COSE474-2024F: Deep Learning HW2

7.1. From Fully Connected Layers to Convolutions

Convolutional Neural Network

- effective at identifying images
- Convolutional Layer: use filters(kernels) to get features from images
- Activation Layer
- Pooling Layer: Decrease the dimension
- Fully Connected Layer: all neurons connected, final result for classification

7.1.1 Invariance

Translation Invariance

- respond equally to the same patch regardless of the LOCATION

Shallow layers: local small-range patches Deepter layers: long-range features

7.1.2. Constraining the MLP

MLP

- Multi-Layer Perceptron
- Input, Hidden, Output Layers

7.1.2.1. Translation Invariance

- CNN에서 filter를 사용한 패턴 매칭이 이미지 상의 location과 무관하기 때문에 hidden layer에 사용되는 parameter가 크게 감소함: ex: 10^12 -> 6 * 10^6

7.1.2.2. Locality

- locality와 filter를 이용하여 각 뉴런에 parameter를 곱하는 것이 아닌 filter단위로 곱하며 계산을 크게 줄인다.

7.1.4. Channels

Channel

- 여러 차원 구성 ex: rgb for each pixel of image
- H는 i,j위치와 채널 d에서의 합성곱 연산의 결과값
- 세모는 filter 범위
- V는 필터 (Weight)
- X는 input

$$\mathbf{H}_{i,j,d} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} \mathbf{V}_{a,b,c,d} \mathbf{X}_{i+a,j+b,c}$$

더블클릭 또는 Enter 키를 눌러 수정

7.1.5 Exercises

1. Assume that the size of the convolution kernel is Δ=0 . Show that in this case the convolution kernel implements an MLP independently for each set of channels. This leads to the Network in Network architectures (Lin et al., 2013).

Answer. 커널의 크기가 1*1이라면, 만약 1000*1000*3의 이미지 일때, 1000*3개의 가중치가 필요. 10^6 * (1000*3)의 연산 횟수가 필요, 주변 픽셀의 연산이 아닌 각 픽셀의 채널 정보를 독립적으로 처리함. 이러한 처리 방식은 MLP에서 각 입력 뉴런에 서로 다른 가중치를 곱해 값을 반환하는 것과 비슷하다.각 입력의 채널마다 가중치를 곱해서 출력 채널로 변환함. 단, local invariance 때문에 차이점이 존재.

7.2. Convolutions for Images

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

```
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
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.
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==
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==
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-consol
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
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
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) (
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
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) 
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
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
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) (0.1
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.
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)
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.
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.
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) (0.1
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (0.
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
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-
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==
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l=
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.
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=
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert->jupyter==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
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert->jupyter==1.0
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert->jupyter==1.0.0->d2
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->jupyt
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.10/dist-packages (from terminado>=0.8.3->notebook->jupyter==1.0.0->d2l=
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook->jupyter==1.0.0-
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert->jupyter==1.0.0->d2
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert->jupyter==1.0.0->d2l==1.0.3)
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->
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->ju
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbfor
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconve
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->j
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
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings->argon2-cffi->notebook->ju
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->n
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.
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2
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->noteb
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->note
```

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

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.]])
```

The kernel K only detects vertical edges.

```
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.]])
```

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
    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 0.959
epoch 4, loss 0.172
epoch 6, loss 0.033
epoch 8, loss 0.008
epoch 10, loss 0.002
```

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

```
tensor([[ 0.9913, -0.9988]])
```

7.2.5. Cross-Correlation and Convolution

- To be precise, cross-correlation and convolution make same results after Transpose.

7.2.6. Feature Map and Receptive Field

- 더 deepter layer로 갈수록 더 넓은 범위를 학습하게 됨.
- 2*2 -> 로 만들어진 층을 다시 2*2로 하면 한 칸에 16칸을 학습 가능.

7.2.7 Exercises

Construct an image X with diagonal edges.

What happens if you apply the kernel K in this section to it?

What happens if you transpose X?

What happens if you transpose K?

```
import torch

# 6x8 크기의 텐서를 1로 초기화
X = torch.ones((6, 8))

# 대각선 방향으로 0을 설정하여 대각선 엣지 생성
for i in range(min(X.size(0), X.size(1))):
    X[i, i] = 0  # 대각선 위치에 0을 설정하여 대각선 엣지 만들기

X
```

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

더블클릭 또는 Enter 키를 눌러 수정

```
corr2d(X, K)
```

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

Edge 탐지해냄.

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

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

결과값이 똑같음

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

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

Edge가 똑같이 탐지되나 탐지 방향이 바뀜.

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
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])
```

7.3.4. Exercise

Given the final code example in this section with kernel size (3,5) , padding (0,1) , and stride (3,4) , calculate the output shape to check if it is consistent with the experimental result.

Answer. with padding -> (8,10) with stride, height:3 and width:4 -> x-axis: 0~~4 then 4~~8 and y-axis: 0-2 then 3-5 (6-8 cannot) => it can make 4 blocks 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)
```

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

7.4.2. Multiple Output Channels

하나의 input에 다중 채널 output을 만들고, jointly하게 사용하여 더 풍부한 특징을 학습시킬 수 있다.

```
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. 1 x 1 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
```

7.4.4. Exercise

Qs.

Assume that we have two convolution kernels of size and , respectively (with no nonlinearity in between).

Prove that the result of the operation can be expressed by a single convolution.

What is the dimensionality of the equivalent single convolution?

Is the converse true, i.e., can you always decompose a convolution into two smaller ones?

Answer.

1. Using the commutative rule. (x * k1) * k2 == x * (k1*k2)
2. Assume k1 is n*m and k2 is p * q -> k1*k2 = n*q, result would be convolution of input by n*q filter
3. Maybe?, multiplication of linear equations can be decomposed.

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'):
    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

Pooling 단계에서도 Convolution과 마찬가지로 원하는 결과값의 형태를 만들기 위해 padding과 stride를 통해 조절할 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)
# 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

Convolution에서와 다르게 multi-input에 대해서 하나의 result로 sum되지 않고, multi -> multi로 pooling이 된다.

```
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 Networks (LeNet)

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

7.6.1. LeNet

At a high level, LeNet (LeNet-5) consists of two parts: (i) a convolutional encoder consisting of two convolutional layers; and (ii) a dense block consisting of three fully connected layers.

Dense Block

- 입력값 + 이전 층들의 출력값 모두를 입력값으로 받는다.
- 입력값의 형태를 맞추기 위해 평탄화 작업이 필요하다.

```
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))
```

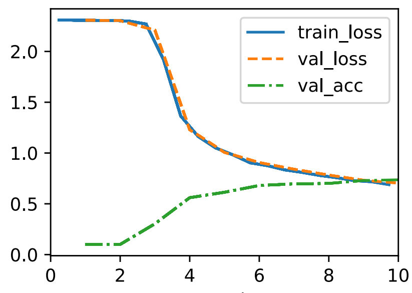
```
Conv2d output shape:     torch.Size([1, 6, 28, 28])
Sigmoid output shape:    torch.Size([1, 6, 28, 28])
AvgPool2d output shape:  torch.Size([1, 6, 14, 14])
Conv2d output shape:     torch.Size([1, 16, 10, 10])
Sigmoid output shape:    torch.Size([1, 16, 10, 10])
AvgPool2d output shape:  torch.Size([1, 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

Train_loss

- train 과정에서의 손실값
- epoch이 진행됨에 따라 model parameter가 학습되어 loss가 줄어듬 Val_loss
- validation과정에서의 손실값
- train_loss와 비슷하게 감소한다는 뜻은 학습된 모델이 validation 데이터에 대해서도 잘 작동하고 있음을 의미한다. Val_accu
- validation과정에서 학습된 모델이 validation데이터에 대해서 얼마나 잘 예측하는지를 보여준다.

```
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. Exercises

Let's modernize LeNet. Implement and test the following changes:

Replace average pooling with max-pooling.

Replace the softmax layer with ReLU.

Avg.pooling -> Max.pooling

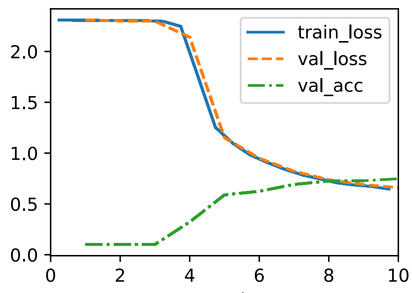
- 눈에 띄는 특징에만 집중하여 세부적인 특징들이 무시될 수 있음.
- avg에서는 중요한 특징이 덜 부각될 수 있음.

```
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.MaxPool2d(kernel_size=2, stride=2),  # Max Pooling으로 변경
            nn.LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
            nn.MaxPool2d(kernel_size=2, stride=2),  # Max Pooling으로 변경
            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])
MaxPool2d 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])
MaxPool2d 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)
```



Softmax -> ReLU ??? sigmoid를 ReLU로 바꿔보았다.

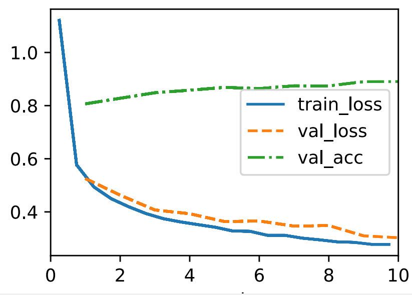
- 더 이상 모델이 0~1사이의 값 나타내지 않고, 점수 또는 활성화 값을 나타낸다. => 네트워그가 더 강한 비선형성을 학습할 수 있다.
- 양수에 대해서 기울기 소실이 적어서 back propagation에 더 유리하다.
- 깊은 층에서 더 학습이 잘 이루어짐. 또한, 학습속도가 빨라짐.
- 더 이상 확률로 해석 불가능.
- 이진 분류에 적합하지 않게됨.
- DeadReLU 발생가능.
- 활성화 값(뉴런의 출력값) 이 너무 커져 학습과정에서 문제가 발생할 수 있다.

```
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.ReLU(),  # Sigmoid -> ReLU
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.LazyConv2d(16, kernel_size=5), nn.ReLU(),  # Sigmoid -> ReLU
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.LazyLinear(120), nn.ReLU(),  # Sigmoid -> ReLU
            nn.LazyLinear(84), nn.ReLU(),   # Sigmoid -> ReLU
            nn.LazyLinear(num_classes))     # 출력층에서는 활성화 함수가 필요하지 않음
```

```
@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])
ReLU 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])
ReLU 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])
ReLU output shape:       torch.Size([1, 120])
Linear output shape:     torch.Size([1, 84])
ReLU 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)
```



8.2. Networks Using Blocks (VGG)

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

8.2.1. VGG Blocks

Back to VGG: a VGG block consists of a sequence of convolutions with 3 x 3 kernels with padding of 1 (keeping height and width) followed by a 2 x 2 max-pooling layer with stride of 2 (halving height and width after each block).

기존 CNN이 output 개수가 너무 rapid하게 줄어들어 deep network를 만들지 못하는 한계가 있었음. => golden standard : kernel size of 3x3, successive application of 3x3 kernel and one 5x5 kernel results touches same pixels, while 25 * c^2 parameters compared to three 3x3 (3 * 9 c^2). = 파라미터는 적게쓰면서 깊게 network 형성 가능.

```
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

Series of VGG blocks (successive 3 x 3 convolutions and 2 x 2 pooling) : 여러 layer를 block으로 묶는 방식

- 파라미터 수는 유지하며 network를 깊게 가져가며 계산 효율성을 높이고, 성능을 극대화함.
- 그 외로, 출력 채널 수를 blcok마다 2배씩 늘려나가 network가 깊어질수록 더 많은 환경에 대해 학습할 수 있도록 한다.

```
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])
```

8.2.3. Training

너무 큰 작업이라 colab 에서 15분 돌아갔는데 안 돌아감...

```
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)
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ../data/FashionMNIST/raw/train-images-idx3-u
100%|██████████| 26421880/26421880 [00:01<00:00, 19871359.04it/s]
Extracting ../data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw/train-labels-idx1-u
100%|██████████| 29515/29515 [00:00<00:00, 300871.74it/s]
Extracting ../data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ../data/FashionMNIST/raw/t10k-images-idx3-uby
100%|██████████| 4422102/4422102 [00:00<00:00, 5617301.95it/s]
Extracting ../data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw/t10k-labels-idx1-uby
100%|██████████| 5148/5148 [00:00<00:00, 4977472.80it/s]Extracting ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw


/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total
  warnings.warn(_create_warning_msg(
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-6-8d3dfa6be93b> in <cell line: 5>()
      3 data = d2l.FashionMNIST(batch_size=128, resize=(224, 224))
      4 model.apply_init([next(iter(data.get_dataloader(True)))[0]], d2l.init_cnn)
----> 5 trainer.fit(model, data)

                              15 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/conv.py in _conv_forward(self, input, weight, bias)
    452                             weight, bias, self.stride,
    453                             _pair(0), self.dilation, self.groups)
--> 454         return F.conv2d(input, weight, bias, self.stride,
    455                         self.padding, self.dilation, self.groups)
    456 

KeyboardInterrupt: 
```

8.2.4. Exercises

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?

Answer.

The reamining 3layers are fully connected layers. Using the fully connected layers, the input then can be classfied into groups. While convolutional layers are described in spatial dimensions like 3D, fully connected layers are described in 1D after flattening.

8.6. Residual Networks (ResNet) and ResNeXt (8.6.1 ~ 8.6.4)

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

8.6.1. Function Classes

- 더 깊게 network를 쌓는다고 해서 성능이 향상되는 것이 아님
- 이러한 문제를 해결하고자 ResNet (Residual Network) 소개됨. = 더 깊은 네트워크에서 정체된 성능을 끌어올림.
- 추가 layer를 설정할 때 Identity function (항등 함수)로 동작하게 하여 추가된 layer가 원래 입력을 그대로 통과시키거나, 학습을 통해 더 발전할 수 있음. => 결과론적으로 layer를 늘려도 성능적으로 떨어지지 않으며, 더 복잡한 모델 설계가 가능해짐.

8.6.2. Residual Blocks

- two 3 x 3 convolutional 'layers' that produce same number of channels (항등 함수 성질을 살려야하기 때문)

- 순서
    1. convolution layer
    2. batch normalization layer
    3. ReLU activation function
    4. Convolution layer
    5. batch norm layer
    6. ReLU

  !! 1~5를 skip하고 input x를 6번 ReLU전에 집어넣을 수 있음 (1 x 1 convolution을 통해 data 형태만 맞춰주면) !!

```
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

초반부

- 7 x 7 convolution layer with stride 2
- activation (ReLU)
- Max Pooling 3 x 3 with stride 2

중반부

- 2 successive Residual blocks

후반부

- global avg. pooling layer

```
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 Discussion

Qs.

Why is there so big gap between Train_loss and Valid_loss in Res Model?

Answer.

- Training is very senstive that it might result in overfitting unless. It has very complex network that it might be overfitted on training data set.

Discussions & Exercises

7.1.5 Exercises

1. Assume that the size of the convolution kernel is $\Delta=0$. Show that in this case the convolution kernel implements an MLP independently for each set of channels. This leads to the Network in Network architectures (Lin et al., 2013).

Answer. 커널의 크기가 1*1이라면, 만약 1000*1000*3의 이미지 일때, 1000*3개의 가중치가 필요. 10^6 * (1000*3)의 연산 횟수가 필요, 주변 픽셀의 연산이 아닌 각 픽셀의 채널 정보를 독립적으로 처리함. 이러한 처리 방식은 MLP에서 각 입력 뉴런에 서로 다른 가중치를 곱해 값을 반환하는 것과 비슷하다.각 입력의 채널마다 가중치를 곱해서 출력 채널로 변환함. 단, local invariance 때문에 차이점이 존재.

7.2.7 Exercises

Construct an image X with diagonal edges.

What happens if you apply the kernel K in this section to it?

What happens if you transpose X?

What happens if you transpose K?

```
import torch

# 6x8 크기의 텐서를 1로 초기화
X = torch.ones((6, 8))

# 대각선 방향으로 0을 설정하여 대각선 엣지 생성
for i in range(min(X.size(0), X.size(1))):
    X[i, i] = 0  # 대각선 위치에 0을 설정하여 대각선 엣지 만들기

X
```

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

더블클릭 또는 Enter 키를 눌러 수정

```
corr2d(X, K)
```

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

Edge 탐지해냄.

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

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

결과값이 똑같음

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

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

Edge가 똑같이 탐지되나 탐지 방향이 바뀜.

7.3.4. Exercise

Qs.

Given the final code example in this section with kernel size (3,5) , padding (0,1) , and stride (3,4) , calculate the output shape to check if it is consistent with the experimental result.

Answer.

with padding -> (8,10) with stride, height:3 and width:4 -> x-axis: 0~~4 then 4~~8 and y-axis: 0-2 then 3-5 (6-8 cannot) => it can make 4 blocks 2*2

7.4.4. Exercise

Qs.

Assume that we have two convolution kernels of size and , respectively (with no nonlinearity in between).

Prove that the result of the operation can be expressed by a single convolution.

What is the dimensionality of the equivalent single convolution?

Is the converse true, i.e., can you always decompose a convolution into two smaller ones?

Answer.

1. Using the commutative rule. (x * k1) * k2 == x * (k1*k2)
2. Assume k1 is n*m and k2 is p * q -> k1*k2 = n*q, result would be convolution of input by n*q filter
3. Maybe?, multiplication of linear equations can be decomposed.

7.6.3. Exercises

Let’s modernize LeNet. Implement and test the following changes:

Replace average pooling with max-pooling.

Replace the softmax layer with ReLU.

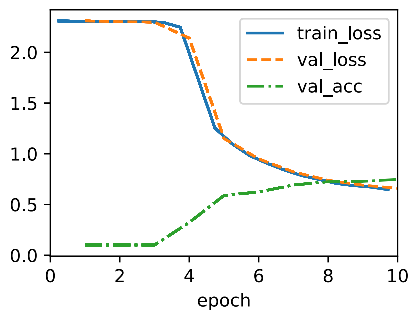
Avg.pooling -> Max.pooling

- 눈에 띄는 특징에만 집중하여 세부적인 특징들이 무시될 수 있음.
- avg에서는 중요한 특징이 덜 부각될 수 있음.

```
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.MaxPool2d(kernel_size=2, stride=2),  # Max Pooling으로 변경
            nn.LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
            nn.MaxPool2d(kernel_size=2, stride=2),  # Max Pooling으로 변경
            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])
MaxPool2d 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])
MaxPool2d 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)
```



Softmax -> ReLU ??? sigmoid를 ReLU로 바꿔보았다.

- 더 이상 모델이 0~1사이의 값 나타내지 않고, 점수 또는 활성화 값을 나타낸다. => 네트워그가 더 강한 비선형성을 학습할 수 있다.
- 양수에 대해서 기울기 소실이 적어서 back propagation에 더 유리하다.
- 깊은 층에서 더 학습이 잘 이루어짐. 또한, 학습속도가 빨라짐.
- 더 이상 확률로 해석 불가능.
- 이진 분류에 적합하지 않게됨.
- DeadReLU 발생가능.
- 활성화 값(뉴런의 출력값) 이 너무 커져 학습과정에서 문제가 발생할 수 있다.

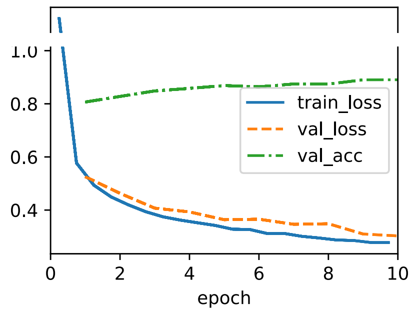
```
def init_cnn(module):
    """Initialize weights for CNNs."""
    if type(module) == nn.Linear or type(module) == nn.Conv2d:
        nn.init.xavier_uniform_(module.weight)

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.ReLU(),  # Sigmoid -> ReLU
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.LazyConv2d(16, kernel_size=5), nn.ReLU(),  # Sigmoid -> ReLU
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.LazyLinear(120), nn.ReLU(),  # Sigmoid -> ReLU
            nn.LazyLinear(84), nn.ReLU(),   # Sigmoid -> ReLU
            nn.LazyLinear(num_classes))     # 출력층에서는 활성화 함수가 필요하지 않음
```

```
@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])
ReLU 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])
ReLU 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])
ReLU output shape:       torch.Size([1, 120])
Linear output shape:     torch.Size([1, 84])
ReLU 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)
```



8.2.4. Exercises

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?

Answer.

The reamining 3layers are fully connected layers. Using the fully connected layers, the input then can be classfied into groups. While convolutional layers are described in spatial dimensions like 3D, fully connected layers are described in 1D after flattening.

8.6.5 Discussion

Qs.

Why is there so big gap between Train_loss and Valid_loss in Res Model?

Answer.

- Training is very senstive that it might result in overfitting unless. It has very complex network that it might be overfitted on training data set.

더블클릭 또는 Enter 키를 눌러 수정

더블클릭 또는 Enter 키를 눌러 수정