COSE474-Homework2. Convolutional Networks

-by Chung Dong Kyun

```
pip install d2l==1.0.3
```

```
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.2.0)
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (7.34.0)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (6.3.3)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==1.0.3) (3.6.9)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==1.0.3) (3.0.13)
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 jupyter-console->jupyter==1.0.0
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter==1.0.0->d2l==1.0.3) (2.18.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (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==1.0.3) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (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==1.0.3) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.10.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (5.10.4)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (1.5.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (1.3.0)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (24.0.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (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==1.0.3) (1.6.0)
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (0.18.1)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (0.21.0)
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (1.1.0)
Requirement already satisfied: qtpy>=2.4.0 in /usr/local/lib/python3.10/dist-packages (from qtconsole->jupyter==1.0.0->d2l==1.0.3) (2.4.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.19.1
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.7.5)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.2.0)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3) (4.9.0
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert->jupyter==1.0.0->d2l==1.
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook->jupyter==1.0.0->d2l=
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->jupyter-console->jupy
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.10/dist-packages (from terminado>=0.8.3->notebook->jupyter==1.0.0->d2l==1.0.3) (0.7
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook->jupyter==1.0.0->d2l==1.0.3) (
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert->jupyter==1.0.0->d2l==1.0.3) (2.
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.5.1)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=5.0.0->ipykernel->jupyter==1.0.
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter==1.0
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyte
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter==
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.10/dist-packages (from notebook-shim>=0.2.3->nbclassic>=0.4.7->notebool
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings->argon2-cffi->notebook->jupyter==1.0.0->
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook->jupyt
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook-shim>=0.
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook-shim>
```

Chapter 7. Convolutional Neural Networks

7.1. From Fully Connected Layers to Convoultions

7.2. Convolution to Images

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

7.2.1. The Cross-Correlation Operation

```
def corr2d(X, K):  ##save
    """Compute 2D cross-correlation."""
    h, w = K.shape   ##h = horizontal, w = width
    Y = torch.zeros((X.shape[0] - h + 1, X.shape[1] - w + 1))  ##coordinates locked in
    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() ##convolution!
    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): ##forward propagation
        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) ##expected all zeros, because the convolution data is vanished
```

```
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.]])
```

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 - book

#translation: no bias, put input data in 1*2 kernel, get the result in 1 output channel
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 - book

#setting input and lr values
X = X.reshape((1, 1, 6, 8))
```

```
Y = Y.reshape((1, 1, 6, 7))
lr = 3e-2  # Learning rate


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

```
epoch 2, loss 10.077
epoch 4, loss 1.960
epoch 6, loss 0.439
epoch 8, loss 0.119
epoch 10, loss 0.039
```

```
conv2d.weight.data.reshape((1, 2)) ##so what's the outcome?
```

```
tensor([[ 1.0027, -0.9661]])
```

7.3. Padding and Stride

```
import torch
from torch import nn
```

7.3.1. Padding

```
# 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

#inshort: yes, it's the shortcut for convolution (yay)
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

##checkem. ##nn->where LazyConv2d is
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 ##result -> same!
```

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

7.3.2. Stride

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

```
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 Channels

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

7.4.1. Multiple Input Channels

```
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) ##expected 2*2 result
```

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

7.4.2. Multiple Output Channels

```
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 ##stacked K dimension check
```

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

```
corr2d_multi_in_out(X, K) ##Results, check!
```

```
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  ##assert -> if the result is smaller than 1e-6, that's it!
```

7.5. Pooling

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

7.5.1. Maximum Pooling and Average Pooling

```
def pool2d(X, pool_size, mode='max'):     ##max/avg pool function
    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) ##4*4->2*2
pool2d(X)
```

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

```
pool2d = nn.MaxPool2d((2, 3), stride=(2, 3), padding=(0, 1))  ## with stride & padding
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) ##multiple channel case
pool2d(X)
```

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

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

7.6. Convolutional Neural Networks (LeNet)

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

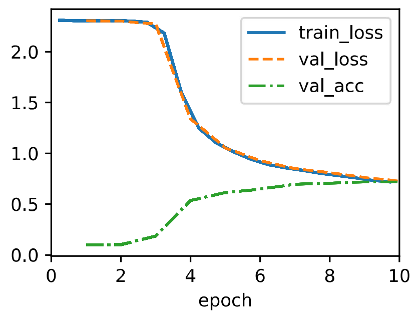
7.6.1. LeNet

```
def init_cnn(module):  ##save
    """Initialize weights for CNNs only if the module is linear or Conv2d."""
    if type(module) == nn.Linear or type(module) == nn.Conv2d:
        nn.init.xavier_uniform_(module.weight)

class LeNet(d2l.Classifier):  ##save
    """The LeNet-5 model."""
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(           ##set size of out model!
            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)  ## that at mark is for adding methods!
def layer_summary(self, X_shape):  ##print our structure!
    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)
```



Chapter 8. Modern Convolutional Neural Networks

8.2. Netswork Using Blocks (VGG)

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

8.2.1. VGG Blocks

```
def vgg_block(num_convs, out_channels): ##making Vgg blocks
    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

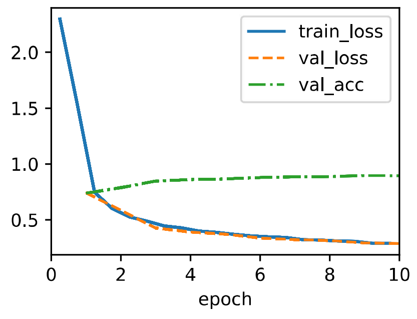
```
class VGG(d2l.Classifier):   ###build network using VGG blocks!
    def __init__(self, arch, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        conv_blks = []
        for (num_convs, out_channels) in arch:  ##add blocks into our model
            conv_blks.append(vgg_block(num_convs, out_channels))
        self.net = nn.Sequential(   ##our structure
            *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])
```

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) ##warning, too much time consumed!
```



8.6. Residual Networks(ResNet) and ResNeXt

```
import torch
from torch import nn
from torch.nn import functional as F
from d2l import torch as d2l
```

8.6.2. Residual Blocks

```
class Residual(nn.Module):  ##Residual models!!
    """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):   ##forward propagation
        Y = F.relu(self.bn1(self.conv1(X))) ##of course relu is included
        Y = self.bn2(self.conv2(Y)) #conv2
        if self.conv3:
            X = self.conv3(X) ##residual add part
        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):  ##building ResNet class with basic environment
    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)
```

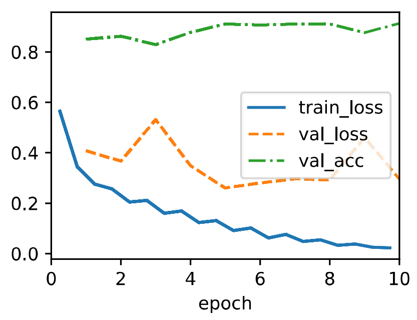
```
@d2l.add_to_class(ResNet)
def __init__(self, arch, lr=0.1, num_classes=10):  #initializing settings
    super(ResNet, self).__init__()
    self.save_hyperparameters()
    self.net = nn.Sequential(self.b1())
    for i, b in enumerate(arch):
        self.net.add_module(f'b{i+2}', self.block(*b, first_block=(i==0)))
    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)
```



Discussions

7.1. Convolution Concept : Identifying input data with fewer parameters. If the goal can be specified, we don't need to look at all characterstics of the input. In other words, we can compress it with only essential parameters.

7.1.Translation Invariance : So the change of input data should affect the hidden layer, and to guarantee this the weight and bias depend on the input data's location. If not, the hidden layer's result cannot distinguish the weight tensor or bias tensor's effect frome that of input.

7.2. It's important to build correct kernels: vertical edges detect in the sample

7.3. Stride size do affect the output tensor's size.

7.4. 1X1 convolution : what they do is not related to compress pattern within the same dimension, it's for compressing different dimension.

7.6. LeNet Layer: Combination of convolution-pooling sequences, and attach FC layers to dense the result.

8.2. VGG Network : VGG blocks, then FC layers to dense the result

8.6. Residual blocks: separate f(x) into x and the residual, put the residual into weight+bias layer, sum the original. This operation is for faster operation.

Exercises

7.1.6.Exercise-1. What happens when the size of kernel is $\Delta = 0$?

If such, $[H]_{i,j,d} = \sum_{a=-\Delta}^{\Delta}\sum_{b=-\Delta}^{\Delta}\sum_c([V]_{a,b,c,d}[X]_{i+a,j+b,c})$ will be reduced into the equation down below.

$[H]_{i,j,d} = \sum_c([V]_{c,d}[X]_{i,j,c})$

This is the simple sum of tensor multiplication of X and V. Each element in this 2d-tensor V, $V_{c,d}$, is doing exactly same operation as MLP. It's not different as implementing MLP independently for each set of channels.

7.1.6.Exercise-2. Audio file Convolution.

1. If you want to extract some special soundwave from the audio file you need to remove other noises. Such cases would require convoultion process.
2. Since audio file is one dimensional and continous, it has to be splitted into discrete intervals and saved into 1d tensor. So an audio file after this preprocessing would be expressed as X, where $[X]_a$ is the element of ath sample interval. Same modification will be applied to the H. Then the weight tensor V should be revised as 3d-tensor. Then we can transform the convolution equation as this: $[H]_{j,d} = \sum_{a=-\Delta}^{\Delta}\sum_c([V]_{a,c,d}[X]_{i+a,c})$.
3. Yes, but you have to use spectrogram for proper result, since this program separates the audio file into different groups, based on the wavelength.

7.3.4.Exercise-2. What does the stride 2 means in the audio file input?

Since audio file stores its data with equal length of sampling intervals, stride 2 means the length of 2 identical intervals.

7.3.4.Exercise-4. Computational benefits for larger strides?

Obviously, it will decrease the amount of total computation. Larger strides guarantees that the filter won't move to adjacent cell but skip them, for faster pace and smaller amount of computation.

7.3.4.Exercise-5. Statistical benefit of larger strides?

If stride = 1, then some cells will be sampled much more often than the other. Specifically, the ones in the side or corner won't be represented as much as the ones in the center. By appling larger strides, this problem would be solved, since some of the cells would be skipped and less sample in the convolution process.

7.4.5.Exercise-1. Two convolution kernels problem.

1. If we set the input layer as X, then the convolution will be conducted like this : $X * k_1 * k_2$. Since the multiplication between those three tensors are allowed, we can choose the order of this multiplication without changing the end result. So $(X * k_1) * k_2$ and $X * (k_1 * k_2)$ is equivalent. If we choose the latter, we can compute $k_1 * k_2$, which is the convolution between two 2d tensors, and it would end up as creating another 2d tensor of k_3. Which has the horizontal size of that of k_1 and the vertical size of that of k_2. It can be used as a single kernel, so the result would also be equivalent to $X * k_3$.
2. Suppose the dimensionality of the original input as a*b, k_1's dimensionality as $k_{1h} * k_{1v}$ and k_2's dimensionality as $k_{2h} * k_{2v}$. After those two kernels are applied, the input's dimension will be shrinked into $(a - ((k_{1h} - 1) - (k_{2h} - 1))) * (b - ((k_{1v} - 1) - (k_{2v} - 1)))$. By substituting k_1 and k_2 to k_3, we can guess the size of the k_3 kernel, which is $(((k_{1h} - 1) - (k_{2h} - 1)) - 1) * (((k_{1v} - 1) - (k_{2v} - 1) - 1))$.
3. Since convolution can be expressed as the linear sum of tensor multiplication between input and kernel, you can split the kernel into two smaller pieces. Although it would take more calculation, it won't change the end result.

7.4.5.Exercise-4. Double the input/output & double the padding.

1. Suppose the original length of input as c_i and that of the output as c_o. The number of calculation would increase by $c_i * c_o$.
2. Suppose the original padding length as p, then the input layer with doubled padding would be $2p$. The number of calculation with original length would be $(h(1 - k_h + p))(v(1 - k_v + p))$, if the stride is 1. The doubled padding length will have to do $(h(1 - k_h + 2p))(v(1 - k_v + 2p))$ amount of calculation, so it'll be slightly increased.

7.5.5.Exercise-2. Why Max pooling cannot be implemented inside the convolution.

Unlike average, finding maximum function is not linear. So it cannot be implemented into the convolution process.

7.5.5.Exercise-5. Why Max pooling and Average pooling act different?

Alongside with linear operation, maybe this would be caused by their different convergence point. While Max pooling gets the maximum value from each cells, average will literally get the average value of them. Which means, those values will converge into two different realm. If we continually pool our input into 1*1 result, the former will get us the maximum value of every cell while the latter will bring us the average of total cell's values.

8.2.5.Exercise-Self. Why does VGG takes up so much computational time?

So I've tried to implement the VGG sample code, but my compiler crashed multiple times due to its computational time. I've solved this problem by using GPU instead of CPU runtime, but I wanted to know why this method took so much time to complete the whole task(It took almost 14 minutes with GPU option for 8.2.2. Training section.)

This problem can be addressed into three points. First, VGG blocks us three consecutive convolution before the pooling. Second, all of the VGG block's filter has 3*3 length. It's small enough to guarantee accuracy, but it would require much more amount of computation too. Finally, after all the VGG blocks are completed, we had to go on FC layers two times, which takes up most of its time.

8.6.7.Exercise-1. Differences between inception layers and residual layers.

So the inception layers is the one used in the Multi-Branched Networks, which is a method that concatenates fiters into a layer. Inception layer puts original data into multiple small filters and pooling layers simultaneously, and the result would be combined into the combinational layer.

Compared with that, residual layer is much more simple in terms of computational path, since all it needs is sequential convolutional layer and addition for residual. Also, it'll be the faster option among two, since inception layers have to do multiple convolution process through different filters and the concatenation of their individual results. But in terms of accuracy, inception layers might have advantage because residual layer's efifciency and accuracy leans on the size of the filter too. And it will be more capable to express complex functions than the residual layer, thanks to its concatenation process. However, the residual layer will guarantee nested class of functions, which will guarantee the model's efficiency.