COES474_HW_1_2019170361_윤동현

[10/10] [10/10] HW 2: CNN

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

```
'Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l==1.0.3) (3.10)',
 'Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l==1.0.3) (2.2.3)',
 'Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l==1.0.3)
(2024.8.30)',
 'Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib==3.7.2-
>d2l==1.0.3) (1.16.0)',
 '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->d2l==1.0.3) (3.0.48)',
 '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)
```

7 Convoultional Neural Networks

7.1 From Fully Connected Layers to Convolutions

7.1.1 Invariance

7.1.2 Constraining the MLP

7.1.2.1 Translation Invariance

7.1.2.2 Locality

7.1.3 Convolutions

Convolution 연산

1차원

$$(f * g)(x) = \int f(z)g(x-z)dz$$
$$(f * g)(x) = \sum f(a)g(i-a)$$

2차원

$$(f * g)(i,j) = \sum_a \sum_b f(a,b)g(i-a, j-b)$$

7.1.4 Channels

일반적인 이미지는 rgb 3개의 채널로 구성되어있음 크기와 채널을 고려한 이미지 텐서의 크기는 $height \times width \times channels$이다.

각 텐서의 첫 번째 $height \times width$는 r. 즉, 빨간색의 정보를 가지고 있다. 이때 적절한 Filter를 적용하면 빨간색에 해당하는 이미지의 특성을 추출할 수 있다.

7.1.5 Summary and Discussion

Input 이미지와 filter (Kernel)의 Convolution 연산을 통해서 Feature Map을 추출할 수 있다.

예시로 Edge를 추출하기 위한 kernel

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$$

이런 Kernel을 적용하게 될 경우 수직 방향의 엣지가 검출될 수 있을 것이다.

7.2 Convolutions for Images

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

7.2.1 The Cross-Corrleation Operation

```
def corr2d(X, K):
  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

### white to black : 1
### black to white : -1
# 세로 테두리 검출
```

```
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)

# 가로로는 테두리가 없음
```

```
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

```
conv2d = nn.LazyConv2d(1, kernel_size=(1,2), bias=False)

X = X.reshape((1,1,6,8))
Y = Y.reshape((1,1,6,7))
lr = 3e-2

for i in range(10):
  Y_hat = conv2d(X)
  l = (Y_hat - Y) ** 2
  conv2d.zero_grad()
  l.sum().backward()

  conv2d.weight.data[:] -= lr * conv2d.weight.grad
  if(i+1)%2==0:
    print(f"epoch {i+1}, loss {l.sum():.3f}")
```

```
epoch 2, loss 3.126
epoch 4, loss 0.936
epoch 6, loss 0.325
epoch 8, loss 0.124
epoch 10, loss 0.049
```

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

```
tensor([[ 1.0164, -0.9712]])
```

7.2.5 Cross-Correlation and Convolution

Cross-Correlation과 Convolution의 차이는 Convolution

Convolution

$$(f*g)(x) = \int f(z)g(x-z)dz$$

Cross-Correlation과

$$(f*g)(x) = \int f(z)g(x+z)dz$$

이다.

즉, 커널을 뒤집느냐 뒤집지 않느냐의 차이이다.

필터를 학습하는 과정에서는 뒤집힘의 유무는 상관없이 동일한 결과나 나오게 될 것이다.

7.2.6 Feature Map and Receptive Field

Feature Map이란 Filter와 Input 이미지의 Convolution 연산을 통해 나온 Output이다. 이 Feature Map은 Filter에 의해 결정되며 Filter에 의해 추출된 해당 이미지의 특성을 의미한다.

Feature Map의 각 Cell은 Input Image와 Filter의 Convolution 연산에 의해 만들어진 Cell 들이다. 각 Cell에 영향을 주는 Input 이미지의 본래 작은 이미지가 Receptive Field가 된다. 이는 Filter의 크기와 동일하다.

7.2.7 Summary

적절한 Filter를 통해 Input 이미지로부터 Feature Map을 추출할 수 있다. 이때 Filter는 학습을 통해서 최적화할 수 있다.

7.3 Padding and Stride

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

7.3.1 Padding

일반적인 Convolution 연산을 반복하여 진행하다보면 크기가 점점 작아지는 것을 알 수 있을 것이다. 이를 조절하기 위해 Input Image에 Padding 즉, 겉에 테두리를 생성하여 Output의 크기를 원하는 크기로 적절히 조절할 수 있다.

```
def comp_conv2d(conv2d, X):
  X = X.reshape((1, 1) + X.shape)
  Y = conv2d(X)
  return Y.reshape(Y.shape[2:])

conv2d = nn.LazyConv2d(1, kernel_size=3, padding=1)
X = torch.rand(size=(8, 8))
comp_conv2d(conv2d, X).shape
```

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

```
conv2d = nn.LazyConv2d(1, kernel_size=(5, 3), padding=(2, 1))
comp_conv2d(conv2d, X).shape
```

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

7.3.2 Stride

Convolution 연산의 2차원 배열 위에 작은 2차원 배열 (Kernel)을 놓고 이동하는 그림을 그릴 수 있을 것이다. 일반적인 경우 한 칸 씩 이동하지만 2칸, 3칸, n칸 씩 이동을 할 수 있을 것이다. 이를 Stride라고 한다,.

이를 통해 Output의 크기를 조절할 수 있다.

```
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])
```

7.3.3 Summary and Discussion

해당 섹션에서는 Convolution의 Output을 적절한 크기로 조절하기 위한 기법으로 Padding과 Stride를 익혔다.

이는 Output 크기 조절 뿐만 아니라 연산량 조절 또한 가능하다.

7.4 Multiple Input and Multiple Output Channels

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

7.4.1 Multiple Input Channels

입력이 여러 채널로 구성되어 있는 경우 커널도 여러 채널로 구성할 수 있을 것이다.

각 채널별로 Convolution한 결과를 합하여 하나의 Cell을 구성할 수 있다.

Output의 Channel의 크기는 1이 된다.

```
def corr2d_multi_in(X, K):
  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 Channels

그러나 일반적인 CNN에서는 1개의 채널이 아닌 다중 채널의 Tensor가 필요하다.

```
def corr2d_multi_in_out(X, K):
  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.]]])
```

더블클릭 또는 Enter 키를 눌러 수정

7.4.3 1 X 1 Convolutional Layer

1 X 1과 Convolution 연산을 하게 되면 그대로 1 X 1이 나오게 될 것이다. 그러나 이 연산을 Channel 수를 조절하기 위한 테크닉으로 사용될 수 있다.

```
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))
  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
```

7.4.4 Discussion

이번 섹션에서는 이미지의 채널수를 조절하기 위한 방법을 배울 수 있었다.

1 X 1 Convolution의 경우 특히, 채널의 수를 조절하기 위해 좋은 테크닉이라 예상된다.

7.5 Pooling

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

7.5.1 Maximum Pooling and Average Pooling

Convolution 연산과 비슷하게 고정된 크기의 Window를 slide한다. 차이점은 Convolution과 다르게 Window가 값을 가지지 않는다.

Pooling 연산은 Feature Map의 크기를 줄이는 연산이다. 크기를 줄이기 위해서 삭제할 데이터와 유지할 데이터를 선정하여 Feature Map의 특성을 유지시키는하는 과정이다.

Max Pooling의 경우 Window에 들어온 값들 중에서 최댓값을 대푯값으로 설정한다. Avg Pooling 의 경우 Window에 들어온 값들의 평균을 대푯값으로 설정한다.

```
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

Convolution과 동일하게 Feature Map의 사이즈를 조절하기 위해 Padding과 Stride를 부여할 수 있다.

```
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)
pool2d(X)
```

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

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

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

7.5.3 Multiple Channels

여러 Channel의 경우, 각 Channel에 대해서 Pooling을 진행한다.

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

7.5.4 Summary

Feature Map의 사이즈를 조절하기 위해 Pooling을 위해 대푯값을 설정할 수 있다.

대표적으로 Max Pool과 Avg Pool이 존재한다.

Convolution 연산과 마찬가지로 Padding과 Stride를 설정할 수 있다.

일반적으로 Convolution Layer 이후 Pooling Layer가 존재하여 두 레이어를 하나의 Convolution Layer로 칭한다.

7.6 Convolutional Neural Networks (LeNet)

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

7.6.1 LeNet

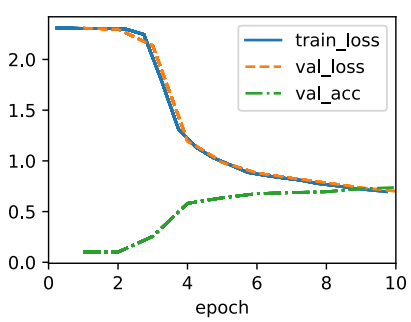
LeNet은 2개의 Convolution 레이어와 3개의 Fully Connected Layer로 구성되어 있다.

```
def init_cnn(module):
  if type(module) == nn.Linear or type(module) == nn.Conv2d:
    nn.init.xavier_uniform_(module.weight)

class LeNet(d2l.Classifier):
  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)
```



7.6.3 Summary

LeNet이라는 이미지 처리 CNN을 예시로 Net을 구성하는 법을 알 수 있었다.

8.2 Networks Using Blocks(VGG)

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

8.2.1 VGG Blocks

5 X 5로 Convolution 한 번 하는 것과 3 X 3로 Convolution을 2번 하는 것은 같은 픽셀을 처리하지만 파라미터의 수가 훨씬 적기때문에 효율적인 연산이 가능하다.

이런 Layer들을 Block으로 모듈화 시킨 것이다.

```
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

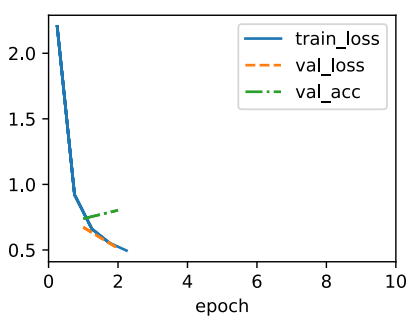
```
class VGG(d2l.Classifier):
    def __init__(self, arch, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        conv_blks = []
        for (num_convs, out_channels) in arch:
            conv_blks.append(vgg_block(num_convs, out_channels))
        self.net = 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])
```

코딩을 시작하거나 AI로 코드를 생성하세요.

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)
```



8.2.4 Summary

현대화된 CNN의 구조이다.

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
```

8.6.1 Function Classes

1. Non-Nested Function Classes 복잡한 네트워크가 단순한 네트워크가 할 수 있는 것을 항상 포함하지 않음

네트워크가 복잡해질수록 파라미터가 필요하고, 학습이 어려워질 수도 있습니다.

2. Nested Function Classes 단순한 네트워크에서 학습할 수 있는 모든 것을 더 복잡한 네트워크에서도 포함됨

복잡한 네트워크가 단순한 네트워크에서 할 수 있는 일을 모두 수행할 수 있으면서 복잡성으로 나은 성능을 낼 수 있음

8.6.2 Residual Blocks

Residual Block이란

VGG처럼 Block 형태로

Input 이미지 x를 block을 통해 x + f(x)로 원본 이미지도 결과에 포함된다.

이 연산을 위해 f(x)의 차원은 x와 동일하여야 한다. 이는 Padding과 Stride를 통해 크기를 조절할 수 있다.

```
class Residual(nn.Module):
  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

```
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])
```