COSE474-2024F: Deep Learning HW2

> Excercises & Discussions은 챕터별 코드 실습 이후 한 번에 정리해두었습니다.

0.1 Installation

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

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from py
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/l
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ju
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconv
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbco
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-package
```

```
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/pytho
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from c
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (fro
```

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

7.1. From Fully Connected Layers to Convolutions

> 챕터에 코드가 없는 관계로 latex 형식 공식을 옮겨 적겠습니다

$$
\begin{aligned}
[H]_{i,j} &= [U]_{i,j} + \sum_k \sum_l [W]_{i,j,k,l} [X]_{k,l} \\
&= [U]_{i,j} + \sum_a \sum_b [V]_{i,j,a,b} [X]_{i+a,j+b}
\end{aligned}
$$

$$
[H]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}
$$

$$
[H]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [V]_{a,b} [X]_{i+a,j+b}
$$

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

$$
(f * g)(i) = \sum_a f(a) g(i-a)
$$

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

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

7.2. Convolutions for Images

```
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.]])
```

```
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.]])
```

```
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 10.804
epoch 4, loss 3.642
epoch 6, loss 1.361
epoch 8, loss 0.535
epoch 10, loss 0.216
```

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

```
tensor([[ 1.0383, -0.9430]])
```

7.3. Padding and Stride

```
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])
```

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

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

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

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

7.4. Multiple Input and Multiple Output Channels

```
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
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):
  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))
  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
```

7.5. 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.]])
```

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

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

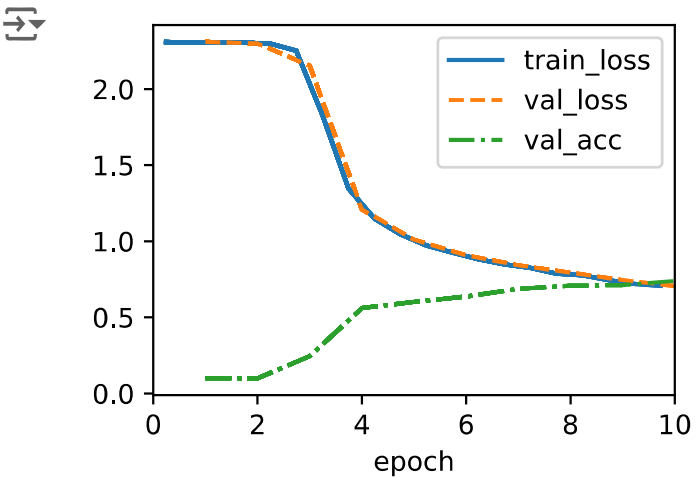
7.6. Convolutional Neural Networks (LeNet)

```
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)
```



8.2. Networks Using Blocks (VGG)

```
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))
```

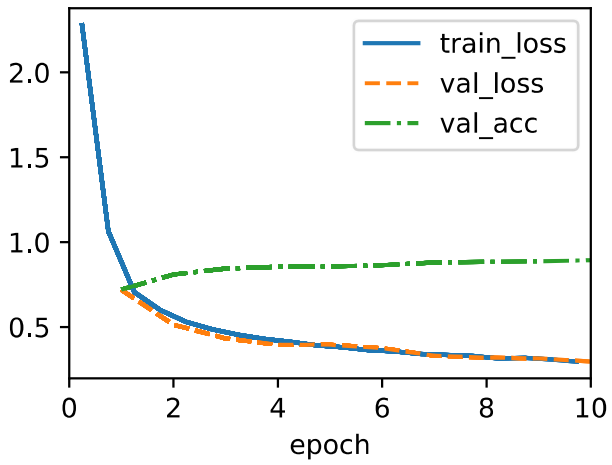
```
Sequential output shape:         torch.Size([1, 64, 112, 112])
Sequential output shape:         torch.Size([1, 128, 56, 56])
Sequential output shape:         torch.Size([1, 256, 28, 28])
Sequential output shape:         torch.Size([1, 512, 14, 14])
Sequential output shape:         torch.Size([1, 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])
```

```
torch.cuda.is_available()
```

```
True
```

> 연산이 d2l의 코드를 그대로 적용할 시 너무 느려서 임의로 gpu에서 연산을 하도록 수정하였습니다.

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



8.6. Residual Networks (ResNet) and ResNeXt

```
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
```

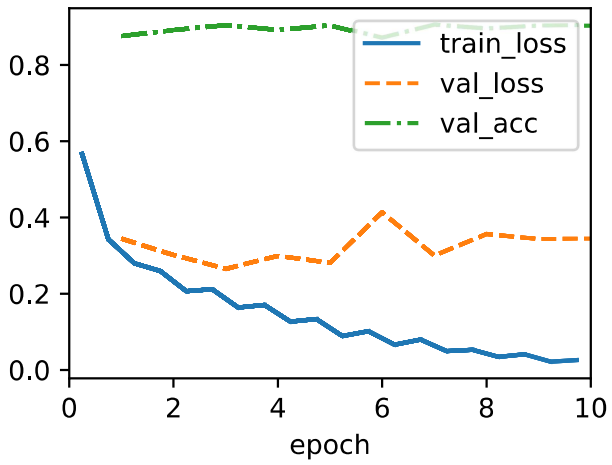
```
torch.Size([4, 6, 3, 3])
```

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


@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.LazyLir
  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])
```

> 위와 마찬가지로 너무 연산이 느려서 gpu에서 연산이 가능하도록 임의로 수정하였습니다

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



Excercises & Discussions

7.1

> Audio data is often represented as a one-dimensional sequence.
> When might you want to impose locality and translation invariance for audio?
>
> 오디오 데이터에서 locality와 translation invariance가 중요한 이유는 오디오의 특징이 시간의 흐름에 따라 일관되게 나타날 수 있기 때문이다. 예를 들어, 특정 주파수의 패턴이 음성 신호에서 반복적으로 나타날 수 있다. 이런 경우에 locality는 해당 패턴이 시간적으로 인접한 부분에서 나타난다는 것을 의미하고, translation invariance는 그 패턴이 시계열의 어디에 나타나든 같은 의미를 지닌다는 것을 뜻한다.

> Do you think that convolutional layers might also be applicable for text data?
> Which problems might you encounter with language?
>
> 위와 같이, 시계열 데이터인 텍스트는 1차원 convolutional으로 유의미한 데이터를 추출할 수 있다. 그러나, 텍스트는 문맥이 존재해 다른 문장들(설령 멀리 떨어져 있는 문장이라 할지라도)에 영향을 받고, 단어의 순서에 큰 영향을 받을 수 있어서 제대로 정보가 추출되지 않을 수 있다는 문제가 존재한다.

7.2

> Construct an image X with diagonal edges. What happens if you apply the kernel K in this section to it?

```
imagex = 2 * torch.eye(5)
K1 = torch.tensor([[0.0, 1.0], [2.0, 3.0]])
K2 = torch.tensor([[1.0, -1.0]])
print(corr2d(imagex, K1))
print(corr2d(imagex, K2))
```

```
tensor([[6., 4., 0., 0.],
        [2., 6., 4., 0.],
        [0., 2., 6., 4.],
        [0., 0., 2., 6.]])
tensor([[ 2.,  0.,  0.,  0.],
        [-2.,  2.,  0.,  0.],
        [ 0., -2.,  2.,  0.],
        [ 0.,  0., -2.,  2.],
        [ 0.,  0.,  0., -2.]])
```

> What happens if you transpose X?

```
print(corr2d(imagex.T, K1))
print(corr2d(imagex.T, K2))
```

```
tensor([[6., 4., 0., 0.],
        [2., 6., 4., 0.],
        [0., 2., 6., 4.],
        [0., 0., 2., 6.]])
tensor([[ 2.,  0.,  0.,  0.],
        [-2.,  2.,  0.,  0.],
        [ 0., -2.,  2.,  0.],
        [ 0.,  0., -2.,  2.],
        [ 0.,  0.,  0., -2.]])
```

> What happens if you transpose K?

```
print(corr2d(imagex, K1.T))
print(corr2d(imagex, K2.T))
```

```
tensor([[6., 2., 0., 0.],
        [4., 6., 2., 0.],
        [0., 4., 6., 2.],
        [0., 0., 4., 6.]])
tensor([[ 2., -2.,  0.,  0.,  0.],
        [ 0.,  2., -2.,  0.,  0.],
        [ 0.,  0.,  2., -2.,  0.],
        [ 0.,  0.,  0.,  2., -2.]])
```

7.3

> 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.

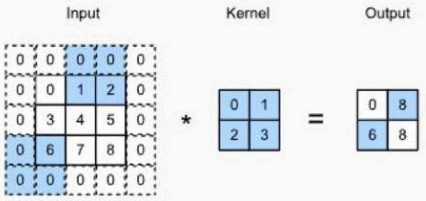


Fig. 7.3.3 Cross-correlation with strides of 3 and 2 for height and width, respectively.

In general, when the stride for the height is s_h and the stride for the width is s_w, the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor. \quad (7.3.2)$$

8 3 0 3 3 8 5 1 4 4 = 8/3 × 8/4 ⇒ 2×2

```
X = torch.rand(size=(8, 8))
conv2d = nn.LazyConv2d(1, kernel_size=(3, 5), padding=(0, 1), stride=(3, 4))
comp_conv2d(conv2d, X).shape
```

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

> For audio signals, what does a stride of 2 correspond to?
>
> 1차원 시계열 데이터가 stride 2를 가진다는 건 다운샘플링이 일어난다는 것이다. 헤르츠가 절반으로 줄어 음질이 감소하며, 사소한 정보들이 사라질 것이다.

7.4

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

```
def corr2d_multi_in(X, K):
  return sum(d2l.corr2d(x, k) for x, k in zip(X, K))

def corr2d_multi_in_out(X, K):
  return torch.stack([corr2d_multi_in(X, k) for k in K], 0)

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
Y1 == Y2
```

```
tensor([[[True, True, True],
         [True, True, True],
         [True, True, True]],

        [[True, True, True],
         [True, True, True],
         [True, True, True]]])
```

> 같은 값을 가진다. 이유는, 당연하게도 K가 1X1의 형식을 따르기 때문.

> Assume that we have a c X c matrix. How much faster is it to multiply with a block-diagonal matrix if the matrix is broken up into b blocks?
>
> 내가 생각하기에는 $b^{0.5}$배 빨라질 것 같다. 그 이유는, b개로 분할한다는 건 한 행렬의 길이가 c에서 $c/b^{0.5}$가 되었다고 (rough하게)고려할 수 있으며, 2차원 행렬의 행렬곱 시간복잡도는 $O(x^3)$(으로 알고 있)으므로, c^3 -> $c^3/b^{1.5}$가 되며 그런 block이 b개 존재하므로 $b^{0.5}$배 빨라지는 것이다.

7.5

> Implement average pooling through a convolution.
>
> average pooling은 단순히 pooling하고자 하는 영역과 동일한 크기의 모든 원소가 1인 커널을 사용하여 진행하는 Cross-Correlation과 같을 것이다(결과를 pooling 영역의 크기로 나누는 과정을 포함한)

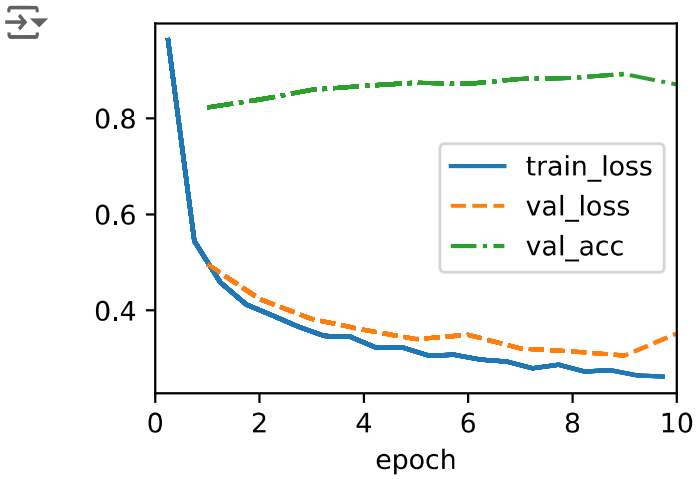
> Prove that max-pooling cannot be implemented through a convolution alone.
>
> max pooling은 pooling하고자 하는 영역과 동일한 크기의, 그러나 그 영역에서 가장 큰 값이 존재하는 구간의 값만 1이며 다른 구간은 모두 0인 값을 커널을 사용하여 진

> 행하는 Cross-Correlation과 같다. 그러나, 1의 값을 가지는 구간은 매번 입력 데이터에 따라 바뀌게 되며, 고정된 커널 값이 존재하지 않기에, 단순한 행렬곱으로는 표현할 수 없다.

7.6

> Let's modernize LeNet. Implement and test the following changes: Replace average pooling with max-pooling, Replace the softmax layer with ReLU.

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

class LeNet_changed(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.ReLU(),
      nn.MaxPool2d(kernel_size=2, stride=2),
      nn.LazyConv2d(16, kernel_size=5), nn.ReLU(),
      nn.MaxPool2d(kernel_size=2, stride=2),
      nn.Flatten(),
      nn.LazyLinear(120), nn.ReLU(),
      nn.LazyLinear(84), nn.ReLU(),
      nn.LazyLinear(num_classes)
    )

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



무척 기존에 비해 나아진 결과를 보여 준다.

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

different images -> 깊은 층에서는 클래스에 따른 고유한 특징이 감지되어 활성화 값이 크게 달라지며, 최종적으로 서로 다른 클래스 점수를 출력할 것이다
random noise -> 모든 층에서 활성화 값이 무작위적이고 구조화되지 않으며, 결과적으로 신뢰할 수 없는 예측이 출력될 것이다.

8.2

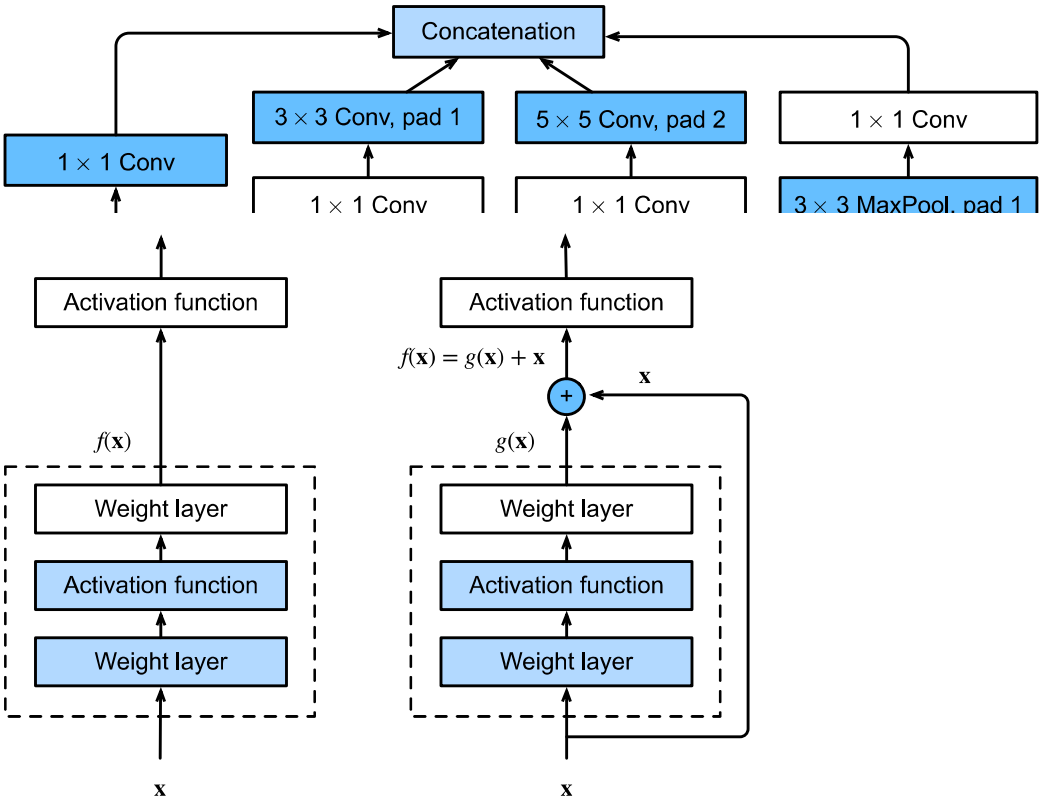
Compared with AlexNet, VGG is much slower in terms of computation, and it also needs more GPU memory. Compare the number of parameters needed for AlexNet and VGG.

AlexNet은 총 8개의 레이어(5개의 컨볼루션 레이어, 3개의 Fully Connected 레이어)로 구성되어 있고, 전체 파라미터 수는 약 6천만 개라고 한다. 반면에, VGG16의 경우 전체 파라미터 수는 약 1억 3천 8백만 개로, AlexNet에 비해 훨씬 많다고 한다. 파라미터 수가 많다는 것은 학습 시 필요한 계산량이 증가하고, 더 많은 GPU 메모리가 필요하다는 것을 의미하기에, 이로 인해 VGG는 AlexNet에 비해 학습 및 추론 속도가 느려지고, 더 큰 GPU 메모리를 사용할 것이다.

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?

나머지 3개의 레이어는 Fully Connected layer로, 네트워크의 최종 분류를 담당하지만, 네트워크 시각화나 구조 설명에서는 생략된다.

8.6



> What are the major differences between the Inception block in Fig. 8.4.1(위의 이미지) and the residual block(밑의 이미지)? How do they compare in terms of computation, accuracy, and the classes of functions they can describe?
>
> 주요 차이점: Inception block에서는 병렬적으로 여러 변환값을 채널 차원에서 합치며, residual block에서는 몇 개의 레이어를 통과한 출력에 입력을 그대로 더한다. accuracy 측면에서, 입력과 비선형적으로 차이가 날 수 있는 고차원적 연산이 진행된 깊은 네트워크 시점에서 입력값을 그대로 연산하면서 한번 체크하는 구간을 만든다는 것에 의미가 있다. 다시 말해, 깊은 네트워크에서 더욱 정확하다.