HW2 Deep Learning (COSE474)

Submitted by : Andi Analta Dwiyanto Palalangan Tunru (2024962230)

```
!pip install d2l
```

```
Requirement already satisfied: d2l in /usr/local/lib/python3.10/dist-packages (1.0.3)
Requirement already satisfied: jupyter==1.0.0 in /usr/local/lib/python3.10/dist-packages (from d2l) (1.0.0)
Requirement already satisfied: numpy==1.23.5 in /usr/local/lib/python3.10/dist-packages (from d2l) (1.23.5)
Requirement already satisfied: matplotlib==3.7.2 in /usr/local/lib/python3.10/dist-packages (from d2l) (3.7.2)
Requirement already satisfied: matplotlib-inline==0.1.6 in /usr/local/lib/python3.10/dist-packages (from d2l) (0.1.6)
Requirement already satisfied: requests==2.31.0 in /usr/local/lib/python3.10/dist-packages (from d2l) (2.31.0)
Requirement already satisfied: pandas==2.0.3 in /usr/local/lib/python3.10/dist-packages (from d2l) (2.0.3)
Requirement already satisfied: scipy==1.10.1 in /usr/local/lib/python3.10/dist-packages (from d2l) (1.10.1)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l) (6.5.5)
Requirement already satisfied: qtconsole in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l) (5.6.0)
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l) (6.1.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l) (6.5.4)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l) (5.5.6)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l) (7.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (10.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (3.0
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l) (2.8
Requirement already satisfied: traitlets in /usr/local/lib/python3.10/dist-packages (from matplotlib-inline==0.1.6->d2l) (5.7.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas==2.0.3->d2l) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas==2.0.3->d2l) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l) (2024.8
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib==3.7.2->
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l) (
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l) (7
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l) (6
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l) (6.3
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from jupyt
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter==1.0.0->d2l) (2
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (4
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (3.1.4
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (2
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (0
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (5.1
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (1.3.0)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l) (24.0
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l) (23.1.0
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l) (
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l) (
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l) (0
```

```
import torch
from torch import nn
from d2l import torch as d2l
```

7 Convolutional Neural Network

7.1 From Fully Connected Layers to Convolutions

1. Notes :

- Local Representation : Small Portions and Simpler, eg. edges, colors, texture
- Longer Range : Abstract / Higher Level Concept, eg. the object itself

2. Questions :

- Why/How did these desideratas come to be
- Comparison with concepts that directly focuses on longer range features
- Performance on challenges such as occlusion

7.2 Convolutions for images

7.2.1 The Cross-Correlation Operation

```
def corr2d(X, K):
    """Compute 2D cross-correlation."""
    h, w = K.shape
    Y = torch.zeros((X.shape[0] - h + 1, X.shape[1] - w + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            Y[i, j] = (X[i:i + h, j:j + w] * K).sum()
    return Y
```

```
X = torch.tensor([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]])
K = torch.tensor([[0.0, 1.0], [2.0, 3.0]])
corr2d(X, K)
```

```
tensor([[19., 25.],
        [37., 43.]])
```

7.2.2 Convolutional Layers

```
class Conv2D(nn.Module):
    def __init__(self, kernel_size):
        super().__init__()
        self.weight = nn.Parameter(torch.rand(kernel_size))
        self.bias = nn.Parameter(torch.zeros(1))

    def forward(self, x):
        return corr2d(x, self.weight) + self.bias
```

7.2.3 Object Edge Detection in Images

```
X = torch.ones((6, 8))
X[:, 2:6] = 0
X
```

```
tensor([[1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.]])
```

```
K = torch.tensor([[1.0, -1.0]])
```

```
Y = corr2d(X, K)
Y
```

```
tensor([[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.]])
```

```
corr2d(X.t(), K)
```

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

The kernel is sensitive to changes in image (tranposedd, etc.). how to fix?

7.2.4 Learning a Kernel

```
# Construct a two-dimensional convolutional layer with 1 output channel and a
# kernel of shape (1, 2). For the sake of simplicity, we ignore the bias here
conv2d = nn.LazyConv2d(1, kernel_size=(1, 2), bias=False)

# The two-dimensional convolutional layer uses four-dimensional input and
# output in the format of (example, channel, height, width), where the batch
# size (number of examples in the batch) and the number of channels are both 1
X = X.reshape((1, 1, 6, 8))
Y = Y.reshape((1, 1, 6, 7))
lr = 3e-2  # Learning rate

for i in range(10):
    Y_hat = conv2d(X)
    l = (Y_hat - Y) ** 2
    print(l)
    conv2d.zero_grad()
    l.sum().backward()
    # Update the kernel
    conv2d.weight.data[:] -= lr * conv2d.weight.grad
    if (i + 1) % 2 == 0:
        print(f'epoch {i + 1}, loss {l.sum():.3f}')
```

```
tensor([[[[0.3675, 1.2287, 0.0000, 0.0000, 0.0000, 0.2523, 0.3675],
          [0.3675, 1.2287, 0.0000, 0.0000, 0.0000, 0.2523, 0.3675],
          [0.3675, 1.2287, 0.0000, 0.0000, 0.0000, 0.2523, 0.3675],
          [0.3675, 1.2287, 0.0000, 0.0000, 0.0000, 0.2523, 0.3675],
          [0.3675, 1.2287, 0.0000, 0.0000, 0.0000, 0.2523, 0.3675],
          [0.3675, 1.2287, 0.0000, 0.0000, 0.0000, 0.2523, 0.3675]]]],
       grad_fn=<PowBackward0>)
tensor([[[[0.2352, 0.0745, 0.0000, 0.0000, 0.0000, 0.5744, 0.2352],
          [0.2352, 0.0745, 0.0000, 0.0000, 0.0000, 0.5744, 0.2352],
          [0.2352, 0.0745, 0.0000, 0.0000, 0.0000, 0.5744, 0.2352],
          [0.2352, 0.0745, 0.0000, 0.0000, 0.0000, 0.5744, 0.2352],
          [0.2352, 0.0745, 0.0000, 0.0000, 0.0000, 0.5744, 0.2352],
          [0.2352, 0.0745, 0.0000, 0.0000, 0.0000, 0.5744, 0.2352]]]],
       grad_fn=<PowBackward0>)
epoch 2, loss 6.716
tensor([[[[0.1505, 0.2744, 0.0000, 0.0000, 0.0000, 0.0185, 0.1505],
          [0.1505, 0.2744, 0.0000, 0.0000, 0.0000, 0.0185, 0.1505],
          [0.1505, 0.2744, 0.0000, 0.0000, 0.0000, 0.0185, 0.1505],
          [0.1505, 0.2744, 0.0000, 0.0000, 0.0000, 0.0185, 0.1505],
          [0.1505, 0.2744, 0.0000, 0.0000, 0.0000, 0.0185, 0.1505],
          [0.1505, 0.2744, 0.0000, 0.0000, 0.0000, 0.0185, 0.1505]]]],
       grad_fn=<PowBackward0>)
tensor([[[[0.0963, 0.0031, 0.0000, 0.0000, 0.0000, 0.1342, 0.0963],
          [0.0963, 0.0031, 0.0000, 0.0000, 0.0000, 0.1342, 0.0963],
          [0.0963, 0.0031, 0.0000, 0.0000, 0.0000, 0.1342, 0.0963],
          [0.0963, 0.0031, 0.0000, 0.0000, 0.0000, 0.1342, 0.0963],
          [0.0963, 0.0031, 0.0000, 0.0000, 0.0000, 0.1342, 0.0963],
          [0.0963, 0.0031, 0.0000, 0.0000, 0.0000, 0.1342, 0.0963]]]],
       grad_fn=<PowBackward0>)
epoch 4, loss 1.980
tensor([[[[0.0616, 0.0672, 0.0000, 0.0000, 0.0000, 0.0001, 0.0616],
          [0.0616, 0.0672, 0.0000, 0.0000, 0.0000, 0.0001, 0.0616],
          [0.0616, 0.0672, 0.0000, 0.0000, 0.0000, 0.0001, 0.0616],
          [0.0616, 0.0672, 0.0000, 0.0000, 0.0000, 0.0001, 0.0616],
          [0.0616, 0.0672, 0.0000, 0.0000, 0.0000, 0.0001, 0.0616],
          [0.0616, 0.0672, 0.0000, 0.0000, 0.0000, 0.0001, 0.0616]]]],
       grad_fn=<PowBackward0>)
tensor([[[[0.0395, 0.0002, 0.0000, 0.0000, 0.0000, 0.0345, 0.0395],
          [0.0395, 0.0002, 0.0000, 0.0000, 0.0000, 0.0345, 0.0395],
          [0.0395, 0.0002, 0.0000, 0.0000, 0.0000, 0.0345, 0.0395],
          [0.0395, 0.0002, 0.0000, 0.0000, 0.0000, 0.0345, 0.0395],
          [0.0395, 0.0002, 0.0000, 0.0000, 0.0000, 0.0345, 0.0395],
          [0.0395, 0.0002, 0.0000, 0.0000, 0.0000, 0.0345, 0.0395]]]],
       grad_fn=<PowBackward0>)
epoch 6, loss 0.682
tensor([[[[0.0253, 0.0182, 0.0000, 0.0000, 0.0000, 0.0006, 0.0253],
          [0.0253, 0.0182, 0.0000, 0.0000, 0.0000, 0.0006, 0.0253],
          [0.0253, 0.0182, 0.0000, 0.0000, 0.0000, 0.0006, 0.0253],
          [0.0253, 0.0182, 0.0000, 0.0000, 0.0000, 0.0006, 0.0253],
          [0.0253, 0.0182, 0.0000, 0.0000, 0.0000, 0.0006, 0.0253],
          [0.0253, 0.0182, 0.0000, 0.0000, 0.0000, 0.0006, 0.0253]]]],
       grad_fn=<PowBackward0>)
tensor([[[[0.0162, 0.0008, 0.0000, 0.0000, 0.0000, 0.0098, 0.0162],
          [0.0162, 0.0008, 0.0000, 0.0000, 0.0000, 0.0098, 0.0162],
          [0.0162, 0.0008, 0.0000, 0.0000, 0.0000, 0.0098, 0.0162],
          [0.0162, 0.0008, 0.0000, 0.0000, 0.0000, 0.0098, 0.0162],
          [0.0162, 0.0008, 0.0000, 0.0000, 0.0000, 0.0098, 0.0162],
          [0.0162, 0.0008, 0.0000, 0.0000, 0.0000, 0.0098, 0.0162]]]],
```

```
conv2d.weight.data.reshape((1, 2))
```

```
tensor([[ 0.9582, -1.0233]])
```

7.2.6 Feature Map and Receptive Field

Notes:

- Receptive Field : Elements (from all the previous layers) that may affect the calculation of X

Questions :

- How can the receptive field be of bigger dimension than the input image

7.3 Padding and Stride

7.3.1 Padding

Common Practice :

- Use an odd number amount for kernel size

```
# We define a helper function to calculate convolutions. It initializes the
# convolutional layer weights and performs corresponding dimensionality
# elevations and reductions on the input and output
def comp_conv2d(conv2d, X):
    # (1, 1) indicates that batch size and the number of channels are both 1
    X = X.reshape((1, 1) + X.shape)
    Y = conv2d(X)
    # Strip the first two dimensions: examples and channels
    return Y.reshape(Y.shape[2:])

# 1 row and column is padded on either side, so a total of 2 rows or columns
# are added
conv2d = nn.LazyConv2d(1, kernel_size=3, padding=1)
X = torch.rand(size=(8, 8))
comp_conv2d(conv2d, X).shape
```

```
torch.Size([8, 8])
```

```
# We use a convolution kernel with height 5 and width 3. The padding on either
# side of the height and width are 2 and 1, respectively
conv2d = nn.LazyConv2d(1, kernel_size=(5, 3), padding=(2, 1))
comp_conv2d(conv2d, X).shape
```

```
torch.Size([8, 8])
```

7.3.2 Stride

```
conv2d = nn.LazyConv2d(1, kernel_size=3, padding=1, stride=2)
comp_conv2d(conv2d, X).shape
```

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

```
conv2d = nn.LazyConv2d(1, kernel_size=(3, 5), padding=(0, 1), stride=(3, 4))
comp_conv2d(conv2d, X).shape
```

```
torch.Size([2, 2])
```

7.4 Multiple Input and Multiple Output Channel

7.4.1 Multiple Input Channel

```
def corr2d_multi_in(X, K):
    # Iterate through the 0th dimension (channel) of K first, then add them up
    return sum(d2l.corr2d(x, k) for x, k in zip(X, K))
```

```
X = torch.tensor([[[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]],
              [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]])
K = torch.tensor([[[0.0, 1.0], [2.0, 3.0]], [[1.0, 2.0], [3.0, 4.0]]])

corr2d_multi_in(X, K)
```

```
tensor([[ 56.,  72.],
        [104., 120.]])
```

7.4.2 Multiple Output Channel

```
def corr2d_multi_in_out(X, K):
    # Iterate through the 0th dimension of K, and each time, perform
    # cross-correlation operations with input X. All of the results are
    # stacked together
    return torch.stack([corr2d_multi_in(X, k) for k in K], 0)


K = torch.stack((K, K + 1, K + 2), 0)
K.shape
```

```
torch.Size([3, 2, 2, 2])
```

```
corr2d_multi_in_out(X, K)
```

```
tensor([[[ 56.,  72.],
         [104., 120.]],

        [[ 76., 100.],
         [148., 172.]],

        [[ 96., 128.],
         [192., 224.]]])
```

7.4.3 1x1 Convolutional Layer

```
def corr2d_multi_in_out_1x1(X, K):
    c_i, h, w = X.shape
    c_o = K.shape[0]
    X = X.reshape((c_i, h * w))
    K = K.reshape((c_o, c_i))
    # Matrix multiplication in the fully connected layer
    Y = torch.matmul(K, X)
    return Y.reshape((c_o, h, w))


X = torch.normal(0, 1, (3, 3, 3))
K = torch.normal(0, 1, (2, 3, 1, 1))
Y1 = corr2d_multi_in_out_1x1(X, K)
Y2 = corr2d_multi_in_out(X, K)
assert float(torch.abs(Y1 - Y2).sum()) < 1e-6
```

Note :

- Acts like a Fully Connected Layer
- Useful for dimensionality reduction (Channel Reduction) while keeping the spatial dimension of the feature
- Introduce non linearity, as it changes the amount of channels, it is often followed by activation functions
- Used typically in cases such as :

1. After regular convolutional layers
2. Before and After pooling layers

7.5 Pooling

7.5.1 Max Pooling and Average Pooling

```
def pool2d(X, pool_size, mode='max'):
    p_h, p_w = pool_size
    Y = torch.zeros((X.shape[0] - p_h + 1, X.shape[1] - p_w + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            if mode == 'max':
```

```
                Y[i, j] = X[i: i + p_h, j: j + p_w].max()
            elif mode == 'avg':
                Y[i, j] = X[i: i + p_h, j: j + p_w].mean()
    return Y


X = torch.tensor([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]])
pool2d(X, (2, 2))
```

```
tensor([[4., 5.],
        [7., 8.]])
```

```
pool2d(X, (2, 2), 'avg')
```

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

7.5.2 Padding and Stride

```
X = torch.arange(16, dtype=torch.float32).reshape((1, 1, 4, 4))
X
```

```
tensor([[[[ 0.,  1.,  2.,  3.],
          [ 4.,  5.,  6.,  7.],
          [ 8.,  9., 10., 11.],
          [12., 13., 14., 15.]]]])
```

```
pool2d = nn.MaxPool2d(3)
# Pooling has no model parameters, hence it needs no initialization
pool2d(X)
```

```
tensor([[[[10.]]]])
```

```
pool2d = nn.MaxPool2d(3, padding=1, stride=2)
pool2d(X)
```

```
tensor([[[[ 5.,  7.],
          [13., 15.]]]])
```

```
pool2d = nn.MaxPool2d((2, 3), stride=(2, 3), padding=(0, 1))
pool2d(X)
```

```
tensor([[[[ 5.,  7.],
          [13., 15.]]]])
```

7.5.3 Multiple Channels

```
X = torch.cat((X, X + 1), 1)
X
```

```
tensor([[[[ 0.,  1.,  2.,  3.],
          [ 4.,  5.,  6.,  7.],
          [ 8.,  9., 10., 11.],
          [12., 13., 14., 15.]],

         [[ 1.,  2.,  3.,  4.],
          [ 5.,  6.,  7.,  8.],
          [ 9., 10., 11., 12.],
          [13., 14., 15., 16.]]]])
```

```
pool2d = nn.MaxPool2d(3, padding=1, stride=2)
pool2d(X)
```

```
tensor([[[[ 5.,  7.],
          [13., 15.]],

         [[ 6.,  8.],
          [14., 16.]]]])
```

7.6 Convolutional Neural Network (LeNet)

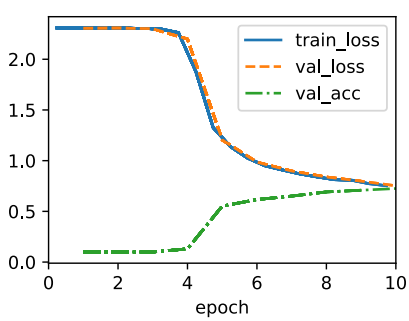
7.6.1 LeNet

```
def init_cnn(module):
    """Initialize weights for CNNs."""
    if type(module) == nn.Linear or type(module) == nn.Conv2d:
        nn.init.xavier_uniform_(module.weight)

class LeNet(d2l.Classifier):
    """The LeNet-5 model."""
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(6, kernel_size=5, padding=2), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.LazyLinear(120), nn.Sigmoid(),
            nn.LazyLinear(84), nn.Sigmoid(),
            nn.LazyLinear(num_classes))


@d2l.add_to_class(d2l.Classifier)
def layer_summary(self, X_shape):
    X = torch.randn(*X_shape)
    for layer in self.net:
        X = layer(X)
        print(layer.__class__.__name__, 'output shape:\t', X.shape)

model = LeNet()
model.layer_summary((1, 1, 28, 28))
```

```
Conv2d output shape:     torch.Size([1, 6, 28, 28])
Sigmoid output shape:    torch.Size([1, 6, 28, 28])
AvgPool2d output shape:  torch.Size([1, 6, 14, 14])
Conv2d output shape:     torch.Size([1, 16, 10, 10])
Sigmoid output shape:    torch.Size([1, 16, 10, 10])
AvgPool2d output shape:  torch.Size([1, 16, 5, 5])
Flatten output shape:    torch.Size([1, 400])
Linear output shape:     torch.Size([1, 120])
Sigmoid output shape:    torch.Size([1, 120])
Linear output shape:     torch.Size([1, 84])
Sigmoid output shape:    torch.Size([1, 84])
Linear output shape:     torch.Size([1, 10])
```

7.6.2 Training

```
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
data = d2l.FashionMNIST(batch_size=128)
model = LeNet(lr=0.1)
model.apply_init([next(iter(data.get_dataloader(True)))[0]], init_cnn)
trainer.fit(model, data)
```



8. Modern Convolutional Neural Network (CNN)

8.1 Deep Convolutional Neural Network (AlexNet)

8.1.2 AlexNet

8.1.2.3 Capacity Control and Preprocessing

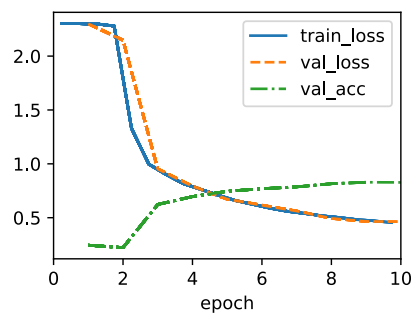
```
class AlexNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(96, kernel_size=11, stride=4, padding=1),
            nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(256, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(256, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2), nn.Flatten(),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(p=0.5),
            nn.LazyLinear(4096), nn.ReLU(),nn.Dropout(p=0.5),
            nn.LazyLinear(num_classes))
        self.net.apply(d2l.init_cnn)
```

```
AlexNet().layer_summary((1, 1, 224, 224))
```

```
Conv2d output shape:     torch.Size([1, 96, 54, 54])
ReLU output shape:       torch.Size([1, 96, 54, 54])
MaxPool2d output shape:  torch.Size([1, 96, 26, 26])
Conv2d output shape:     torch.Size([1, 256, 26, 26])
ReLU output shape:       torch.Size([1, 256, 26, 26])
MaxPool2d output shape:  torch.Size([1, 256, 12, 12])
Conv2d output shape:     torch.Size([1, 384, 12, 12])
ReLU output shape:       torch.Size([1, 384, 12, 12])
Conv2d output shape:     torch.Size([1, 384, 12, 12])
ReLU output shape:       torch.Size([1, 384, 12, 12])
Conv2d output shape:     torch.Size([1, 256, 12, 12])
ReLU output shape:       torch.Size([1, 256, 12, 12])
MaxPool2d output shape:  torch.Size([1, 256, 5, 5])
Flatten output shape:    torch.Size([1, 6400])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 10])
```

8.1.3 Training

```
model = AlexNet(lr=0.01)
data = d2l.FashionMNIST(batch_size=128, resize=(224, 224))
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
```



Discussion :

- Convolutional Neural Networks are sensitive to the image size (input dimension), mentioned by author that to put the MNIST dataset into the original AlexNet, they needed to upscale the images, is there a way to make this not necessary
- Is there a model/architecture that can generalize on different image resolutions

8.2 Network Using Blocks (VGG)

8.2.1 VGG Blocks

```
def vgg_block(num_convs, out_channels):
    layers = []
```

```
    for _ in range(num_convs):
        layers.append(nn.LazyConv2d(out_channels, kernel_size=3, padding=1))
        layers.append(nn.ReLU())
    layers.append(nn.MaxPool2d(kernel_size=2,stride=2))
    return nn.Sequential(*layers)
```

8.2.2 VGG Network

```
class VGG(d2l.Classifier):
    def __init__(self, arch, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        conv_blks = []
        for (num_convs, out_channels) in arch:
            conv_blks.append(vgg_block(num_convs, out_channels))
        self.net = nn.Sequential(
            *conv_blks, nn.Flatten(),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(0.5),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(0.5),
            nn.LazyLinear(num_classes))
        self.net.apply(d2l.init_cnn)
```

```
VGG(arch=((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))).layer_summary(
    (1, 1, 224, 224))
```

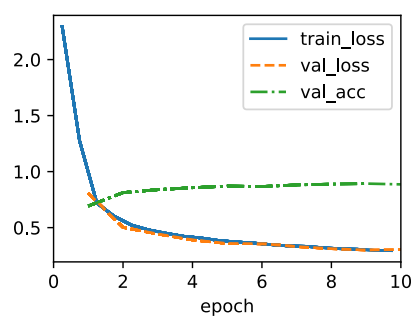
```
Sequential output shape:         torch.Size([1, 64, 112, 112])
Sequential output shape:         torch.Size([1, 128, 56, 56])
Sequential output shape:         torch.Size([1, 256, 28, 28])
Sequential output shape:         torch.Size([1, 512, 14, 14])
Sequential output shape:         torch.Size([1, 512, 7, 7])
Flatten output shape:    torch.Size([1, 25088])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 10])
```

Notes :

- Essentially seperating and restructuring larger kernel (filters) from the typical CNN such as AlexNet into blocks of 3x3 Convolution (2 3x3 Conv is similar to 1 5x5, 3 3x3 Conv is similar to 1 7x7, etc.)

8.2.3 Training

```
model = VGG(arch=((1, 16), (1, 32), (2, 64), (2, 128), (2, 128)), lr=0.01)
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
data = d2l.FashionMNIST(batch_size=128, resize=(224, 224))
model.apply_init([next(iter(data.get_dataloader(True)))[0]], d2l.init_cnn)
trainer.fit(model, data)
```



8.6 Residual Network (ResNet) and ResNeXt

```
from torch.nn import functional as F
```

8.6.2 Residual Blocks

```
class Residual(nn.Module):
    """The Residual block of ResNet models."""
    def __init__(self, num_channels, use_1x1conv=False, strides=1):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=strides)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.LazyConv2d(num_channels, kernel_size=1,
                                       stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.LazyBatchNorm2d()
        self.bn2 = nn.LazyBatchNorm2d()

    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)
        Y += X
        return F.relu(Y)


blk = Residual(3)
X = torch.randn(4, 3, 6, 6)
blk(X).shape
```

```
torch.Size([4, 3, 6, 6])
```

```
blk = Residual(6, use_1x1conv=True, strides=2)
blk(X).shape
```

```
torch.Size([4, 6, 3, 3])
```

8.6.3 ResNet Model

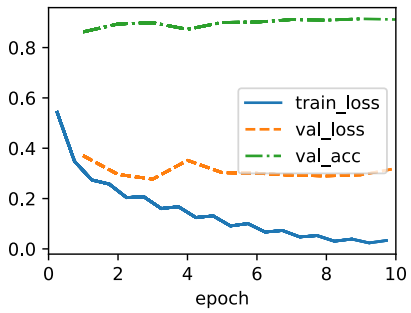
```
class ResNet(d2l.Classifier):
    def b1(self):
        return nn.Sequential(
            nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3),
            nn.LazyBatchNorm2d(), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


@d2l.add_to_class(ResNet)
def block(self, num_residuals, num_channels, first_block=False):
    blk = []
    for i in range(num_residuals):
        if i == 0 and not first_block:
            blk.append(Residual(num_channels, use_1x1conv=True, strides=2))
        else:
            blk.append(Residual(num_channels))
    return nn.Sequential(*blk)


@d2l.add_to_class(ResNet)
def __init__(self, arch, lr=0.1, num_classes=10):
    super(ResNet, self).__init__()
    self.save_hyperparameters()
    self.net = nn.Sequential(self.b1())
    for i, b in enumerate(arch):
        self.net.add_module(f'b{i+2}', self.block(*b, first_block=(i==0)))
    self.net.add_module('last', nn.Sequential(
        nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten(),
        nn.LazyLinear(num_classes)))
    self.net.apply(d2l.init_cnn)


class ResNet18(ResNet):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__(((2, 64), (2, 128), (2, 256), (2, 512)),
                         lr, num_classes)

ResNet18().layer_summary((1, 1, 96, 96))
```

```
Sequential output shape:         torch.Size([1, 64, 24, 24])
Sequential output shape:         torch.Size([1, 64, 24, 24])
Sequential output shape:         torch.Size([1, 128, 12, 12])
Sequential output shape:         torch.Size([1, 256, 6, 6])
Sequential output shape:         torch.Size([1, 512, 3, 3])
```

```
Sequential output shape:         torch.Size([1, 10])
```

8.6.4 Training

```
model = ResNet18(lr=0.01)
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
model.apply_init([next(iter(data.get_dataloader(True)))[0]], d2l.init_cnn)
trainer.fit(model, data)
```



8.6.5 ResNeXt

```
class ResNeXtBlock(nn.Module):
    """The ResNeXt block."""
    def __init__(self, num_channels, groups, bot_mul, use_1x1conv=False,
                 strides=1):
        super().__init__()
        bot_channels = int(round(num_channels * bot_mul))
        self.conv1 = nn.LazyConv2d(bot_channels, kernel_size=1, stride=1)
        self.conv2 = nn.LazyConv2d(bot_channels, kernel_size=3,
                                   stride=strides, padding=1,
                                   groups=bot_channels//groups)
        self.conv3 = nn.LazyConv2d(num_channels, kernel_size=1, stride=1)
        self.bn1 = nn.LazyBatchNorm2d()
        self.bn2 = nn.LazyBatchNorm2d()
        self.bn3 = nn.LazyBatchNorm2d()
        if use_1x1conv:
            self.conv4 = nn.LazyConv2d(num_channels, kernel_size=1,
                                       stride=strides)
            self.bn4 = nn.LazyBatchNorm2d()
        else:
            self.conv4 = None

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = F.relu(self.bn2(self.conv2(Y)))
        Y = self.bn3(self.conv3(Y))
        if self.conv4:
            X = self.bn4(self.conv4(X))
        return F.relu(Y + X)


blk = ResNeXtBlock(32, 16, 1)
X = torch.randn(4, 32, 96, 96)
blk(X).shape
```

```
torch.Size([4, 32, 96, 96])
torch.Size([4, 32, 96, 96])
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.