第 5 章　现代卷积神经网络

第 4 章介绍了卷积神经网络的基本原理，本章介绍现代的卷积神经网络结构，许多现代卷积神经网络的研究都是建立在这一章知识的基础上。本章中的每一个模型都曾一度占据主导地位，其中许多模型都是 ImageNet 竞赛的优胜者。ImageNet 竞赛自 2010 年以来，一直是计算机视觉中监督学习进展的指向标。

这些模型包括如下几种。

（1）深度卷积神经网络（AlexNet）。它是第一个在大规模视觉竞赛中击败传统计算机视觉模型的大型神经网络。

（2）使用重复块的网络（VGG）。它利用许多重复的神经网络块。

（3）网络中的网络（NiN）。它重复使用由卷积层和 1×1 卷积层（用来代替全连接层）来构建深层网络。

（4）含并行连结的网络（GoogLeNet）。它使用并行连结的网络，通过不同窗口大小的卷积层和最大池化层来并行抽取信息。

（5）残差网络（ResNet）。它通过残差块构建跨层的数据通道，是计算机视觉中最流行的体系结构。

（6）稠密连接网络（DenseNet）。它的计算成本很高，但能实现更好的效果。

虽然深度神经网络的概念非常简单——将神经网络堆叠在一起。但由于不同的网络结构和超参数选择，这些神经网络的性能会发生很大变化。本章介绍的神经网络是将人类直觉和相关数学见解结合后，经过大量研究试错后的结晶。本章按时间顺序介绍这些模型，在追寻历史脉络的同时，培养读者对该领域发展的直觉。这有助于读者研究、开发自己的结构。例如，本章介绍的批量归一化（batch normalization）和残差网络为设计和训练深度神经网络提供了重要的思想指导。

5.1　深度卷积神经网络

在 LeNet 提出后，卷积神经网络在计算机视觉和机器学习领域中很有名气。但卷积神经网络并没有主导这些领域。这是因为虽然 LeNet 在小数据集上取得了很好的效果，但是在更大、更真实的数据集上训练卷积神经网络的性能和可行性还有待研究。事实上，在 20 世纪 90 年代初到 2012 年之间的大部分时间里，神经网络往往被其他机器学习方法超越，如支持向量机。

在计算机视觉中，直接将神经网络与其他机器学习方法进行比较也许不公平。这是因为，卷积神经网络的输入是由原始像素值或是经过简单预处理（例如居中、缩放）的像素值组成的。但在使用传统机器学习方法时，从业者永远不会将原始像素作为输入。在传统机器学习方法中，计算机视觉流水线是由经过人的手工精心设计的特征流水线组成的。对于这些传统方法，大部分的进展都来自对特征有了更聪明的想法，并且学习到的算法往往归于事后的解释。

虽然 20 世纪 90 年代就有了一些神经网络加速器，但仅靠它们还不足以开发出有大量参数的深层多通道多层卷积神经网络。此外，当时的数据集仍然相对较小。除了这些障碍，训练神经网络的一些关键技巧仍然缺失，包括启发式参数初始化、随机梯度下降的巧妙变体、非挤压激活函数和有效的正则化技术。

因此，与训练端到端（从像素到分类结果）系统不同，经典机器学习的流水线看起来更像如下这样。

（1）获取一个有趣的数据集。在早期，收集这些数据集需要昂贵的传感器（在当时最先进的图像也就 100 万像素）。

（2）根据光学、几何学、其他知识以及偶然的发现，手工对特征数据集进行预处理。

（3）通过标准的特征提取算法，如 SIFT（尺度不变特征变换）、SURF（加速鲁棒特征）或其他手动调整的流水线来输入数据。

（4）将提取的特征放到最喜欢的分类器中（例如线性模型或其他核方法），以训练分类器。

机器学习研究人员相信机器学习既重要又美丽，他们希望用优雅的理论去证明各种模型的性质，他们相信机器学习是一个正在蓬勃发展、严谨且非常有用的领域。然而，计算机视觉研究人员认为推动领域进步的是数据特征，而不是学习算法。计算机视觉研究人员相信，从对最终模型精度的影响来说，更大或更干净的数据集或是稍微改进的特征提取，比任何学习算法带来的进步要大得多。

5.1.1　学习表征

另一种预测这个领域发展的方法——观察图像特征的提取方法。在 2012 年前，图像特征都是机械地计算出来的。事实上，设计一套新的特征函数、改进结果，并撰写论文是盛极一时的潮流。SIFT、SURF、HOG（定向梯度直方图）、bags of visual words 和类似的特征提取方法占据了主导地位。

另一组研究人员（包括 Yann LeCun、Geoff Hinton、Yoshua Bengio、Andrew Ng、Shun ichi Amari 和 Juergen Schmidhuber）的想法则与众不同：他们认为特征本身应该被学习。此外，他们还认为，在合理的复杂性前提下，特征应该由多个共同学习的神经网络层组成，每个层都有可学习的参数。在机器视觉中，最底层可能检测边缘、颜色和纹理。事实上，Alex Krizhevsky、Ilya Sutskever 和 Geoff Hinton 提出了一种新的卷积神经网络变体——AlexNet。AlexNet 在 2012 年 ImageNet 挑战赛中取得了轰动一时的成绩。AlexNet

以 Alex Krizhevsky 的名字命名，他是论文的第一作者。

有趣的是，在网络的最底层，模型学习到了一些类似于传统滤波器的特征抽取器。图 5-1 描述了底层图像特征。

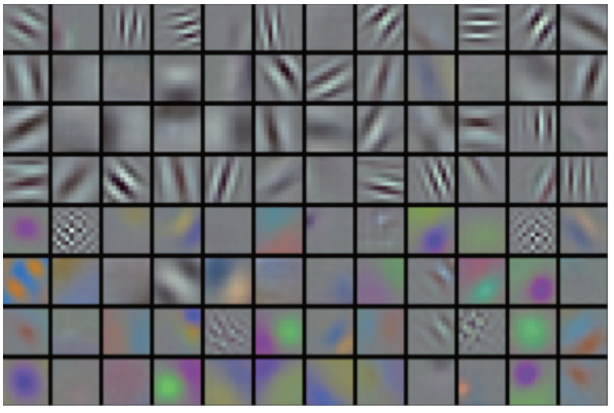


图 5-1　AlexNet 第一层学习到的特征抽取器

AlexNet 的更高层建立在这些底层表示的基础上，以表示更大的特征，如眼睛、鼻子、草叶等。而更高的层可以检测整个物体，如人、飞机、狗或飞盘。最终的隐藏神经元可以学习图像的综合表示，从而使属于不同类别的数据易于区分。尽管一直有一群执着的研究者不断钻研，试图学习视觉数据的逐级表征，然而很长一段时间里这些尝试都未有突破。深度卷积神经网络的突破出现在 2012 年。突破可归因于如下两个关键因素。

1. 缺少的成分：数据

包含许多特征的深度模型需要大量的有标签数据，才能显著优于基于凸优化的传统方法（如线性方法和核方法）。然而，限于早期计算机有限的存储和 20 世纪 90 年代有限的研究预算，大部分研究只基于小的公开数据集。例如，不少研究论文基于加州大学欧文分校（UCI）提供的若干公开数据集，其中许多数据集只有几百至几千张在非自然环境下以低分辨率拍摄的图像。这一状况在 2010 年前后兴起的大数据浪潮中得到改善。2009 年，ImageNet 数据集发布，并发起 ImageNet 挑战赛：要求研究人员从 100 万个样本中训练模型，以区分 1000 个不同类别的对象。ImageNet 数据集由斯坦福大学李飞飞教授小组的研究人员开发，利用谷歌图像搜索（Google image search）对每一类图像进行预筛选，并利用亚马逊众包（Amazon mechanical turk）来标注每张图片的相关类别。这种规模是前所未有的。这项被称为 ImageNet 的挑战赛推动了计算机视觉和机器学习研究的发展，挑战研究人员确定哪些模型能够在更大的数据规模下表现更好。

2. 缺少的成分：硬件

深度学习对计算资源要求很高，训练可能需要数百个迭代周期，每次迭代都需要通过代价高昂的许多线性代数层传递数据。这也是为什么在 20 世纪 90 年代至 21 世纪初，优化凸目标的简单算法是研究人员的首选。然而，用 GPU（Graphics Processing Unit，图形处理器）训练神经网络改变了这一格局。GPU 早年用来加速图形处理，使计算机游戏玩

家受益。GPU 可优化高吞吐量的 4×4 矩阵和向量乘法，从而服务于基本的图形任务。幸运的是，这些数学运算与卷积层的计算惊人地相似。由此，英伟达（NVIDIA）和 ATI 已经开始为通用计算操作优化 GPU，甚至把它们作为通用 GPU（General-Purpose GPUs，GPGPU）来销售。

那么 GPU 比 CPU 强在哪里呢？

中央处理器（Central Processing Unit，CPU）的每个核心都拥有高时钟频率的运行能力，和高达数 MB 的三级缓存 (L3 Cache)。它们非常适合执行各种指令，具有分支预测器、深层流水线和其他使 CPU 能够运行各种程序的功能。然而，这种明显的优势也是它的致命弱点：通用核心的制造成本非常高。它们需要大量的芯片面积、复杂的支持结构（内存接口、内核之间的缓存逻辑、高速互连等），而且它们在任何单个任务上的性能都相对较差。现代笔记本电脑一般有 4 核，即使是高端服务器也很少超过 64 核，因为它们的性价比不高。

相比于 CPU，GPU 由 100×1000 个小的处理单元组成（NVIDIA、ATI、ARM 和其他芯片供应商之间的细节稍有不同），通常被分成更大的组（NVIDIA 称为 warps）。虽然每个 GPU 核心都相对较弱，有时甚至以低于 1GHz 的时钟频率运行，但庞大的核心数量使 GPU 比 CPU 快几个数量级。例如，NVIDIA 的 Ampere GPU 架构为每个芯片提供了高达 312 TFlops 的浮点性能，而 CPU 的浮点性能到目前为止还没有超过 1TFlops。之所以有如此大的差距，原因其实很简单：首先，功耗往往会随时钟频率呈二次方增长。对于一个 CPU 核心，假设它的运行速度比 GPU 快 4 倍，就可以使用 16 个 GPU 内核取代，那么 GPU 的综合性能就是 CPU 的 $16 \times 1/4 = 4$ 倍。其次，GPU 内核要简单得多，这使得它们更节能。此外，深度学习中的许多操作需要相对较高的内存带宽，而 GPU 拥有 10 倍于 CPU 的带宽。

回到 2012 年的重大突破，当 Alex Krizhevsky 和 Ilya Sutskever 实现了可以在 GPU 硬件上运行的深度卷积神经网络时，一个重大突破出现了。他们意识到卷积神经网络中的计算瓶颈：卷积和矩阵乘法，都是可以在硬件上并行化的操作。于是，他们使用两个显存为 3GB 的 NVIDIA GTX580 GPU 实现了快速卷积运算。他们的创新成果——cuda-convnet，多年来一直是行业标准，并推动了深度学习热潮的到来。

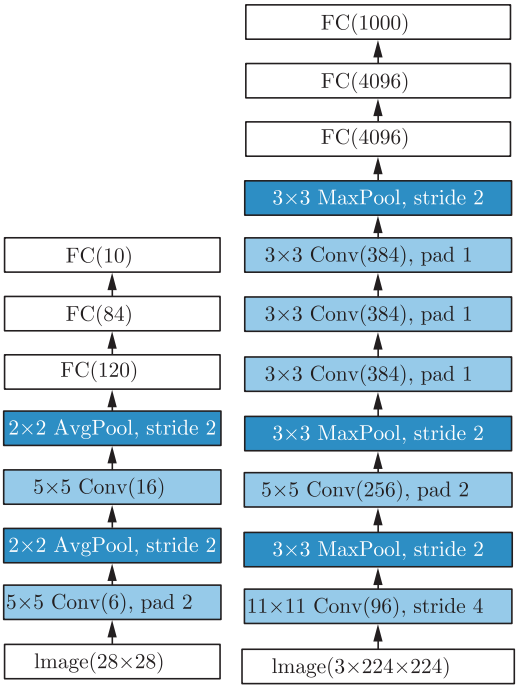
5.1.2　AlexNet

2012 年，AlexNet 横空出世。它首次证明了学习到的特征可以超越手工设计的特征。它一举打破了计算机视觉研究的现状。AlexNet 使用 8 层卷积神经网络，并以很大的优势赢得了 2012 年 ImageNet 图像识别挑战赛。

AlexNet 和 LeNet 的架构非常相似，如图 5-2 所示。注意，这里提供了一个稍精简版本的 AlexNet，去除了当年需要两个小型 GPU 同时运算的设计特点。

AlexNet 和 LeNet 的设计理念非常相似，但也存在显著差异。首先，AlexNet 比相对较小的 LeNet5 要深得多。AlexNet 由 8 层组成：5 个卷积层、2 个全连接隐藏层和 1 个全

连接输出层。其次，AlexNet 使用 ReLU 而不是 Sigmoid 作为其激活函数。接下来深入研究 AlexNet 的细节。



（a）LeNet　　（b）AlexNet

图 5-2　从 LeNet 到 AlexNet

1. 模型设计

在 AlexNet 的第一层，卷积窗口的形状是 11×11。由于大多数 ImageNet 中图像的宽和高比 MNIST 图像多 10 倍以上，因此，需要一个更大的卷积窗口来捕获目标。第二层中的卷积窗口形状被缩减为 5×5，然后是 3×3。此外，在第一层、第二层和第五层之后，加入窗口形状为 3×3、步幅为 2 的最大池化层。此外，AlexNet 的卷积通道是 LeNet 的 10 倍。

在最后一个卷积层后有两个全连接隐藏层，分别有 4096 个输出。这两个巨大的全连接隐藏层拥有将近 1GB 的模型参数。由于早期 GPU 显存有限，原版的 AlexNet 采用了双数据流设计，使得每个 GPU 只负责存储和计算模型的一半参数。幸运的是，现在 GPU 显存相对充裕，所以现在很少需要跨 GPU 分解模型。

2. 激活函数

此外，AlexNet 将 Sigmoid 激活函数改为更简单的 ReLU 激活函数。一方面，ReLU 激活函数的计算更简单，它没有如 Sigmoid 激活函数那样复杂的求幂运算。另一方面，当使用不同的参数初始化方法时，ReLU 激活函数使训练模型更加容易。当 Sigmoid 激活函数的输出非常接近于 0 或 1 时，这些区域的梯度几乎为 0，因此反向传播无法继续更新一

些模型参数。相反，ReLU 激活函数在正区间的梯度总是 1。因此，如果模型参数没有正确初始化，Sigmoid 函数可能在正区间内得到几乎为 0 的梯度，从而使模型无法得到有效的训练。

3. 容量控制和预处理

AlexNet 通过 dropout 控制全连接层的模型复杂度，而 LeNet 只使用了权重衰减。为了进一步扩充数据，AlexNet 在训练时增加了大量的图像增强数据，如翻转、裁切和变色。这使得模型更健壮，更大的样本量有效地减少了过拟合。具体内容将在 sec_image_augmentation 中更详细地讨论数据扩充。

```
import tensorflow as tf
from d2l import tensorflow as d2l

def net():
    return tf.keras.models.Sequential([
        # 这里使用一个 11*11 的更大窗口来捕捉对象
        # 同时，步幅为 4，以减少输出的高度和宽度
        # 另外，输出通道的数目远大于 LeNet
        tf.keras.layers.Conv2D(filters=96, kernel_size=11, strides=4,
                               activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=3, strides=2),
        # 减小卷积窗口，使用填充为 2 来让输入与输出的高和宽一致，且增大输出通道数
        tf.keras.layers.Conv2D(filters=256, kernel_size=5, padding='same',
                               activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=3, strides=2),
        # 使用 3 个连续的卷积层和较小的卷积窗口
        # 除了最后的卷积层，输出通道的数量进一步增加
        # 在前两个卷积层之后，池化层不用于减少输入的高度和宽度
        tf.keras.layers.Conv2D(filters=384, kernel_size=3, padding='same',
                               activation='relu'),
        tf.keras.layers.Conv2D(filters=384, kernel_size=3, padding='same',
                               activation='relu'),
        tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                               activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=3, strides=2),
        tf.keras.layers.Flatten(),
        # 这里，全连接层的输出数量是 LeNet 中的几倍，使用 dropout 层来减轻过度拟合
        tf.keras.layers.Dense(4096, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(4096, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        # 最后是输出层。由于这里使用 Fashion-MNIST，所以用类别数为 10，而非论文中的 1000
        tf.keras.layers.Dense(10)])
```

构造一个高度和宽度都为 224 的单通道数据来观察每一层输出的形状。它与图 5-2 中的 AlexNet 架构相匹配。

```
X = tf.random.uniform((1, 224, 224, 1))
for layer in net().layers:
        X = layer(X)
        print(layer.__class__.__name__, 'Output shape:\t', X.shape)
```

```
Conv2D Output shape:         (1, 54, 54, 96)
MaxPooling2D Output shape:   (1, 26, 26, 96)
Conv2D Output shape:         (1, 26, 26, 256)
MaxPooling2D Output shape:   (1, 12, 12, 256)
Conv2D Output shape:         (1, 12, 12, 384)
Conv2D Output shape:         (1, 12, 12, 384)
Conv2D Output shape:         (1, 12, 12, 256)
MaxPooling2D Output shape:   (1, 5, 5, 256)
Flatten Output shape:        (1, 6400)
Dense Output shape:  (1, 4096)
Dropout Output shape:        (1, 4096)
Dense Output shape:  (1, 4096)
Dropout Output shape:        (1, 4096)
Dense Output shape:  (1, 10)
```

5.1.3 读取数据集

尽管 AlexNet 是在 ImageNet 上进行训练的，但这里使用的是 Fashion-MNIST 数据集。因为即使在现代 GPU 上训练 ImageNet 模型，同时使其收敛可能需要数小时或数天的时间。而将 AlexNet 直接应用于 Fashion-MNIST 的一个问题是，Fashion-MNIST 的图像分辨率（28×28 像素）低于 ImageNet 图像的分辨率。为了解决这个问题，将分辨率增加到 224×224 像素（通常来讲这不是一个明智的做法，但在这里是为了有效使用 AlexNet 结构）。使用 d2l.load_data_fashion_mnist 函数中的 resize 参数执行此调整。

```
batch_size = 128
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size, resize=224)
```

5.1.4 训练 AlexNet

现在可以开始训练 AlexNet 了。与 LeNet 相比，主要变化是使用更小的学习速率训练，这是因为网络更深、更广，图像分辨率更高，训练卷积神经网络就更昂贵。迭代结果如图 5-3 所示。

```
lr, num_epochs = 0.01, 10
d2l.train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())
```

```
loss 0.325, train acc 0.881, test acc 0.871
4700.9 examples/sec on /GPU:0
```

```
<tensorflow.python.keras.engine.sequential.Sequential at 0x7ff809eb3d00>
```

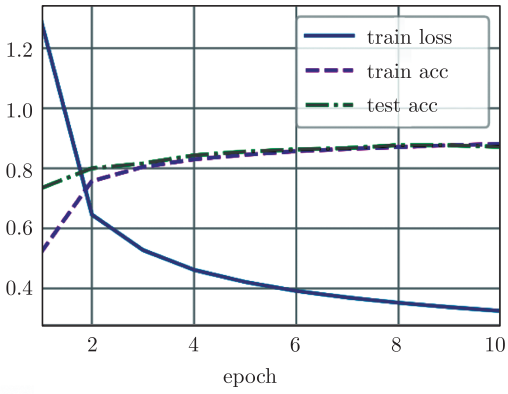


图 5-3　AlexNet 迭代结果

5.2　使用块的网络

虽然 AlexNet 证明深层神经网络卓有成效，但它没有提供一个通用的模板来指导后续的研究人员设计新的网络。本章介绍一些常用于设计深层神经网络的启发式概念。

与芯片设计中的工程师从放置晶体管到逻辑元件再到逻辑块的过程类似，神经网络结构的设计也逐渐变得更加抽象。研究人员开始从单个神经元的角度思考问题，发展到整个层次，现在又转向模块，重复各层的模式。

使用块的想法首先出现在牛津大学的视觉几何组（Visual Geometry Group，VGG）的网络中。通过使用循环和子程序，可以很容易地在任何现代深度学习框架的代码中实现这些重复的结构。

5.2.1　VGG 块

经典卷积神经网络的基本组成部分的序列如下：①带填充以保持分辨率的卷积层；②非线性激活函数，如 ReLU；③池化层，如最大池化层。

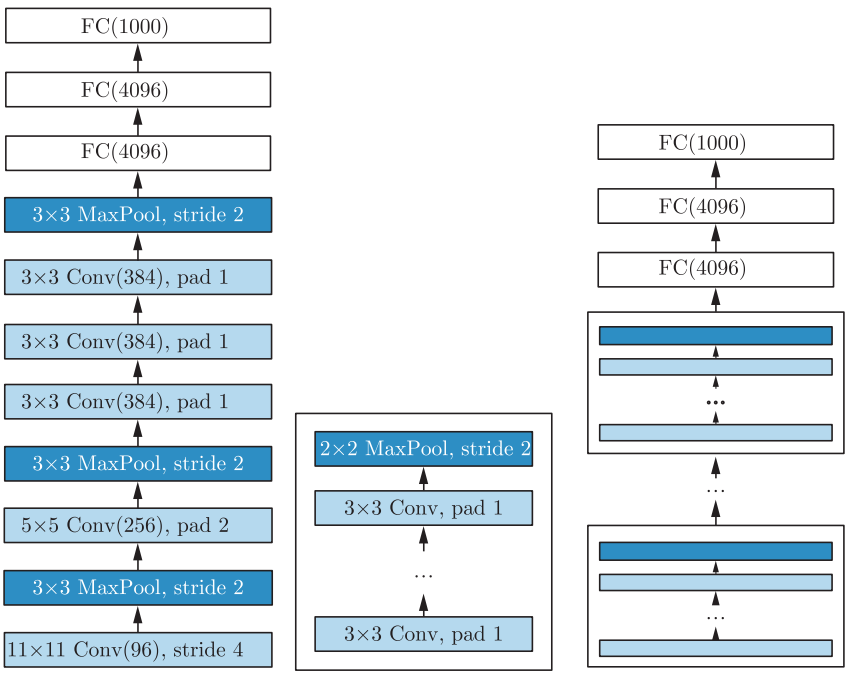
而一个 VGG 块与之类似，由一系列卷积层组成，后面再加上用于空间下采样的最大池化层。在最初的 VGG 论文中，作者使用了带有 3×3 卷积核、填充为 1（保持高度和宽度）的卷积层，带有 2×2 池化窗口、步幅为 2（每个块后的分辨率减半）的最大池化层。在下面的代码中，定义了一个名为 vgg_block 的函数来实现 VGG 块。

```
import tensorflow as tf
from d2l import tensorflow as d2l

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

5.2.2 VGG 网络

与 AlexNet、LeNet 一样，VGG 网络可以分为两部分：第一部分主要由卷积层和池化层组成，第二部分由全连接层组成，如图 5-4 所示。



（a）AlexNet　　（b）VGG block　　（c）VGG

图 5-4　从 AlexNet 到 VGG

VGG 神经网络连续连接图 5-4 的几个 VGG 块（在 vgg_block 函数中定义）。其中有超参数变量 conv_arch。该变量指定了每个 VGG 块里卷积层个数和输出通道数。全连接模块则与 AlexNet 中相同。

原始 VGG 网络有 5 个卷积块，其中前两个块各有一个卷积层，后三个块各包含两个卷积层。第一个模块有 64 个输出通道，每个后续模块将输出通道数量翻倍，直到该数字达到 512。由于该网络使用 8 个卷积层和 3 个全连接层，因此它通常被称为 VGG-11。

```
conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))
```

下面的代码实现了 VGG-11，可以通过在 conv_arch 上执行 for 循环来简单实现。

```
def vgg(conv_arch):
    net = tf.keras.models.Sequential()
    # 卷积层部分
    for (num_convs, num_channels) in conv_arch:
        net.add(vgg_block(num_convs, num_channels))
    # 全连接层部分
    net.add(
        tf.keras.models.Sequential([
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(4096, activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(4096, activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(10)]))
    return net

net = vgg(conv_arch)
```

接下来，构建一个高度和宽度为 224 的单通道数据样本，以观察每个层输出的形状。

```
X = tf.random.uniform((1, 224, 224, 1))
for blk in net.layers:
        X = blk(X)
        print(blk.__class__.__name__, 'output shape:\t', X.shape)
```

```
Sequential output shape:     (1, 112, 112, 64)
Sequential output shape:     (1, 56, 56, 128)
Sequential output shape:     (1, 28, 28, 256)
Sequential output shape:     (1, 14, 14, 512)
Sequential output shape:     (1, 7, 7, 512)
Sequential output shape:     (1, 10)
```

从上面的代码中可以看到，每个块的高度和宽度减半，最终高度和宽度都为 7。最后再展平表示，送入全连接层处理。