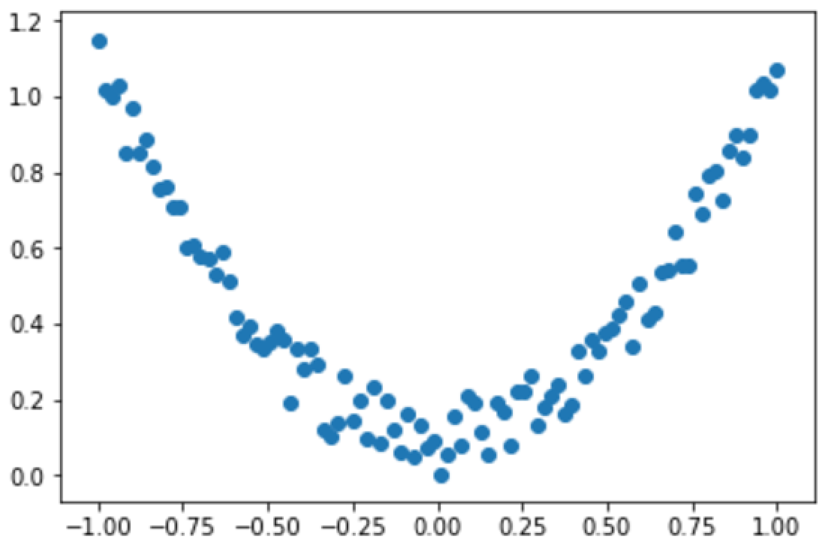
计算机视觉

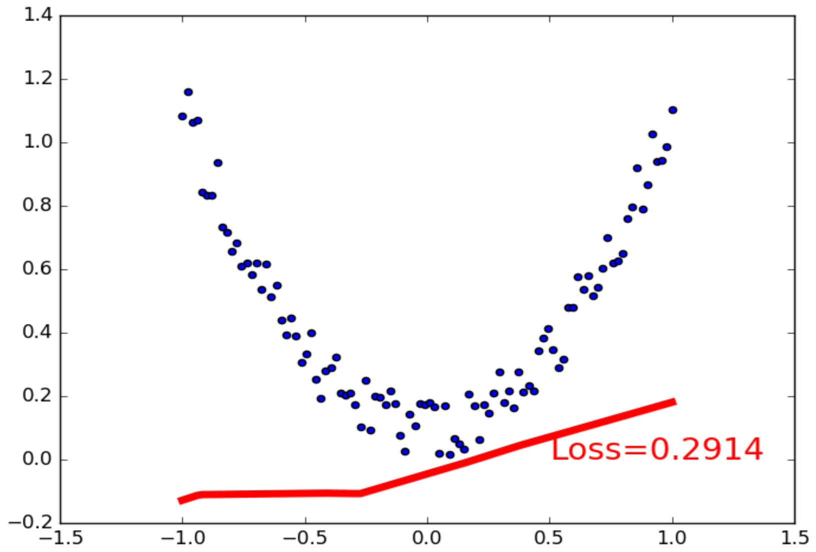
张健

数字媒体研究中心
信息工程学院
北京大学深圳研究生院

2021.10.27

作业：PyTorch搭建两层全连接网络-作业






```
torch.manual_seed(1)    # reproducible

x = torch.unsqueeze(torch.linspace(-1, 1, 100), dim=1)
y = x.pow(2) + 0.2*torch.rand(x.size())
```

1. 补全两层全连接代码 W4_Homework.ipynb
2. 给出变量W1,b1,W2,b2导数表达式

$$h = XW_1 + b_1$$

$$h_{\text{sigmoid}} = sigmoid(h)$$

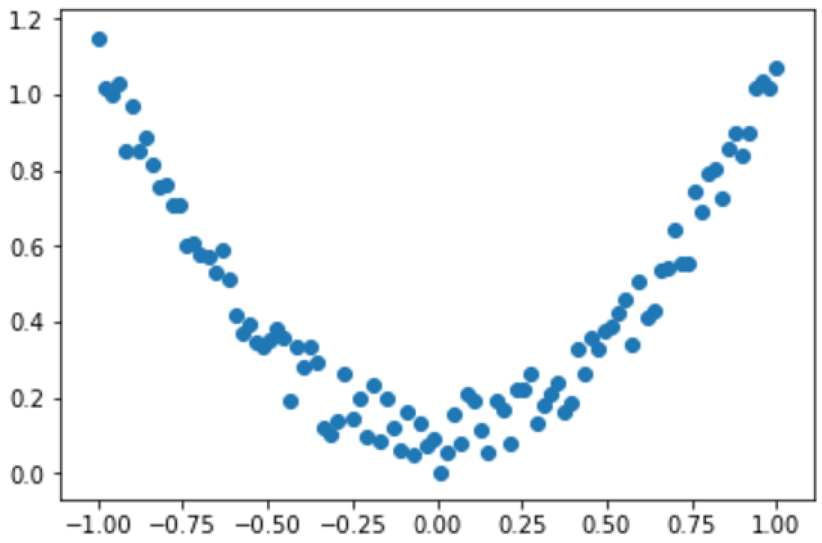
$$Y_{\text{pred}} = h_{\text{sigmoid}} W_2 + b_2$$

$$f = ||Y - Y_{\text{pred}}||_F^2$$

搭建深度神经网络步骤

- 准备训练数据
- 设计网络架构，构建损失函数
- 批量输入数据，利用反向传播算法训练参数
  - 正向计算损失函数
  - 计算网络参数梯度
  - 利用梯度下降算法更新网络参数

小批量随机梯度下降



$$h = XW_1 + b_1$$
$$h_{\text{sigmoid}} = sigmoid(h)$$
$$Y_{\text{pred}} = h_{\text{sigmoid}} W_2 + b_2$$
$$f = ||Y - Y_{\text{pred}}||_F^2$$

```
torch.manual_seed(1)    # reproducible

x = torch.unsqueeze(torch.linspace(-1, 1, 10000000), dim=1)
y = x.pow(2) + 0.2*torch.rand(x.size())
```

Batch 概念

小批量随机梯度下降

$$f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x})$$

$$\nabla f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{x})$$

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \nabla f_i(\boldsymbol{x})$$

$$\nabla f_{\mathcal{B}}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|}\sum_{i \in \mathcal{B}} \nabla f_i(\boldsymbol{x})$$

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \nabla f_{\mathcal{B}}(\boldsymbol{x})$$

W5_Regression_Batch.ipynb

模型保存

北京大学 PEKING UNIVERSITY

1. 保存整个网络结构和参数

```
torch.save(net, 'net_all.pkl')  # save entire net
```

2. 只保存网络参数

```
torch.save(net.state_dict(), 'net_params.pkl')   # save only the parameters
```

W5_Save_Model.ipynb

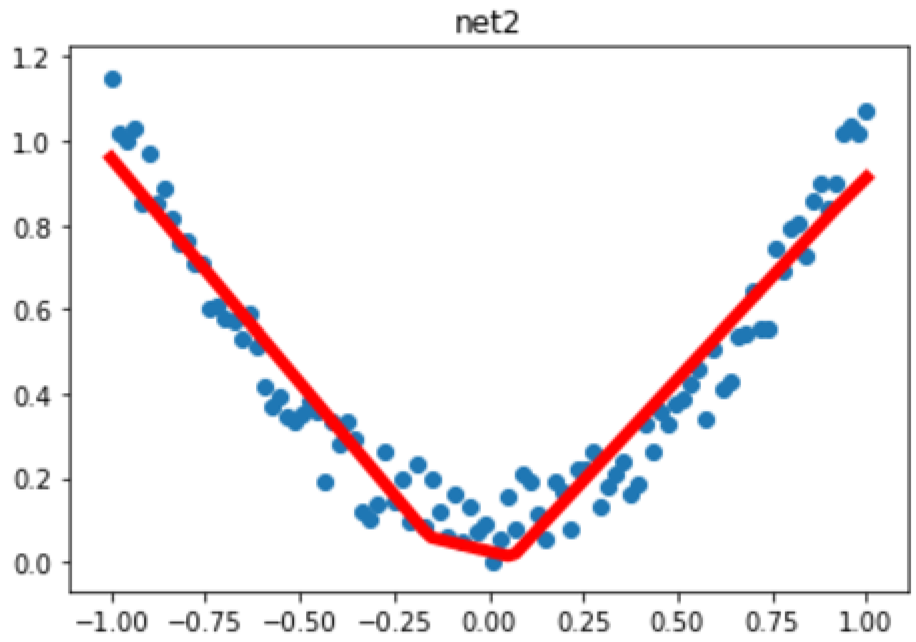
模型保存

1. 保存整个网络结构和参数

torch.save(net, 'net_all.pkl') # save entire net

```python
# restore entire net
# Restore_Network()
net2 = torch.load('net_all.pkl')
prediction = net2(x)
print(net2)
# plot result
plt.title('net2')
plt.scatter(x.data.numpy(), y.data.numpy())
plt.plot(x.data.numpy(), prediction.data.numpy(), 'r-', lw=5)
```

```
Net(
  (hidden): Linear(in_features=1, out_features=10, bias=True)
  (predict): Linear(in_features=10, out_features=1, bias=True)
)
```



W5_Save_Model.ipynb

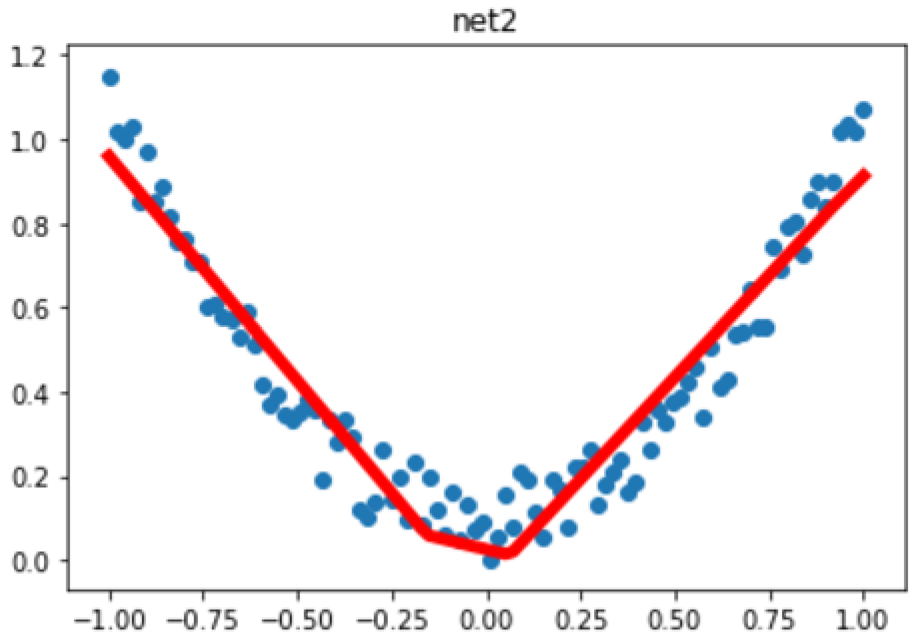
模型保存

2. 只保存网络参数

torch.save(net.state_dict(), 'net_params.pkl')   # save only the parameters

```
# restore only the net parameters
# Restore_Net_Para()
net3 = Net(n_feature=1, n_hidden=10, n_output=1) # define the network
# copy net's parameters into net3
net3.load_state_dict(torch.load('net_params.pkl'))
prediction = net3(x)
print(net3)
# plot result
plt.title('net3')
plt.scatter(x.data.numpy(), y.data.numpy())
plt.plot(x.data.numpy(), prediction.data.numpy(), 'r-', lw=5)
```

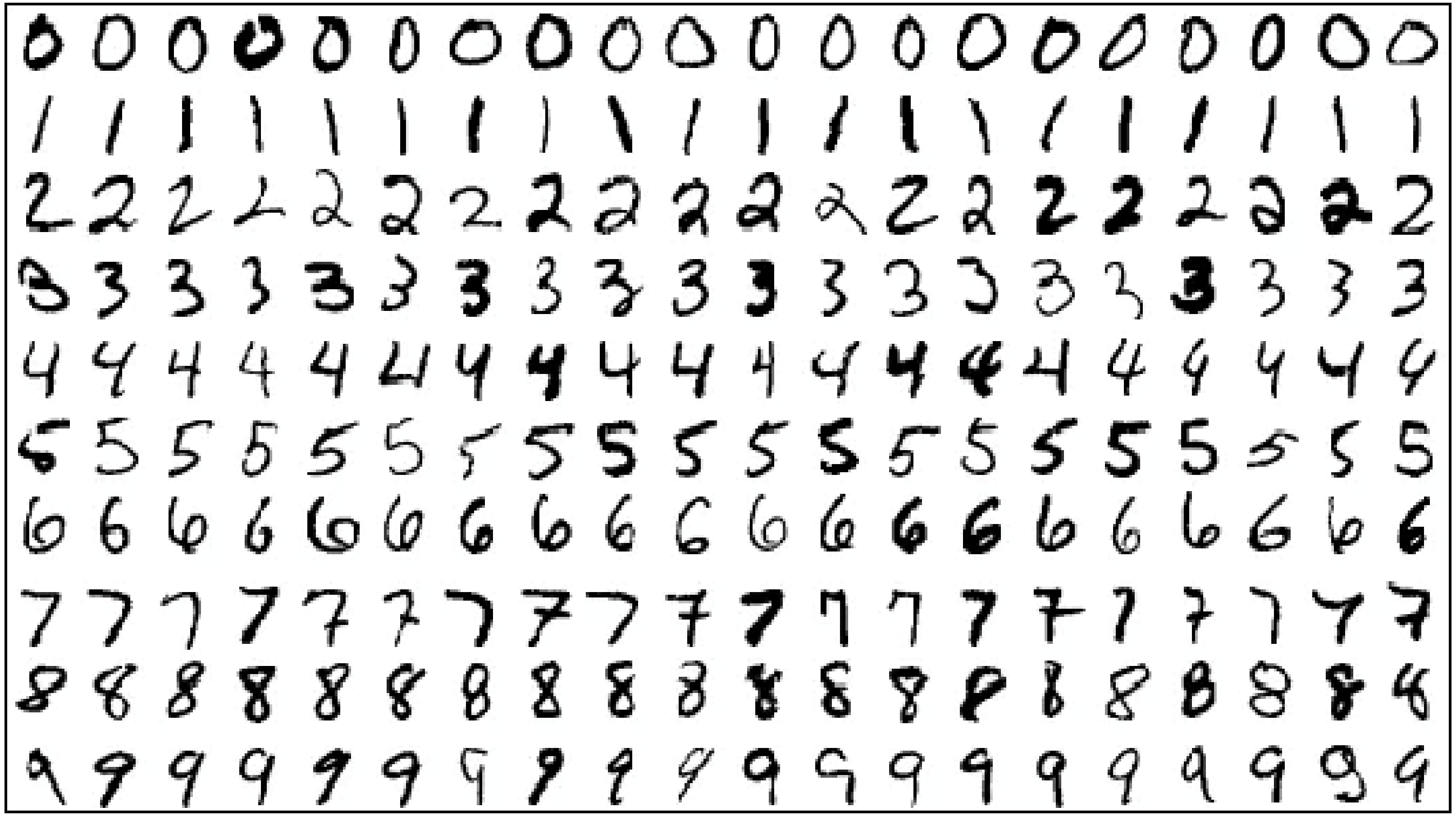
```
Net(
  (hidden): Linear(in_features=1, out_features=10, bias=True)
  (predict): Linear(in_features=10, out_features=1, bias=True)
)
```



W6_Save_Model.ipynb

全连接网络分类-MNIST

- 居中和缩放
- 50,000 个训练数据
- 10,000 个测试数据
- 28 x 28 大小图片
- 10 类



https://tensorspace.org/html/playground/lenet.html

全连接网络分类-MNIST

```
import torch
import torch.nn as nn
import torch.utils.data as Data
import torchvision
import torch.nn.functional as F
import numpy as np

# torch.manual_seed(1)

EPOCH = 1
LR = 0.001
DOWNLOAD_MNIST = True

train_data = torchvision.datasets.MNIST(root='./mnist/', train=True,
transform=torchvision.transforms.ToTensor(), download=DOWNLOAD_MNIST,)
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)

train_x = torch.unsqueeze(train_data.train_data, dim=1).type(torch.FloatTensor)/255.
train_y = train_data.train_labels
```


W6_MNIST_FC.ipynb

全连接网络分类-MNIST

```
class FC(nn.Module):
    def __init__(self):
        super(FC, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 10)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)

        output = x
        return output

fc = FC()
optimizer = torch.optim.Adam(fc.parameters(), lr=LR)
loss_func = nn.CrossEntropyLoss()
data_size = 20000
batch_size = 50
```

```
for epoch in range(EPOCH):
    random_indx = np.random.permutation(data_size)
    for batch_i in range(data_size//batch_size):
        indx = random_indx[batch_i*batch_size:(batch_i+1)*batch_size]

        b_x = train_x[indx,:]
        b_y = train_y[indx]

        output = fc(b_x)
        loss = loss_func(output, b_y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

全连接网络的不足

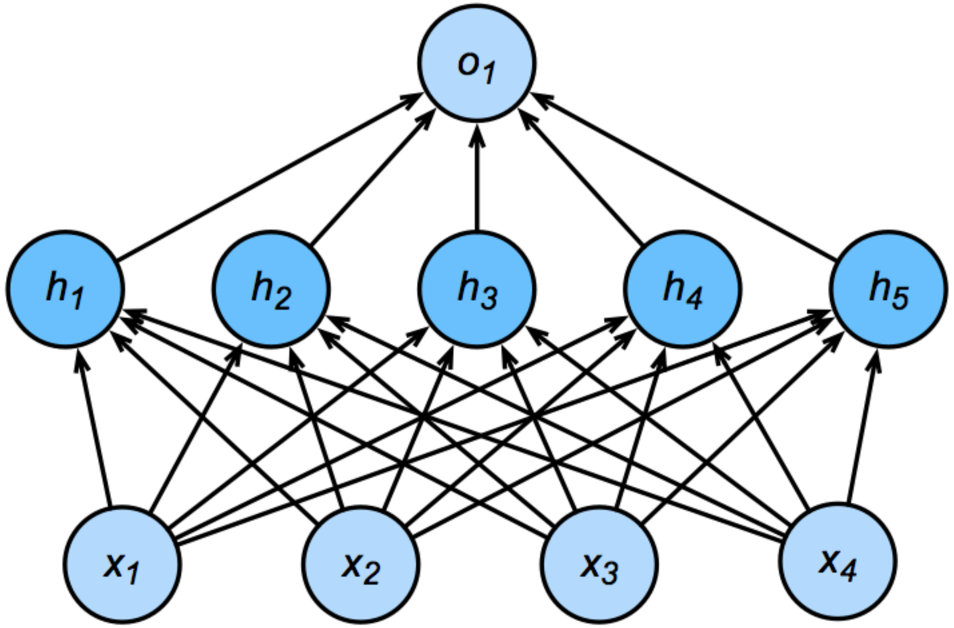
Dual
12MP
wide-angle and
telephoto cameras

Output layer

100 神经元 — Hidden layer

3.6B 参数 = 14GB

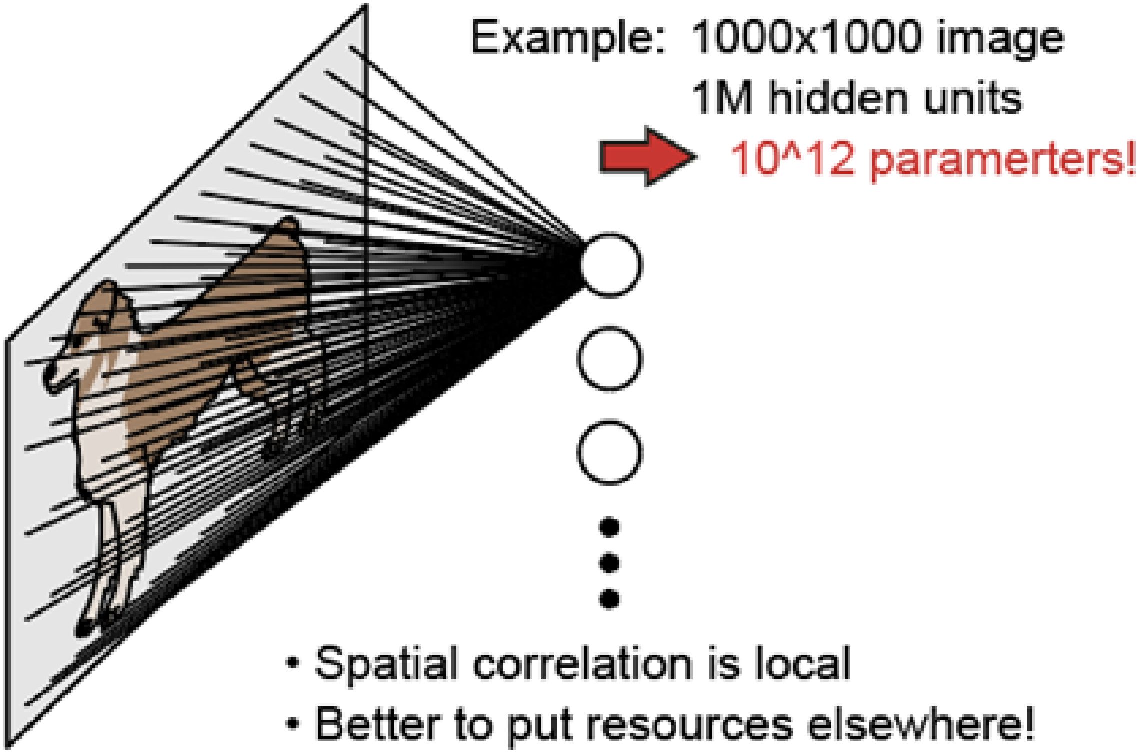
36M 特征 — Input layer



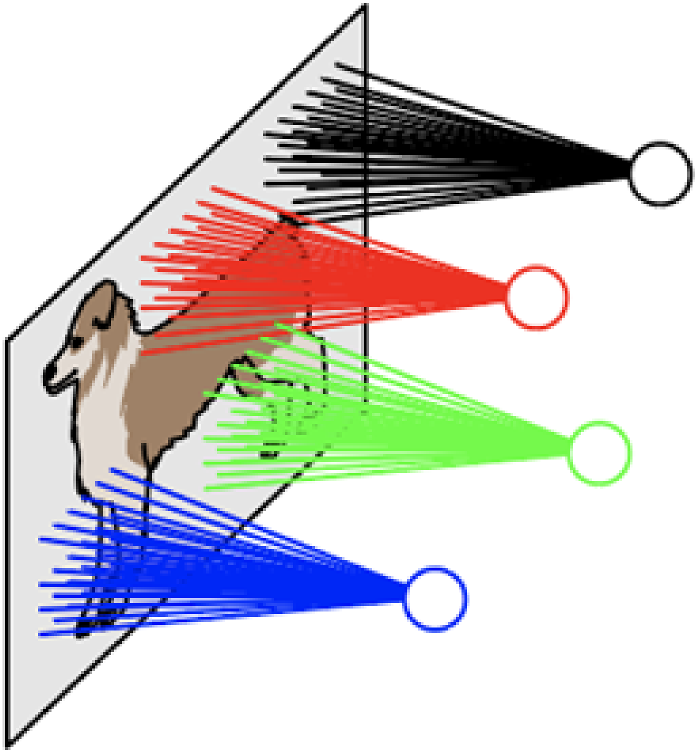
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

权重共享

Fully connected neural net

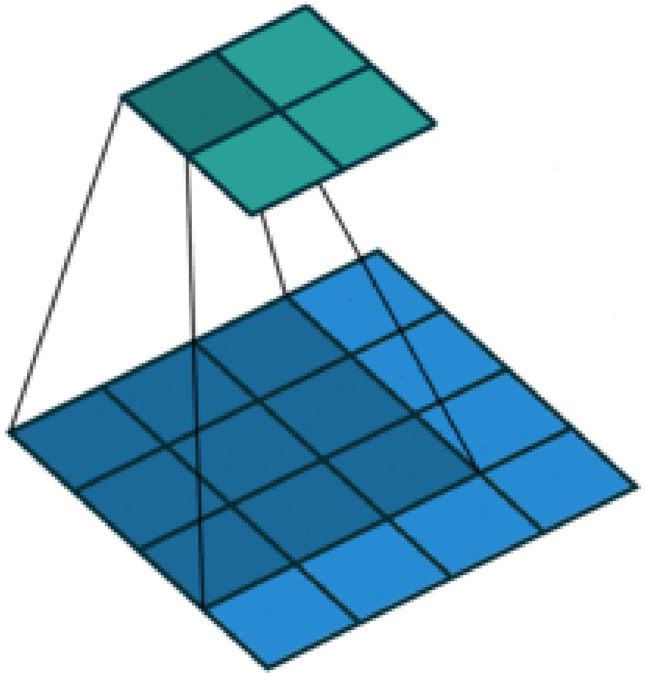


Locally connected neural net



二维卷积层

$$\underset{\text{Input}}{\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array}} * \underset{\text{Kernel}}{\begin{array}{|c|c|}\hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array}} = \underset{\text{Output}}{\begin{array}{|c|c|}\hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}}$$



$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

二维卷积层

$$\begin{bmatrix}0&1&2\\3&4&5\\6&7&8\end{bmatrix} * \begin{bmatrix}0&1\\2&3\end{bmatrix} = \begin{bmatrix}19&25\\37&43\end{bmatrix}$$

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

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

- W 和 b 是可学习的参数

二维卷积层

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

边缘检测

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

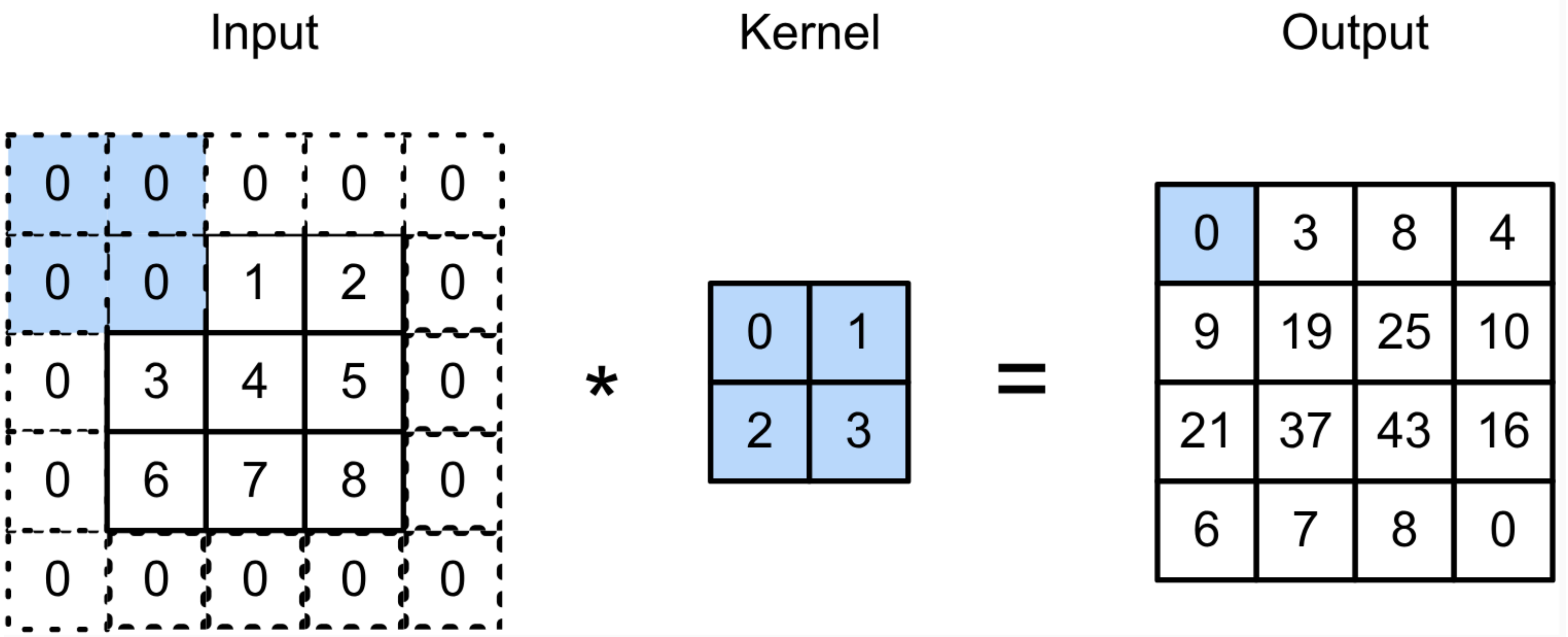
锐化

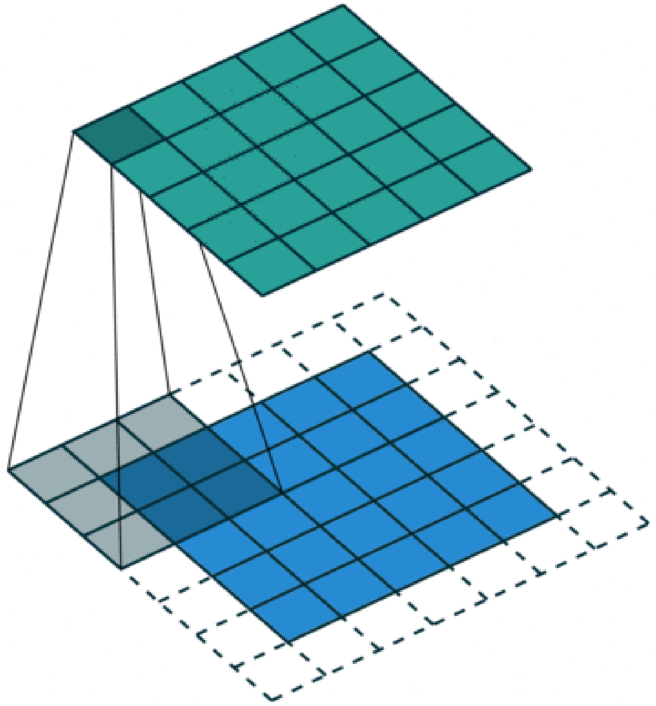
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

高斯模糊

填充(Padding)

填充： 在输入周围添加行 / 列

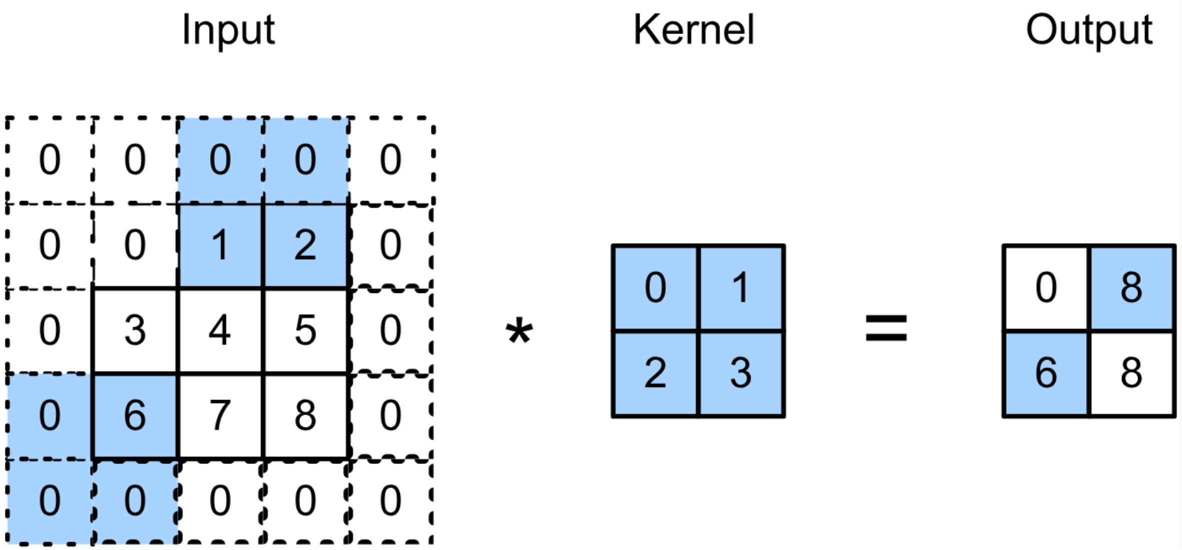


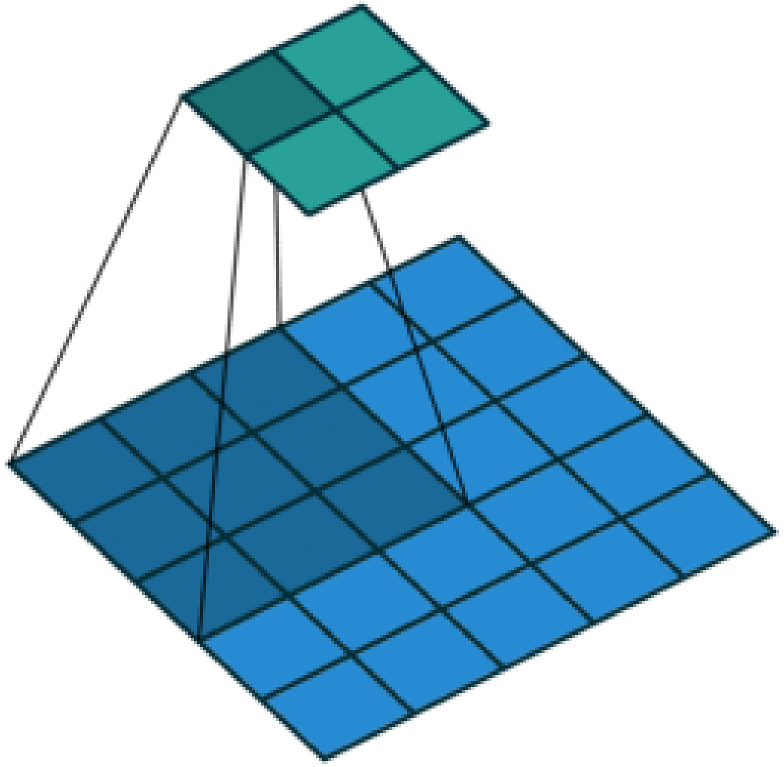


$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

步幅（Stride）

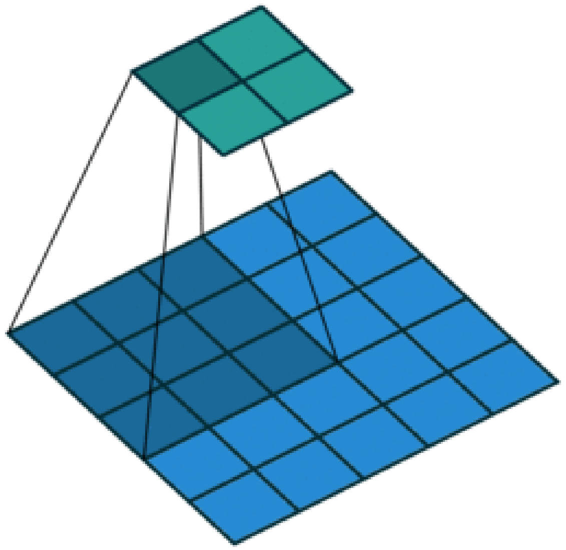
- 每步幅是 “行数量 / 列数量”



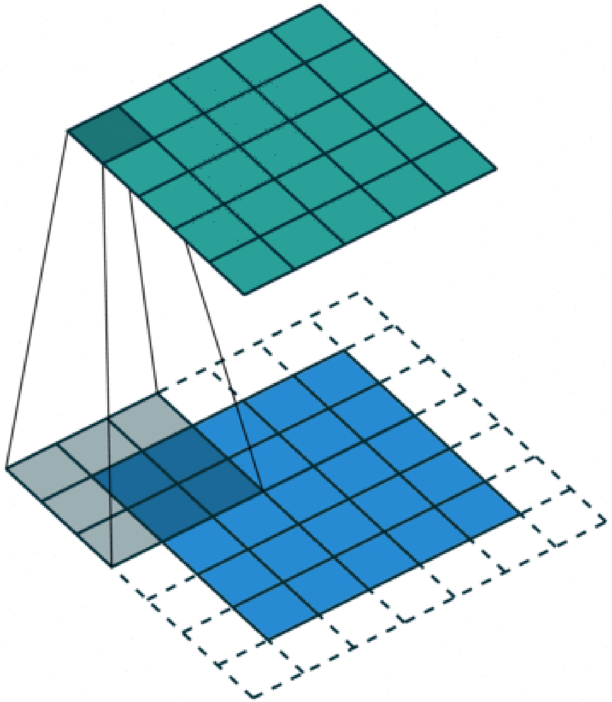


$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

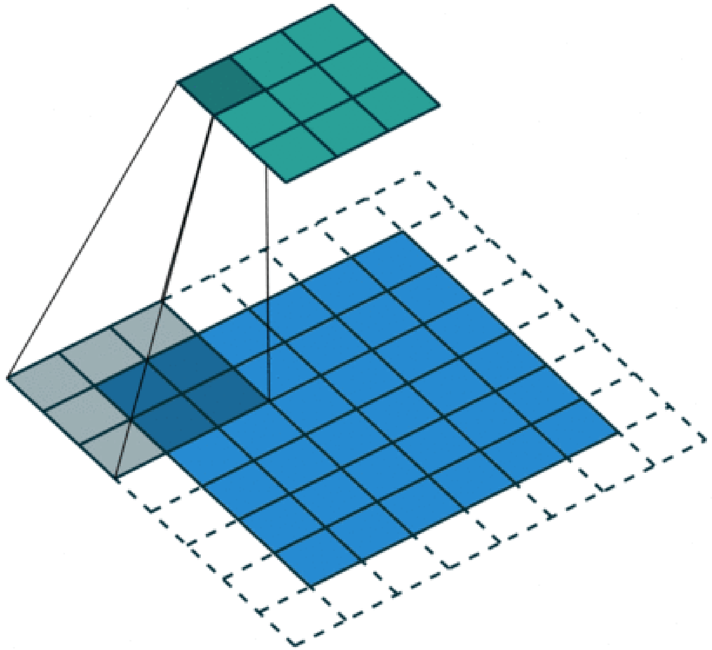
二维卷积层



Padding=0, stride=2

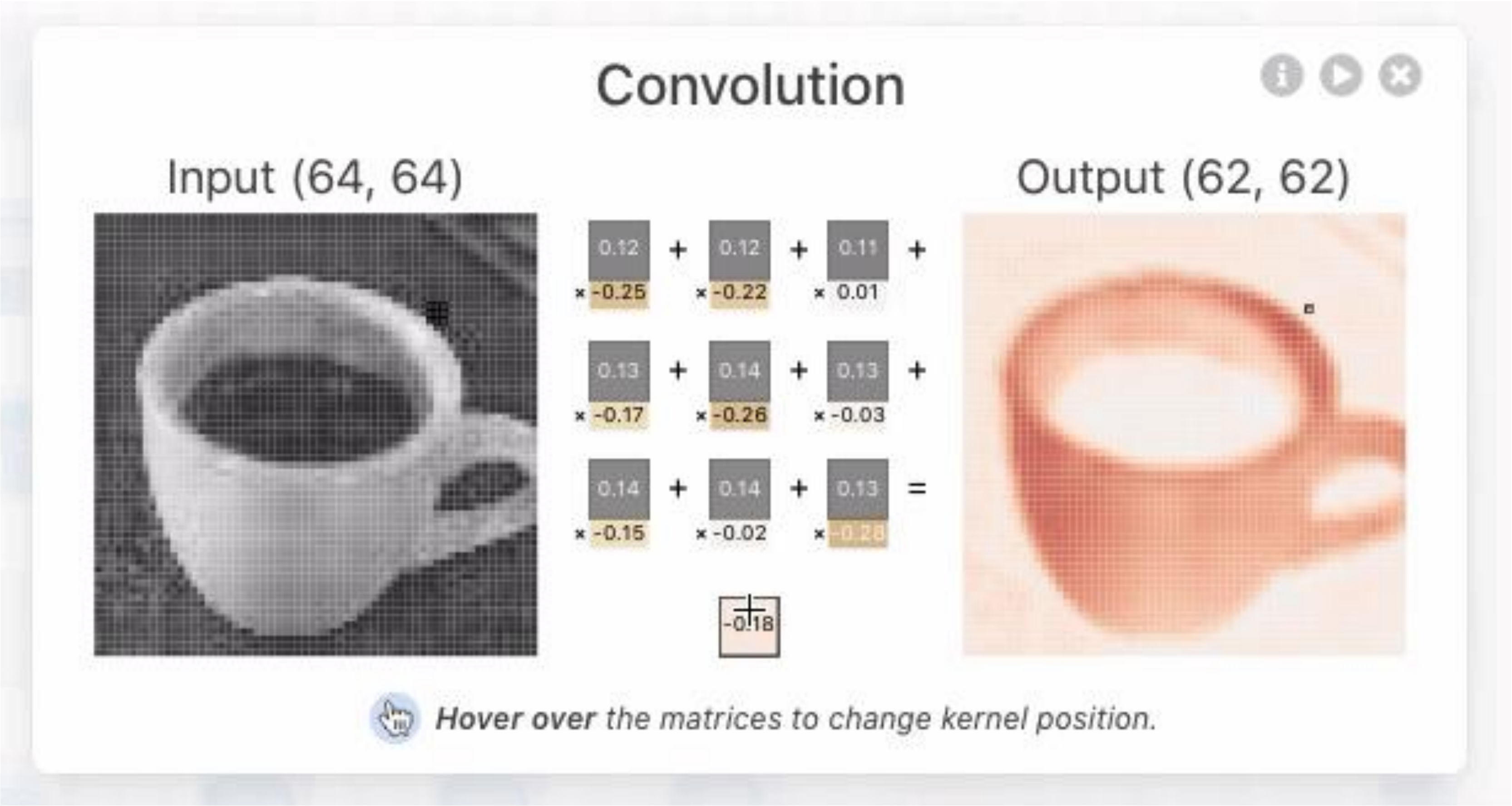


Padding=1, stride=1

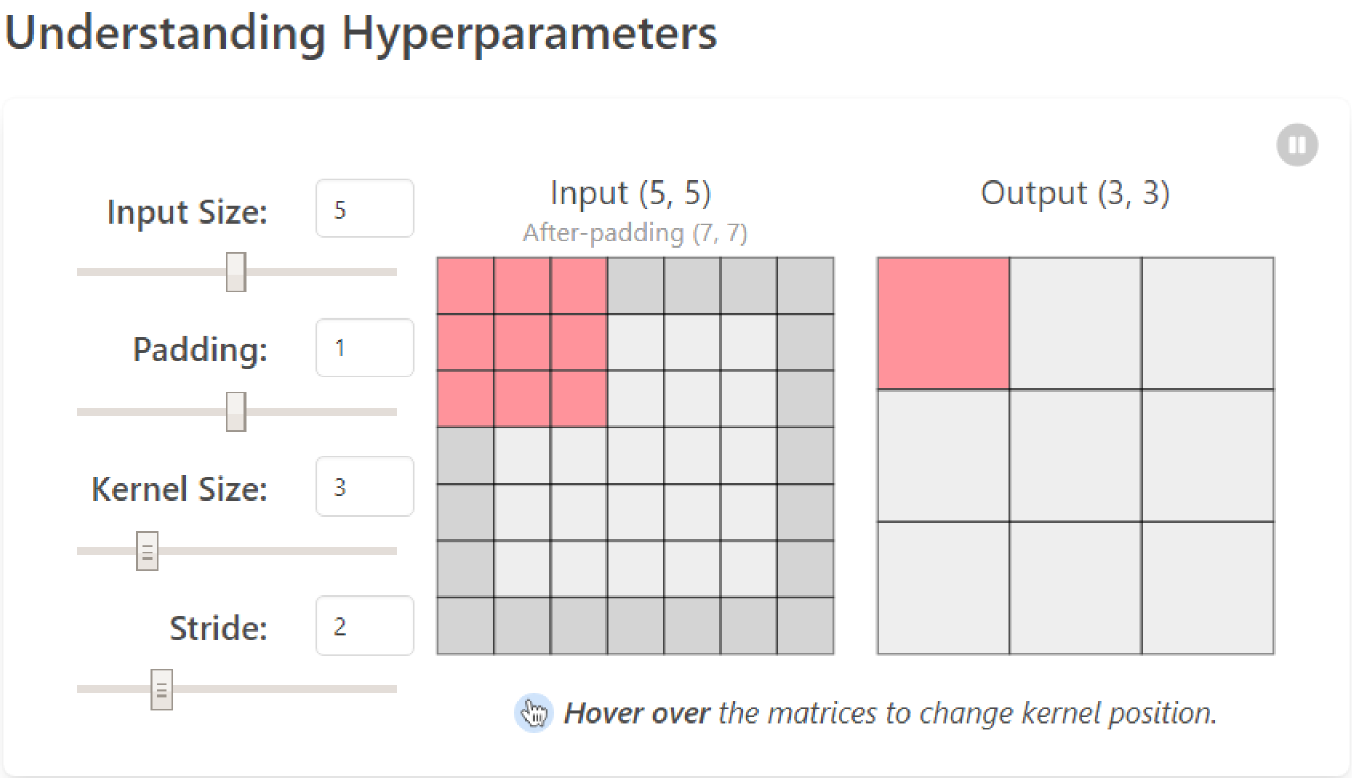


Padding=1, stride=2

二维卷积层


Convolution
Input (64, 64)
Output (62, 62)
0.12 × -0.25 + 0.12 × -0.22 + 0.11 × 0.01 +
0.13 × -0.17 + 0.14 × -0.26 + 0.13 × -0.03 +
0.14 × -0.15 + 0.14 × -0.02 + 0.13 × -0.28 =
-0.18
Hover over the matrices to change kernel position.

二维卷积层



https://poloclub.github.io/cnn-explainer/#article-input

多个输入通道

北京大学 PEKING UNIVERSITY

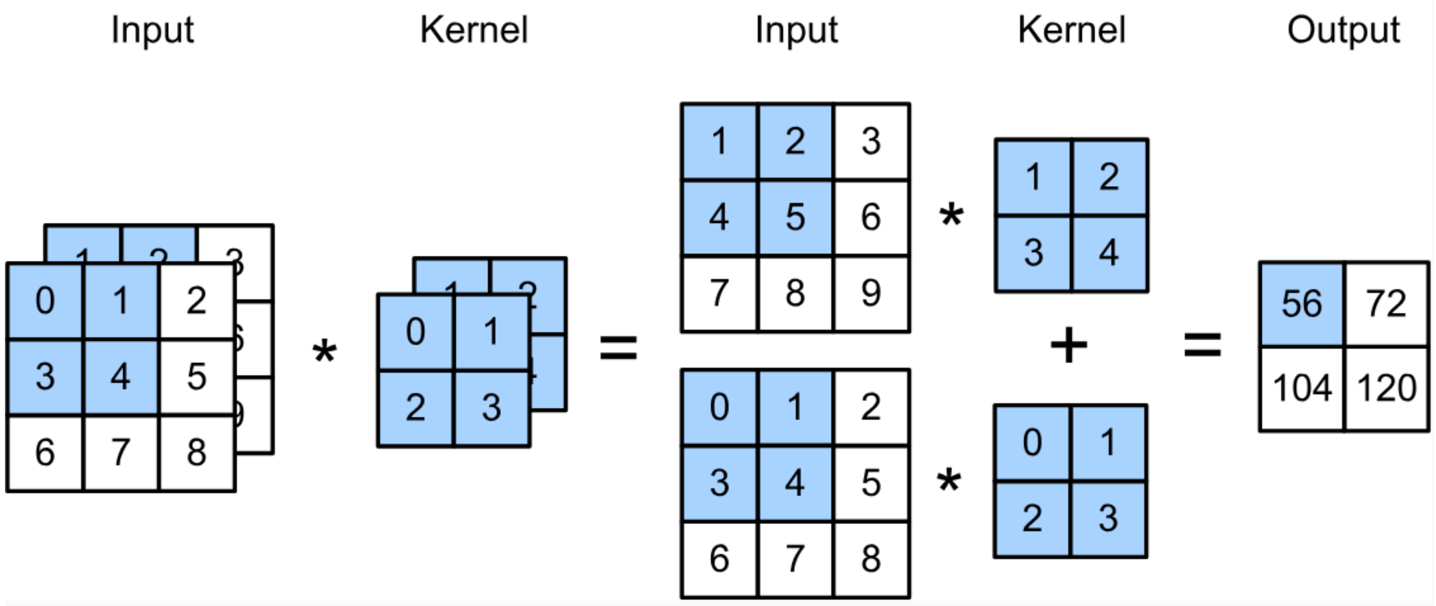
- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息

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

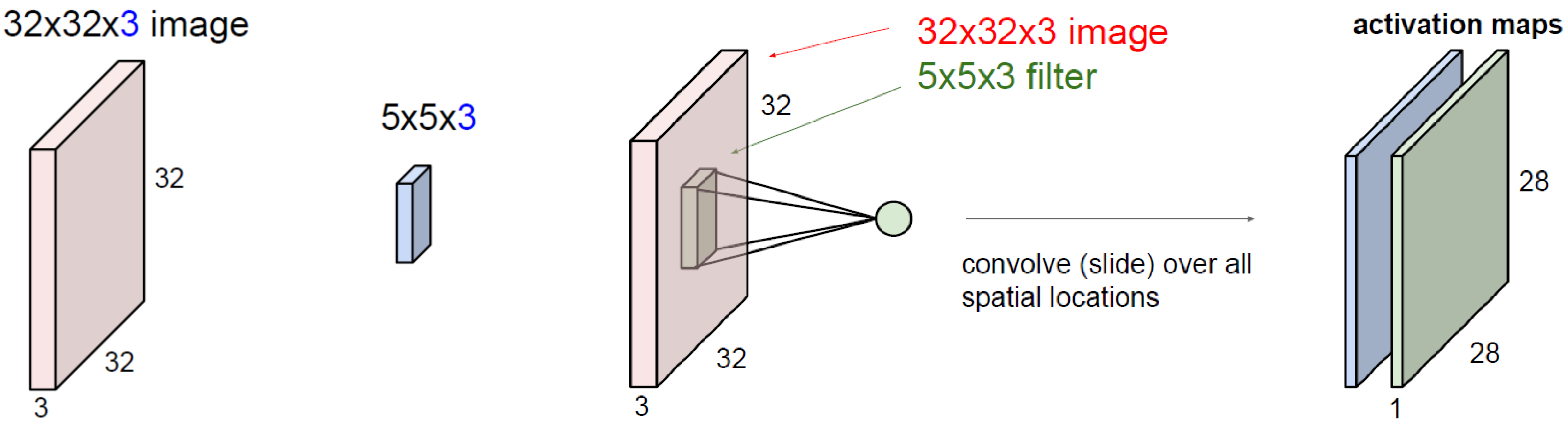
多个输入通道

每个通道都有一个内核，对结果进行求和



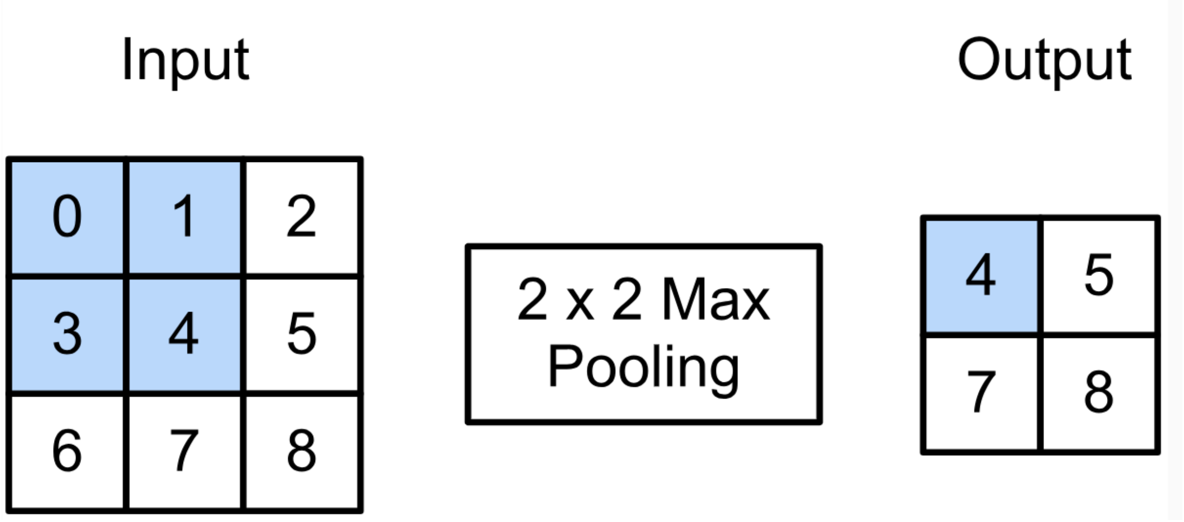
$$
\begin{aligned}
&(1\times1+2\times2+4\times3+5\times4)\\
&+(0\times0+1\times1+3\times2+4\times3)\\
&=56
\end{aligned}
$$

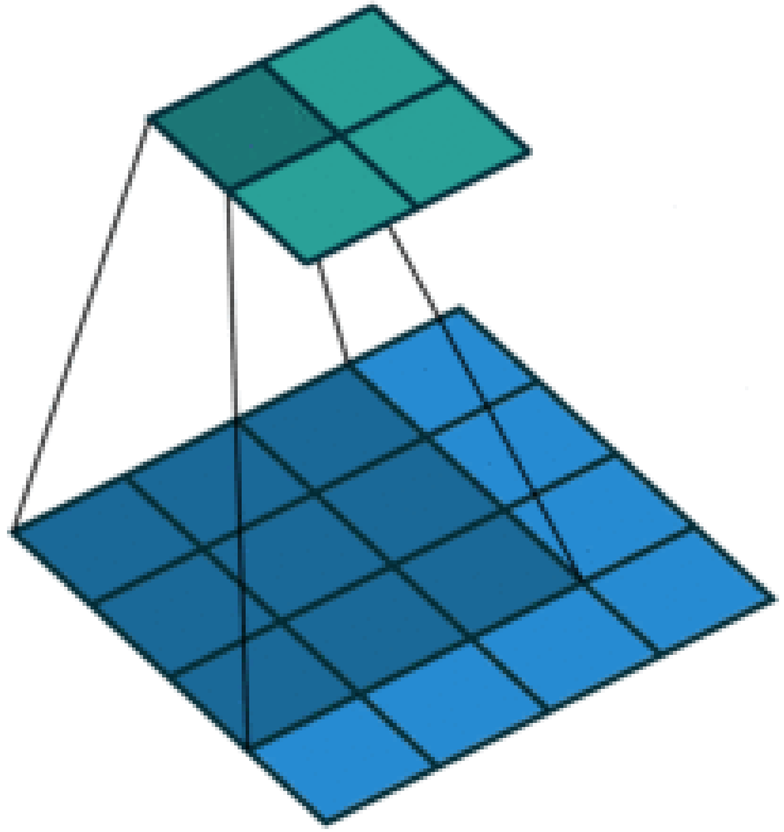
多个输入通道


32x32x3 image
32
32
3
5x5x3
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all spatial locations
activation maps
28
28
1

池化（pooling）

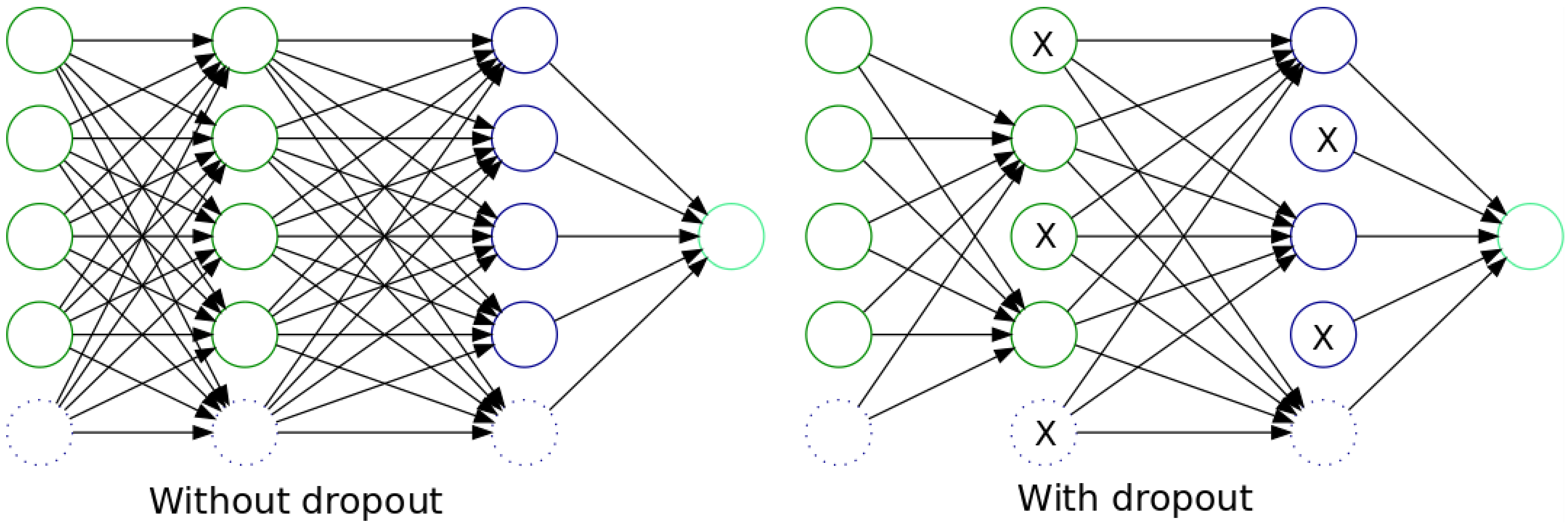
- 返回滑动窗口中的最大值





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

丢弃法 – 训练（Dropout）



丢弃法 – 训练（Dropout）

DROPOUT

CLASS `torch.nn.Dropout(p: float = 0.5, inplace: bool = False)` [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

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`

批归一化（Batch Normalization）

BATCHNORM2D

CLASS `torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)` [SOURCE]

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

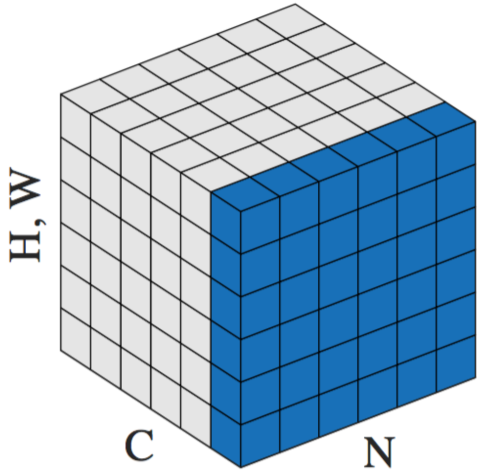
$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

The mean and standard-deviation are calculated per-dimension over the mini-batches and γ and β are learnable parameter vectors of size C (where C is the input size). By default, the elements of γ are set to 1 and the elements of β are set to 0. The standard-deviation is calculated via the biased estimator, equivalent to *torch.var(input, unbiased=False)*.

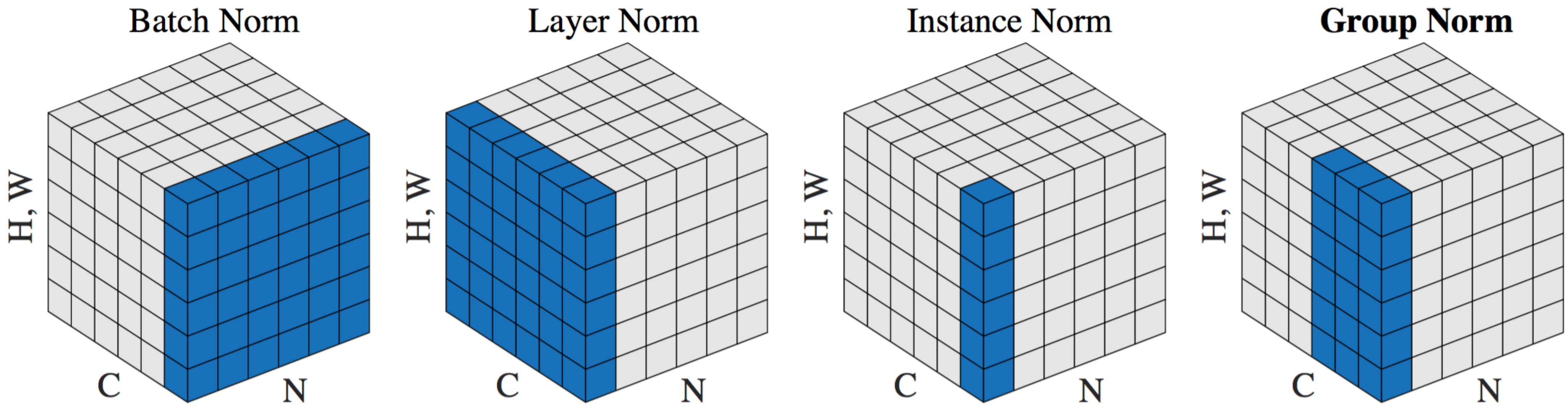
(N, C, H, W)

(batchsize, channel, height, weight)

Batch Norm



其他归一化



二维卷积

class `torch.nn.Conv2d`*(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)* [source]

Applies a 2D convolution over an input signal composed of several input planes.

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

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

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.

二维卷积

卷积层的输入必须为四维张量

例子1：PyTorch卷积层实现

```python
# Convolution Example 1:
import torch.nn as nn
input = torch.randn(1,1,28,28) # (BatchSize, NumChannels, Height, Width)
conv1 = nn.Conv2d(in_channels=1, out_channels=5, kernel_size=3, padding=1, stride=1, bias=True)
output_conv1 = conv1(input)

print("Size of Input is", input.shape)
print("Size of Conv1 Output is",output_conv1.shape)

params = list(conv1.parameters())
print("The Number of Conv1 is %d " % len(params))

for name, parameters in conv1.named_parameters():
    print(name, ':', parameters.size())
```

```
Size of Input is torch.Size([1, 1, 28, 28])
Size of Conv1 Output is torch.Size([1, 5, 28, 28])
The Number of Conv1 is 2
weight : torch.Size([5, 1, 3, 3])
bias : torch.Size([5])
```

例子2：PyTorch卷积层(padding=0)

```
# Convolution Example 2(padding=0):

import torch.nn as nn
input = torch.randn(1,1,28,28) # (BatchSize, NumChannels, Height, Width)
conv1 = nn.Conv2d(in_channels=1, out_channels=5, kernel_size=3, padding=0, stride=1, bias=True)
output_conv1 = conv1(input)

print("Size of Input is", input.shape)
print("Size of Conv1 Output is",output_conv1.shape)

params = list(conv1.parameters())
print("The Number of Conv1 is %d " % len(params))

for name, parameters in conv1.named_parameters():
    print(name, ':', parameters.size())
```

```
Size of Input is torch.Size([1, 1, 28, 28])
Size of Conv1 Output is torch.Size([1, 5, 26, 26])
The Number of Conv1 is 2
weight : torch.Size([5, 1, 3, 3])
bias : torch.Size([5])
```

例子3：PyTorch卷积层(stride=2)

```
# Convolution Example 3(stride=2):

import torch.nn as nn
input = torch.randn(1,1,28,28) # (BatchSize, NumChannels, Height, Width)
conv1 = nn.Conv2d(in_channels=1, out_channels=5, kernel_size=3, padding=1, stride=2, bias=True)
output_conv1 = conv1(input)

print("Size of Input is", input.shape)
print("Size of Conv1 Output is",output_conv1.shape)

params = list(conv1.parameters())
print("The Number of Conv1 is %d " % len(params))

for name, parameters in conv1.named_parameters():
    print(name, ':', parameters.size())
```

```
Size of Input is torch.Size([1, 1, 28, 28])
Size of Conv1 Output is torch.Size([1, 5, 14, 14])
The Number of Conv1 is 2
weight : torch.Size([5, 1, 3, 3])
bias : torch.Size([5])
```

PEKING UNIVERSITY 北京大学

例子4：PyTorch卷积层(bias=False):

```
# Convolution Example 4(bias=False):

input = torch.randn(1,1,28,28) # (BatchSize, NumChannels, Height, Width)
conv1 = nn.Conv2d(in_channels=1, out_channels=5, kernel_size=3, padding=1, stride=2, bias=False)
output_conv1 = conv1(input)

print("Size of Input is", input.shape)
print("Size of Conv1 Output is",output_conv1.shape)

params = list(conv1.parameters())
print("The Number of Conv1 is %d " % len(params))

for name, parameters in conv1.named_parameters():
    print(name, ':', parameters.size())
```

```
Size of Input is torch.Size([1, 1, 28, 28])
Size of Conv1 Output is torch.Size([1, 5, 14, 14])
The Number of Conv1 is 1
weight : torch.Size([5, 1, 3, 3])
```

例子5：PyTorch卷积层(in_channels=3):

```
# Convolution Example 5(in_channels=3):

input = torch.randn(1,3,28,28) # (BatchSize, NumChannels, Height, Width)
conv1 = nn.Conv2d(in_channels=3, out_channels=5, kernel_size=3, padding=1, stride=1, bias=True)
output_conv1 = conv1(input)

print("Size of Input is", input.shape)
print("Size of Conv1 Output is",output_conv1.shape)

params = list(conv1.parameters())
print("The Number of Conv1 is %d " % len(params))

for name, parameters in conv1.named_parameters():
    print(name, ':', parameters.size())
```

```
Size of Input is torch.Size([1, 3, 28, 28])
Size of Conv1 Output is torch.Size([1, 5, 28, 28])
The Number of Conv1 is 2
weight : torch.Size([5, 3, 3, 3])
bias : torch.Size([5])
```

例子6：PyTorch卷积层(BatchSize=10): ¶

```
# Convolution Example 6(BatchSize=10):

input = torch.randn(10,1,28,28) # (BatchSize, NumChannels, Height, Width)
conv1 = nn.Conv2d(in_channels=1, out_channels=5, kernel_size=3, padding=1, stride=1, bias=True)
output_conv1 = conv1(input)

print("Size of Input is", input.shape)
print("Size of Conv1 Output is",output_conv1.shape)

params = list(conv1.parameters())
print("The Number of Conv1 is %d " % len(params))

for name, parameters in conv1.named_parameters():
    print(name, ':', parameters.size())
```

```
Size of Input is torch.Size([10, 1, 28, 28])
Size of Conv1 Output is torch.Size([10, 5, 28, 28])
The Number of Conv1 is 2
weight : torch.Size([5, 1, 3, 3])
bias : torch.Size([5])
```

二维卷积

如何输出网络中间层结果？

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, stride=1, padding=2)
        self.conv2 = nn.Conv2d(16, 32, 5, 1, 2)
        self.out = nn.Linear(32 * 7 * 7, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, (2,2))
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, (2,2))
        x = x.view(x.size(0), -1)

        output = self.out(x)
        return output

cnn = CNN()
```

例子7：输出神经网络的中间结果

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, stride=1, padding=2)
        self.conv2 = nn.Conv2d(16, 32, 5, 1, 2)
        self.out = nn.Linear(32 * 7 * 7, 10)

    def forward(self, x):
        x1 = self.conv1(x)
        x1 = F.relu(x1)
        x2 = F.max_pool2d(x1, (2,2))
        x3 = self.conv2(x2)
        x3 = F.relu(x3)
        x4 = F.max_pool2d(x3, (2,2))
        x5 = x4.view(x4.size(0), -1)

        output = self.out(x5)
        return [output, x1, x2, x3, x4, x5]

cnn = CNN()
```

二维卷积

北京大学 PEKING UNIVERSITY

例子7：输出神经网络的中间结果

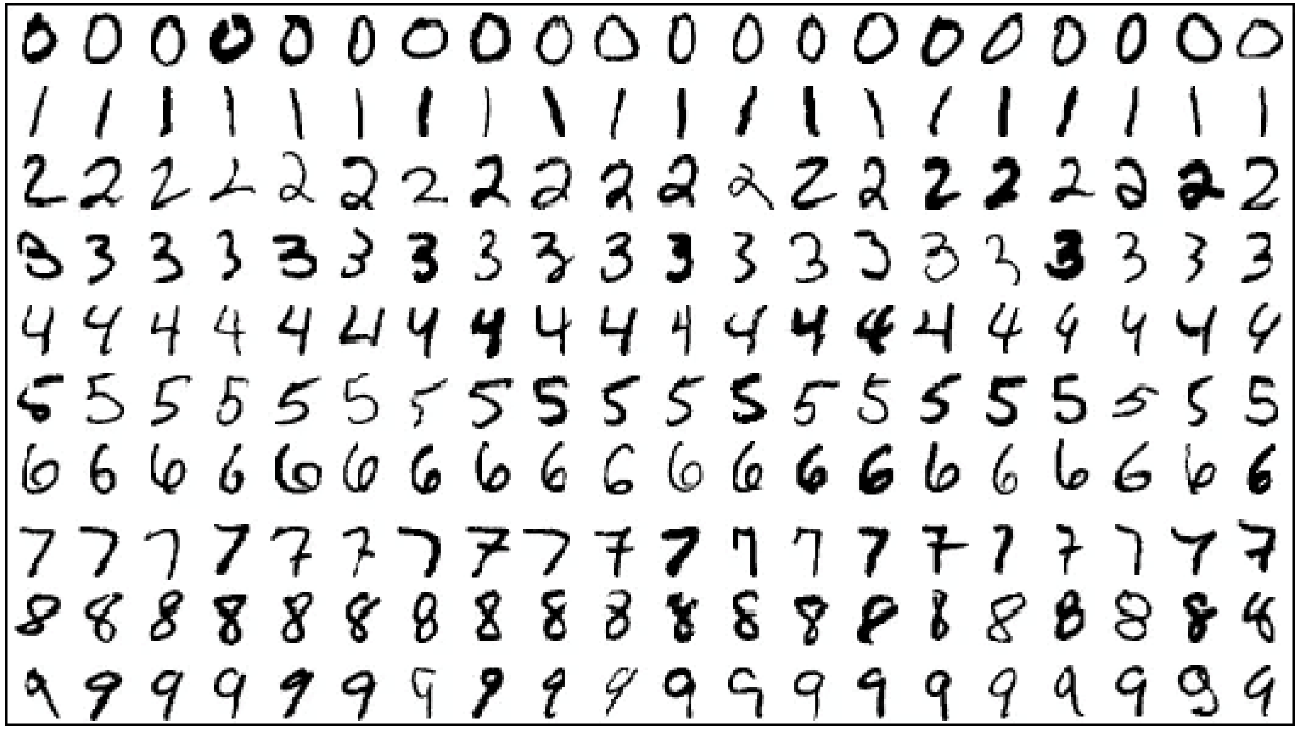
```
input = torch.randn(1,1,28,28)
out, xx1, xx2, xx3, xx4, xx5 = cnn(input)
print(out.shape)
print(xx1.shape)
print(xx2.shape)
print(xx3.shape)
print(xx4.shape)
print(xx5.shape)
```

```
torch.Size([1, 10])
torch.Size([1, 16, 28, 28])
torch.Size([1, 16, 14, 14])
torch.Size([1, 32, 14, 14])
torch.Size([1, 32, 7, 7])
torch.Size([1, 1568])
```

CNN.ipynb

本次作业

- 在W6_MNIST_FC.ipynb基础上，增加卷积层结构/增加dropout或者BN技术等，训练出尽可能高的MNIST分类效果。



交流&问题？