

Artificial Intelligence

人工智能实验

深度学习

中山大学计算机学院
2024年春季

目录

1. 预备知识

1.1 PyTorch介绍

1.2 CNN

1.3 CNN 网络训练实例

2. 实验任务

2.1 中药图片分类任务

3. 参考资料

1.1 PyTorch 介绍

□ PyTorch 安装

- 官网: <https://pytorch.org>
- CPU 版安装（无 Nvidia 显卡）：直接在官网主页选择配置，然后复制生成的 Command 粘贴到终端运行（注意需提前激活 Python 虚拟环境）。
- 建议：安装1.7.0及以上版本。

PyTorch Build	Stable (2.3.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	ROCm 6.0	CPU
Run this Command:	<pre>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118</pre>			

1.1 PyTorch 介绍

GPU 版安装：先在终端使用 `nvidia-smi` 命令查看当前显卡支持的 CUDA 版本

```
(base) PS C:\Users\yanghl> nvidia-smi
Mon May 23 19:42:28 2022
```

NVIDIA-SMI 512.77		Driver Version: 512.77		CUDA Version: 11.6	
GPU	Name	TCC/WDDM	Bus-Id	Disp.A	Volatile Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util Compute M. MIG M.
0	NVIDIA GeForce ...	WDDM	00000000:01:00.0	On	N/A
40%	35C	P8	12W / 160W	982MiB / 6144MiB	11% Default N/A

然后在主页选择 CUDA xx.x 生成 Command 命令，PyTorch CUDA 的版本不能高于显卡支持的 CUDA 版本。

Linux 和 MacOS 同理，在主页选择对应 OS 选项即可。

1.1 PyTorch 介绍

安装完成后验证是否安装成功：在终端键入 `python`，进入 `python` 交互环境。

- CPU 版直接执行 `import torch`，如无报错即安装成功。
- GPU 版执行 `import torch` 后，执行 `torch.cuda.is_available()`，如返回 `True` 说明 GPU 版 PyTorch 安装成功。

```
(pytorch) PS C:\Users\yanghl> python
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.cuda.is_available()
True
```

PyTorch 安装到此结束，更多内容可参考官方文档：

<https://pytorch.org/docs/stable/index.html>

1.1 PyTorch 介绍——Tensor

- tensor 是 PyTorch 的基本数据类型，在使用 torch 框架进行操作时，对象一般都要求是 tensor 类型。
- 要初始化一个 tensor，通常有以下三种方式：

1. 直接初始化

```
beta = torch.tensor([[1., 2., 3.], [4., 5., 6.]], dtype=torch.float32)
print(beta)

[2] ✓ 0.6s

... tensor([[1., 2., 3.],
           [4., 5., 6.]])
```

2. 通过原始数据转化

```
alpha = [[1., 2., 3.], [4., 5., 6.]]
beta = torch.tensor(alpha)
print(beta)

[3] ✓ 0.3s

... tensor([[1., 2., 3.],
           [4., 5., 6.]])
```

3. 通过 numpy 数据转化

```
alpha = np.array([[1., 2., 3.], [4., 5., 6.]])
beta = torch.tensor(alpha)
print(beta)

[5] ✓ 0.5s

... tensor([[1., 2., 3.],
           [4., 5., 6.]], dtype=torch.float64)
```

1.1 PyTorch 介绍——Tensor

- 当初始化时未指定数据类型时，`torch.tensor()` 将会根据数据本身的类型自行判断，如：

```
beta = torch.tensor([[1., 2., 3.], [4., 5., 6.]])
print(beta.dtype)

beta = torch.tensor([[1, 2, 3], [4, 5, 6]])
print(beta.dtype)
```

[6] ✓ 0.3s

... torch.float32
torch.int64

- 改类型：`t.type(torch.float32)`

- 也可以通过 `torch.ones()`, `torch.zeros()` 等创建指定大小的全 0 或者全 1 张量：

```
torch.ones((2, 3))
```

[8] ✓ 0.5s

... tensor([[1., 1., 1.],
[1., 1., 1.]])

```
torch.zeros((2, 3))
```

[9] ✓ 0.5s

... tensor([[0., 0., 0.],
[0., 0., 0.]])

1.1 PyTorch 介绍——Tensor

Tensor的维度

1. 标量

2. 向量

3. 矩阵

```
t0 = torch.tensor(6)
print(t0,t0.shape,t0.ndim)
✓ 0.0s
tensor(6) torch.Size([]) 0

t1 = torch.tensor([6])
print(t1,t1.shape,t1.ndim)
✓ 0.0s
tensor([6]) torch.Size([1]) 1

t22 = torch.tensor([[6,6],[6,6]])
print(t22,t22.shape,t22.ndim)
✓ 0.0s
tensor([[6, 6],
        [6, 6]]) torch.Size([2, 2]) 2
```


1.1 PyTorch 介绍—— torch 常用数据操作

维度变换

- `t.view()`或者`t.reshape()` 维度重置（但总数要一致），
若根据已有维度可推算出剩下的维度，
则可使用 -1 替代

- `t.size(dim)`返回第dim维的大小
- `t.view(...)` 并不改变t本身的shape

你可能需要: `t = t.view(...)`

- `t.squeeze(dim)` 若不指定维度，则
会将 tensor 中为1的dim压缩，若指定
则只会压缩对应的维度（必须为1）

```
temp = torch.rand((4, 4, 6))
print(temp.shape)
print(temp.view(4, 24).shape)
print(temp.view(4, -1).shape)
```

✓ 0.6s

```
torch.Size([4, 4, 6])
torch.Size([4, 24])
torch.Size([4, 24])
```

```
temp = torch.rand((4, 1, 6, 1))
print(temp.shape)
print(temp.squeeze().shape)
print(temp.squeeze(1).shape)
print(temp.squeeze(-1).shape)
```

✓ 0.4s

```
torch.Size([4, 1, 6, 1])
torch.Size([4, 6])
torch.Size([4, 6, 1])
torch.Size([4, 1, 6])
```

1.1 PyTorch 介绍—— torch 常用数据操作

维度变换

- `t.unsqueeze(dim)` 维度扩展

因为神经网络一般默认 batch 输入，所以测试数据时，如果输入为单个数据，需要对数据进行 `unsqueeze` 处理，即将其看成 `batch=1` 的特殊情况

- `torch.cat(List[tensor, tensor], dim)`

向量拼接，需指定维度

```
temp = torch.rand((4, 6))
print(temp.shape)
print(temp.unsqueeze(0).shape)
```

✓ 0.5s

`torch.Size([4, 6])`

`torch.Size([1, 4, 6])`

1 2 5 6

3 4 7 8

```
a = torch.tensor([[1, 2], [3, 4]])
b = torch.tensor([[5, 6], [7, 8]])
c = torch.cat([a, b], dim=0)
c
```

✓ 0.5s

`tensor([[1, 2],
 [3, 4],
 [5, 6],
 [7, 8]])`

```
a = torch.tensor([[1, 2], [3, 4]])
b = torch.tensor([[5, 6], [7, 8]])
c = torch.cat([a, b], dim=-1)
c
```

✓ 0.4s

`tensor([[1, 2, 5, 6],
 [3, 4, 7, 8]])`

1.1 PyTorch 介绍——gradient

requires_grad

- 输入神经网络的数据需保证 `tensor(XXX, requires_grad=True)`

e.g. $z = x^2 + 2x + 1$, 求导为 $2x + 2$

- `requires_grad` 为 `True` 时会自动记录梯度, $x=5$ 时 $2x+2$ 结果为 12。



```
x = torch.tensor(5.)

y1 = x**2 # y1 = x^2
y2 = 2*x # y2 = 2x
z = y1 + y2 + 1 # z = x^2 + 2x + 1

print("x :",x)
print("y1:",y1)
print("y2:",y2)
print("z :",z)

z.backward()
⊗ 0.0s

x : tensor(5.)
y1: tensor(25.)
y2: tensor(10.)
z : tensor(36.)

-----
RuntimeError
Cell In[42], line 12
      9 print("y2:",y2)
     10 print("z :",z)
--> 12 z.backward()
```

```
x = torch.tensor(5.,requires_grad=True)

y1 = x**2 # y1 = x^2
y2 = 2*x # y2 = 2x
z = y1 + y2 + 1 # z = x^2 + 2x + 1

print("x :",x)
print("y1:",y1)
print("y2:",y2)
print("z :",z)

z.backward()
print(x.grad)
✓ 0.0s

x : tensor(5., requires_grad=True)
y1: tensor(25., grad_fn=<PowBackward0>)
y2: tensor(10., grad_fn=<MulBackward0>)
z : tensor(36., grad_fn=<AddBackward0>)
tensor(12.)
```



若 `requires_grad` 为 `False` (默认值), 梯度反向传播时会直接报错。

1.1 PyTorch 介绍——gradient

手动梯度下降

更新参数时不记录梯度，用
with torch.no_grad():

...

更新后清除梯度
x.grad.zero_()

```
def loss_fn(x,y):  
    return (x - y)**2  
  
x = torch.tensor(5.,requires_grad=True)  
y = 2  
learning_rate = 0.1  
  
for epoch in range(20):  
    loss = loss_fn(x,y)  
    # 反向传播  
    loss.backward()  
    # 更新参数  
    with torch.no_grad():  
        x -= learning_rate*(x.grad)  
    # 监测loss  
    if epoch % 5 == 0:  
        print(f"loss: ",loss,"\tx: ",x.item())  
    # 清除梯度  
    x.grad.zero_()
```

✓ 0.0s

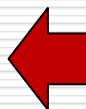
```
loss: tensor(9., grad_fn=<PowBackward0>)    x: 4.400000095367432  
loss: tensor(5.9037, grad_fn=<PowBackward0>) x: -1.0103679895401  
loss: tensor(0.0052, grad_fn=<PowBackward0>) x: 3.406068801879883  
loss: tensor(5.4697, grad_fn=<PowBackward0>) x: 3.232646942138672
```

e.g. $\text{loss_function}(x,y) = (x - y)^2$
 $x = 5$
 $y = 2$

```
def loss_fn(x,y):  
    return (x - y)**2  
  
x = torch.tensor(5.,requires_grad=True)  
y = 2  
learning_rate = 0.1  
  
for epoch in range(20):  
    loss = loss_fn(x,y)  
    # 反向传播  
    loss.backward()  
    # 更新参数  
    with torch.no_grad():  
        x -= learning_rate*(x.grad)  
    # 监测loss  
    if epoch % 5 == 0:  
        print(f"loss: ",loss,"\tx: ",x.item())  
    # 清除梯度  
    x.grad.zero_()
```

✓ 0.0s

```
loss: tensor(9., grad_fn=<PowBackward0>)    x: 4.400000095367432  
loss: tensor(0.9664, grad_fn=<PowBackward0>) x: 2.7864320278167725  
loss: tensor(0.1038, grad_fn=<PowBackward0>) x: 2.2576980590820312  
loss: tensor(0.0111, grad_fn=<PowBackward0>) x: 2.084442615509033
```



不清除梯度会有问题

1.1 PyTorch 介绍——gradient

清除梯度

e.g. $\text{loss_function}(x, y) = (x - y)^2$, 对 x 求导为 $2(x - y)$

```
def loss_fn(x,y):  
    return (x - y)**2  
  
x = torch.tensor(5.,requires_grad=True)  
  
for _ in range(2):  
    loss = loss_fn(x,2)  
    print(f"loss: ",loss)  
    loss.backward()  
    print(x.grad,"\n")  
  
✓ 0.0s  
loss: tensor(9., grad_fn=<PowBackward0>)  
tensor(6.)  
  
loss: tensor(9., grad_fn=<PowBackward0>)  
tensor(12.)
```



当 $x=5$, $y=2$ 时, (在这里我们没有更新 x) x 的梯度应该一直都是6——多次迭代会带来梯度的累计

清除梯度



```
def loss_fn(x,y):  
    return (x - y)**2  
  
x = torch.tensor(5.,requires_grad=True)  
  
for _ in range(2):  
    loss = loss_fn(x,2)  
    print(f"loss: ",loss)  
    loss.backward()  
    print(x.grad,"\n")  
    x.grad.zero_()  
  
✓ 0.0s  
loss: tensor(9., grad_fn=<PowBackward0>)  
tensor(6.)  
  
loss: tensor(9., grad_fn=<PowBackward0>)  
tensor(6.)
```

1.1 PyTorch 介绍—— torch.nn

□ torch.nn

- 自定义神经网络类的基本框架：继承 `nn.Module` 神经网络基本类，该类实例化后输入数据将自动调用 `forward` 前向计算。

```
from torch import nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        ...

    def forward(x):
        ...
        return ...
```

```
net = Net()
out = net(x)
```

1.1 PyTorch 介绍—— torch.nn

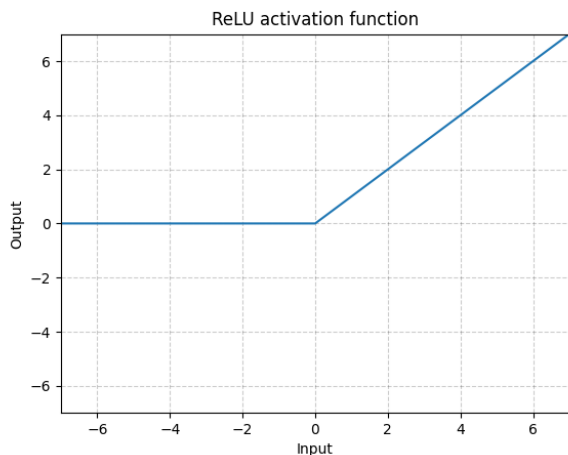
- 全连接层 `nn.Linear(in_dim, out_dim, bias=True)`

```
from torch import nn
m = nn.Linear(20, 30)
input = torch.randn(128, 20)
output = m(input)
print(output.size())
```

✓ 0.7s

`torch.Size([128, 30])`

- 激活函数 `nn.ReLU()`



```
m = nn.ReLU()
input = torch.tensor([-2.4, 1.8])
print(input)
output = m(input)
print(output)
```

✓ 0.3s

`tensor([-2.4000, 1.8000])`
`tensor([0.0000, 1.8000])`

1.1 PyTorch 介绍—— torch.nn

parameters()

net.parameters() 中包含了网络/Module中的参数，作为参数传给优化器，可以在后续实现自动梯度下降

```
class Net(torch.nn.Module):
    def __init__(self) -> None:
        super().__init__()
        # self.fc = torch.nn.Linear(20,30,bias=False)
        self.fc = torch.nn.Linear(2,3,bias=True)

    def forward(self,x):
        x = self.fc(x)
        return x
```

```
net = Net()
for param in net.parameters():
    print(param)
```

✓ 0.0s

```
Parameter containing:
tensor([[ -0.2871, -0.0903],
        [ 0.1767,  0.5264],
        [ 0.2817, -0.0701]], requires_grad=True)
Parameter containing:
tensor([ 0.0533, -0.6758, -0.1570], requires_grad=True)
```

```
conv1 = torch.nn.Conv2d(3,10,3)
for param in conv1.parameters():
    print(param.shape)
```

✓ 0.0s

```
torch.Size([10, 3, 3, 3])
torch.Size([10])
```

Loss

Loss Functions

- nn.L1Loss
- nn.MSELoss
- nn.CrossEntropyLoss
 - 已经包含了Softmax

1.1 PyTorch 介绍—— torch.nn

网络训练一般步骤

实例化网络 `net = Net()` 后，定义[网络优化器](#)，如：

```
optim = torch.optim.SGD(net.parameters(), lr=lr)
```

计算得到 `loss`，在更新前，需清除上一步的梯度，即

```
optim.zero_grad()
```

然后反向传播：

```
loss.backward()
```

最后优化器更新：

```
optim.step()
```

1.2 CNN

卷积神经网络（CNN）

什么是卷积神经网络？

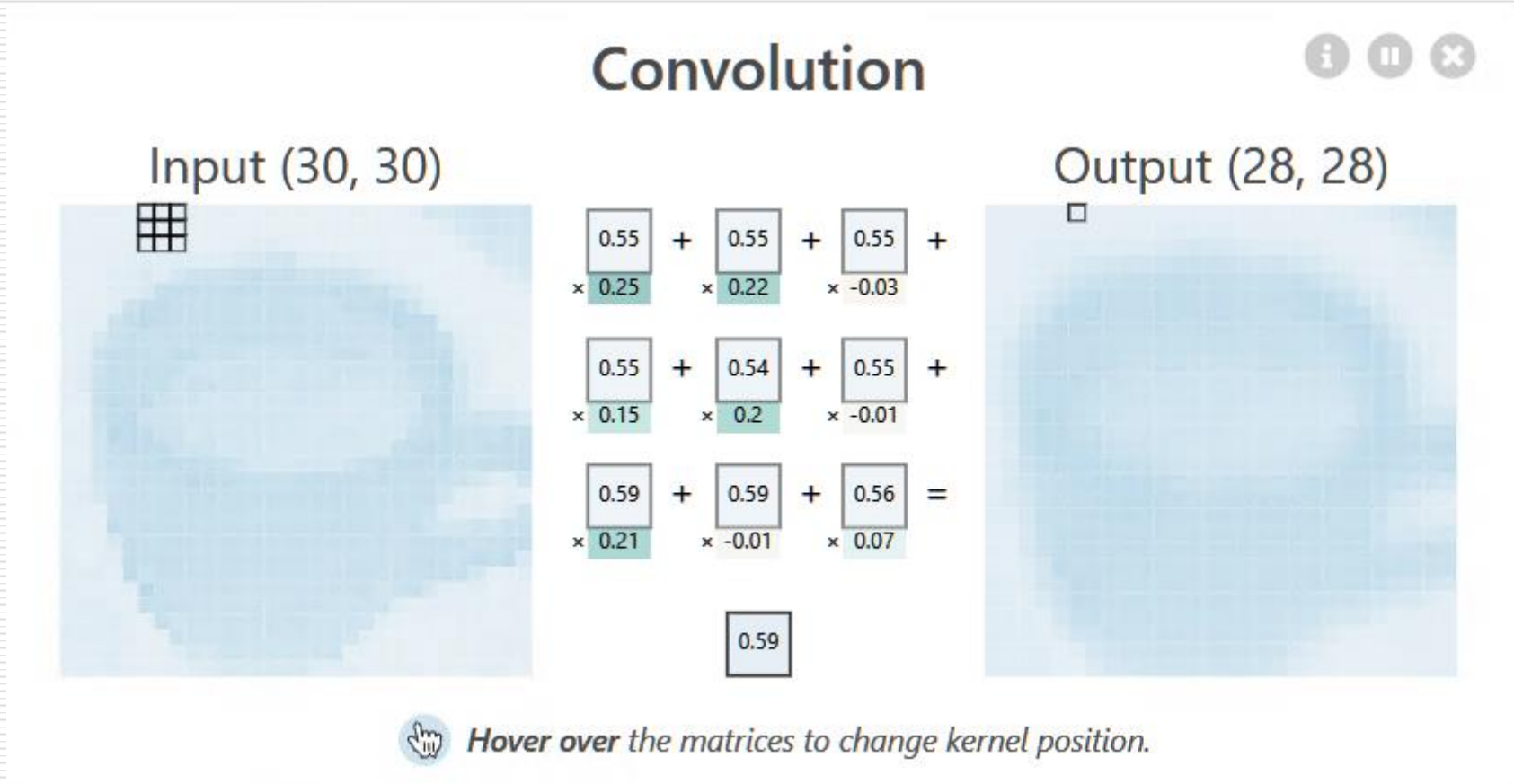
➤ 卷积

- 不再是对图像中每一个像素做处理，而是对图片上每一小块像素区域做处理，加强了图片中像素的连续性，从而处理的一个图形而不是单个像素点

➤ 神经网络

- 神经网络是一种计算模型，由大量的神经元以及层与层之间的激活函数组成。

1.2 CNN



- [CNN Explainer](#)

1.2 CNN——nn.Conv2d

- 卷积神经网络 `nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, bias=True)`
 - NOTE: PyTorch 卷积网络输入默认格式为 (N, C, H, W)**，其中 **N** 为 batch 大小（输入默认batch处理），**C** 为图像通道数（黑白1维，彩色RGB三维），**H**和**W**分别为图像的高度和宽度。
- `Conv2d` 的前两个参数分别为输入和输出的通道数，`kernel_size` 为卷积核大小，`stride`为步长默认为1，`padding` 为填充默认0。
- 一般情况下，计算公式为：

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

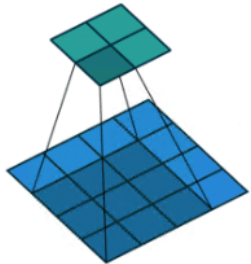
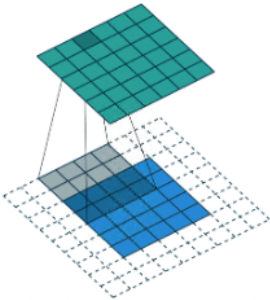
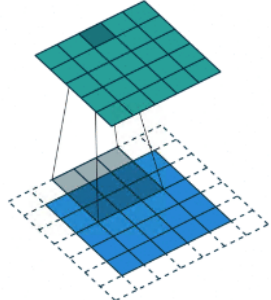
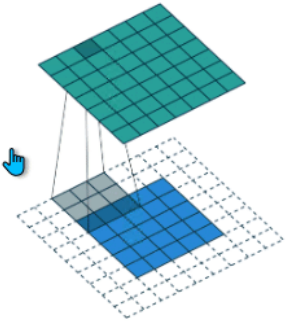
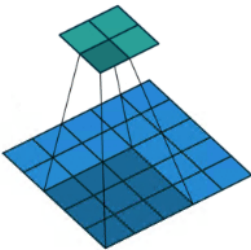
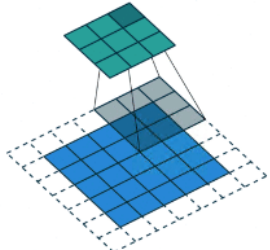
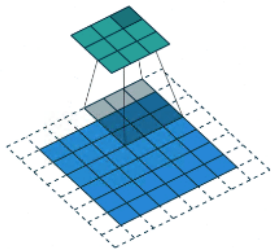
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

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

1.2 CNN——nn.Conv2d

Convolution animations

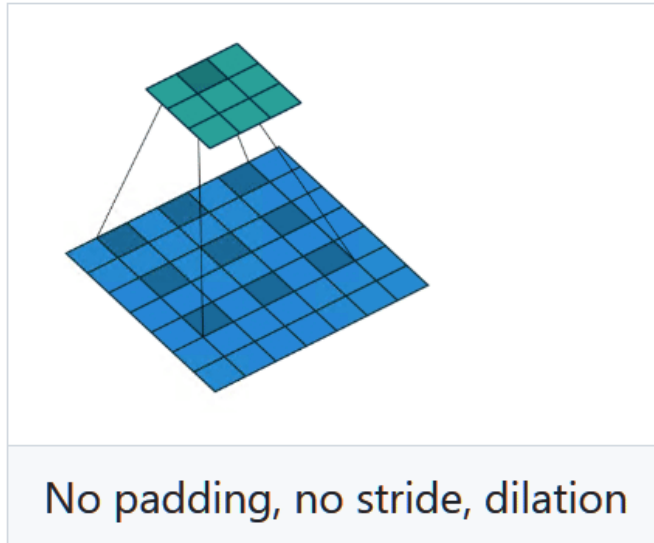
N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

1.2 CNN——nn.Conv2d

Dilated convolution animations

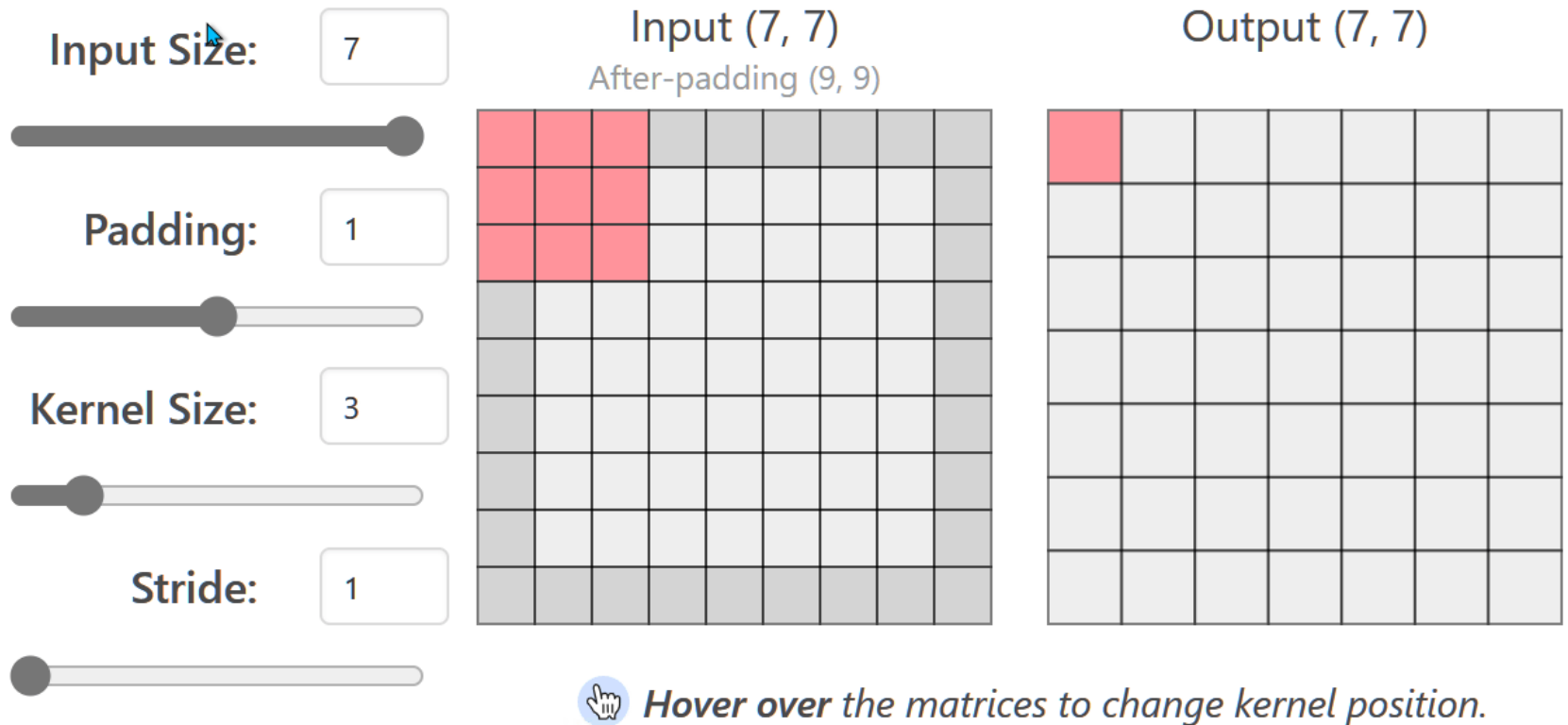
N.B.: Blue maps are inputs, and cyan maps are outputs.



链接: [vdumoulin/conv_arithmetic](https://vdumoulin.com/conv_arithmetic)

1.2 CNN——nn.Conv2d

Understanding Hyperparameters



1.2 CNN——Pooling

MaxPool2d

```
CLASS torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
    return_indices=False, ceil_mode=False) [SOURCE]
```

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

In the simplest case, the output value of the layer with input size (N, C, H, W) , output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0, \dots, kH-1} \max_{n=0, \dots, kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly padded with negative infinity on both sides for `padding` number of points. `dilation` controls the spacing between the kernel points. It is harder to describe, but this [link](#) has a nice visualization of what `dilation` does.

AvgPool2d

```
CLASS torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False,
    count_include_pad=True, divisor_override=None) [SOURCE]
```

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

In the simplest case, the output value of the layer with input size (N, C, H, W) , output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly zero-padded on both sides for `padding` number of points.

1.2 CNN——Pooling

Max Pooling

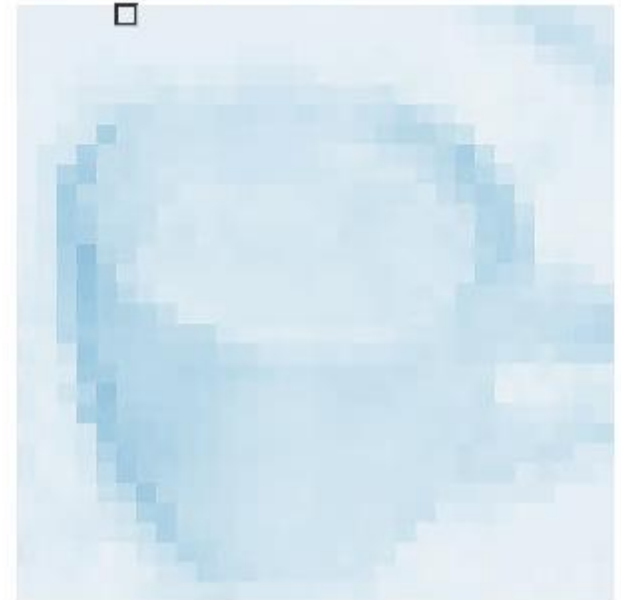


Input (60, 60)



$$\max\left(\begin{array}{|c|c|} \hline 0.3 & 0.3 \\ \hline 0.31 & 0.31 \\ \hline \end{array}\right) = \boxed{0.31}$$

Output (30, 30)



Hover over the matrices to change kernel position.

1.3 CNN 网络训练实例

卷积神经网络（CNN）（仅供参考）

假设用手写数字识别作为样例：

- 1.读入训练集和测试集中的数字图片信息以及对图片预处理
- 2.用pytorch搭建神经网络（包括卷积和全连接神经网络）
- 3.将一个batch的训练集中的图片输入至神经网络，得到所有数字的预测分类概率（总共10个数字,0123456789）
- 4.根据真实标签和预测标签，利用交叉熵损失函数计算loss值，并进行梯度下降
- 5.根据测试集计算准确率，如果准确率没收敛，跳转回步骤3
- 6.画出loss、测试集准确率的曲线图

参考视频：<https://www.bilibili.com/video/BV1Vx411j7kT?p=19>

1.3 CNN 网络训练实例

卷积神经网络（CNN）（仅供参考）

步骤2：用pytorch搭建神经网络（包括卷积和全连接神经网络）

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(          # input shape (1, 28, 28)
            nn.Conv2d(                        # input height
                in_channels=1,                # n_filters
                out_channels=16,              # filter size
                kernel_size=5,                # filter movement/step
                stride=1,                     # if want same width and length of this image after con2d, padding=(kernel_size-1)/2 if stride=1
                padding=2,                    # output shape (16, 28, 28)
            ),
            nn.ReLU(),                        # activation
            nn.MaxPool2d(kernel_size=2),      # choose max value in 2x2 area, output shape (16, 14, 14)
        )
        self.conv2 = nn.Sequential(          # input shape (1, 28, 28)
            nn.Conv2d(16, 32, 5, 1, 2),      # output shape (32, 14, 14)
            nn.ReLU(),                       # activation
            nn.MaxPool2d(2),                 # output shape (32, 7, 7)
        )
        self.out = nn.Linear(32 * 7 * 7, 10) # fully connected layer, output 10 classes

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        output = self.out(x)
        return output, x                    # return x for visualization
```

1.3 CNN 网络训练实例

卷积神经网络（CNN）（仅供参考）

步骤3：将一个batch的训练集中的图片输入至神经网络，得到所有数字的预测分类概率（下面的代码中实际上output是得到logit）

步骤4：根据真实标签和预测标签，利用交叉熵损失函数计算loss值，并进行梯度下降

```
for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader): # gives batch data, normalize x when iterate train_loader
        b_x = Variable(x) # batch x
        b_y = Variable(y) # batch y

        output = cnn(b_x)[0] # cnn output
        loss = loss_func(output, b_y) # cross entropy loss
        optimizer.zero_grad() # clear gradients for this training step
        loss.backward() # backpropagation, compute gradients
        optimizer.step() # apply gradients
```

2. 实验任务

□ 中药图片分类任务

- 利用pytorch框架搭建神经网络实现中药图片分类，具体见给出的数据集和测试集。
- 要求
 - 搭建合适的网络框架，利用训练集完成网络训练，统计网络模型的训练准确率和测试准确率，画出模型的训练过程的loss曲线、准确率曲线。
- Deadline
 - 两周，即**2024年5月27日23:59**
- 提交格式
 - E7_学号.zip

3. 参考资料

一些可能有用的链接（仅供参考）

Overview:

- [Pytorch QuickStart](#)

Load Custom Data:

- [ImageFolder](#)

CNN Visualization:

- [vdumoulin/conv_arithmetic](#)
- [CNN Explainer](#)