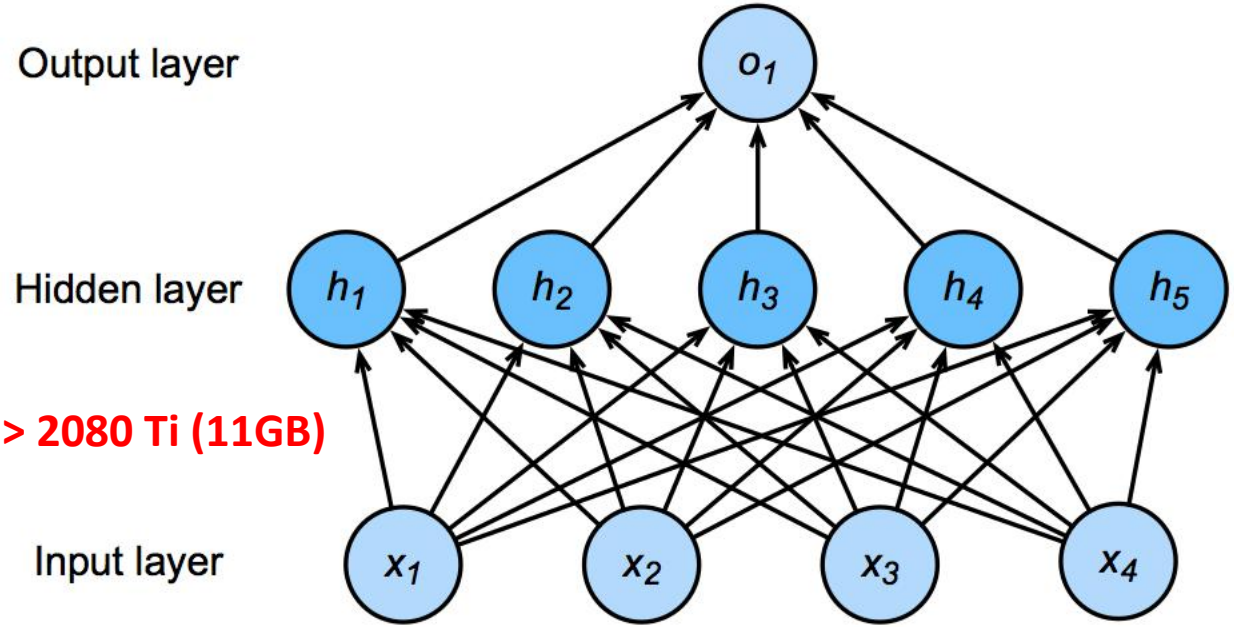
- 使用较好相机，得到的RGB图像具有 36M 个元素
- 使用 100 个神经元单隐含层的 MLP 模型：
  - 有 36 亿个参数
  - 超过地球上的狗和猫的数量（900M 狗+ 600M 猫）
  - 占用14.4GB存储

单隐层网络

100个 神经元

3.6B 参数 = **14.4GB > 2080 Ti (11GB)**

36M 特征



$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

重新思考 - 稠密层

- 将输入和输出变形为矩阵（宽度，高度）
- 将权重变形为4-D张量（h, w）到（h’, w’）

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

原则 #1 - 平移不变性

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

- x 的变化也导致 h 的变化
- V 不应该依赖于（i，j），所以 $v_{i,j,a,b} = v_{a,b}$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

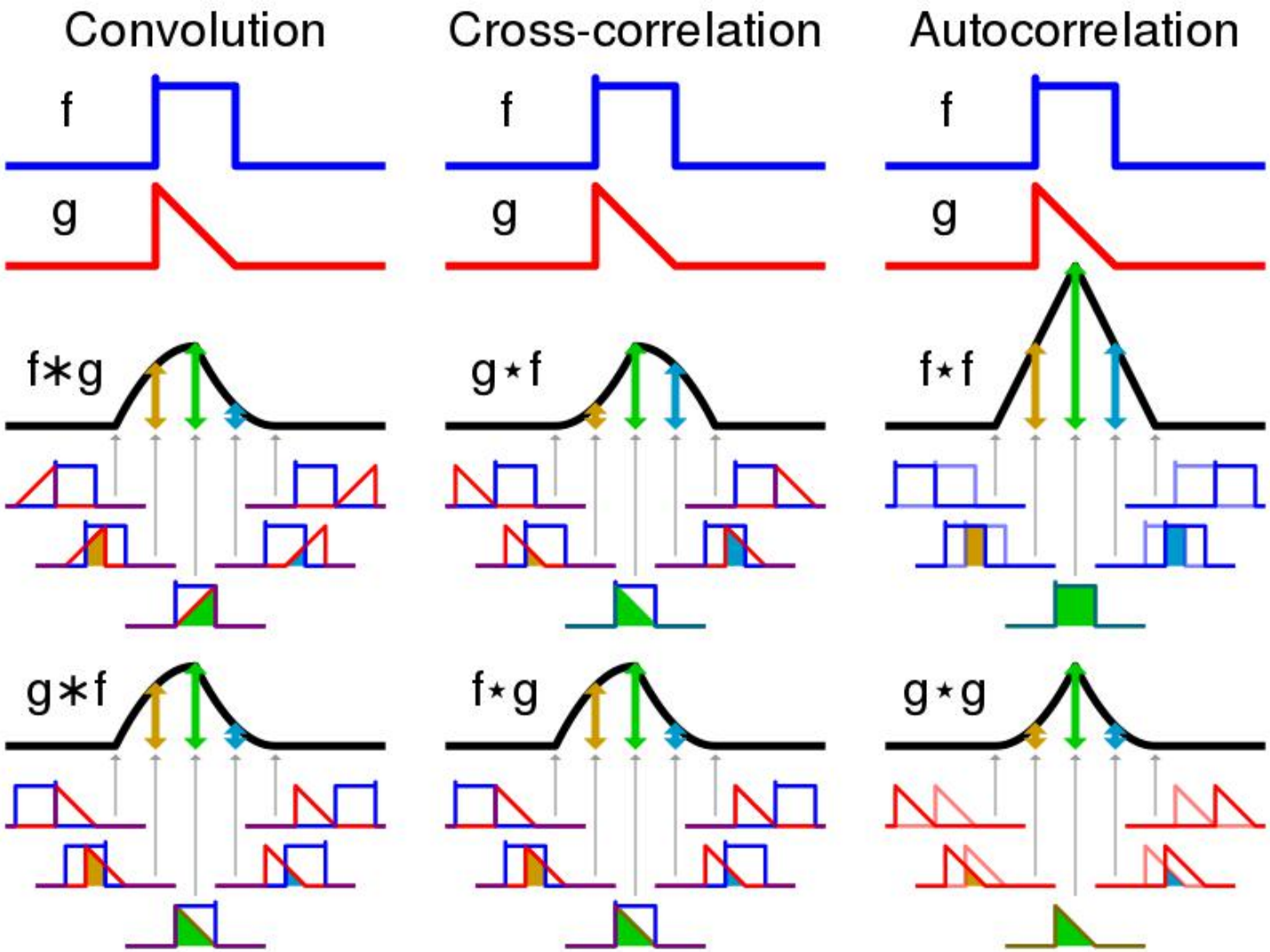
原则 #2 - 局部性

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

- 当评估 h(i,j) 时，我们不应该用远离 x(i, j) 的参数
- 外部参数 $|a|, |b| > \Delta$ 消失，$v_{a,b} = 0$

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

卷积层


Convolution
Cross-correlation
Autocorrelation
f
g
f∗g
g⋆f
f⋆f
g∗f
f⋆g
g⋆g

二维互相关

Input

0	1	2
3	4	5
6	7	8

*

Kernel

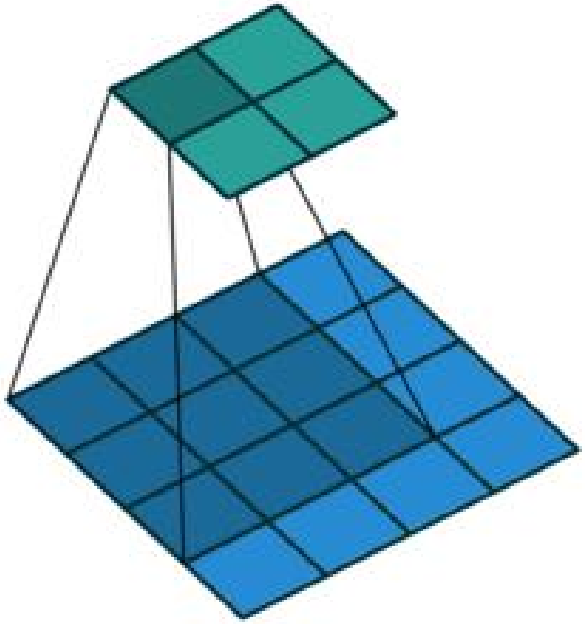
0	1
2	3

=

Output

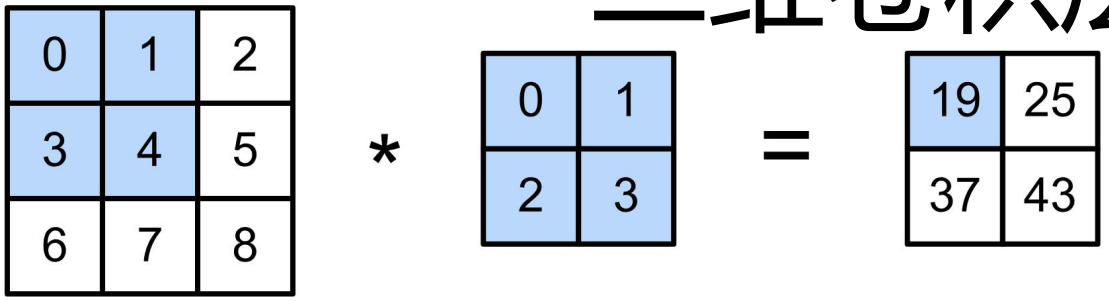
19	25
37	43

$$0\times0+1\times1+3\times2+4\times3=19,$$
$$1\times0+2\times1+4\times2+5\times3=25,$$
$$3\times0+4\times1+6\times2+7\times3=37,$$
$$4\times0+5\times1+7\times2+8\times3=43.$$



(vdumoulin@ Github)

二维卷积层



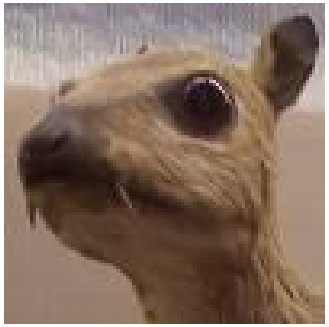
- **X**: $n_h \times n_w$ 输入矩阵
- **W**: $k_h \times k_w$核矩阵
- b:偏差标量
- **Y**: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ 输出矩阵

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- **W** 和 b 是可学习的参数

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

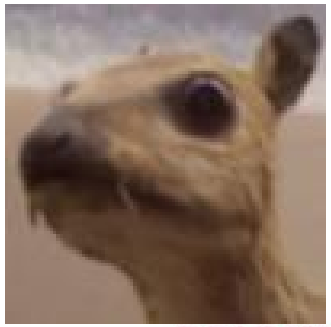
边缘检测



$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

锐化

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



高斯模糊

互相关与卷积

- 2-D 互相关

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{a,b} x_{i+a,j+b}$$

- 2-D 卷积

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{-a,-b} x_{i+a,j+b}$$

- 在对称性方面没有差别

1-D 和 3-D 互相关

- 1-D

$$y_i = \sum_{a=1}^{h} w_a x_{i+a}$$

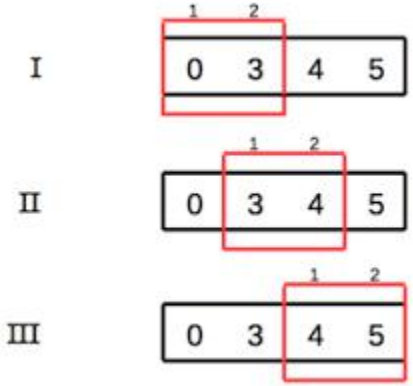
  - 文本
  - 语音
  - 时间序列

- 3-D

$$y_{i,j,k} = \sum_{a=1}^{h} \sum_{b=1}^{w} \sum_{c=1}^{d} w_{a,b,c} x_{i+a,j+b,k+c}$$

  - 视频
  - 医学图像

Conv1d

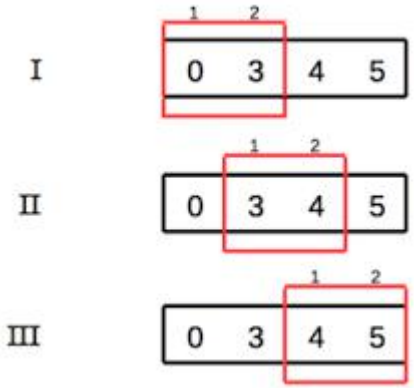


torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'**)**

In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output $(N, C_{\text{out}}, L_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

Conv1d



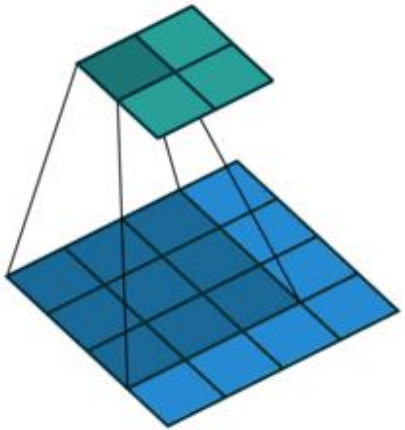
torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'**)**

Shape:

- Input: (N, C_{in}, L_{in})
- Output: (N, C_{out}, L_{out}) where

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

Conv2d

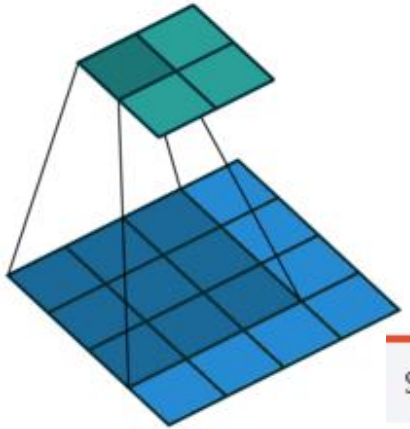


torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

Conv2d



torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

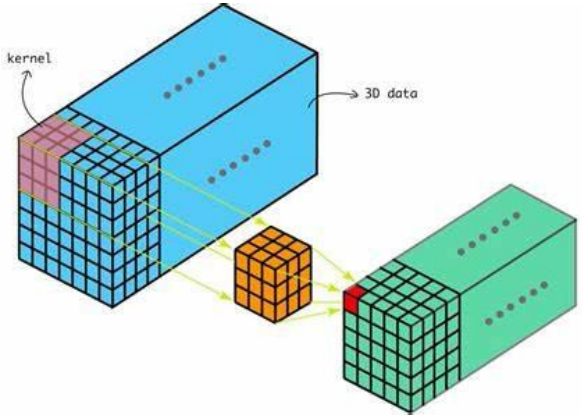
Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Conv3d

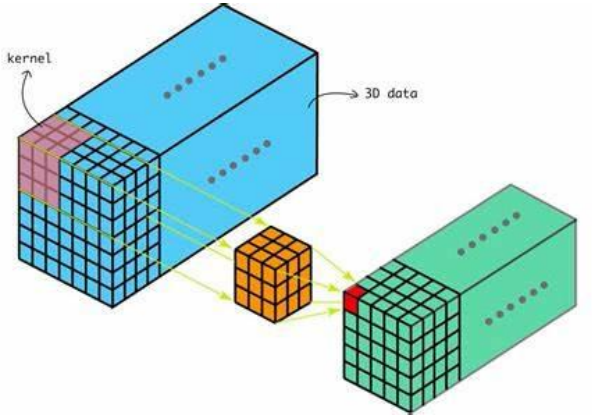


torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

In the simplest case, the output value of the layer with input size (N, C_{in}, D, H, W) and output $(N, C_{out}, D_{out}, H_{out}, W_{out})$ can be precisely described as:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k)$$

Conv3d



torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

Shape:

- Input: $(N, C_{in}, D_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, D_{out}, H_{out}, W_{out})$ where

$$D_{out} = \left\lfloor \frac{D_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

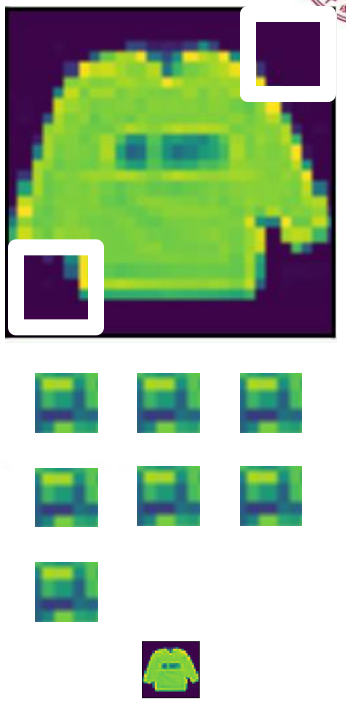
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[2] - \text{dilation}[2] \times (\text{kernel\_size}[2] - 1) - 1}{\text{stride}[2]} + 1 \right\rfloor$$

填充

- 给定输入图像（32 x 32）
- 应用5 x 5大小的 卷积核
- 第1层得到输出大小28 x 28
- 第7层得到输出大小4 x 4
- 更大的卷积核可以更快地减小输出
- 形状从$n_h \times n_w$减少到

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



填充

填充： 在输入周围添加额外的行 / 列

Input

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

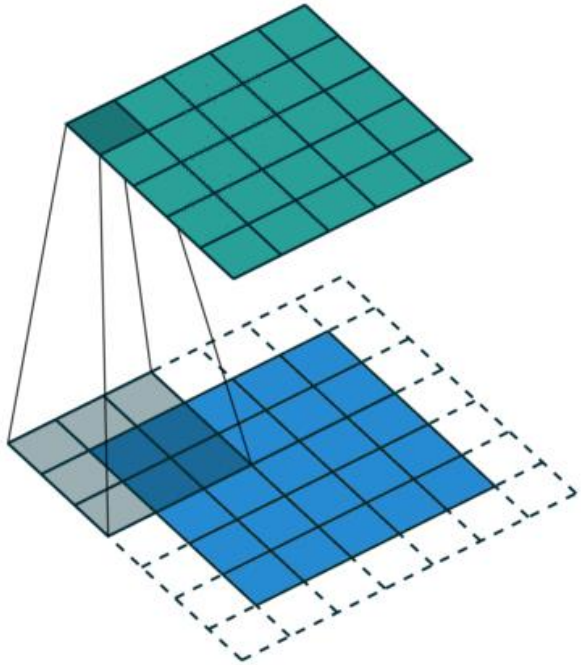
Kernel

0	1
2	3

=

Output

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0



$$0\times0 + 0\times1 + 0\times2 + 0\times3 = 0$$

填充

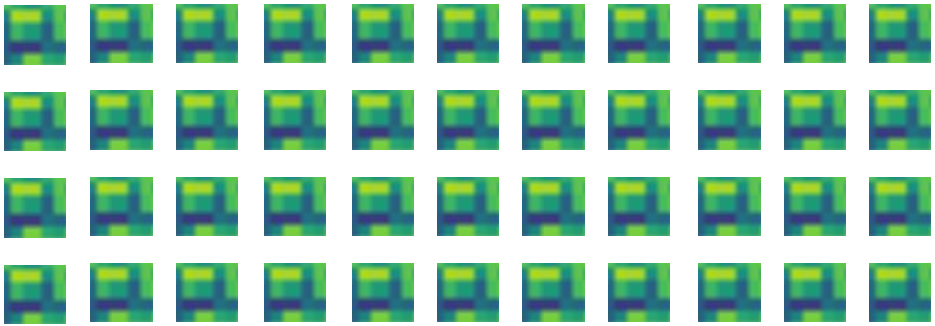
- 填充 p_h 行和 p_w 列，则输出为：

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- 通常取 $p_h = k_h - 1$， $p_w = k_w - 1$
  - 当 k_h为奇数：在上下两侧填充 $p_h/2$
  - 当 k_h为偶数：在上侧填充 $\lceil p_h/2 \rceil$,在下侧填充 $\lfloor p_h/2 \rfloor$

步幅

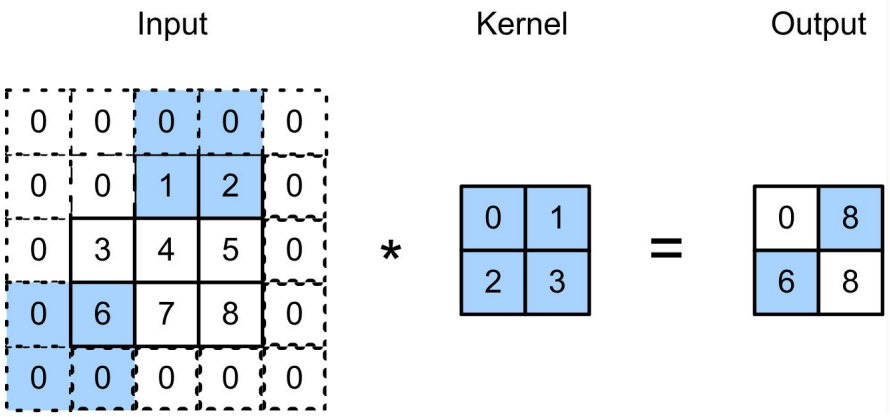
- 填充降低的输出大小与层数线性相关
  - 给定输入大小224*224，在使用5*5卷积核的情况下，需要44层将输出降低到4*4
  - 需要大量的计算



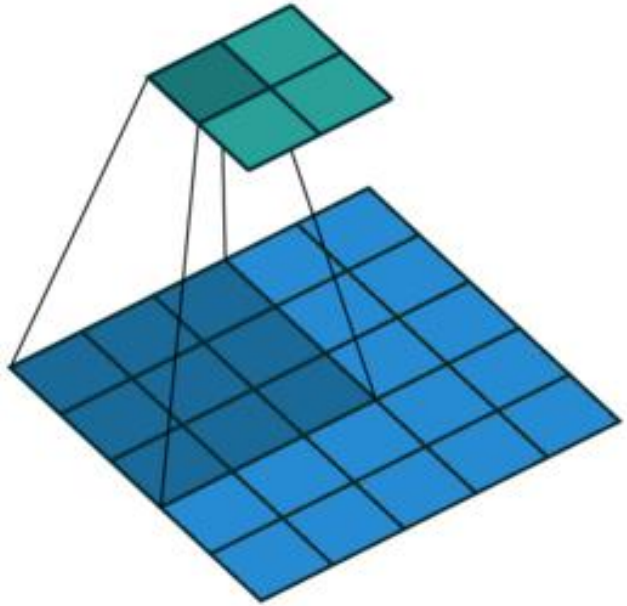
步幅

- 步幅是指行/列的滑动步长

例：高度3 宽度2 的步幅



$$0\times0 + 0\times1 + 1\times2 + 2\times3 = 8$$
$$0\times0 + 6\times1 + 0\times2 + 0\times3 = 6$$



步幅

- 给出高度 s_h 和宽度 s_w 的步幅，输出形状是

  $\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$

- 如果 $p_h = k_h - 1$， $p_w = k_w - 1$

  $\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$

- 如果输入高度和宽度可以被步幅整除

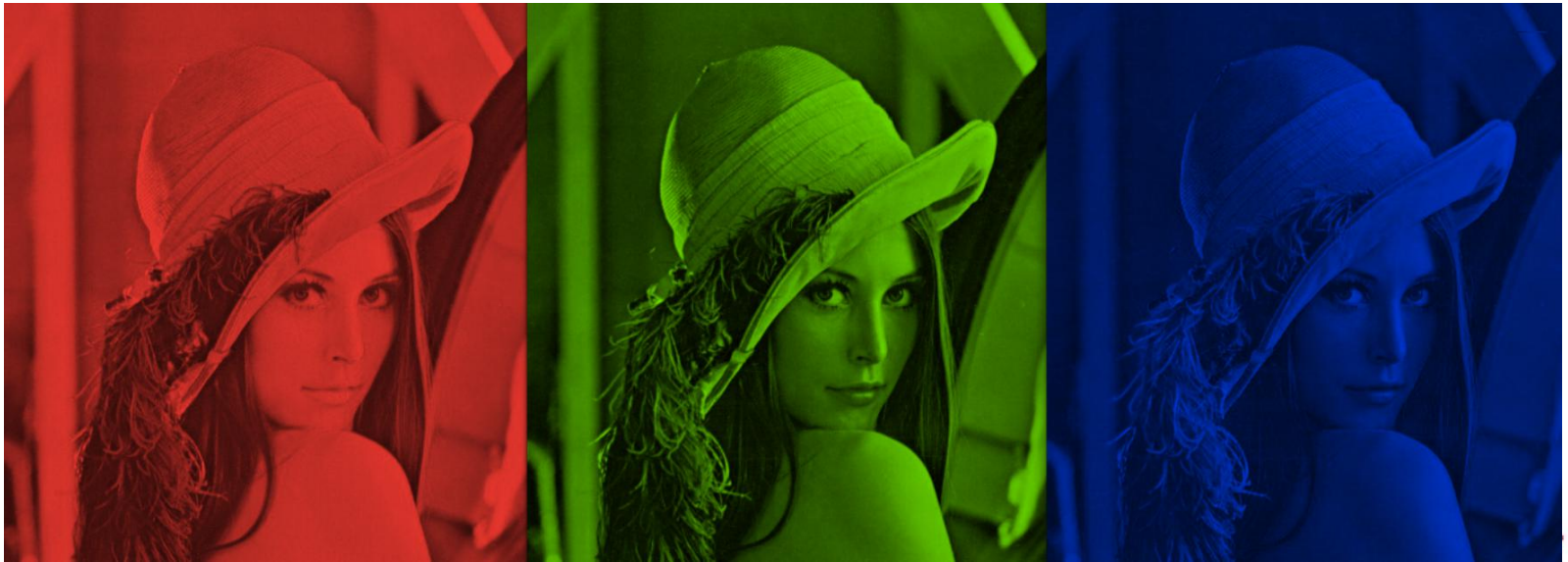
  $(n_h/s_h) \times (n_w/s_w)$

多个输入通道

- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息

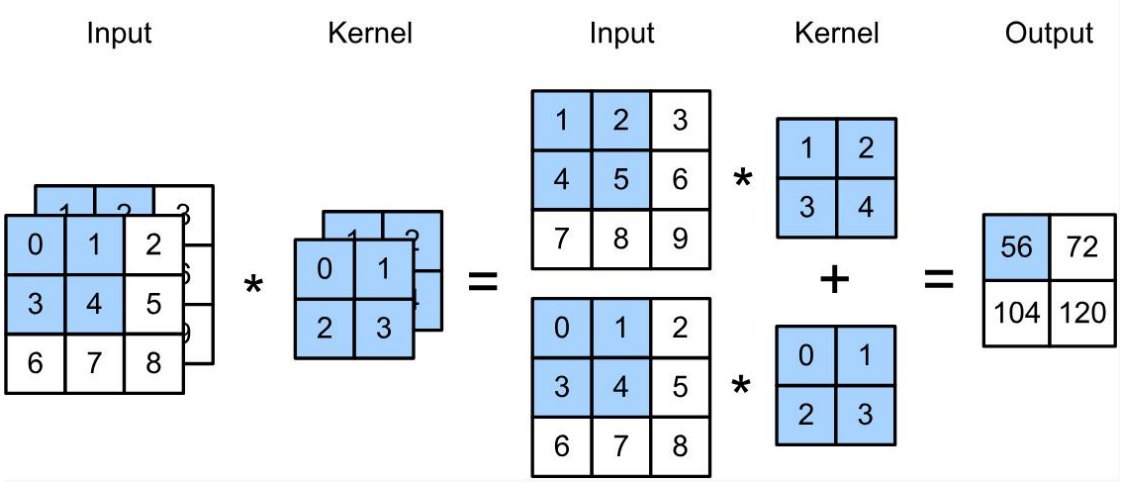
多个输入通道

- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息



多个输入通道

- 每个通道都有一个卷积核，结果是所有通道卷积结果的和



$$(1\times1+2\times2+4\times3+5\times4)$$
$$+(0\times0+1\times1+3\times2+4\times3$$
$$=56$$

多个输入通道

- $\mathbf{X}: c_i \times n_h \times n_w$ 输入
- $\mathbf{W}: c_i \times k_h \times k_w$ 卷积核
- $\mathbf{Y}: m_h \times m_w$ 输出

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

多个输出通道

- 无论有多少输入通道，到目前为止我们只用到单输出通道
- 我们可以有多个3-D卷积核，每个核生成一个输出通道

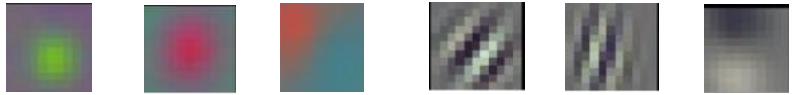
- 输入 $\mathbf{X}: c_i \times n_h \times n_w$
- 内核 $\mathbf{W}: c_o \times c_i \times k_h \times k_w$
- 输出 $\mathbf{Y}: c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

$$for i = 1, \ldots, c_o$$

多个输入和输出通道

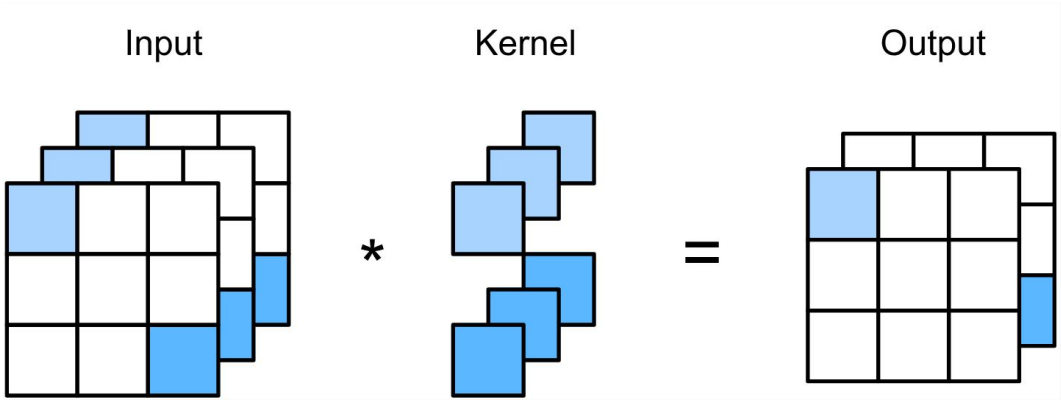
- 每个输出通道可以识别特定模式



- 输入通道内核识别并组合输入中的模式

1x1 卷积层

$k_h = k_w = 1$ 是一个受欢迎的选择。
它不识别空间模式，只是融合通道。



相当于具有输入 $n_h n_w \times c_i$ 和重量 $c_o \times c_i$ 的稠密层 。

2-D 卷积层

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + \mathbf{B}$$

- 输入 X：$c_i \times n_h \times n_w$
- 卷积核 W：$c_i \times c_o \times k_h \times k_w$
- 偏差 B：$c_i \times c_o$
- 输出 Y：$c_o \times m_h \times m_w$

复杂性（浮点运算FLOP的数量）

$$c_i = c_o = 100$$
$$k_h = h_w = 5$$
$$m_h = m_w = 64$$

$O(c_i c_o k_h k_w m_h m_w)$ = 1GF

10层，1M示例：10PF
(CPU：0.98 TF = 2.8h,
GPU：12 TF = 14min)

池化层

- 卷积对位置敏感
  - 检测垂直边缘

0输出，1像素移位

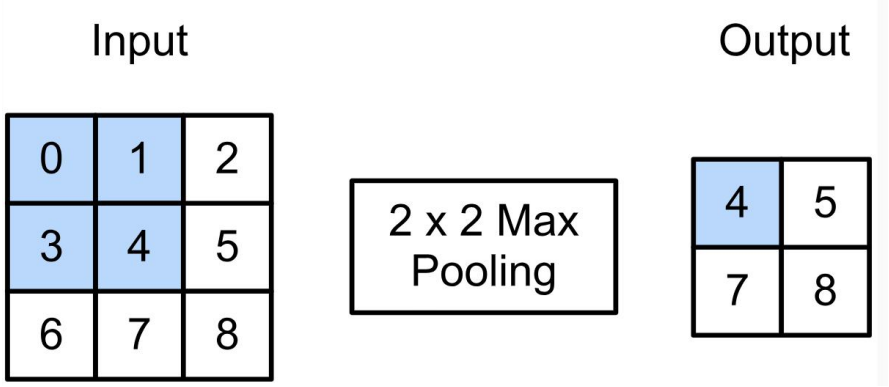
```
X  [[1. 1. 0. 0. 0.
    [1. 1. 0. 0. 0.
    [1. 1. 0. 0. 0.
    [1. 1. 0. 0. 0.

Y  [[ 0.  1.  0.  0.
    [ 0.  1.  0.  0.
    [ 0.  1.  0.  0.
    [ 0.  1.  0.  0.
```

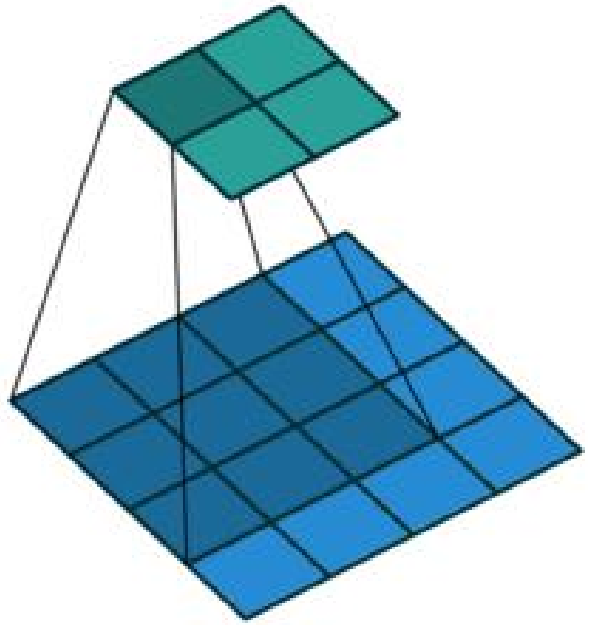
- 需要一定程度的平移不变性
  - 照明，物体位置，比例，外观等等因图像而异

2-D 最大池化

- 返回滑动窗口中的最大值



$$max(0,1,3,4) = 4$$



2-D 最大池化

- 返回滑动窗口中的最大值

垂直边缘检测

```
[[1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
```

卷积输出

```
[[ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
```

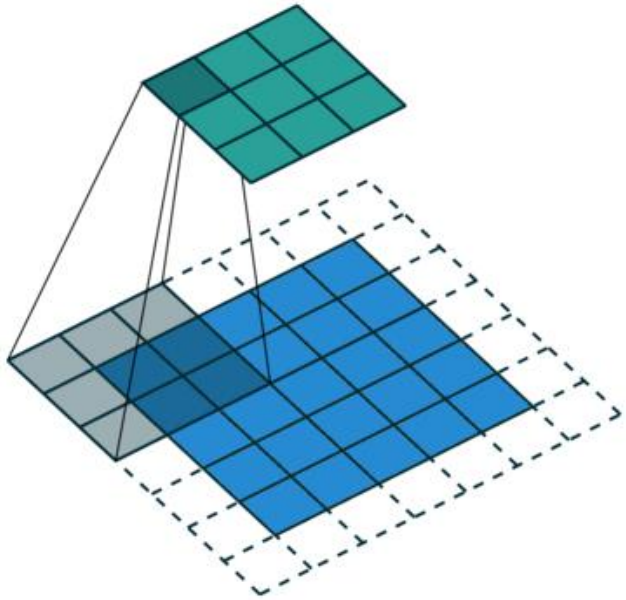
2-D 最大池化

```
[[ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
```

可容1像素移位

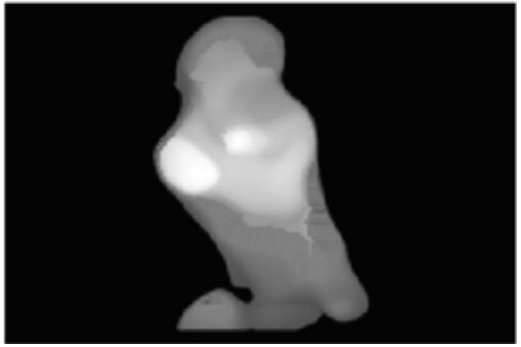
池化层 - 填充，步幅和多个通道

- 池化层与卷积层类似，都具有填充和步幅
- 没有可学习的参数
- 在每个输入通道应用池化层以获得相应的输出通道
- #输出通道 = #输入通道

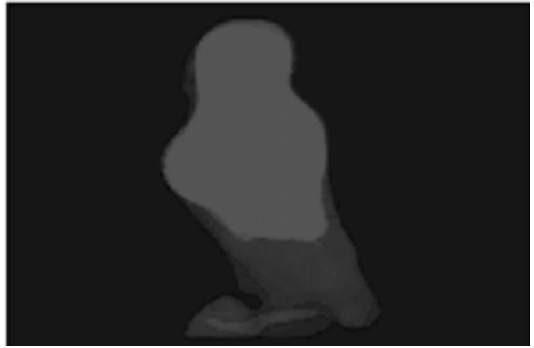


平均池化层

- 最大池化层：每个窗口中最强的模式信号
- 平均池化层：
  - 将最大池化层中的“最大”操作替换为“平均”
  - 窗口中的平均信号强度



最大池化层

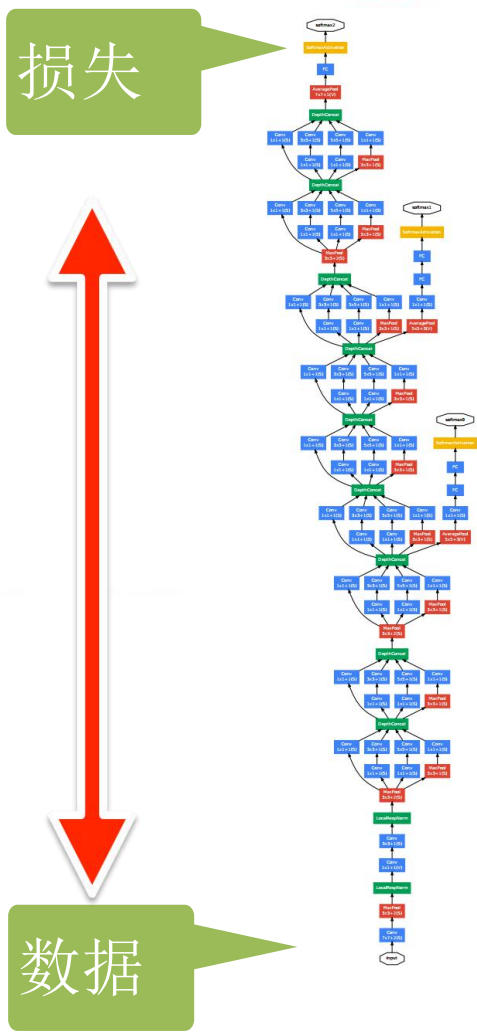


平均池化层

批量归一化

- 损失发生在最后一层
  - 最后一层可快速学习
- 数据插入在第一层（底层）
  - 底层变化 - 一切都变化
  - 最后一层需要多次重新学习
  - 收敛缓慢
- 称为协变量偏移

  我们可以避免在学习第一层时改变最后一层吗?



批量归一化

- 学习第一层时，我们可以避免更改最后一层吗？
  - 修正均值和方差

$$\mu_B = \frac{1}{|B|}\sum_{i\in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|}\sum_{i\in B}(x_i - \mu_B)^2 + \epsilon$$

  - 单独调整：

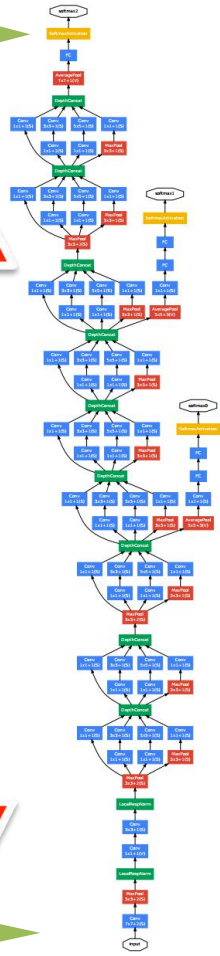
$$x_{i+1} = \gamma\frac{x_i - \mu_B}{\sigma_B} + \beta$$

均值

方差

损失

数据



批量归一化

- 通过注入噪声进行正则化

$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Random offset

Random scale

  - 每个小批量随机偏移（shift）
  - 每个小批量的随机转换比例（scale）

- 与丢弃法共用（两者都是正则化控制）
- 理想的小批量大小为 64-256

细节

- **稠密层**
  对所有神经元进行批量归一化
- **卷积层**
  每个通道一次批量归一化
- **训练计算每个小批量的新均值和方差**
- **推理时不再计算批量，重用训练批量数据**

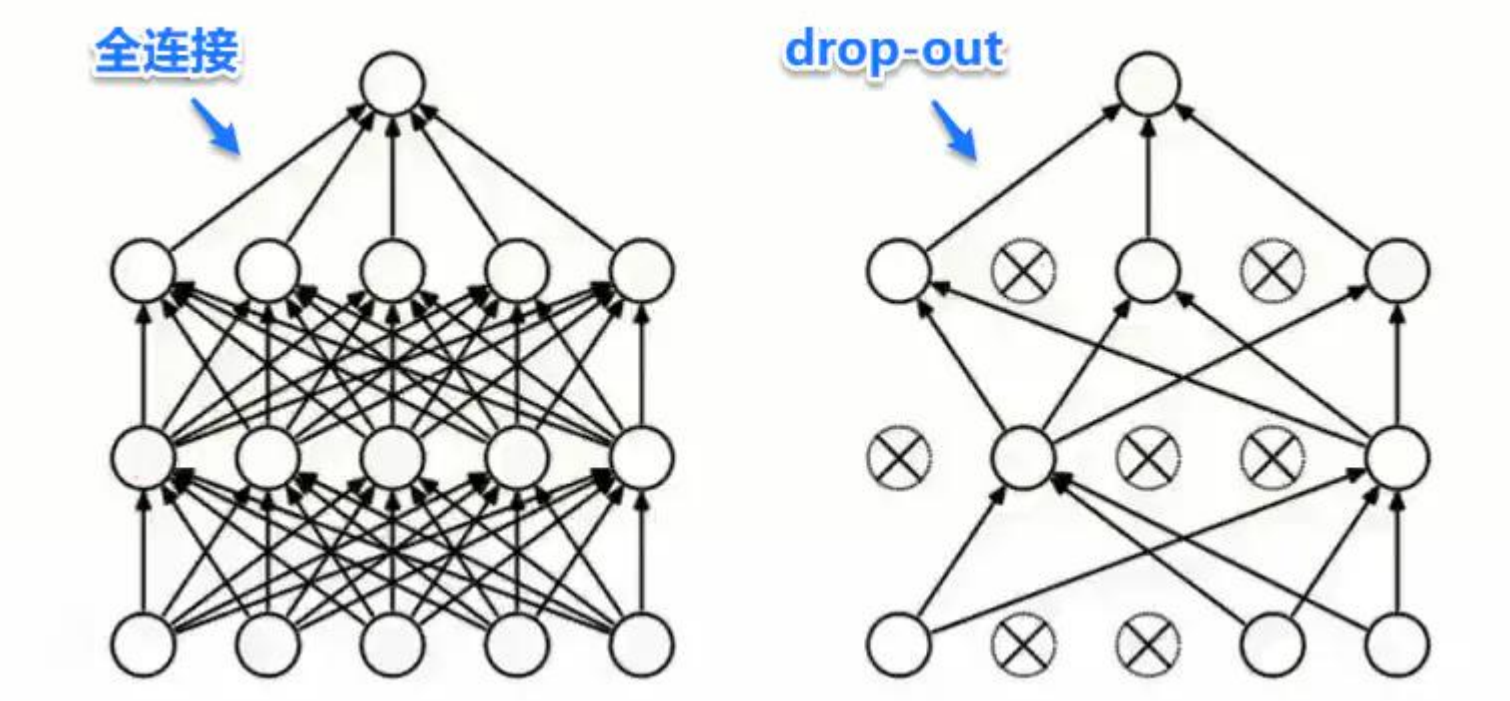
细节

- torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

Applies Batch Normalization over a 2D or 3D input (a mini-batch of 1D inputs with optional additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

丢弃法



动机

- 在输入的适度变化下，一个好的模型应该是稳定的
  - 用输入噪声训练相当于一种正则化
  - 丢弃法：将噪音注入内部隐藏层

丢弃法 – 训练

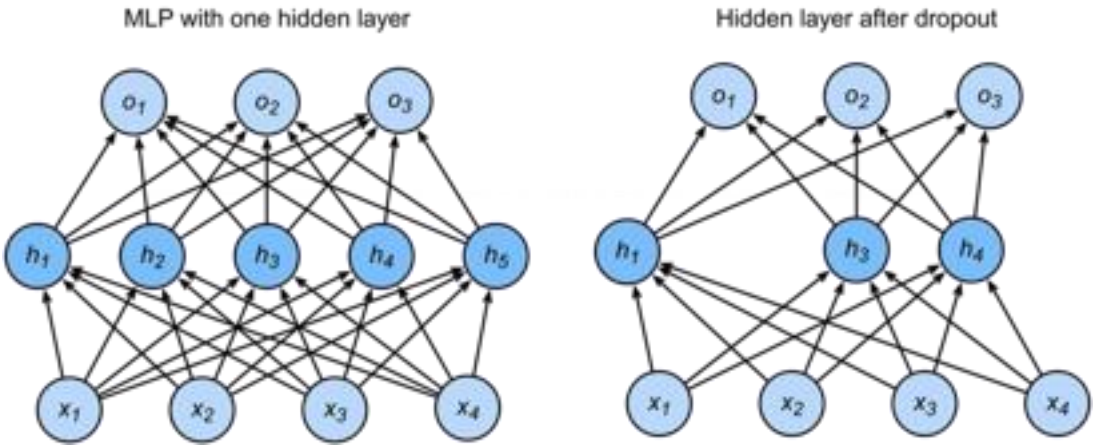
- 通常在全连接层的输出上使用丢弃法

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$

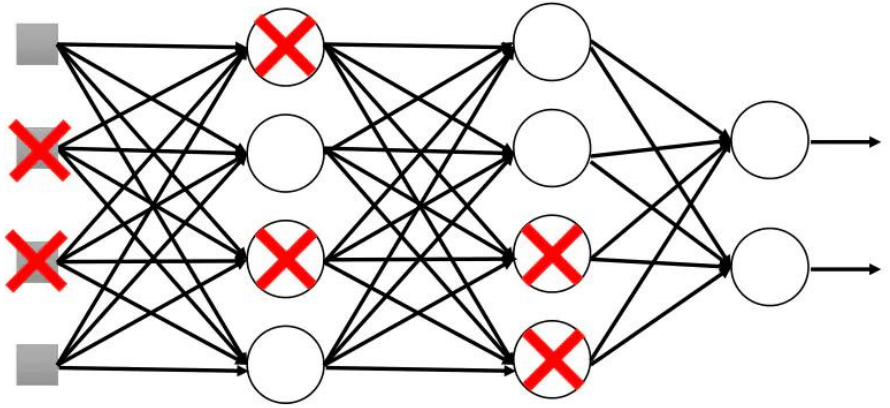
$$\mathbf{y} = \text{softmax}(o)$$



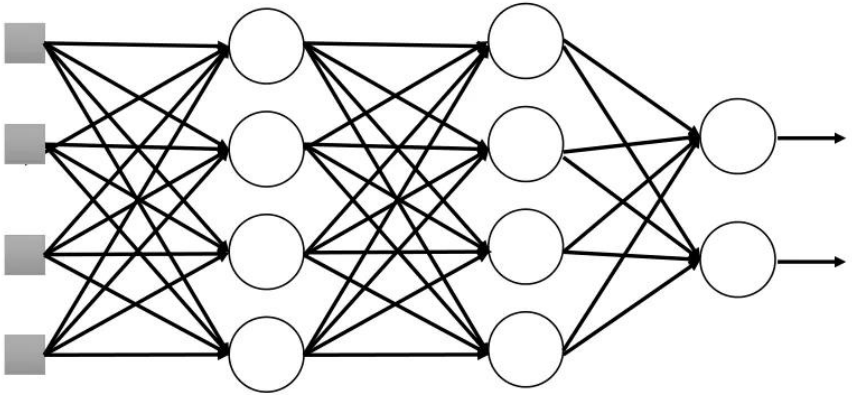
丢弃法 – 推理阶段

- 正规化仅用于模型训练

Training:



Testing:



丢弃法 – 细节

- torch.nn.Dropout(p=0.5, inplace=False)

Parameters

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

Shape:

- Input: $(*)$. Input can be of any shape
- Output: $(*)$. Output is of the same shape as input

总结

- 卷积层
  - 与稠密层相比，模型容量降低
  - 有效地检测空间模式
  - 计算复杂度高
  - 通过填充，步幅和通道控制输出形状
- 最大 / 平均池化层
  - 提供一定程度的平移不变性
- 批标准化
  - 提供一定程度的平移不变性
- 丢弃
  - 正则化