第七章 构建卷积神经网络模型识别多个目标对象

7.1 场景描述

组长又召集大家开会了，今天，组长给小武布置了任务，本小组需要给场景识别应用提供支持，要训练一个多目标对象的模型，以便在场景中对多个目标进行识别，组长要求小武先做 10 个目标识别的初始模型，将来再这上面进行优化。

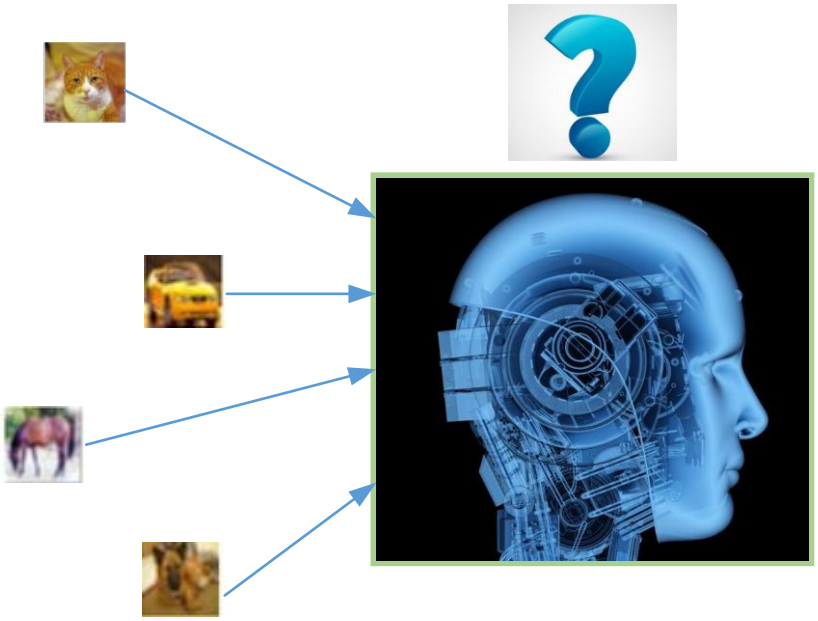

图 7-1 多目标的对象识别

7.2 任务描述

针对 10 种图像，建立一个卷积神经网络（CNN）训练模型，模型要求：对图像进行两次卷积运算，第一次使用 32 个卷集核进行卷积，第二次使用 64 个卷积核进行卷积，一个平坦层，一个隐藏层和一个输出层，采用搭建的模型对 CIFAR-10 图像集进行训练，并利用训练好的模型进行 10 类图像的分类和预测。

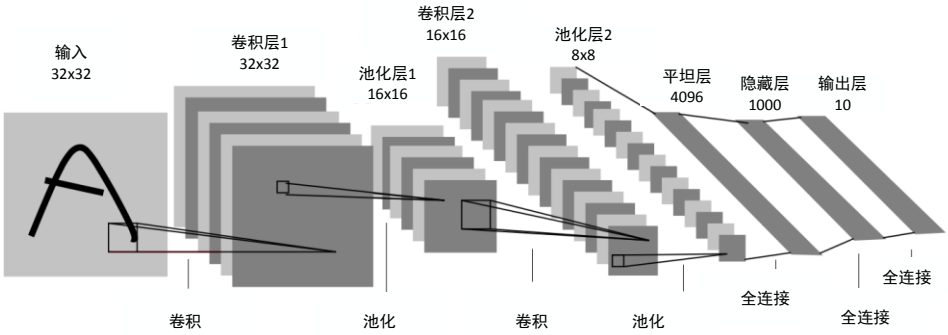

图 7-2 卷积神经网络训练与识别模型

7.3 任务分解

按照任务要求，我们卷积神经网络识别多目标的过程描述如图 7-3 所示。

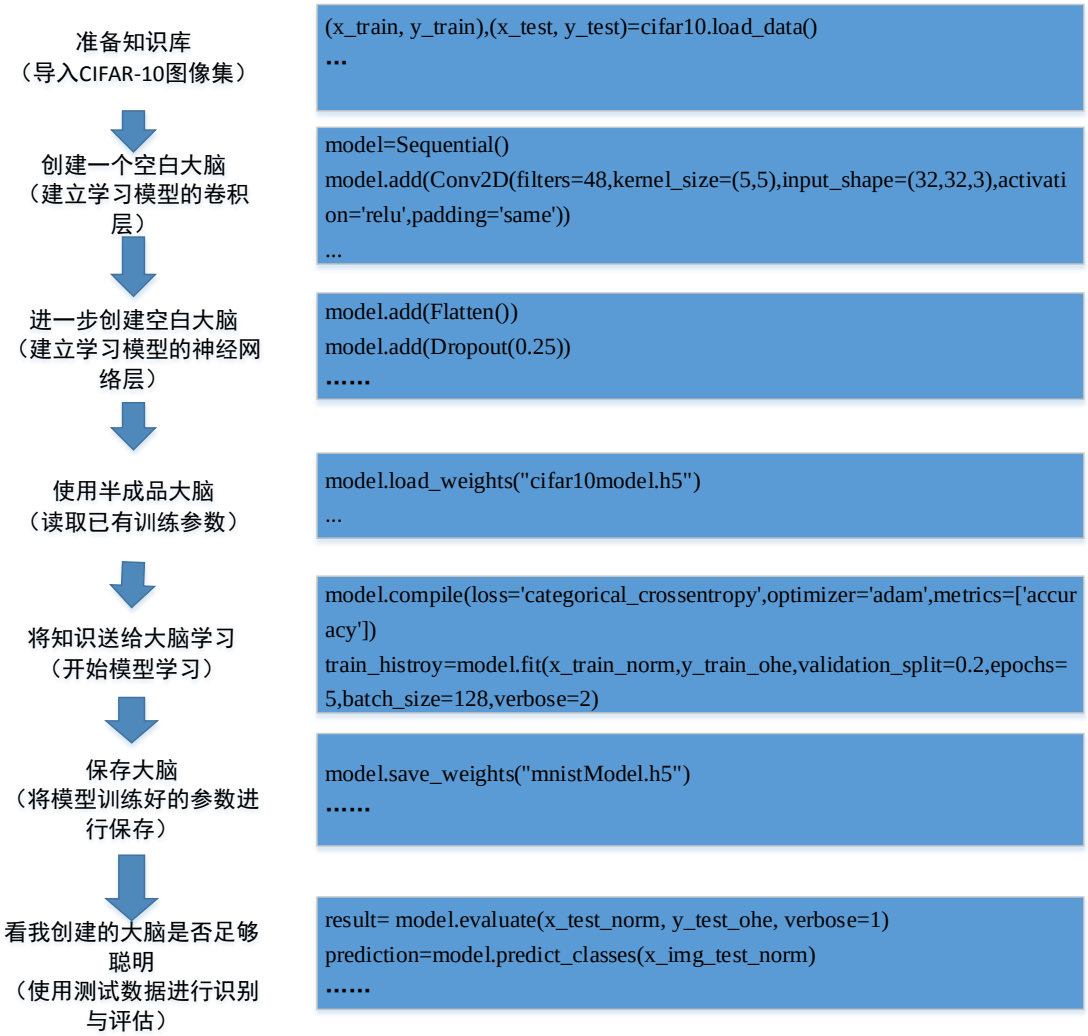

图 7-3 多图像目标识别的流程图

7.4 知识储备

7.4.1 CIFAR-10 图像集简介

该数据集共有 60000 张彩色图像，这些图像是 32*32，分为 10 个类，每类 6000 张图。这里面有 50000 张用于训练，构成了 5 个训练批，每一批 10000 张图；另外 10000 用于测试，单独构成一批。测试批的数据里，取自 10 类中的每一类，每一类随机取 1000 张。抽剩下的就随机排列组成了训练批。注意一个训练批中的各类图像并不一定数量相同，总的来看训练批，每一类都有 5000 张图。

下面这幅图就是列举了 10 个类，每一类展示了随机的 10 张图片：

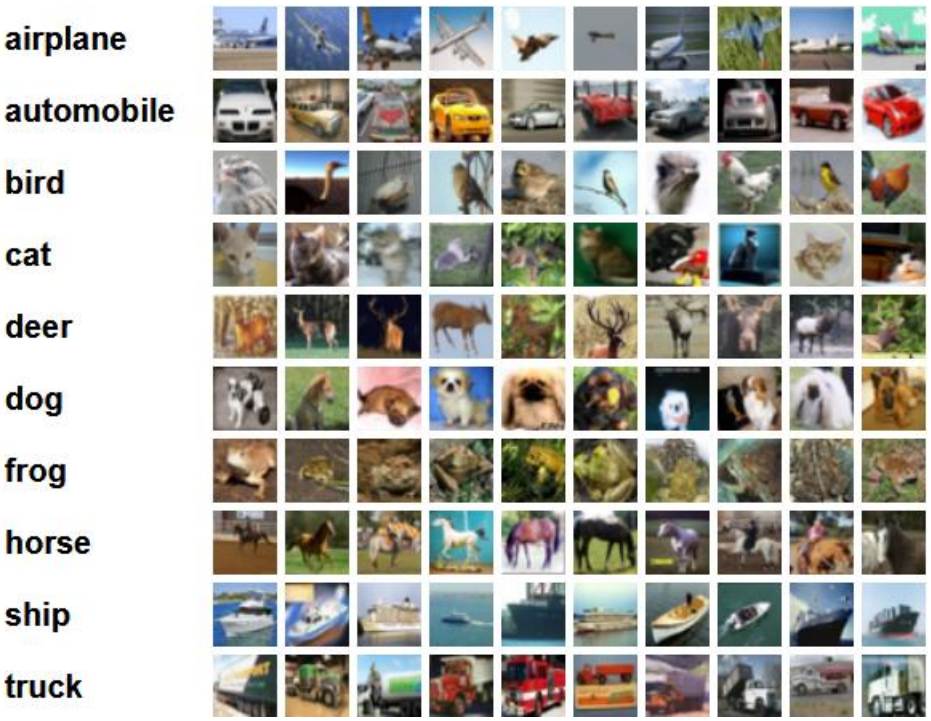

图 7-4 CIFAR-10 图像集中部分照片

需要说明的是，这 10 类都是各自独立的，不会出现重叠。数据的下载地址为：http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

7.4.2 彩色数字图像（RGB 图像）

在第四章中我们已经学习了什么叫灰度图像，但在现实生活中，我们常常看到的并不是灰度的图像，而是五彩斑斓的图像，我们把这种图像叫做彩色图像。在中学学习美术时，我们都知道红、黄、蓝是三原色。通过这三个颜色的颜料画在一起叠加即可组合成任意一种颜色。在计算机里，也类似，但三原色不再是红、黄、蓝，而是红（R）、绿（G)、蓝(B)。在计算机里通过控制这三个颜色的量组

合在一起，也可以合成任意一种颜色，于是就有了 RGB 图。RGB 图像里，每个像素点由三个数值控制颜色，分别对应红、绿、蓝的分量大小。范围一般也为 0~255 之间，0 表示这个颜色分量没有，255 表示这个颜色分量取到最大值。例如：某 RGB 图中一个像素点的红、绿、蓝分量均为 255，则根据光学叠加的理论可知，该点为纯白色。

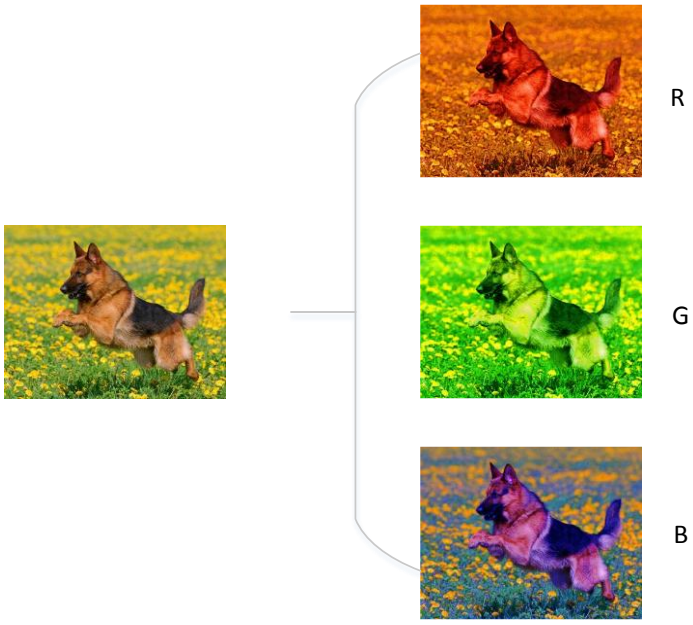

图 7-5 彩色图像的 RGB 三个图层

如图 7-5 所示，一副彩色数字图像实际上是由三幅图像组成的，分别是 R 图像、G 图像和 B 图像，前面我们已经知道，在计算机中，一副单色图像实际上就是一个二维数组，因此，一副彩色图像在计算机中就是由 3 个二维数组所存储的。

7.4.3 图像的卷积

卷积，有时也叫算子。用一个模板去和另一个图片对比，进行卷积运算。目的是使目标与目标之间的差距变得更大。卷积在数字图像处理中最常见的应用为锐化和边缘提取。

假设卷积核 h 为，

1	2	1
0	0	0
-1	-2	-1

图 7-6　卷积核 h

待处理的图像矩阵 x 为

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

图 7-7 待处理的图像矩阵

图像 x 与卷积核的卷积，首先将卷积核旋转 180°，则卷积核变为

-1	-2	-1
0	0	0
1	2	1

图 7-8 旋转 180°的卷积核

将卷积核 h 的中心对准 x 的第一个元素，然后 h 和 x 重叠的元素相乘，h 中不与 x 重叠的地方 x 用 0 代替，再将相乘后 h 对应的元素相加，得到结果矩阵中 Y 的第一个元素。如：

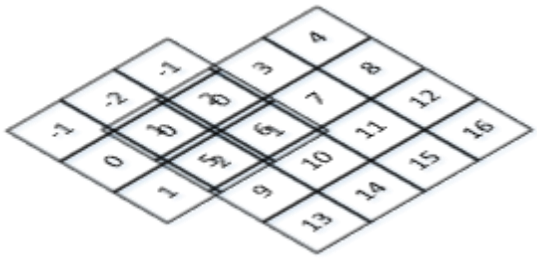

图 7-9 进行卷积运算

所以结果矩阵中的第一个元素

Y11 = -1 * 0 + -2 * 0 + -1 * 0 + 0 * 0 + 0 * 1 + 0 * 2 + 1 * 0 + 2 * 5 + 1 * 6 = 16

x 中的每一个元素都用这样的方法来计算，得到的卷积结果矩阵为

16	24	28	23
24	32	32	24
24	32	32	24
-28	-40	-44	-35

图 7-10　待处理图像进行卷积后的结果

给出一个更直观的例子，从左到右看，原像素经过卷积由 1 变成-8。

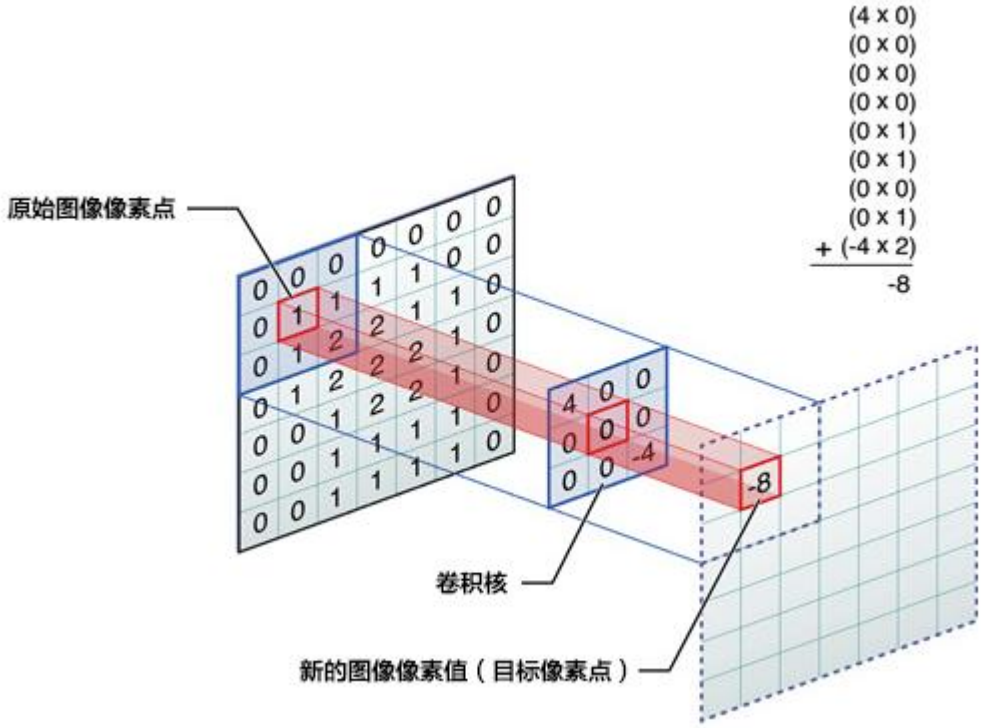

图 7-11 直观的图像卷积过程

通过滑动卷积核，就可以得到整张图片的卷积结果。

对于 RGB 彩色图像来说，原始图像的大小为 mxnx3，因此，相应的卷积核也为要相应改变，大小为 txtx3，如下图所示，原始图像的大小为 6x6x3，卷积核为 3x3x3 大小，6*6*3 分别代表 RGB 图像的高、宽、通道数；3*3*3 分别代表卷积核的高、宽、通道数

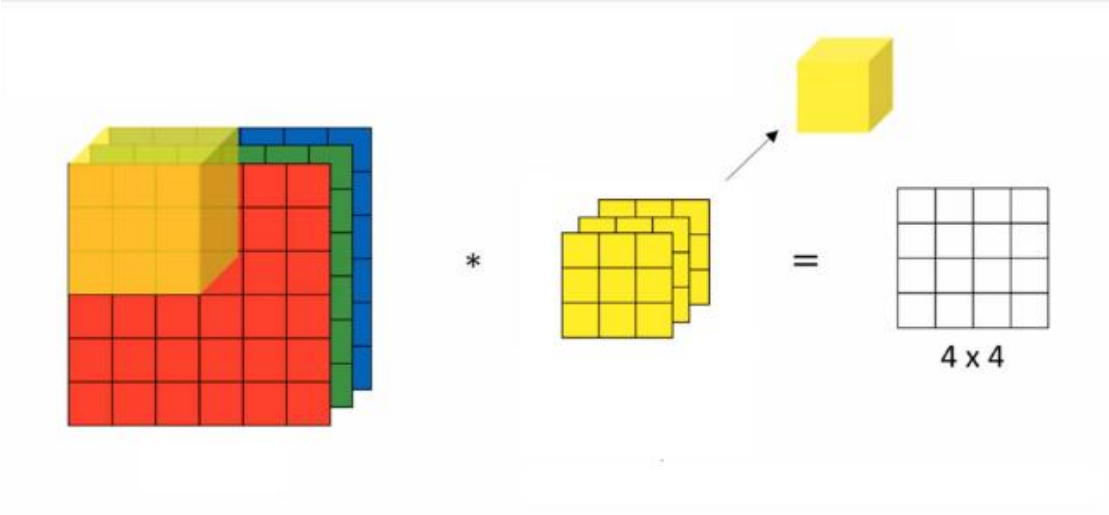

图 7-12　彩色图像的卷积

将 3*3*3 卷积核转换成立方体，一共 3^3=27 个数值。分别乘与滤波器对应的 RGB 图像三个通道的数值，再相加得到 6*6 输出矩阵的值。

7.5 任务实现步骤

按照任务流程，我们可以将该任务分解成如下几个子任务，依次完成：

第一步：准备知识库，导入 CIFAR-10 图像集，并对图像集进行处理以适应卷积神经网络模型（CNN）的输入数据格式要求。

第二步：创建空白大脑，建立卷积神经网络中卷积层模型。使用 keras 模型中的函数来建立卷积层模型，完成两层卷积层的构建。

第三步：进一步创建空白大脑，建立卷积神经网络中神经网络层模型。使用 keras 模型中的函数来建立神经网络层模型，完成平坦层、隐含层和输出层的构建。

第四步：读取半层品大脑，获取已有模型中的参数，在此基础上进行训练，使得大脑更加完善。

第五步：将 CIFAR-10 图像知识送给大脑学习，设置模型的训练参数，启动模型进行训练，并动态查看模型的训练状态。

第六步：保存学习好的大脑，将大脑模型中的参数进行保存，已方便将来在此基础上继续学习。

第七步：通过查看模型训练过程中的准确率和误差变化，了解大脑的学习过程和效果。

第八步：使用训练好的“大脑”模型，对 CIFAR 中的测试数据进行预测和识别。

7.5.1 创建 jupyter notebook 项目

（1）打开 jupyter notebook，如图 7-13 所示。

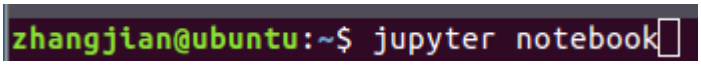

图 7-13 启动 jupyter notebook

（2）在 python3 下新建一个 notebook 项目，命名为 rask7-1，如图 7-14 所示。

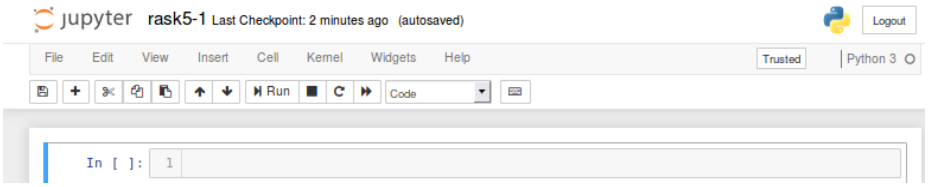

图 7-14 新建 notebook 项目示意图

7.5.2 处理 CIFAR-10 图像数据集

（1）在 jupyter notebook 中输入图 7-15 中显示的代码，并确认代码无错误。

```
from keras.utils import np_utils
import numpy as np
np.random.seed(10)
from keras.datasets import cifar10
(x_train, y_train),(x_test, y_test)=cifar10.load_data()
x_train_norm=x_train.astype('float32')/255
x_test_norm=x_test.astype('float32')/255
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
print(y_train[0:10])
y_train_ohe=np_utils.to_categorical(y_train)
y_test_ohe=np_utils.to_categorical(y_test)
```

图 7-15 处理 CIFAR-10 图像数据集代码

（2）按 Ctrl+Enter 组合键执行代码，代码没有错误提示后按 Shift+Enter 组合键新建下一个单元格，结果显示如图 7-16 所示：

```
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
[[6]
 [9]
 [9]
 [4]
 [1]
 [1]
 [2]
 [7]
 [8]
 [3]]
```

图 7-16 导入手写数字图像数据集后结果显示图

（3）下面对 CIFAR-10 图像集的下载和处理代码进行解析，以便更加了解该图像集。

- from keras.utils import np_utils
- import numpy as np
- np.random.seed(10)
- from keras.datasets import cifar10

导入所需的模块包

- (x_train, y_train),(x_test, y_test)=cifar10.load_data()

下载 CIFAR-10 图像数据集，并把数据分别保存在 x_train、y_train、x_test 和 y_test 四个变量中。

- print(x_train.shape)
- print(y_train.shape)
- print(x_test.shape)
- print(y_test.shape)

查看四个变量的大小。有输出结果可以看到，训练图像有 50000 张图片，每张图片的大小为 32*32 大小，每张图片是 RGB3 通道的彩色图片。同样测试图像有 10000 张图片，每张图片的大小为 32*32 大小，每张图片是 RGB3 通道的彩色图片。训练图像和测试图像对应的 label 都是一维的数组。

- print(y_train[0:10])

查看前 10 个训练图像对应的标签，由结果可以看到，前 10 个训练图像分别对应的类别为 6、9、9、4、1、1、2、7、8、3。该图像集中每个数字对应一个图像类别，具体的对应关系如下所示：

- y_train_ohe=np_utils.to_categorical(y_train)
- y_test_ohe=np_utils.to_categorical(y_test)

将对应的 label 数据进行 one-hot-encoding 编码。

7.5.3 建立学习模型的卷积部分

（1）在 jupyter notebook 中输入图 7-17 中显示的代码，并确认代码无错误。

```
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,ZeroPadding2D,Dropout
from keras.layers import Flatten,Dense
model=Sequential()
model.add(Conv2D(filters=48,kernel_size=(5,5),input_shape=(32,32,3),activation='relu',padding='same'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu',padding='same'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(2,2)))
```

图 7-17 建立 MLP 学习模型代码

（2）按 Ctrl+Enter 组合键执行代码确认代码正确无误。

（3）按下 Shift+Enter 组合键新建下一个单元格。

（4）下面对学习模型中的卷积层创建代码进行逐行解析，对整个模型的创建进行更深入的了解。

- from keras.models import Sequential
- from keras.layers import Conv2D,MaxPooling2D,ZeroPadding2D,Dropout
- from keras.layers import Flatten,Dense

导入所需的层的创建函数。

- model=Sequential()

建立贯序模型。

- model.add(Conv2D(filters=48,kernel_size=(3,3),input_shape=(32,32,3),activation='relu',padding='same'))

为模型添加卷积层 1。

其参数解析如下表 7-1。

表 7-1 卷积层 1 构建代码解析

filters=48	表示建立 48 个卷积核，即 48 个滤波器。
kernel_size=(5,5)	卷积核大小为 5x5
padding='same'	代表保留边界处的卷积结果，输出 shape 与输入 shape 相同
input_shape=(32,32,3)	代表 32*32 的彩色图像,当使用该层作为第一层时，应提供 input_shape 参数
activation='relu'	采用 relu 激励函数，CNN 采用的激励函数一般为 ReLU(The Rectified Linear Unit/修正线性单元)，它的特点是收敛快，求梯度简单，但较脆弱

由于有 48 个卷积核，因此，每张图片和一个卷积核进行计算，会得到一个 32*32 的图片，48 个卷积核计算完，会得到 48 个 32*32 的图片，因此，进行第一层卷积运算后，每张彩色图片变成了 48 个 32*32 的图片，这个时候，可以把该结果看成是一个有 48 层的 32*32 大小的照片，进行下一次卷积计算。

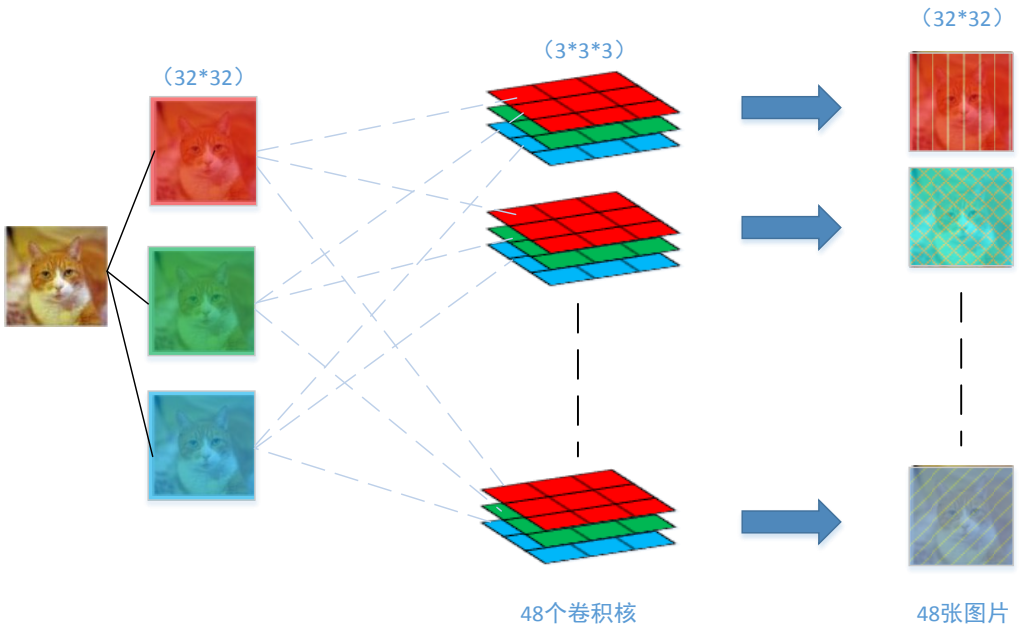

图 7-18 第一次卷积的过程示意图

- model.add(Dropout(0.25))

构建 dropout 层，每次计算随机丢弃 25%的神经元。

- model.add(MaxPooling2D(pool_size=(2,2)))

构建池化层 1，将 2x2 的数变成一个数。池化层实际上是对图像进行降尺寸，即把原来的图像变小。pool_size 决定了所见尺寸，pool_size=(2,2)即把图像的长

缩减一倍，图像的宽缩减一倍。之前的图像是 32*32 的图像，缩减后变成了 16*16 的图像了。

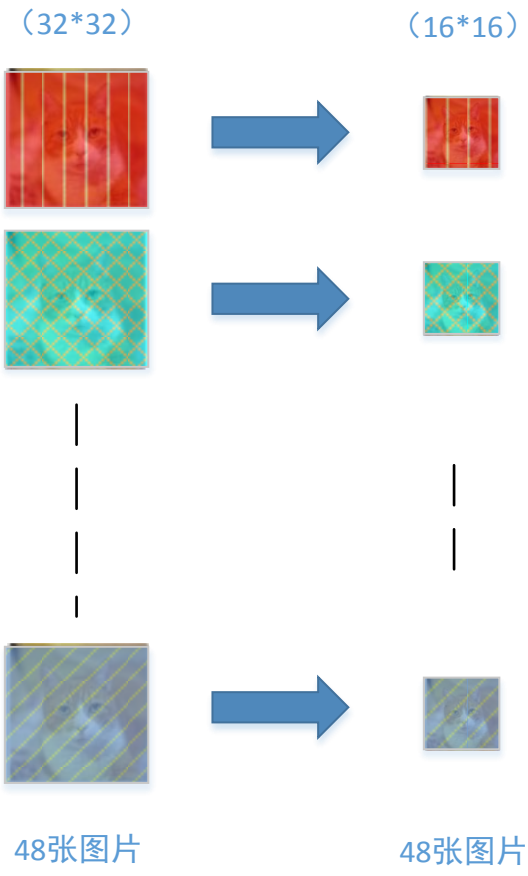

图 7-19 池化以后的图片

- model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu',padding='same'))
- model.add(Dropout(0.25))
- model.add(MaxPooling2D(pool_size=(2,2)))

创建第二层卷积层，卷积核个数为 64 个，卷积核尺寸为 3*3，采用 relu 激励函数，输出图像尺寸和原尺寸保持相同。

表 7-2 卷积层 2 构建代码解析

filters=64	表示建立 64 个卷积核，即 64 个滤波器。
kernel_size=(3,3)	卷积核大小为 3x3
padding='same'	代表保留边界处的卷积结果，输出 shape 与输入 shape 相同
activation='relu'	采用 relu 激励函数，CNN 采用的激励函数一般为 ReLU(The Rectified Linear Unit/修正线性单元)，它的特点是收敛快，求梯度

	简单，但较脆弱

构建 dropout 层，每次计算随机丢弃 25%的神经元。构建池化层 1，将 2x2 的数变成一个数。之前的图像是 16*16 的图像，缩减后变成了 8*8 的图像了。

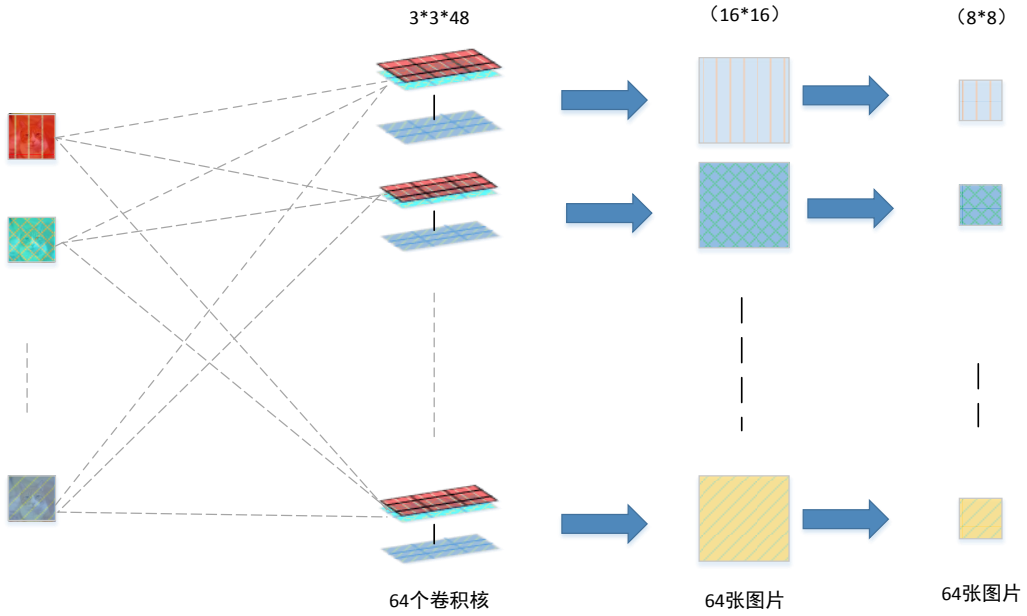

图 7-20　第二次卷积与池化示意图

7.5.4 建立学习模型的神经网络部分

（1）在 jupyter notebook 中输入图 7-21 中显示的代码，并确认代码无错误。

```
model.add(Flatten())
model.add(Dropout(0.25))

model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(10,activation='softmax'))

print(model.summary())
```

图 7-21 建立模型的神经网络代码

（2）按 Ctrl+Enter 组合键执行代码，代码没有错误提示后按 Shift+Enter 组合键新建下一个单元格，结果显示如图 7-22 所示：

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 48)	3648
dropout_1 (Dropout)	(None, 32, 32, 48)	0
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 48)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	27712
dropout_2 (Dropout)	(None, 16, 16, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 64)	0
flatten_1 (Flatten)	(None, 4096)	0
dropout_3 (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 1000)	4097000
dropout_4 (Dropout)	(None, 1000)	0
dense_2 (Dense)	(None, 10)	10010

Total params: 4,138,370
Trainable params: 4,138,370
Non-trainable params: 0

None

图 7-22 模型的打印结果

（3）下面对学习模型中的神经网络部分创建代码进行逐行解析，对整个模型的创建进行更深入的了解。

- model.add(Flatten())

构建平坦层，将池化层后的数据转化为一维数组。平坦层会将上一层的多维数据转换为一维数据。上层池化层池化后数据维度为 64x8x8,即每副照片经过卷积层 2 和池化层 2 以后，64 个过滤器得到 64 副照片，每幅照片规格大小为 8x8。经过平坦层后一副图像转换为 1*4096 长度的一维数组。

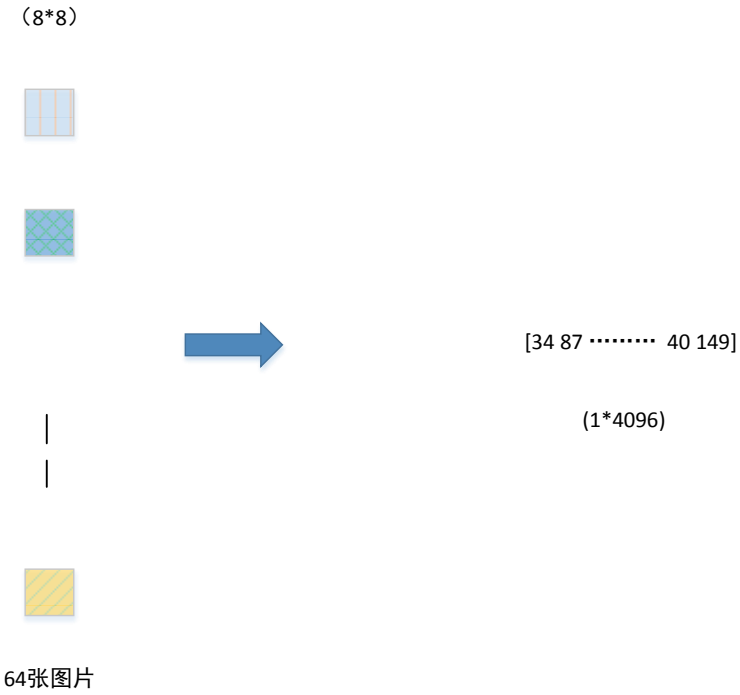

图 7-23　经过平坦层后数据的变化

- model.add(Dropout(0.25))

构建 dropout 层，每次计算随机丢弃 25%的神经元。

- model.add(Dense(1000,activation='relu'))

构建隐藏层，全连接结构，神经元个数 1024 个，初始化权重为默认值(kernel_initializer='glorot_uniform')，激活函数为 relu。

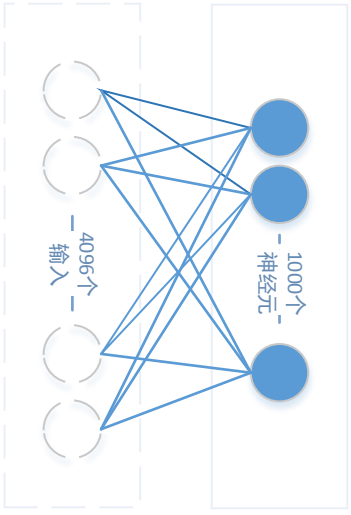

图 7-24 隐藏层示意图

- model.add(Dropout(0.25))

构建 dropout 层，每次计算随机丢弃 25%的神经元

- model.add(Dense(10,activation='softmax'))

构建输出层，全连接结构，神经元个数 10 个，初始化权重为默认值（kernel_initializer='glorot_uniform'），激活函数为 softmax。

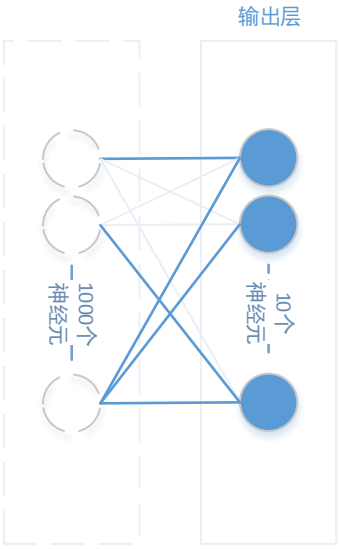

图 7-25 输出层示意图

- print(model.summary())

打印模型结构，其输出入图所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 48)        3648
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 48)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 48)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        27712
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_4 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10010
=================================================================
Total params: 4,138,370
Trainable params: 4,138,370
Non-trainable params: 0
_________________________________________________________________
None
```

图 7-26 模型各层的名称与参数

打印出模型概况，如图 5-12 所示。它实际调用的是 keras.utils.print_summary。

conv2d_1 为卷积层，有 3648 个参数，因为该卷积层有 48 个卷积核，每个卷积核大小为 5*5*3，即总参数为 48*（75+1）=3648 个。同理，conv2d_2 为卷积层，有 27712 个参数，因为该卷积层有 64 个卷积核，每个卷积核大小为 3*3*48，即总参数为 64*（432+1）=27712 个。

dense_1 为隐藏层，有 4097000 个参数，因为输入层有 4096 个单元，隐藏层有 1000 个单元，按照全连接模式，一共需要（4096+1）x1000=785000 个权重参数进行训练。dense_2 为输出层，按照全连接模式，一共有参数（1000+1）x10=10010 个参数。

整个模型参数一共有 4138370 个参数需要通过数据集进行训练获得。此外，在卷积层之间还有 dropout 层（dropout_1 和 dropout_2）和池化层（max_pooling2d_1 和 max_pooling2d_1）、平坦层、隐含层和输出层之间还有

dropout 层（dropout_3 和 dropout_4），由于 dropout 层只随机丢弃神经元，不需要权重参数，因此，权重参数个数均为 0，池化层只需要缩减尺寸，也不需要参数。

7.5.5 读取已有模型的参数

（1）在 jupyter notebook 中输入图 7-28 中显示的代码，并确认代码无错误。

```
try:
    model.load_weights("cifar10model.h5")
    print("成功加载已有模型,继续训练该模型")
except:
    print("没有模型加载，开始训练新模型")
```

图 7-28 读取已有模型参数的代码

（2）按 Ctrl+Enter 组合键执行代码。如果该路径下存在已经训练过的模型参数，显示如图 7-29，

```
没有模型加载，开始训练新模型
```

图 7-29 无模型加载时的显示

如果该路径下还没有已经训练过的模型参数，则如图 7-30 显示

```
没有模型加载，开始训练新模型
```

图 7-30 有模型加载时的显示

（3）按下 Shift+Enter 组合键新建下一个单元格。

（4）代码解析

- model.load_weights("cifar10model.h5")

keras 源码 engine 中 toplogy.py 定义了加载权重的函数:load_weights(self, filepath, by_name=False)，其中默认 by_name 为 False,这时候加载权重按照网络拓扑结构加载,适合直接使用 keras 中自带的网络模型。本网络模型中有三层结构，一共有 4138370 个参数，调用此函数实际上就是把这 4138370 个权重参数赋了初始值，这个初始值就是之前训练过的模型权重参数值。

7.5.6 对模型进行训练

（1）在 jupyter notebook 中输入图 7-31 中显示的代码，并确认代码无错误。

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
train_histroy=model.fit(x_train_norm,y_train_ohe,validation_split=0.2,epochs=5,batch_size=128,verbose=2)
```

图 7-31 对模型进行训练的代码

（2）按下 Ctrl+Enter 组合键,显示代码的运行结果

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/5
 - 139s - loss: 1.5332 - acc: 0.4447 - val_loss: 1.3438 - val_acc: 0.5411
Epoch 2/5
 - 138s - loss: 1.1900 - acc: 0.5783 - val_loss: 1.1635 - val_acc: 0.6151
Epoch 3/5
 - 139s - loss: 1.0317 - acc: 0.6380 - val_loss: 1.0715 - val_acc: 0.6477
Epoch 4/5
 - 139s - loss: 0.9193 - acc: 0.6753 - val_loss: 1.0080 - val_acc: 0.6785
Epoch 5/5
 - 139s - loss: 0.8136 - acc: 0.7136 - val_loss: 0.9289 - val_acc: 0.6904
```

图 7-32 模型训练过程中的准确率和误差动态数据

（3）按下 Shift+Enter 组合键新建下一个单元格。

（4）下面对模型进行训练的代码进行逐行解析，能对模型的学习设置和学习过程进行更深入的了解。

- model.compile（loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy']）

  调用 model.compile()函数对训练模型进行设置，参数设置为：

  loss='categorical_crossentropy'：loss（损失函数）设置为交叉熵模式，在深度学习中用交叉熵模式训练效果会比较好。

  optimizer='adam'：optimizer（优化器）设置为 adam，在深度学习中可以让训练更快收敛，并提高准确率。

  metrics=['accuracy']：评估模式设置为准确度评估模式。

- train_history=model.fit(x=x_train_norm,y=y_train_ohe,validation_split=0.2,epochs=5,batch_size=128,verbose=2)

调用 model.fit 配置训练参数，开始训练，并保存训练结果。

x=x_train_normalize：MNIST 数据集中已经经过预处理的训练集图像

y=y_label_ohe: MNIST 数据集中已经经过预处理的训练集 label

validation_split=0.2：训练之前将输入的训练数据集中 80%作为训练数据，20%作为测试数据。

epochs=5：设置训练周期为 5 次。

batch_size=128：设置每一次训练周期中，训练数据每次输入多少个。

verbose=2：设置成显示训练过程。

7.5.7 保存训练完的模型

（1）在 jupyter notebook 中输入图 7-33 中显示的代码，并确认代码无错误。

```
model.save_weights("cifar10Model.h5")
print("保存刚训练的模型")
```

图 7-33 处理手写数字图像数据集代码

（2）按 Ctrl+Enter 组合键执行代码，代码没有错误提示后按 Shift+Enter 组合键新建下一个单元格，结果显示如图 7-34 所示：

```
保存刚训练的模型
```

图 7-34 保存模型结果显示

（4）代码解析

- model.save_weights("cifar10model.h5")

保存模型的权重，可通过该函数利用 HDF5 进行保存。注意，在使用前需要确保你已安装了 HDF5 和其 Python 库 h5py。

model.save_weights(filepath)：将模型权重保存到指定路径，文件类型是 HDF5（后缀是.h5）。

7.5.8 显示模型准确率与误差

（1）在 jupyter notebook 中输入图 7-35 中显示的代码，并确认代码无错误。

```
import matplotlib.pyplot as plt
def  show_train(train_histroy,train,validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title("train history")
    plt.xlabel("train epoch")
    plt.ylabel(train)
    plt.legend(["train data","validation data"],loc="upper left")
    plt.show()
show_train(train_history,'acc','val_acc')
show_train(train_history,'loss','val_loss')
```

图 7-35 显示模型准确率与误差代码

（2）按下 Ctrl+Enter 组合键,显示代码的运行结果如图 7-36，并按下 Shift+Enter 组合键新建下一个单元格。

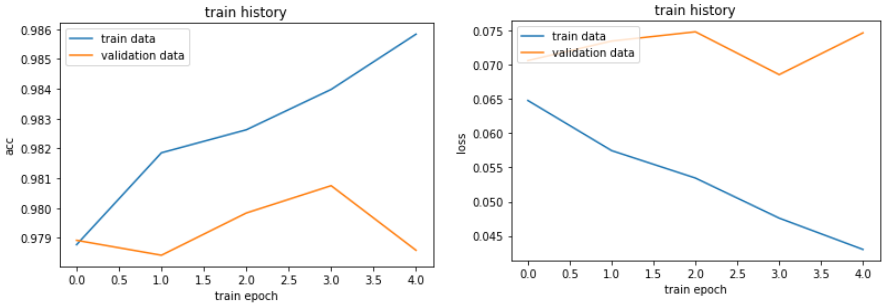

图 7-36 训练数据和测试数据准确率和误差变化曲线对比图

（3）显示模型准确率和误差代码的详细解析可参考第五章。

7.5.9 利用测试数据进行预测评估与识别

（1）在 jupyter notebook 中输入图 7-37 中显示的代码，并确认代码无错误。

```
scores= model.evaluate(x_test_norm, y_test_ohe, verbose=1)
prediction=model.predict_classes(x_img_test_norm)
prediction[0:10]
predicted_probability=model.predict(x_test_norm)
predicted_probability[0]
```

图 7-37 利用测试数据进行预测评估与识别代码

（2）按下 Ctrl+Enter 组合键,显示代码的运行结果，按下 Shift+Enter 组合键新建下一个单元格。

```
10000/10000 [==============================] - 1s 87us/step
acc= 0.9828

array([7, 2, 1, ..., 4, 5, 6])
```

图 7-38 测试数据进行预测的结果显示

（3）下面对使用测试数据进行预测的代码进行详细的解析，以便掌握使用模型进行预测的方法。

- result= model.evaluate(x_test_norm, y_test_ohe, verbose=1)

x_test_norm：输入数据位预处理后的测试数据集

y_test_ohe：标签为预处理后的测试标签集

- prediction=model.predict_classes(x_test)

对测试数据集进行预测，测试数据集是未经过归一化处理的数据。测试结果返回到 prediciton 变量中。

- prediction[0:10]

显示预测前 10 项结果，由图 7-38 中可以看到，预测结果保存为一维数组，显示预测到的对应图像的数字。

- predicted_probability=model.predict(x_test_norm)

显示预测概率，因为本结果有 10 中类别，该方法可以获得在每种类别下的预测概率值。

- predicted_probability[0]

显示第 1 副图像在每个类别下的预测概率。

7.6 项目代码完整示例

```
### rask5-1.py
#使用MNIST数据集
#采用MLP模型进行训练
#
#**************** 第一步：准备知识库，导入手写数字图像数据集************************
from keras.utils import np_utils #导入keras模块中的utils函数内容
import numpy as np           #导入numpy模块
#用于指定随机数生成时所用算法开始的整数值，不指定则每次随机数都一样
np.random.seed(10)
from keras.datasets import mnist #导入keras模块中的datasets函数内容
#下载mnist数据集
(x_train_image, y_train_label),(x_test_image, y_test_label)=mnist.load_data()
#将二维图像数据转换成一维向量并改变类型为浮点数类型
x_train=x_train_image.reshape(60000,784).astype('float32')
x_test=x_test_image.reshape(10000,784).astype('float32')
#将一维向量数据归一化使得数据处于[0,1]区间
x_train_norm=x_train/255
x_test_norm=x_test/255
#将测试数据进行one-hot-encode处理
y_train_ohe=np_utils.to_categorical(y_train_label)
y_test_ohe=np_utils.to_categorical(y_test_label)
print('第一张训练照片数据为：')
print(x_train_image[0])
print('第一张训练照片转换成一维向量后数据为：')
print(x_train[0])
print('前3个训练label进行one-hot-encoding转换后的数据为：')
print(y_train_ohe[:3])
```

```

#**************** 第二步：创建空白大脑，建立MLP学习模型************************
from keras.models import Sequential #导入keras模块中的models函数内容
from keras.layers import Dense#导入keras模块中的layers函数内容
model=Sequential() # 建立线性堆叠模型
model.add(Dense(units=256,input_dim=784,kernel_initializer='normal',activation='relu'))  #添加隐藏层
model.add(Dense(units=10,kernel_initializer='normal',activation='softmax')) #添加输出层
print(model.summary())#打印建立的模型内容
```

```

#**************** 第三步：将手写数字图像知识送给大脑学习，对模型进行训练************************
model.compile(loss='categorical_crossentropy',/
optimizer='adam',metrics=['accuracy']) #对训练模型进行参数设置
train_history=model.fit(x=x_train_normalize,/
    y=y_label_onddd,validation_split=0.2,/
    epochs=10,batch_size=200,verbose=2) #设置训练参数，并启动训练
```

```
43
44  #*************** 第四步：显示大脑的学习过程和效果，显示模型训练过程中的准确率和误差变化***
45  import matplotlib.pyplot as plt #导入matplotlib模块中的pyplot函数进行图形绘制
46  def  show_train(train_histroy,train,validation):  #定义训练准确率和误差绘制函数
47      plt.plot(train_history.history[train])
48      plt.plot(train_history.history[validation])
49      plt.show()
50  #绘制训练数据准确率变化曲线和测试数据准确率变化曲线对比图
51  show_train(train_history,'acc','val_acc')
52  #绘制训练数据误差变化曲线和测试数据误差变化曲线对比图
53  show_train(train_history,'loss','val_loss')
```

```
54
55  #*************** 第五步：看看我创建的大脑是否足够聪明，利用模型进行预测和识别**********
56  results=model.evaluate(x_test_norm,y_test_ohe)#利用测试数据对模型进行评估
57  print('acc=',results[1]) #打印测试数据进行评估的准确率
58  prediction=model.predict_classes(x_test) #使用训练好的模型对测试数据进行分类
59  prediction[:10] #打印前10个测试数据预测分类的结果
```

7.7 小结与应用

本节主要介绍了利用 CNN 卷积神经网络对彩色图像进行多分类的处理，训练过程中每次训练都会保存训练后的模型数据，下次使用时会检查是否有已经训练过的模型，如果有，则读取模型数据继续进行训练，如果没有，则重新训练一个新模型。

习题

1.简述彩色图像的卷积过程。

2.在本章训练模型中，改变模型的卷积层层数、卷积核个数、隐藏层层数、对应的 dropout 层配置、模型的损失函数、优化器等，记录测试数据进行预测的准确率。

测试序号	隐藏层层数	模型的损失函数	优化器	测试数据准确率评估
1	例如：2 层隐藏层 卷积层 1（32） 卷积层 2（28） 隐藏层 1（2000），dropout 层（0.5） 隐藏层 2（1800），dropout 层（0.4）	categorical_crossentropy	adam	
2				
3				

4				
5				

3.编写函数，显示指定图像以及该图像的真实值和图像在模型中的预测结果。