函数式API能够多次重复使用一个层实例。如果你对一个层实例调用两次，而不是每次调用都实例化一个新层，那么每次调用可以重复使用相同的权重。这样你可以构建具有共享分支的模型，即几个分支全都共享相同的知识并执行相同的运算。也就是说，这些分支共享相同的表示，并同时对不同的输入集合学习这些表示。

举个例子，假设一个模型想要评估两个句子之间的语义相似度。这个模型有两个输入（需要比较的两个句子），并输出一个范围在 0~1 的分数，0 表示两个句子毫不相关，1 表示两个句子完全相同或只是换一种表述。这种模型在许多应用中都很有用，其中包括在对话系统中删除重复的自然语言查询。

在这种设置下，两个输入句子是可以互换的，因为语义相似度是一种对称关系，A 相对于 B 的相似度等于 B 相对于 A 的相似度。因此，学习两个单独的模型来分别处理两个输入句子是没有道理的。相反，你需要用一个 LSTM 层来处理两个句子。这个 LSTM 层的表示（即它的权重）是同时基于两个输入来学习的。我们将其称为连体 LSTM（Siamese LSTM）或共享 LSTM（shared LSTM）模型。

In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [4]:

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

In [5]:

```
import numpy as np
```

In [6]:

```
from tensorflow.keras import Model
```

In [7]:

```
lstm = layers.LSTM(32)
```

将一个 LSTM 层实例化一次，左右两个分支使用同一个权值

一个层实例可能被多次重复使用，它可以被调用任意多次，每次都重复使用一组相同的权重，会重复使用这个实例已经学到的表示

In [8]:

```
left_input = Input(shape=(None, 128))
left_output = lstm(left_input)
```

In [9]:

```
right_input = Input(shape=(None, 128))
right_output = lstm(right_input)
```

In [11]:

```
merged_data = layers.concatenate([left_output, right_output], axis=-1)
```

In [12]:

```
pred = layers.Dense(1, activation='sigmoid')(merged_data)
```

In [13]:

```
model = Model([left_input, right_input], pred)
```

In [ ]:

```
model.fit([left_data, right_data], targets)
```

通过重复使用模型实例可以构建一个简单的例子，就是一个使用双摄像头作为输入的视觉模型：两个平行的摄像头，相距几厘米（一英寸）。这样的模型可以感知深度，这在很多应用中都很有用。你不需要两个单独的模型从左右两个摄像头中分别提取视觉特征，然后再将二者合并。这样的底层处理可以在两个输入之间共享，即通过共享层（使用相同的权重，从而共享相同的表示）来实现。

In [14]:

```
from tensorflow.keras.applications import Xception
```

In [15]:

```
xception_base = Xception(weights=None, include_top=False)
```

In [18]:

```
xception_base.layers
```

```
8f3630>,
 <tensorflow.python.keras.layers.normalization.BatchNormalization at 0x20
0d3970c50>,
 <tensorflow.python.keras.layers.core.Activation at 0x200cebb0eb8>,
 <tensorflow.python.keras.layers.convolutional.SeparableConv2D at 0x200d3
97af60>,
 <tensorflow.python.keras.layers.normalization.BatchNormalization at 0x20
0d39e8e10>,
 <tensorflow.python.keras.layers.core.Activation at 0x200d3a08710>,
 <tensorflow.python.keras.layers.convolutional.SeparableConv2D at 0x200d3
a487b8>,
 <tensorflow.python.keras.layers.normalization.BatchNormalization at 0x20
0d3ca8828>,
 <tensorflow.python.keras.layers.merge.Add at 0x200d3cb1cf8>,
 <tensorflow.python.keras.layers.core.Activation at 0x200d38f3400>,
 <tensorflow.python.keras.layers.convolutional.SeparableConv2D at 0x200d3
cb1cc0>,
 <tensorflow.python.keras.layers.normalization.BatchNormalization at 0x20
0d3d21eb8>,
```

In [19]:

```
left_input = Input(shape=(250, 250, 3))
right_input = Input(shape=(250, 250, 3))
```

In [20]:

```
left_features = xception_base(left_input)
right_features = xception_base(right_input)        # 对相同的视觉模型调用两次
merged_features = layers.concatenate([left_features, right_features], axis=-1)
```

In [22]:

```
# 合并后的特征包含来自左右两个视觉输入中的信息，这时候变成一个合并后的特征
# 不用单独训练两个视觉特征模型取提取左右视觉输入的信息后再合并
```

In [ ]:

```
```