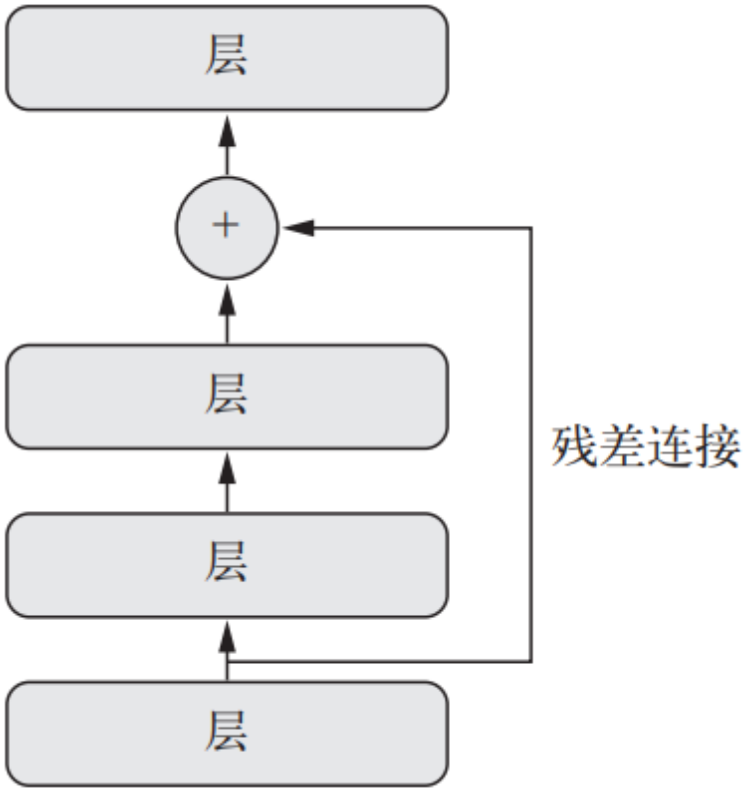
让模型由“具有不错的性能”上升到“性能卓越且能够赢得机器学习竞赛

特殊结构

残差连接



批标准化

标准化（normalization）是一大类方法，用于让机器学习模型看到的不同样本彼此之间更加相似，这有助于模型的学习与对新数据的泛化。最常见的数据标准化形式就是你已经在本书中多次见到的那种形式：将数据减去其平均值使其中心为 0，然后将数据除以其标准差使其标准差为 1。实际上，这种做法假设数据服从正态分布（也叫高斯分布），并确保让该分布的中心为 0，同时缩放到方差为 1。

```
normalized_data = (data - np.mean(data, axis=...)) / np.std(data, axis=...)
```

前面的示例都是在将数据输入模型之前对数据做标准化。但在网络的每一次变换之后都应该考虑数据标准化。即使输入 `Dense` 或 `Conv2D` 网络的数据均值为 0、方差为 1，也没有理由假定网络输出的数据也是这样。

批标准化（batch normalization）是 Ioffe 和 Szegedy 在 2015 年提出的一种层的类型[①]（在 Keras 中是 `BatchNormalization`），即使在训练过程中均值和方差随时间发生变化，它也可以适应性地将数据标准化。批标准化的工作原理是，训练过程中在内部保存已读取每批数据均值和方差的指数移动平均值。批标准化的主要效果是，它有助于梯度传播（这一点和残差连接很像），因此允许更深的网络。对于有些特别深的网络，只有包含多个 `BatchNormalization` 层时才能进行训练。例如，`BatchNormalization` 广泛用于 Keras 内置的许多高级卷积神经网络架构，比如 ResNet50、Inception V3 和 Xception。

BatchNormalization 层通常在卷积层或密集连接层之后使用

深度可分离卷积

如果我告诉你，有一个层可以替代 `Conv2D`，并可以让模型更加轻量（即更少的可训练权重参数）、速度更快（即更少的浮点数运算），还可以让任务性能提高几个百分点，你觉得怎么样？我说的正是**深度可分离卷积**（depthwise separable convolution）层（`SeparableConv2D`）的作用。这个层对输入的每个通道分别执行空间卷积，然后通过逐点卷积（1×1 卷积）将输出通道混合，如图 7-16 所示。这相当于将空间特征学习和通道特征学习分开，如果你假设输入中的空间位置高度相关，但不同的通道之间相对独立，那么这么做是很有意义的。它需要的参数要少很多，计算量也更小，因此可以得到更小、更快的模型。因为它是一种执行卷积更高效的方法，所以往往能够使用更少的数据学到更好的表示，从而得到性能更好的模型。

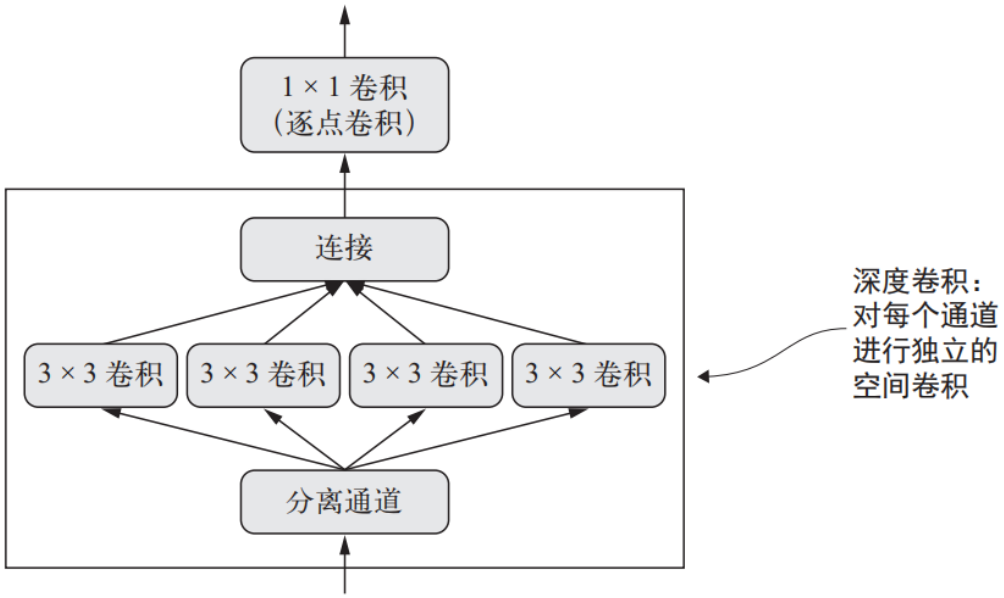


图 7-16　深度可分离卷积：深度卷积 + 逐点卷积

未来，无论是一维、二维还是三维应用，深度可分离卷积很可能会完全取代普通卷积，因为它的表示效率更高

In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [4]:

```
from tensorflow.keras import Sequential, layers, models
```

In [5]:

```
height = 64
width = 64
channels = 3
num_classes = 10
```

In [6]:

```
model = Sequential()
model.add(layers.SeparableConv2D(32, 3, activation='relu',
                                 input_shape=(height, width, channels,)))
model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))
model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))
model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
```

In [7]:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

对于规模更大的模型，深度可分离卷积是 Xception 架构的基础，Xception 是一个高性能的卷积神经网络，内置于 Keras 中。在我的论文“Xception: deep learning with depthwise separable convolutions”中，你可以进一步了解深度可分离卷积和 Xception 的理论基础。

超参数选择

构建深度学习模型时，你必须做出许多看似随意的决定：应该堆叠多少层？每层应该包含多少个单元或过滤器？激活应该使用 `relu` 还是其他函数？在某一层之后是否应该使用 `BatchNormalization`？应该使用多大的 dropout 比率？还有很多。这些在架构层面的参数叫作**超参数**（hyperparameter），以便将其与模型参数区分开来，后者通过反向传播进行训练。

在实践中，经验丰富的机器学习工程师和研究人员会培养出直觉，能够判断上述选择哪些可行、哪些不可行。也就是说，他们学会了调节超参数的技巧。但是调节超参数并没有正式成文的规则。如果你想要在某项任务上达到最佳性能，那么就不能满足于一个容易犯错的人随意做出的选择。即使你拥有很好的直觉，最初的选择也几乎不可能是最优的。你可以手动调节你的选择、重新训练模型，如此不停重复来改进你的选择，这也是机器学习工程师和研究人员大部分时间都在做的事情。但是，整天调节超参数不应该是人类的工作，最好留给机器去做。

因此，你需要制定一个原则，系统性地自动探索可能的决策空间。你需要搜索架构空间，并根据经验找到性能最佳的架构。这正是超参数自动优化领域的内容。这个领域是一个完整的研究领域，而且很重要。

超参数优化的过程通常如下所示。

(1) 选择一组超参数（自动选择）。

(2) 构建相应的模型。

(3) 将模型在训练数据上拟合，并衡量其在验证数据上的最终性能。

(4) 选择要尝试的下一组超参数（自动选择）。

(5) 重复上述过程。

(6) 最后，衡量模型在测试数据上的性能。

这个过程的关键在于，给定许多组超参数，使用验证性能的历史来选择下一组需要评估的超参数的算法。有多种不同的技术可供选择：贝叶斯优化、遗传算法、简单随机搜索等。

训练模型权重相对简单：在小批量数据上计算损失函数，然后用反向传播算法让权重向正确的方向移动。与此相反，更新超参数则非常具有挑战性。我们来考虑以下两点。

- 计算反馈信号（这组超参数在这个任务上是否得到了一个高性能的模型）的计算代价可能非常高，它需要在数据集上创建一个新模型并从头开始训练。
- 超参数空间通常由许多离散的决定组成，因而既不是连续的，也不是可微的。因此，你通常不能在超参数空间中做梯度下降。相反，你必须依赖不使用梯度的优化方法，而这些方法的效率比梯度下降要低很多。

这些挑战非常困难，而这个领域还很年轻，因此我们目前只能使用非常有限的工具来优化模型。通常情况下，随机搜索（随机选择需要评估的超参数，并重复这一过程）就是最好的解决方案，虽然这也是最简单的解决方案。但我发现有一种工具确实比随机搜索更好，它就是 Hyperopt。它是一个用于超参数优化的 Python 库，其内部使用 Parzen 估计器的树来预测哪组超参数可能会得到好的结果。另一个叫作 Hyperas 的库将 Hyperopt 与 Keras 模型集成在一起。一定要试试。

注意 在进行大规模超参数自动优化时，有一个重要的问题需要牢记，那就是验证集过拟合。因为你是使用验证数据计算出一个信号，然后根据这个信号更新超参数，所以你实际上是在验证数据上训练超参数，很快会对验证数据过拟合。请始终记住这一点。

总之，超参数优化是一项强大的技术，想要在任何任务上获得最先进的模型或者赢得机器学习竞赛，这项技术都必不可少。思考一下：曾经人们手动设计特征，然后输入到浅层机器学习模型中，这肯定不是最优的。现在，深度学习能够自动完成分层特征工程的任务，这些特征都是利用反馈信号学到的，而不是手动调节的，事情本来就应该如此。同样，你也不应该手动设计模型架构，而是应该按照某种原则对其进行最优化。在写作本书时，超参数自动优化还是一个非常年轻且不成熟的领域，正如几年前的深度学习，但我预计这一领域会在未来数年内蓬勃发展。

模型集成

想要在一项任务上获得最佳结果，另一种强大的技术是**模型集成**（model ensembling）。集成是指将一系列不同模型的预测结果汇集到一起，从而得到更好的预测结果。观察机器学习竞赛，特别是 Kaggle 上的竞赛，你会发现优胜者都是将很多模型集成到一起，它必然可以打败任何单个模型，无论这个模型的表现多么好。

集成依赖于这样的假设，即对于独立训练的不同良好模型，它们表现良好可能是因为**不同的原因**：每个模型都从略有不同的角度观察数据来做出预测，得到了“真相”的一部分，但不是全部真相。你可能听说过盲人摸象的古代寓言：一群盲人第一次遇到大象，想要通过触摸来了解大象。每个人都摸到了大象身体的不同部位，但只摸到了一部分，比如鼻子或一条腿。这些人描述的大象是这样的，“它像一条蛇”“像一根柱子或一棵树”，等等。这些盲人就好比机器学习模型，每个人都试图根据自己的假设（这些假设就是模型的独特架构和独特的随机权重初始化）并从自己的角度来理解训练数据的多面性。每个人都得到了数据真相的一部分，但不是全部真相。将他们的观点汇集在一起，你可以得到对数据更加准确的描述。大象是多个部分的组合，每个盲人说的都不完全准确，但综合起来就成了一个相当准确的故事。

我们以分类问题为例。想要将一组分类器的预测结果汇集在一起［即**分类器集成**（ensemble the classifiers）］，最简单的方法就是将它们的预测结果取平均值作为预测结果。

```
preds_a = model_a.predict(x_val)
preds_b = model_b.predict(x_val)
preds_c = model_c.predict(x_val)
preds_d = model_d.predict(x_val)

final_preds = 0.25 * (preds_a + preds_b + preds_c + preds_d)
```

使用 4 个不同的模型来计算初始预测

这个新的预测数组应该比任何一个初始预测都更加准确

只有这组分类器中每一个的性能差不多一样好时，这种方法才奏效。如果其中一个分类器性能比其他的差很多，那么最终预测结果可能不如这一组中的最佳分类器那么好。

将分类器集成有一个更聪明的做法，即加权平均，其权重在验证数据上学习得到。通常来说，更好的分类器被赋予更大的权重，而较差的分类器则被赋予较小的权重。为了找到一组好的集成权重，你可以使用随机搜索或简单的优化算法（比如 Nelder-Mead 方法）。

```
preds_a = model_a.predict(x_val)
preds_b = model_b.predict(x_val)
preds_c = model_c.predict(x_val)
preds_d = model_d.predict(x_val)

final_preds = 0.5 * preds_a + 0.25 * preds_b + 0.1 * preds_c + 0.15 * preds_d
```

假设 (0.5, 0.25, 0.1, 0.15) 这些权重是根据经验学到的

还有许多其他变体，比如你可以对预测结果先取指数再做平均。一般来说，简单的加权平均，其权重在验证数据上进行最优化，这是一个很强大的基准方法。

想要保证集成方法有效，关键在于这组分类器的**多样性**（diversity）。多样性就是力量。如果所有盲人都只摸到大象的鼻子，那么他们会一致认为大象像蛇，并且永远不会知道大象的真实模样。是多样性让集成方法能够取得良好效果。用机器学习的术语来说，如果所有模型的偏差都在同一个方向上，那么集成也会保留同样的偏差。如果各个模型的**偏差在不同方向上**，那么这些偏差会彼此抵消，集成结果会更加稳定、更加准确。

因此，集成的模型应该**尽可能好**，同时**尽可能不同**。这通常意味着使用非常不同的架构，甚至使用不同类型的机器学习方法。有一件事情基本上是**不**值得做的，就是对相同的网络，使用不同的随机初始化多次独立训练，然后集成。如果模型之间的唯一区别是随机初始化和训练数据的读取顺序，那么集成的多样性很小，与单一模型相比只会有微小的改进。

我发现有一种方法在实践中非常有效（但这一方法还没有推广到所有问题领域），就是将基于树的方法（比如随机森林或梯度提升树）和深度神经网络进行集成。2014 年，合作者 Andrei Kolev 和我使用多种树模型和深度神经网络的集成，在 Kaggle 希格斯玻色子衰变探测挑战赛中获得第四名。值得一提的是，集成中的某一个模型来源于与其他模型都不相同的方法（它是正则化的贪婪森林），并且得分也远远低于其他模型。不出所料，它在集成中被赋予了一个很小的权重。但出乎我们的意料，它极大地改进了总体的集成结果，因为它和其他所有模型都完全不同，提供了其他模型都无法获得的信息。这正是集成方法的关键之处。集成不在于你的最佳模型有多好，而在于候选模型集合的多样性。

近年来，一种在实践中非常成功的基本集成方法是**宽且深**（wide and deep）的模型类型，它结合了深度学习与浅层学习。这种模型联合训练一个深度神经网络和一个大型的线性模型。对多种模型联合训练，是实现模型集成的另一种选择。

将非常相似的模型集成基本上是没有意义的。最好的集成方法是将尽可能不同的一组模型集成（这组模型还需要具有尽可能高的预测能力）

In [ ]: