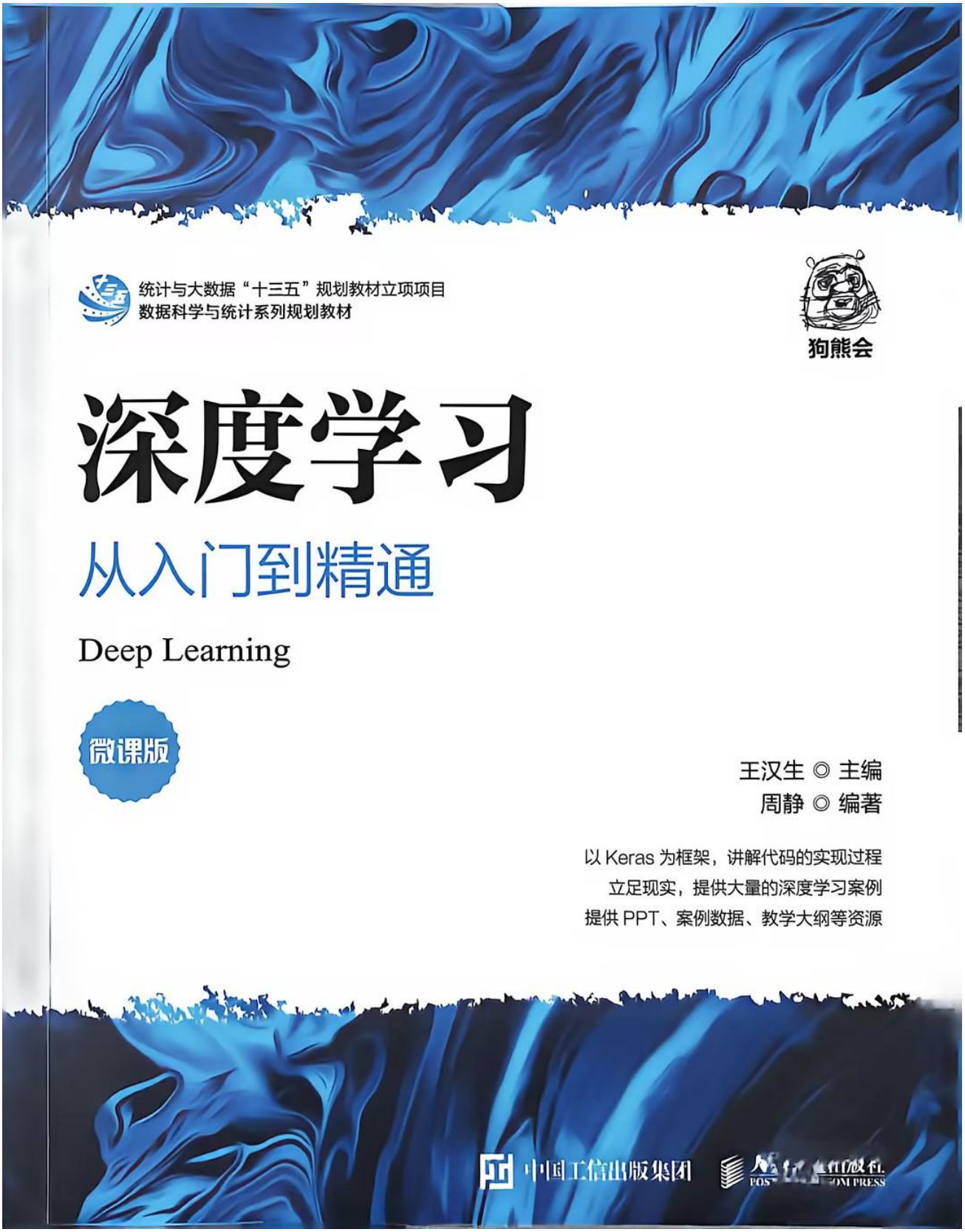
大数据专业教师岗试讲具体内容



输出结果为：

```
Epoch 1/20
10/10 [==============================] - 15s 1s/step - loss: 3.9327 - acc: 0.0499 - val_loss: 5.5232 - val_acc: 0.0613
Epoch 2/20
10/10 [==============================] - 3s 294ms/step - loss: 3.5016 - acc: 0.1187 - val_loss: 7.1849 - val_acc: 0.0882
Epoch 3/20
10/10 [==============================] - 3s 329ms/step - loss: 3.0862 - acc: 0.1648 - val_loss: 8.6394 - val_acc: 0.0809
Epoch 4/20
10/10 [==============================] - 3s 347ms/step - loss: 2.6561 - acc: 0.2314 - val_loss: 10.2452 - val_acc: 0.0907
Epoch 5/20
10/10 [==============================] - 3s 331ms/step - loss: 2.4108 - acc: 0.2788 - val_loss: 8.9433 - val_acc: 0.0980
Epoch 6/20
10/10 [==============================] - 4s 356ms/step - loss: 2.0467 - acc: 0.3647 - val_loss: 6.4456 - val_acc: 0.1593
Epoch 7/20
10/10 [==============================] - 4s 351ms/step - loss: 1.7387 - acc: 0.4452 - val_loss: 6.3859 - val_acc: 0.1446
Epoch 8/20
10/10 [==============================] - 3s 348ms/step - loss: 1.5493 - acc: 0.5028 - val_loss: 5.4671 - val_acc: 0.2059
Epoch 9/20
10/10 [==============================] - 4s 354ms/step - loss: 1.3673 - acc: 0.5316 - val_loss: 5.5319 - val_acc: 0.2132
Epoch 10/20
10/10 [==============================] - 3s 333ms/step - loss: 1.0962 - acc: 0.6415 - val_loss: 4.6930 - val_acc: 0.2623
Epoch 11/20
10/10 [==============================] - 4s 350ms/step - loss: 0.9661 - acc: 0.6870 - val_loss: 4.3933 - val_acc: 0.3162
Epoch 12/20
10/10 [==============================] - 3s 342ms/step - loss: 0.7762 - acc: 0.7569 - val_loss: 4.5305 - val_acc: 0.3088
Epoch 13/20
10/10 [==============================] - 3s 350ms/step - loss: 0.6023 - acc: 0.8101 - val_loss: 3.9272 - val_acc: 0.3431
Epoch 14/20
10/10 [==============================] - 3s 330ms/step - loss: 0.5423 - acc: 0.8203 - val_loss: 3.4937 - val_acc: 0.3897
Epoch 15/20
10/10 [==============================] - 3s 328ms/step - loss: 0.3840 - acc: 0.8784 - val_loss: 3.6174 - val_acc: 0.3946
Epoch 16/20
10/10 [==============================] - 3s 338ms/step - loss: 0.3868 - acc: 0.8781 - val_loss: 3.0238 - val_acc: 0.4485
Epoch 17/20
10/10 [==============================] - 3s 344ms/step - loss: 0.3547 - acc: 0.8861 - val_loss: 3.1737 - val_acc: 0.4363
Epoch 18/20
10/10 [==============================] - 4s 353ms/step - loss: 0.2760 - acc: 0.9054 - val_loss: 2.8368 - val_acc: 0.4510
Epoch 19/20
10/10 [==============================] - 3s 346ms/step - loss: 0.2576 - acc: 0.9178 - val_loss: 2.7349 - val_acc: 0.4583
Epoch 20/20
10/10 [==============================] - 3s 326ms/step - loss: 0.2117 - acc: 0.9384 - val_loss: 2.6844 - val_acc: 0.4853
```

图 6.7 Inception V1 模型训练结果

最后总结一下，Inception V1 通过增加网络的宽度来提升训练效果，主要有两个创新点，一是使用不同尺寸的卷积核来提炼更多丰富的细节，二是大量使用小卷积核来达到降维的目的。

6.2 ResNet

ResNet（Residual Neural Network）是由微软研究院何恺明等人提出的，该算法获得了 2015 年大规模视觉识别挑战赛的冠军[①]。不仅如此，在 ImageNet Detection、ImageNet Localization、COCO Detection 等多项竞赛中，该模型也都获得过冠军。截止到本书写作为止，提出 ResNet 的文章已有约两万次的引用。有评价说，ResNet 是过去几年中计算机视觉和深度学习领域最具开创性的工作，影响了这之后深度学习在学术界和工业界的发展方向。

① He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition.

6.2.1 ResNet 网络结构

在讲解 ResNet 网络结构之前，先介绍 ResNet 中的重要结构——残差学习模块。

1．ResNet 的残差学习模块

ResNet 声名鹊起的一个很重要的原因是，它提出了残差学习的思想。图 6.8 为 ResNet 的一个残差学习模块，该模块包含多个卷积层，多个卷积层对这个残差学习模块的输入数据 X 进行 $f(X)$ 的变化，同时原始输入信息跳过多个卷积层直接传导到后面的层中，最终将 $f(X)+X$ 的整体作为输入，并用激活函数激活，从而得到这个残差学习模块的输出结果。所以 $f(X)$ 本质上是输出结果和输入结果之间的差值，即残差。ResNet 学习的就是 $f(X)$，因此 ResNet 又叫作残差网络。

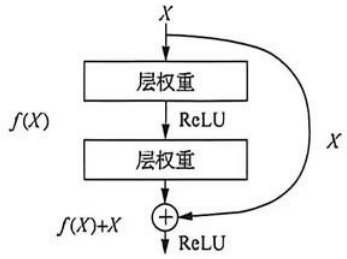


图 6.8　原论文中对残差学习模块的图解

2．残差学习模块的优势

传统的卷积神经网络或者全连接网络，在信息传递时，或多或少会存在信息丢失、损耗等问题，同时还会导致梯度消失或梯度爆炸，使得很深的网络无法训练。ResNet 通过提出残差学习的思想，在一定程度上解决了这个问题。通过将输入信息 X “绕道”传导到输出，极大保护了信息的完整性，整个网络只需要学习输入、输出和残差部分，即 $f(X)$，就能简化学习的目标和难度。

以图 6.9 为例，最左边是 19 层的 VGG，中间是 34 层的普通神经网络，最右边是 34 层的 ResNet。将三者对比发现，ResNet 与其他两个网络结构最大的区别是有很多的旁路将输入直接连接到后面的层，这种结构也被称为 shortcuts，每一个 shortcuts 连线中间包含的是一个残差学习模块。

3．ResNet 中常用的残差学习模块

图 6.10 所示为 ResNet 网络结构中常用的两种残差学习模块，左边是由两个 3×3 卷积网络串联在一起作为一个残差学习模块，右边是以 1×1、3×3、1×1 3 个卷积网络串联在一起作为一个残差学习模块。ResNet 大都是以这两种学习模块堆叠在一起实现的。比较常见的 ResNet 有 50 层、101 层和 152 层。表 6.1 列出了 ResNet 不同层数的网络结构。

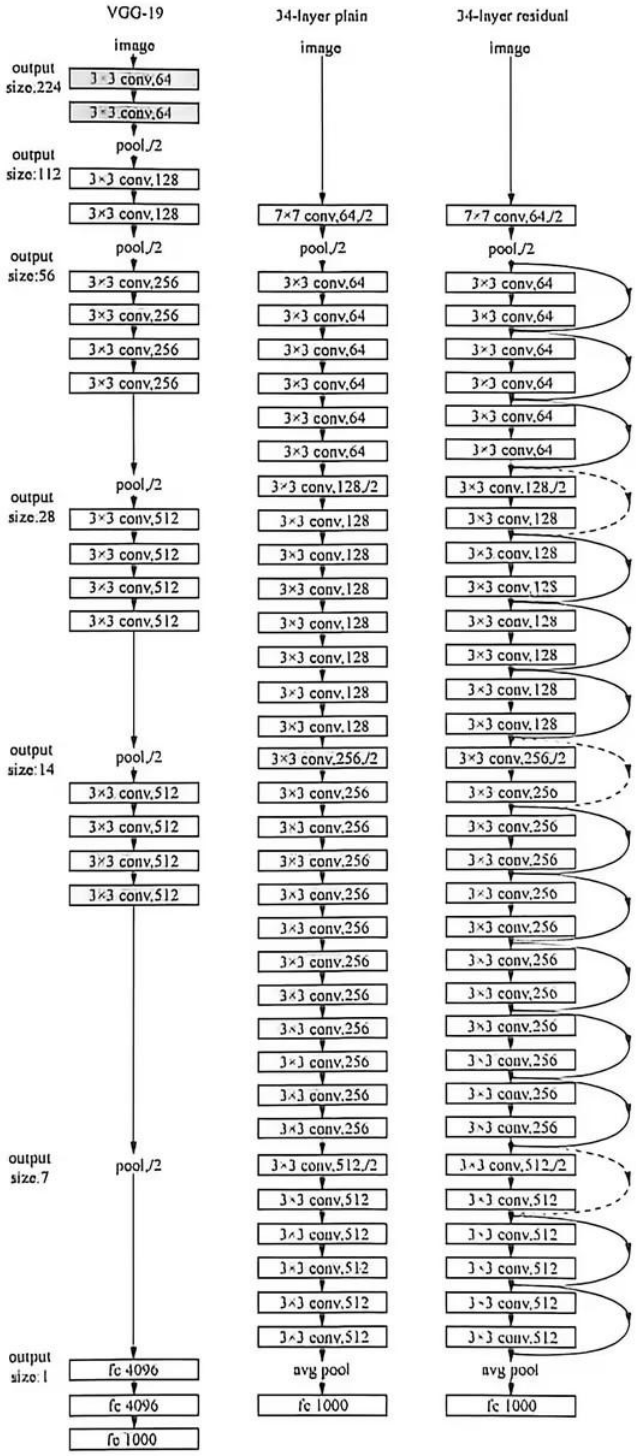


图 6.9　3 种神经网络结构的对比图

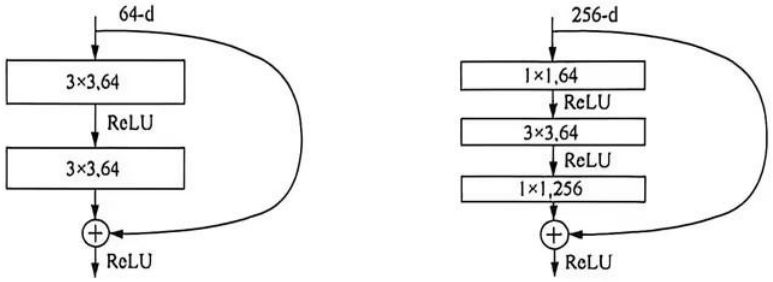


图 6.10　两种残差学习模块

表 6.1　原论文中不同层数的 ResNet 结构

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7,64,stride2				
conv2_x	56×56	3×3max pool,stride2				
		$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$
	1×1	average pool,1000-d fc,softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

4．ResNet 网络结构详解

下面以 34 层的 ResNet 为例，对照表 6.1，详解其网络结构。

（1）conv1 层。该层使用 64 个 7×7 的卷积核，步长为 2，将 224×224 大小的彩色图像降

维到112×112。

（2）conv2_x 层。首先进行3×3的最大值池化，步长为 2，将维度进一步降低为 56×56；然后是 3 个残差学习模块，每一个模块都由两个卷积层组成，卷积核大小为3×3，64 个通道。

（3）conv3_x 层。由 4 个残差学习模块组成。由于 conv2_3 的输出结果是56×56，因此在 conv3_1 的某一卷积层，需要将步长调整为 2，从而将 conv3_4 的输出维度降低到28×28。

（4）conv4_x 层。由 6 个残差学习模块组成。同理，在 conv4_1 中的某一卷积层需要将步长调整为 2，从而将 conv4_6 的输出维度降低到14×14。

（5）conv5_x 层。由 3 个残差学习模块组成。同理，在 conv5_1 将步长调整为 2，最后输出7×7维的图像。

（6）最后是一个全连接层，输出到 1 000 分类。

6.2.2 案例：花的三分类问题

1. 数据准备与处理

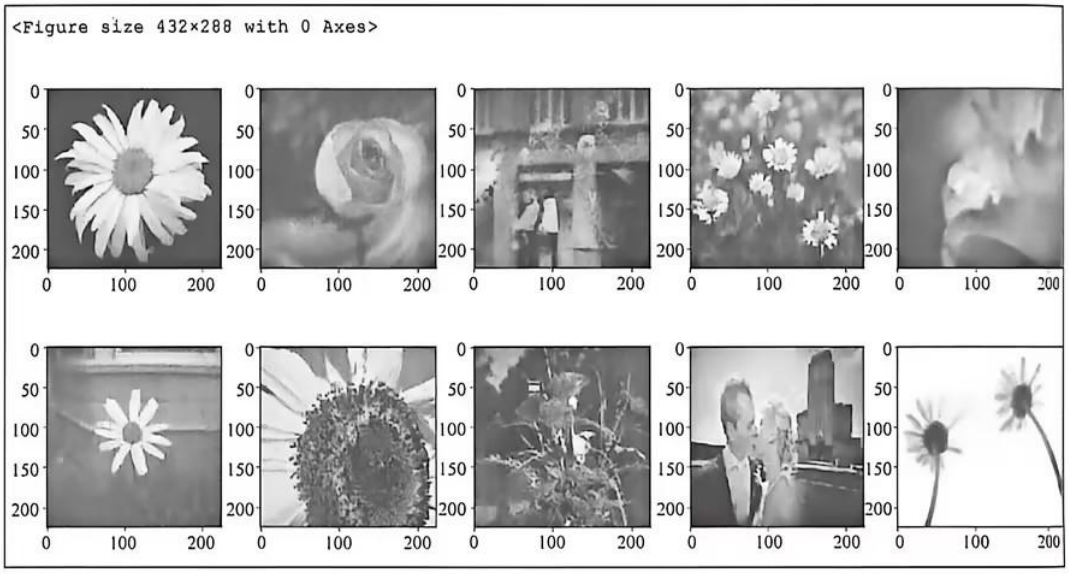
本节将通过花的三分类问题来演示 ResNet 的代码实现。数据集分别存放在 train 和 validation 两个文件夹中。首先使用 ImageDataGenerator 将数据读入并展示其中的 10 张图像，具体如代码示例 6-5 所示。

代码示例 6-5：读入数据并展示图像

```
from matplotlib import pyplot as plt
from keras.preprocessing.image import ImageDataGenerator

IMSIZE=224
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    'data_res/train',
    target_size=(IMSIZE, IMSIZE),
    batch_size=100,
    class_mode='categorical')

validation_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    './data_res/validation',
    target_size=(IMSIZE, IMSIZE),
    batch_size=100,
    class_mode='categorical')

plt.figure()
fig,ax = plt.subplots(2,5)
fig.set_figheight(7)
fig.set_figwidth(15)
```

```
ax=ax.flatten()

X,Y=next(train_generator)
for i in range(10): ax[i].imshow(X[i,:,:,:])
```

输出结果为：



2. ResNet 代码实现

扫一扫

ResNet 代码实现

首先构建残差学习模块之前的网络结构，具体如代码示例 6-6 所示。

代码示例 6-6：构建残差学习模块之前的网络结构

```
from keras.layers import Input
from keras.layers import Activation, Conv2D, BatchNormalization,
add, MaxPooling2D

NB_CLASS=3
IM_WIDTH=224
IM_HEIGHT=224
inpt = Input(shape=(IM_WIDTH, IM_HEIGHT, 3))
x = Conv2D(64, (7,7), padding='same', strides=(2,2), activation='relu')(inpt)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)
x0 = x
```

其中 Conv2D 函数设置第 1 个卷积层，采取 64 个 7×7 的卷积核进行 same 卷积，步长为 2，激活函数选择 ReLU，接下来是 BN 层，然后是池化层，采取 3×3 的卷积核进行 same 池化，步长为 2。x0 = x 表示把当前的 x 储存下来，之后调用原始输入信息时，可以使用 x0 。

接下来进入第 1 个残差学习模块，这里将1×1、3×3、1×1 3 个卷积网络串联在一起作为一个残差学习模块，参数和层数的设置如代码示例 6-7 所示。

代码示例 6-7：残差学习模块代码

```
# 一个 block
x = Conv2D(64, (1,1), padding='same', strides=(1,1), activation='relu')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3,3), padding='same', strides=(1,1), activation='relu')(x)
x = BatchNormalization()(x)
x = Conv2D(256, (1,1), padding='same', strides=(1,1), activation=None)(x)
x = BatchNormalization()(x)

# 下面两步为了把输入 64 通道的数据转换为 256 个通道，用来让 x0 和 x 维数相同，可以进行加法计算
x0 = Conv2D(256,(1,1),padding='same',strides=(1,1),activation='relu')(x0)
x0 = BatchNormalization()(x0)
x = add([x,x0])                # add 把输入的 x 和经过一个 block 之后输出的结果加在一起
x = Activation('relu')(x)      #求和之后的结果再做一次 relu
x0 = x
```

需要注意的是，ResNet 的每一个残差学习模块，最后一个卷积层没有激活函数，即 activation=None。由于最后一层的输出通道是 256，但 x0 是 64 通道，二者维度不一样，无法直接相加，因此需要先将它们的维度统一。于是使用 256 个大小为1×1的卷积核对 x0 进行卷积，之后使用 add 函数完成加法运算，即 x = add([x,x0])，求和之后的结果再做一次 ReLU 变换，这样就完成了第 1 个残差学习模块的代码编写。在进入下一个残差学习模块之前，仍然需要将输出的 x 存入 x0 当中，即 x0 = x 。

作为教学展示，我们构建一个包含 3 个残差学习模块的 ResNet，第 2 个、第 3 个残差学习模块和第 1 个非常类似，此处不再赘述。

课堂思考

依照代码示例 6-7，把第 2 个和第 3 个残差学习模块完成，从而构建一个完整的 ResNet 模型结构。

至此，ResNet 的网络搭建完成，可以用 model.summary 查看模型参数报表。由于层数太多，无法展示全部，这里仅展示部分结果，具体如代码示例 6-8 所示。

代码示例 6-8：ResNet 主体部分模型结构展示

```
from keras.models import Model
model = Model(inputs=inpt,outputs=x)
model.summary()
```