코딩을 시작하거나 AI로 코드를 생성하세요.

DeepLearningAssignment2

0.1.Installation

```
!pip install d2l
```

```
Requirement already satisfied: d2l in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: jupyter==1.0.0 in /usr/local/lib/python3.10/
Requirement already satisfied: numpy==1.23.5 in /usr/local/lib/python3.10/d
Requirement already satisfied: matplotlib==3.7.2 in /usr/local/lib/python3.
Requirement already satisfied: matplotlib-inline==0.1.6 in /usr/local/lib/p
Requirement already satisfied: requests==2.31.0 in /usr/local/lib/python3.1
Requirement already satisfied: pandas==2.0.3 in /usr/local/lib/python3.10/d
Requirement already satisfied: scipy==1.10.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: qtconsole in /usr/local/lib/python3.10/dist-
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in /usr/local/lib/pyth
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
Requirement already satisfied: traitlets in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/p
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.1
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/pytho
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/d
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dis
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.1
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.1
```

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

7.1.From Fully Connected Layers to Convolutions

7.2.Convolutions for Images

```python
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.]])
```

Kernal K only detects vertical Edges!

```
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 1.714
epoch 4, loss 0.294
epoch 6, loss 0.052
epoch 8, loss 0.010
epoch 10, loss 0.002
```

더블클릭 또는 Enter 키를 눌러 수정

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

```
tensor([[ 0.9904, -0.9961]])
```

learned kernel tensor is close to the kernel tensor K.

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))
print(X)
print(comp_conv2d(conv2d, X))
comp_conv2d(conv2d, X).shape
```

```
tensor([[0.1984, 0.9992, 0.1228, 0.3939, 0.3344, 0.0318, 0.4032, 0.6669],
        [0.7731, 0.9311, 0.8539, 0.8132, 0.0864, 0.6334, 0.1031, 0.2290],
        [0.9472, 0.1253, 0.0837, 0.8574, 0.7952, 0.9861, 0.2615, 0.8785],
        [0.3074, 0.6434, 0.1558, 0.8075, 0.0695, 0.8649, 0.2298, 0.7232],
        [0.4754, 0.8081, 0.7717, 0.4876, 0.5995, 0.8315, 0.8118, 0.1277],
        [0.3213, 0.2945, 0.9800, 0.2948, 0.5631, 0.3483, 0.0778, 0.0732],
        [0.7381, 0.3695, 0.6687, 0.3519, 0.8901, 0.1738, 0.2934, 0.4053],
        [0.5241, 0.6170, 0.7517, 0.0782, 0.8749, 0.8771, 0.5941, 0.4258]])
tensor([[ 0.0285,  0.3734,  0.3200,  0.2331,  0.3226,  0.0457,  0.0369,  0.
        [-0.0775, -0.2024,  0.2204,  0.1777,  0.3506,  0.4708,  0.3703,  0.
        [ 0.0735,  0.3133,  0.3567,  0.0382,  0.2949, -0.0772,  0.2700,  0.
        [-0.0861,  0.5962,  0.2353,  0.2680,  0.3163,  0.4221,  0.4313,  0.
        [-0.1023,  0.0214,  0.2884,  0.0540,  0.2676, -0.2302,  0.3151, -0.
        [ 0.1208,  0.1493,  0.2795,  0.3827,  0.2503,  0.3838,  0.3247,  0.
        [ 0.0550,  0.2597,  0.0686,  0.3886,  0.2339,  0.4377,  0.3708,  0.
        [-0.2445, -0.0024, -0.0225, -0.0057, -0.2981,  0.0364, -0.1316,  0.
       grad_fn=<ViewBackward0>)
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.MultipleInput 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, 6.0], [7.0, 8.0, 9.0]]])
K = torch.tensor([[[0.0, 1.0], [2.0, 3.0]], [[1.0, 2.0], [3.0, 4.0]]])

corr2d_multi_in(X, K)
```

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

```
def corr2d_multi_in_out(X, K):
    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)
print(Y1)
print(Y2)
assert float(torch.abs(Y1 - Y2).sum()) < 1e-6
```

```
tensor([[[-1.9662, -0.4338, -3.4396],
         [-2.2989, -0.4998, -0.4554],
         [ 1.0167, -1.6806,  0.1215]],

        [[ 2.6687,  0.8972, -1.9620],
         [-2.7739, -0.9719, -1.1503],
         [-2.8555,  2.6781, -5.5760]]])
tensor([[[-1.9662, -0.4338, -3.4396],
         [-2.2989, -0.4998, -0.4554],
         [ 1.0167, -1.6806,  0.1215]],

        [[ 2.6687,  0.8972, -1.9620],
         [-2.7739, -0.9719, -1.1503],
         [-2.8555,  2.6781, -5.5760]]])
```

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.]]]])
```

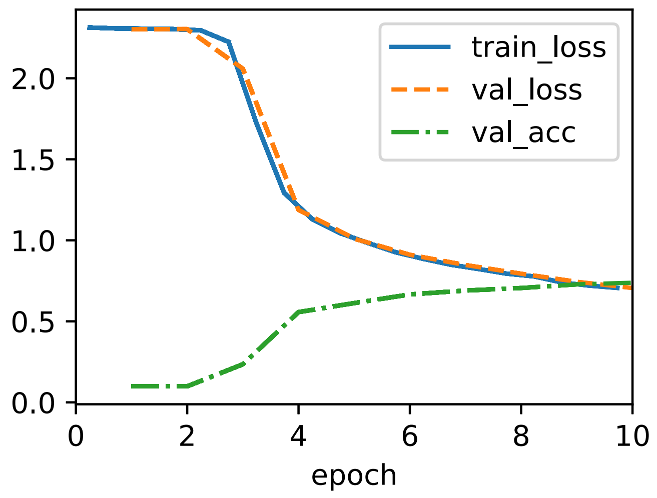
7.6.Convolutional Neural Networks(LeNet)

```
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)
```

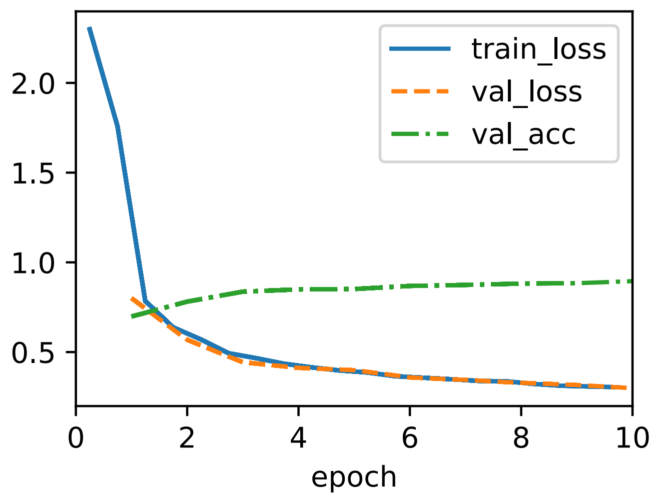


8.2.Networks Using Blocks (VGG)

```python
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])
```

```
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)
```



8.6.Residual Networks (ResNEt) and ResNeXt

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

```
class Residual(nn.Module):
    def __init__(self, num_channels, use_1x1conv=False, strides=1):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=strides)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.LazyConv2d(num_channels, kernel_size=1,
                                       stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.LazyBatchNorm2d()
        self.bn2 = nn.LazyBatchNorm2d()

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        Y += X
        return F.relu(Y)
```

```
blk = Residual(3)
X = torch.randn(4, 3, 6, 6)
blk(X).shape
```

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

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

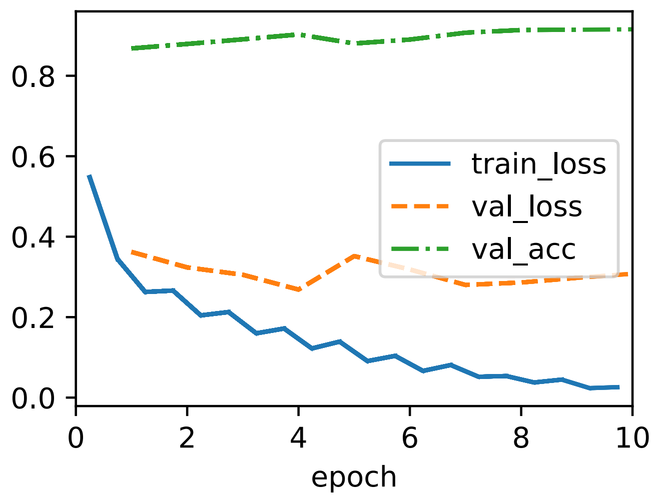
```
torch.Size([4, 6, 3, 3])
```

```
class ResNet(d2l.Classifier):
    def b1(self):
        return nn.Sequential(
            nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3),
            nn.LazyBatchNorm2d(), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
@d2l.add_to_class(ResNet)
def block(self, num_residuals, num_channels, first_block=False):
    blk = []
    for i in range(num_residuals):
        if i == 0 and not first_block:
            blk.append(Residual(num_channels, use_1x1conv=True, strides=2))
        else:
            blk.append(Residual(num_channels))
    return nn.Sequential(*blk)


@d2l.add_to_class(ResNet)
def __init__(self, arch, lr=0.1, num_classes=10):
    super(ResNet, self).__init__()
    self.save_hyperparameters()
    self.net = nn.Sequential(self.b1())
    for i, b in enumerate(arch):
        self.net.add_module(f'b{i+2}', self.block(*b, first_block=(i==0)))
    self.net.add_module('last', nn.Sequential(
        nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten(),
        nn.LazyLinear(num_classes)))
    self.net.apply(d2l.init_cnn)


class ResNet18(ResNet):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__(((2, 64), (2, 128), (2, 256), (2, 512)),
                       lr, num_classes)

ResNet18().layer_summary((1, 1, 96, 96))
```

```
Sequential output shape:         torch.Size([1, 64, 24, 24])
Sequential output shape:         torch.Size([1, 64, 24, 24])
Sequential output shape:         torch.Size([1, 128, 12, 12])
Sequential output shape:         torch.Size([1, 256, 6, 6])
Sequential output shape:         torch.Size([1, 512, 3, 3])
Sequential output shape:         torch.Size([1, 10])
```

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



```
class ResNeXtBlock(nn.Module):
    def __init__(self, num_channels, groups, bot_mul, use_1x1conv=False,
                 strides=1):
        super().__init__()
        bot_channels = int(round(num_channels * bot_mul))
        self.conv1 = nn.LazyConv2d(bot_channels, kernel_size=1, stride=1)
        self.conv2 = nn.LazyConv2d(bot_channels, kernel_size=3,
                                   stride=strides, padding=1,
                                   groups=bot_channels//groups)
        self.conv3 = nn.LazyConv2d(num_channels, kernel_size=1, stride=1)
        self.bn1 = nn.LazyBatchNorm2d()
        self.bn2 = nn.LazyBatchNorm2d()
        self.bn3 = nn.LazyBatchNorm2d()
        if use_1x1conv:
            self.conv4 = nn.LazyConv2d(num_channels, kernel_size=1,
                                       stride=strides)
            self.bn4 = nn.LazyBatchNorm2d()
        else:
            self.conv4 = None

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = F.relu(self.bn2(self.conv2(Y)))
        Y = self.bn3(self.conv3(Y))
        if self.conv4:
            X = self.bn4(self.conv4(X))
        return F.relu(Y + X)


blk = ResNeXtBlock(32, 16, 1)
X = torch.randn(4, 32, 96, 96)
blk(X).shape
```

```
torch.Size([4, 32, 96, 96])
```

Discussions & Exercises

7.1.2. Discussions

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

- $\mathbf{H}$: hidden representations
- $\mathbf{X}$: inputs

7.1.6.Exercises

What happens with convolution when an object is at the boundary of an image?

-> 경계 근처에 있다면 이웃 픽셀이 적기 때문에 합성곱 연산 결과에 영향을 미칠 수 있음. 해결하기 위한 방법:

- Zero Padding : 이미지 경계 주변에 0을 추가하여 필요로 하는 픽셀의 수를 만족하도록 함.
- Replicate Padding : 경계 픽셀 값을 그대로 복제하여 경계 바깥으로 확장하는 방법. 가장자리 픽셀 값을 "늘려" 필터가 경계 부분에서도 추가 정보를 사용할 수 있게 함.

7.2.8. Exercises

```
X = torch.zeros(10, 10)
for i in range(10):
    X[i, i] = 1
```

```
K = torch.tensor([[1.0, -1.0]])
Y = corr2d(X, K)
Y
```

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

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

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

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

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

7.3.Discussions

출력 크기 공식:

$$H_{\text{out}} = \frac{H_{\text{in}} - K_h + 2P_h}{S_h} + 1 \; W_{\text{out}} = \frac{W_{\text{in}} - K_w + 2P_w}{S_w} + 1$$

여기서,

- $(H_{\text{in}}, W_{\text{in}})$: 입력 이미지의 높이와 너비
- (K_h, K_w): 커널의 높이와 너비
- (P_h, P_w): 패딩의 높이와 너비
- (S_h, S_w): 스트라이드의 높이와 너비
- $(H_{\text{out}}, W_{\text{out}})$: 출력 특성 맵의 높이와 너비

7.4.Discussions

When the input data contains multiple channels, we need to construct a convolution kernel with the same number of input channels as the input data.

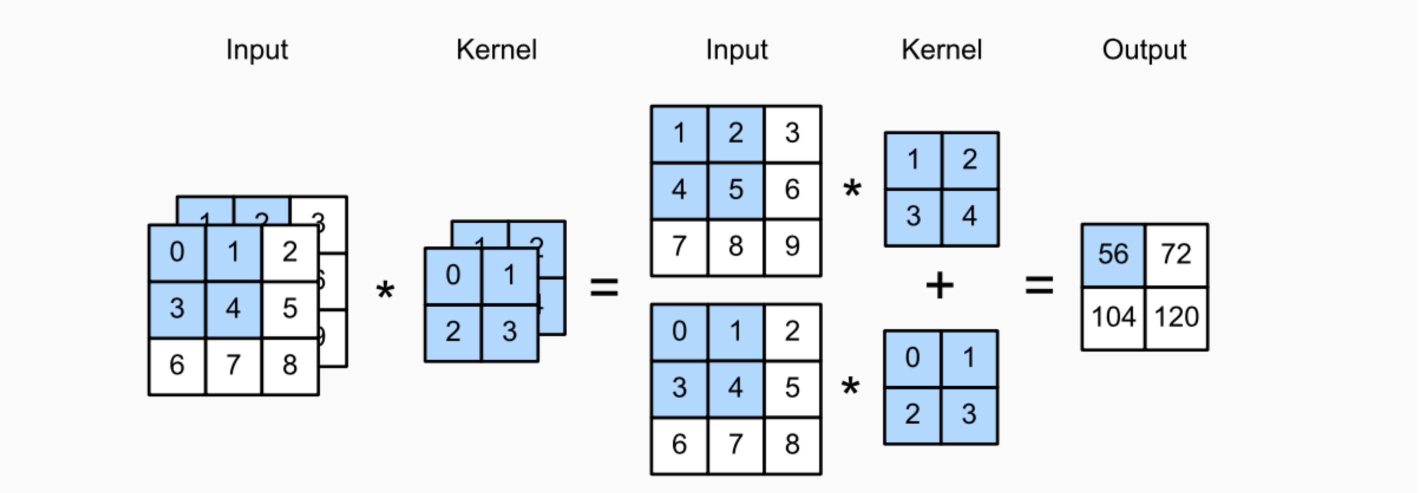


Fig. 7.4.1 Cross-correlation computation with two input channels.

7.5.Discussions

- Maximum Pooling : 각 픽셀의 가장 큰 수만 추출하는 방법
- Average Pooling : 각 픽셀 값의 평균을 대표하는 수로 하는 방법.

7.5.Exercises

```
X = torch.arange(16, dtype=torch.float32).reshape((1, 1, 4, 4))
X_new_concat = torch.cat((X, X + 5, X - 3), dim=1)
print(X_new_concat)
print('-------------')
pool2d = nn.MaxPool2d(3, padding=1, stride=4)
output_new = pool2d(X_new_concat)
print(output_new)
```

```
tensor([[[[ 0.,  1.,  2.,  3.],
          [ 4.,  5.,  6.,  7.],
          [ 8.,  9., 10., 11.],
          [12., 13., 14., 15.]],

         [[ 5.,  6.,  7.,  8.],
          [ 9., 10., 11., 12.],
          [13., 14., 15., 16.],
          [17., 18., 19., 20.]],

         [[-3., -2., -1.,  0.],
          [ 1.,  2.,  3.,  4.],
          [ 5.,  6.,  7.,  8.],
          [ 9., 10., 11., 12.]]]])
-------------
tensor([[[[ 5.]],

         [[10.]],

         [[ 2.]]]])
```

7.6.Exercises

- Display the activations of the first and second layer of LeNet for different inputs
- Train with CIFAR-10

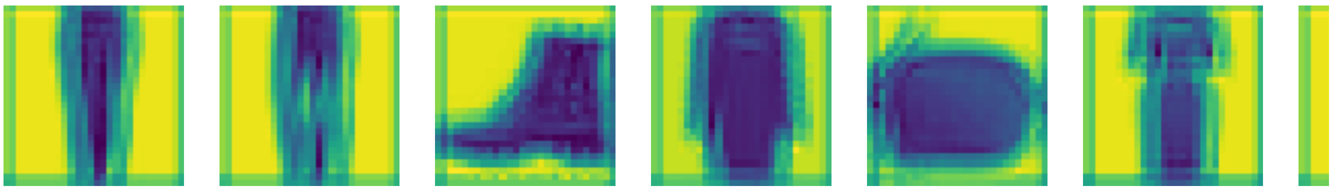
```
class Reshape(torch.nn.Module) :
  def forward(self, x):
    return x.view(-1, 1, 28, 28)
net = torch.nn.Sequential(Reshape(), nn.Conv2d(1, 6, kernel_size=5, padding=2),
                          nn.AvgPool2d(kernel_size=2, stride=2),
                          nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
                          nn.AvgPool2d(kernel_size=2, stride=2), nn.Flatten(),
                          nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
                          nn.Linear(120, 84), nn.Sigmoid(), nn.Linear(84,10)).t
```

```
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, dc
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, dc

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

```
import matplotlib.pyplot as plt

def show_images(images, nrows, ncols, titles=None):
    fig, axes = plt.subplots(nrows, ncols, figsize=(12, 12))
    for i in range(nrows * ncols):
        ax = axes[i] if nrows == 1 else axes[i // ncols, i % ncols]
        ax.imshow(images[i], cmap='viridis')
        ax.axis('off')
        if titles:
            ax.set_title(titles[i])
    plt.show()
```

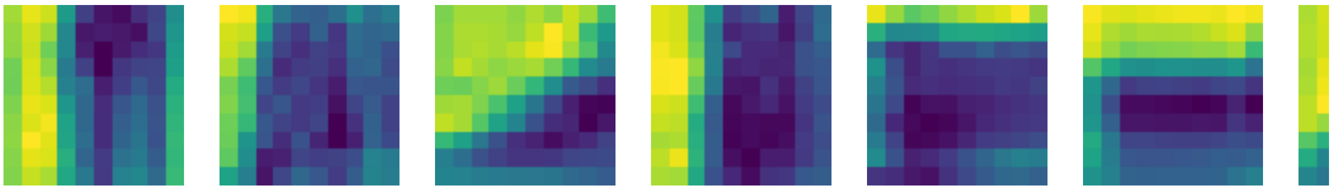
```
data_iter = iter(train_loader)
images, labels = next(data_iter)
device = d2l.try_gpu()

x_first_Sigmoid_layer = net[0:3](images.to(device)).cpu().detach().numpy()[0:9,
x_first_Sigmoid_layer.shape
```

```
show_images(x_first_Sigmoid_layer.reshape(9, 28, 28), 1, 9)
```



```
data_iter = iter(train_loader)
images, labels = next(data_iter)
device = d2l.try_gpu()

x_second_Sigmoid_layer = net[0:6](images.to(device)).cpu().detach().numpy()[0:9
x_second_Sigmoid_layer.shape
```

```
show_images(x_second_Sigmoid_layer.reshape(9, 10, 10), 1, 9)
```



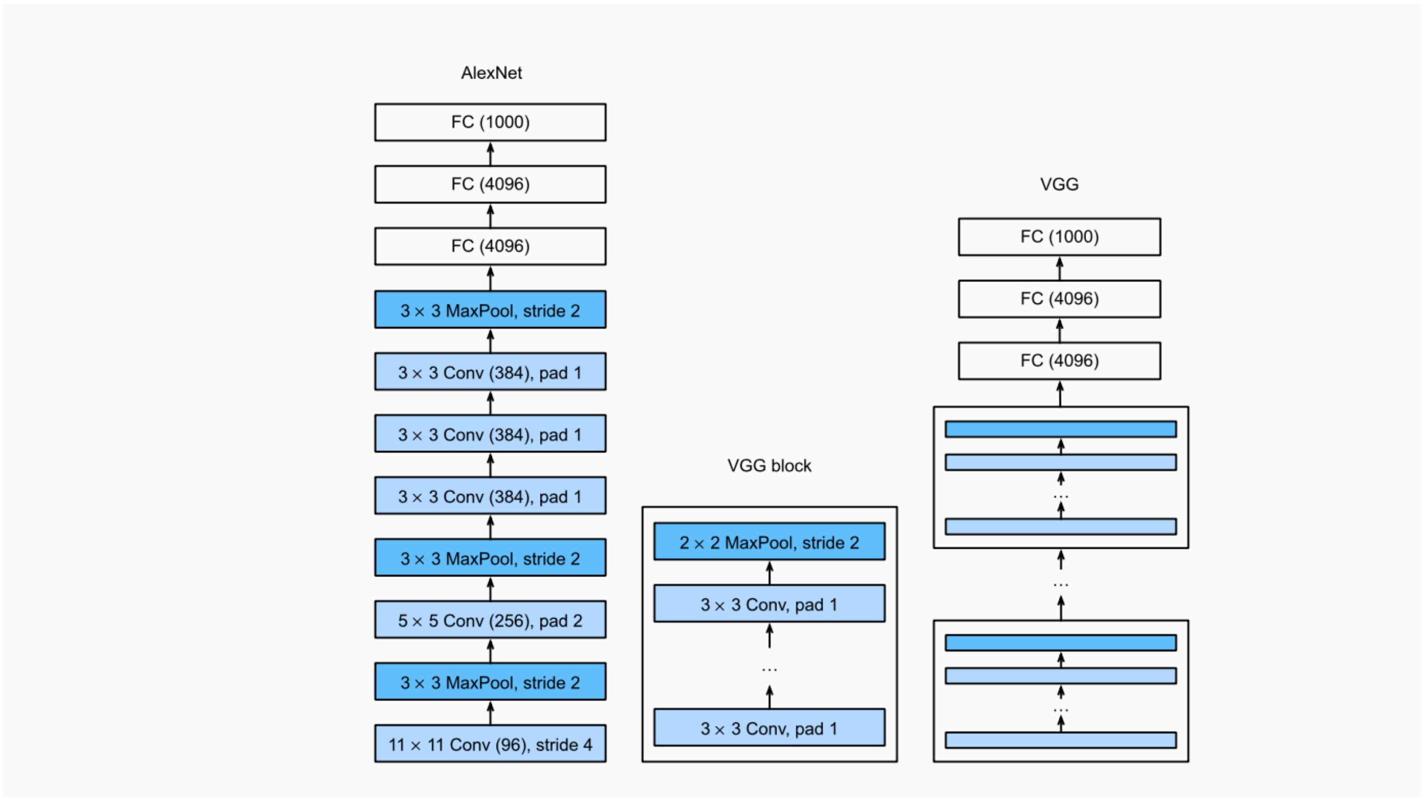
```
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
from torch import nn, optim

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

```
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, downloa
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, downloa

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)



model = LeNet(num_classes=10)
optimizer = optim.SGD(model.parameters(), lr=0.1)
loss_fn = nn.CrossEntropyLoss()


for epoch in range(10):
    model.train()
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()

    print(f'Epoch {epoch+1}, Loss: {loss.item()}')

model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy: {100 * correct / total}%')
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1, Loss: 2.294325828552246
Epoch 2, Loss: 2.3035967350006104
Epoch 3, Loss: 2.2967653274536133
Epoch 4, Loss: 2.3061301708221436
Epoch 5, Loss: 2.29586124420166
Epoch 6, Loss: 2.3058841228485107
Epoch 7, Loss: 2.3222591876983643
Epoch 8, Loss: 2.3006088733673096
Epoch 9, Loss: 2.3026089668273926
Epoch 10, Loss: 2.305776357650757
Accuracy: 10.0%
```

8.2.2.Discussions

-VGG Network



8.2.Exercises

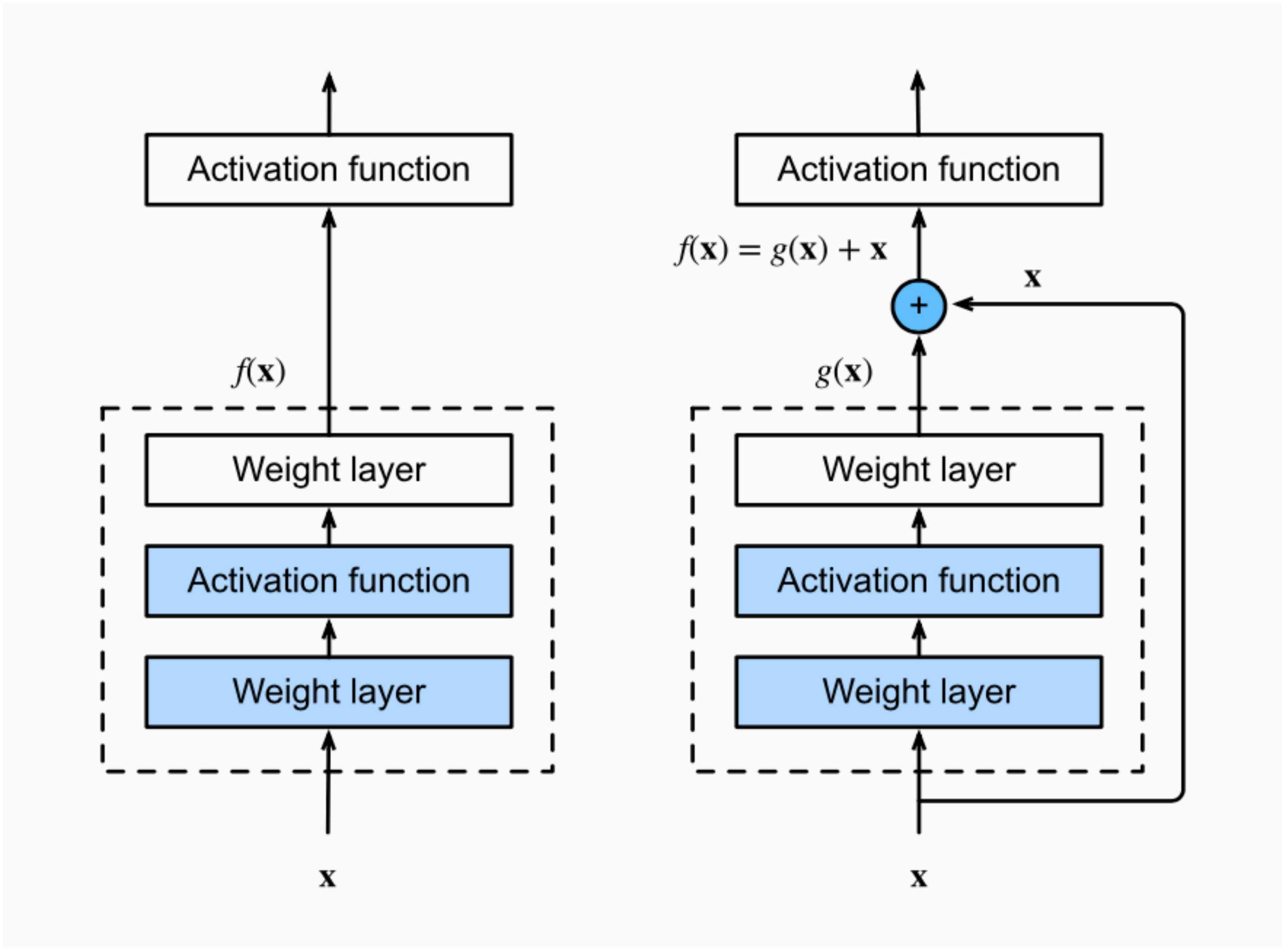
Comapre the number of parameters needed for AlexNet and VGG.

```
import torch
import torchvision.models as models

alexnet = models.alexnet()
vgg16 = models.vgg16()

def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

alexnet_params = count_parameters(alexnet)
vgg16_params = count_parameters(vgg16)

print(f"AlexNet Total Parameters: {alexnet_params}")
print(f"VGG16 Total Parameters: {vgg16_params}")
```

```
AlexNet Total Parameters: 61100840
VGG16 Total Parameters: 138357544
```

8.6.Summary & Discussions

- ResNet은 Residual 연결을 통하여 입력을 출력으로 직접 전달할 수 있도록 함. 이 방식은 네트워크의 귀납적 편향을 변경하여 단순 함수가 특정 형태가 아닌 더 복잡한 형태로 나타날 수 있도록 함.
- Residual 을 통해 입력이 더 빠르게 전달되어, 효과적으로 깊게 학습할 수 있도록 함. (깊어도 안정적임)
- 'Gradient Vanishing' 문제를 방지할 수 있음. <- 이전 층의 출력이 다음 층으로 직접 전달되므로, 기울기가 소실되지 않고 잘 전달됨.



코딩을 시작하거나 AI로 코드를 생성하세요.