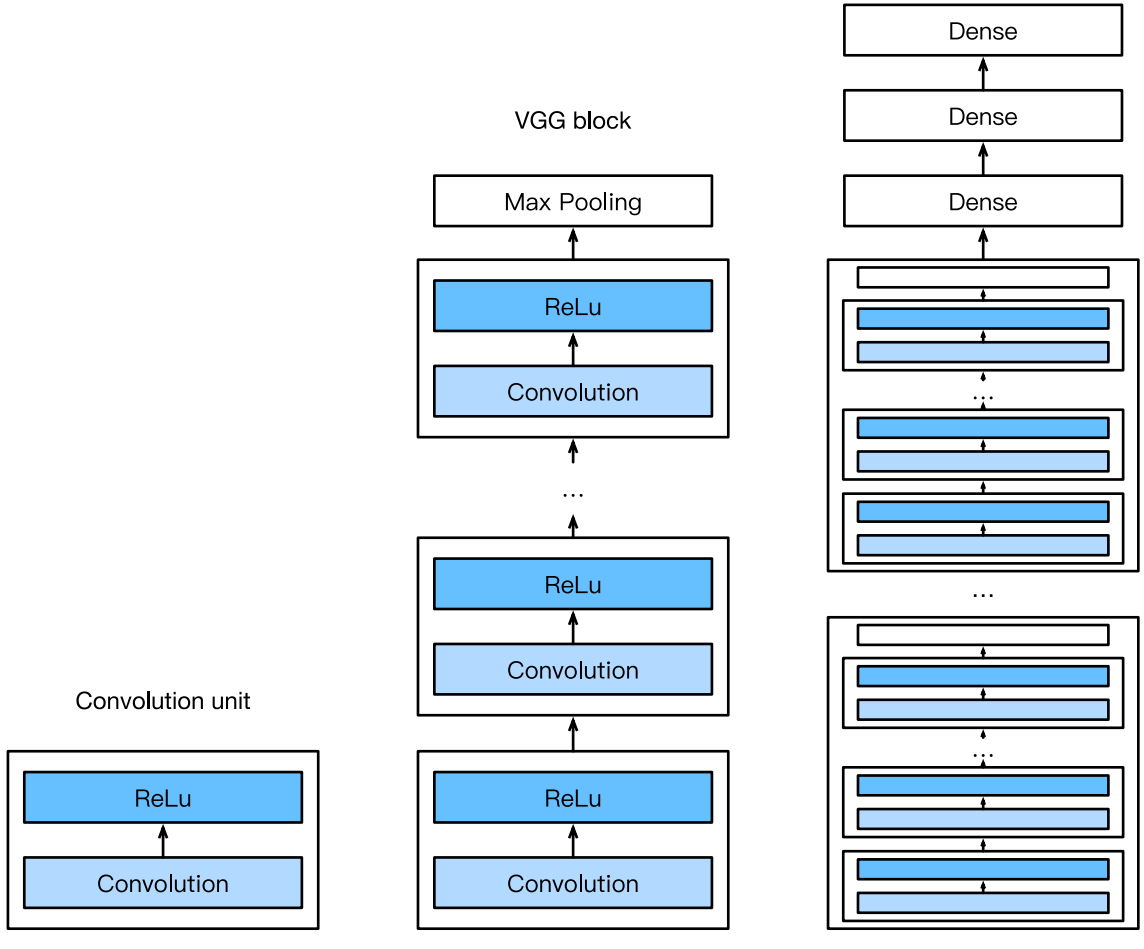
Networks Using Blocks (VGG)

We use the `vgg_block` function to implement this basic VGG block. This function takes the number of convolutional layers `num_convs` and the number of output channels `num_channels` as input.

In [1]:
```
import d2l
from mxnet import gluon, init, nd
from mxnet.gluon import nn

def vgg_block(num_convs, num_channels):
    blk = nn.Sequential()
    for _ in range(num_convs):
        blk.add(nn.Conv2D(num_channels, kernel_size=3,
                          padding=1, activation='relu'))
    blk.add(nn.MaxPool2D(pool_size=2, strides=2))
    return blk
```

VGG Architecture



In [2]:
```
conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))
```

Now, we will implement VGG-11. This is a simple matter of executing a for loop over `conv_arch`.

In [3]:
```
def vgg(conv_arch):
    net = nn.Sequential()
    # The convolutional layer part.
    for (num_convs, num_channels) in conv_arch:
        net.add(vgg_block(num_convs, num_channels))
    # The fully connected layer part.
    net.add(nn.Dense(4096, activation='relu'), nn.Dropout(0.5),
            nn.Dense(4096, activation='relu'), nn.Dropout(0.5),
            nn.Dense(10))
    return net

net = vgg(conv_arch)
```

Memory usage for single input

Next, we will construct a single-channel data example with a height and width of 224 to observe the output shape of each layer.

In [4]:
```
net.initialize()
X = nd.random.uniform(shape=(1, 1, 224, 224))
for blk in net:
    X = blk(X)
    print(blk.name, 'output shape:\t', X.shape)
```

```
sequential1 output shape:	 (1, 64, 112, 112)
sequential2 output shape:	 (1, 128, 56, 56)
sequential3 output shape:	 (1, 256, 28, 28)
sequential4 output shape:	 (1, 512, 14, 14)
sequential5 output shape:	 (1, 512, 7, 7)
dense0 output shape:	 (1, 4096)
dropout0 output shape:	 (1, 4096)
dense1 output shape:	 (1, 4096)
dropout1 output shape:	 (1, 4096)
dense2 output shape:	 (1, 10)
```

Model Training

Since VGG-11 is more complicated than AlexNet let's use a smaller network.,

In [5]:
```
ratio = 4
small_conv_arch = [(pair[0], pair[1] // ratio) for pair in conv_arch]
net = vgg(small_conv_arch)
```

Training Loop

In [6]:
```
lr, num_epochs, batch_size, ctx = 0.05, 5, 128, d2l.try_gpu()
net.initialize(ctx=ctx, init=init.Xavier())
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': lr})
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size, resize=224)
d2l.train_ch5(net, train_iter, test_iter, batch_size, trainer, ctx, num_epochs)
```

```
training on gpu(0)
epoch 1, loss 0.9533, train acc 0.654, test acc 0.852, time 37.7 sec
epoch 2, loss 0.4086, train acc 0.850, test acc 0.886, time 35.8 sec
epoch 3, loss 0.3319, train acc 0.878, test acc 0.900, time 35.9 sec
epoch 4, loss 0.2915, train acc 0.894, test acc 0.904, time 35.9 sec
epoch 5, loss 0.2605, train acc 0.905, test acc 0.911, time 35.8 sec
```