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

```
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/d
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dis
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/pytho
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.1
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.1
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/pyth
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/di
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3
```

7.1 From Fully Connected Layers to Convolutions

Exercise 3: Why might translation invariance not be a good idea after all? Give an example.

A: translation invariance 는 입력 데이터의 위치에 따라 다르게 처리하지 않고 동일하게 처리되는 특성인데 사진에 따라 이 정보가 중요하게 처리될 수 있다. 예를 들어 얼굴 인식 시스템은 눈, 코, 입과 같은 부위가 고유한 위치에 있어야 하기에 여기서는 좋지 못하다.

7.2 Convolutions for Images

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

```
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.]])
```

```
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
```

```
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.]])
```

```
# 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
    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}')
```

```
epoch 2, loss 9.263
epoch 4, loss 1.559
epoch 6, loss 0.264
epoch 8, loss 0.045
epoch 10, loss 0.008
```

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

```
tensor([[ 0.9816, -0.9867]])
```

Exercise 3: When you try to automatically find the gradient for the Conv2D class we created, what kind of error message do you see?

A: RuntimeError: One of the differentiated Tensors does not require grad. 와 같이 텐서 중 일부가 requires_grad=True 가 되어 있지 않아 기울기 계산을 하지 못하기 때문에 발생한다.

7.3 Padding and Stride

```
import torch
from torch import nn


# 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])
```

```
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])
```

Exercise 3: For audio signals, what does a stride of 2 correspond to?

A: stride를 2로 한다는 것은 오디오 신호를 처리할 때 두 샘플씩 건너뛰면서 계산을 진행하는 것을 의미한다. 2, 4, 6... 번째 신호를 필터링하거나 다운샘플링하여 데이터의 크기를 절반으로 줄여 연산 속도를 높이고 메모리 사용을 줄일 수 있다.

7.4. Multiple Input and Multiple Output Channels

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

```
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.]])
```

```
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.]]])
```

```
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
```

Exercise 4: Are the variables Y1 and Y2 in the final example of this section exactly the same? Why?

A: Y1은 행렬 곱셈 방식으로 1x1 합성곱을 수행하였고, Y2는 합성곱 연산으로 1x1 필터를 사용하였다. 수학적으로 동일한 결과를 내지만 그 과정이 다르다.

7.5. Pooling

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

```
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.]])
```

```
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.]]]])
```

```
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.]]]])
```

Exercise 5: Why do you expect max-pooling and average pooling to work differently?

A: Max-pooling은 영역 내 가장 큰 값을 남기고 average pooling은 영역 내 값을 평균해서 남긴다. 전자는 그 영역에서 가장 두드러진 특성을 강조하고, 후자는 영역 내 모든 값이 반영되기에 이미지가 더 부드러워진다.

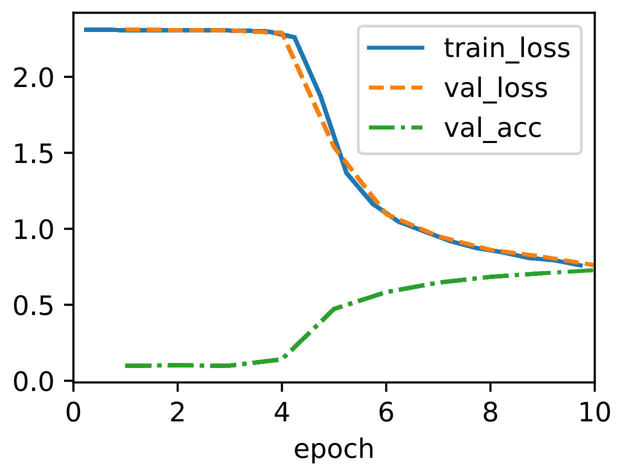
7.6. Convolutional Neural Networks

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


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])
```

```
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)
```



Exercise 5: What happens to the activations when you feed significantly different images into the network (e.g., cats, cars, or even random noise)?

A: 네트워크의 각 계층에서 활성화 값이 크게 달라지게 된다. 그 차이는 저수준에서는 크게 달라지진 않지만 학습을 진행할수록 각 이미지의 고유한 특징을 학습하여 다른 활성화 출력 값을 보이게 된다. 랜덤 노이즈는 패턴이 오히려 없기에 비정상적인 활성화 값을 가지며 분류 결과도 확실한 값을 가지지 않고 불완전한 분류를 하게 된다.

8.2. Networks Using Blocks (VGG)

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

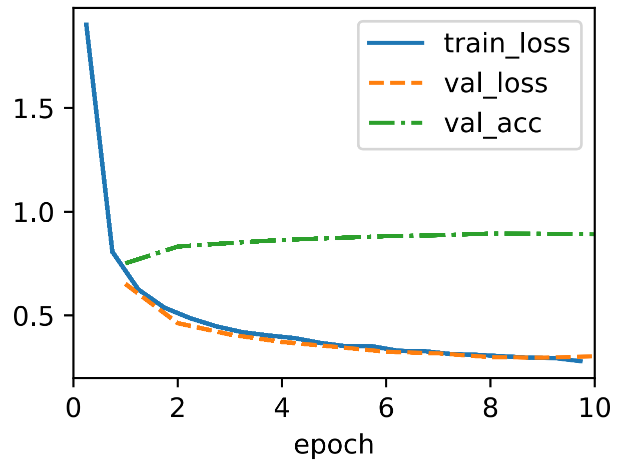
```
def vgg_block(num_convs, out_channels):
    layers = []
    for _ in range(num_convs):
        layers.append(nn.LazyConv2d(out_channels, kernel_size=3, padding=1))
        layers.append(nn.ReLU())
    layers.append(nn.MaxPool2d(kernel_size=2,stride=2))
    return nn.Sequential(*layers)
```

```
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])
```

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



Exercise 2: When displaying the dimensions associated with the various layers of the network, we only see the information associated with eight blocks (plus some auxiliary transforms), even though the network has 11 layers. Where did the remaining three layers go?

A: 11개 레이어 중 3개의 레이어가 출력에 나타나지 않는 이유는 주로 활성화 함수, 풀링 레이어가 학습 가능한 파라미터가 없어 따로 표현되지 않거나 네트워크 출력에서 압축된 블록으로 표현하기 때문에 보이지 않는다.

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


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])
```

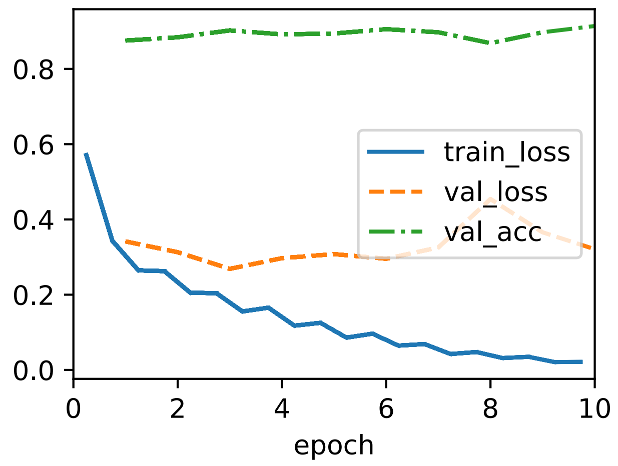
```
blk = Residual(6, use_1x1conv=True, strides=2)
blk(X).shape
```

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

```
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])
```

```
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)
```



Exercise 5: Why can’t we just increase the complexity of functions without bound, even if the function classes are nested?

A: 함수의 복잡성을 늘리면 우선 overfitting이 될 확률이 높다. 학습 데이터에 너무 지나치게 적합하되면 이러한 단점이 발생한다. 또한 학습 과정에서 필요한 계산 비용도 늘어 시간이 많이 증가되고 실용성이 떨어진다.