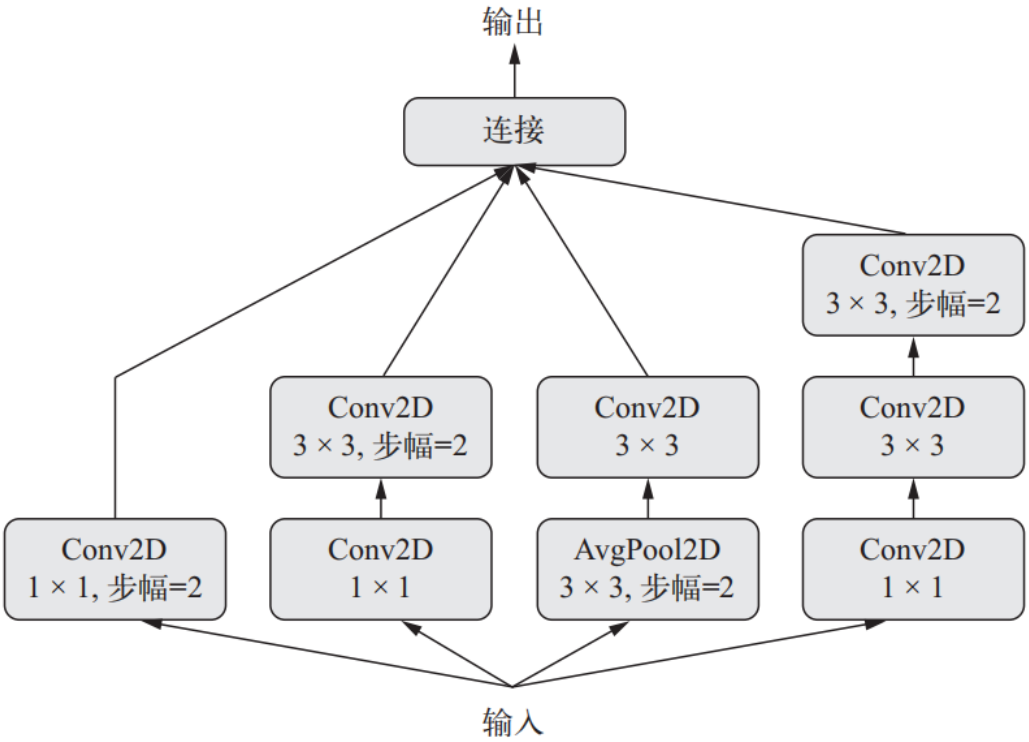
实现具有复杂的内部拓扑结构的网络

Inception 模块

GoogleNet CNN网络，灵感来源于早期的 network-in-network 架构，它是模块的堆叠，这些模块本身看起来像是小型的独立网络，被分为多个并行分支

多个卷积并行计算，最后将所得到的特征连接在一起。这种设置有助于网络分别学习空间特征和逐通道的特征，这比联合学习(一个学习1x1，一个学习3x3)这两种特征更加有效



Inception 模块也可能具有更复杂的形式，通常会包含池化运算、不同尺寸的空间卷积，（比如在某些分支上使用 5×5 的卷积代替 3×3 的卷积）和不包含空间卷积的分支（只有一个1×1 卷积）

1×1 卷积的作用

我们已经知道，卷积能够在输入张量的每一个方块周围提取空间图块，并对所有图块应用相同的变换。极端情况是提取的图块只包含一个方块。这时卷积运算等价于让每个方块向量经过一个 Dense 层：它计算得到的特征能够将输入张量通道中的信息混合在一起，但不会将跨空间的信息混合在一起（因为它一次只查看一个方块）。这种 1×1 卷积［也叫作**逐点卷积**（pointwise convolution）］是 Inception 模块的特色，它有助于区分开通道特征学习和空间特征学习。如果你假设每个通道在跨越空间时是高度自相关的，但不同的通道之间可能并不高度相关，那么这种做法是很合理的。

In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [3]:

```
from tensorflow.keras import Input, layers, Model
```

In [4]:

```
x = Input(shape=(None, None, 3))
```

In [5]:

```
x.shape
```

Out[5]:

```
TensorShape([None, None, None, 3])
```

In [6]:

```
brach1 = layers.Conv2D(128, 1, activation='relu', strides=2)(x)
```

In [7]:

```
brach2 = layers.Conv2D(128, 1, activation='relu')(x)
brach2 = layers.Conv2D(128, 3, activation='relu', strides=2)(brach2)
```

In [8]:

```
brach3 = layers.AveragePooling2D(3, strides=2)(x)
brach3 = layers.Conv2D(128, 3, activation='relu')(brach3)
```

In [9]:

```
brach4 = layers.Conv2D(128, 1, activation='relu')(x)
brach4 = layers.Conv2D(128, 3, activation='relu')(brach4)
brach4 = layers.Conv2D(128, 3, activation='relu', strides=2)(brach4)
```

In [10]:

```
brach1.shape, brach2.shape, brach3.shape, brach4.shape
```

Out[10]:

```
(TensorShape([None, None, None, 128]),
 TensorShape([None, None, None, 128]),
 TensorShape([None, None, None, 128]),
 TensorShape([None, None, None, 128]))
```

In [11]:

```
output = layers.concatenate([brach1, brach2, brach3, brach4], axis=-1)
```

In [13]:

```
output.shape
```

Out[13]:

```
TensorShape([None, None, None, 512])
```

Xception 代表极端 Inception（extreme inception），它是一种卷积神经网络架构，其灵感可能来自于 Inception。Xception 将分别进行通道特征学习与空间特征学习的想法推向逻辑上的极端，并将 Inception 模块替换为深度可分离卷积，其中包括一个逐深度卷积（即一个空间卷积，分别对每个输入通道进行处理）和后面的一个逐点卷积（即一个 1×1 卷积）。这个深度可分离卷积实际上是 Inception 模块的一种极端形式，其空间特征和通道特征被完全分离。Xception 的参数个数与 Inception V3 大致相同，但因为它对模型参数的使用更加高效，所以在 ImageNet 以及其他大规模数据集上的运行性能更好，精度也更高。

残差连接 residual connection

残差连接解决了困扰所有大规模深度学习模型的两个共性问题：梯度消失和表示瓶颈。通常来说，向任何多于 10 层的模型中添加残差连接，都可能会有所帮助

残差连接是让前面某层的输出作为后面某层的输入，从而在序列网络中有效地创造了一条捷径。前面层的输出没有与后面层的激活连接在一起，而是与后面层的激活相加（这里假设两个激活的形状相同）。如果它们的形状不同，我们可以用一个线性变换将前面层的激活改变成目标形状（例如，这个线性变换可以是不带激活的 Dense 层；对于卷积特征图，可以是不带激活1×1 卷积）。

In [15]:

```
x = Input(shape=(None, None, 128))
```

In [16]:

```
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.add([y, x])
```

In [17]:

```
y.shape
```

Out[17]:

```
TensorShape([None, None, None, 128])
```

In [18]:

```
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.MaxPooling2D(2, strides=2)(y)
residual = layers.Conv2D(128, 1, strides=2, padding='same')(x)
y = layers.add([y, residual])
```

In [19]:

```
y.shape
```

Out[19]:

```
TensorShape([None, None, None, 128])
```

In [20]:

```
residual.shape
```

Out[20]:

```
TensorShape([None, None, None, 128])
```

深度学习中的表示瓶颈

在 Sequential 模型中，每个连续的表示层都构建于前一层之上，这意味着它只能访问前一层激活中包含的信息。如果某一层太小（比如特征维度太低），那么模型将会受限于该层激活中能够塞入多少信息。

你可以通过类比信号处理来理解这个概念：假设你有一条包含一系列操作的音频处理流水线，每个操作的输入都是前一个操作的输出，如果某个操作将信号裁剪到低频范围（比如 0~15 kHz），那么下游操作将永远无法恢复那些被丢弃的频段。任何信息的丢失都是永久性的。残差连接可以将较早的信息重新注入到下游数据中，从而部分解决了深度学习模型的这一问题。

深度学习中的梯度消失

反向传播是用于训练深度神经网络的主要算法，其工作原理是将来自输出损失的反馈信号向下传播到更底部的层。如果这个反馈信号的传播需要经过很多层，那么信号可能会变得非常微弱，甚至完全丢失，导致网络无法训练。这个问题被称为**梯度消失**（vanishing gradient）。

深度网络中存在这个问题，在很长序列上的循环网络也存在这个问题。在这两种情况下，反馈信号的传播都必须通过一长串操作。我们已经知道 LSTM 层是如何在循环网络中解决这个问题的：它引入了一个**携带轨道**（carry track），可以在与主处理轨道平行的轨道上传播信息。残差连接在前馈深度网络中的工作原理与此类似，但它更加简单：它引入了一个纯线性的信息携带轨道，与主要的层堆叠方向平行，从而有助于跨越任意深度的层来传播梯度。

In [ ]: