第四章 准备训练所用知识库

4.1 场景描述

小豪是某著名互联网+公司数据处理程序员，最近小豪所在的项目组接到任务，智能算法开发组需要手写数字图像数据集来做深度学习的训练，因此，需要小豪提供一个能用来进行深度学习的手写数字图像知识库。

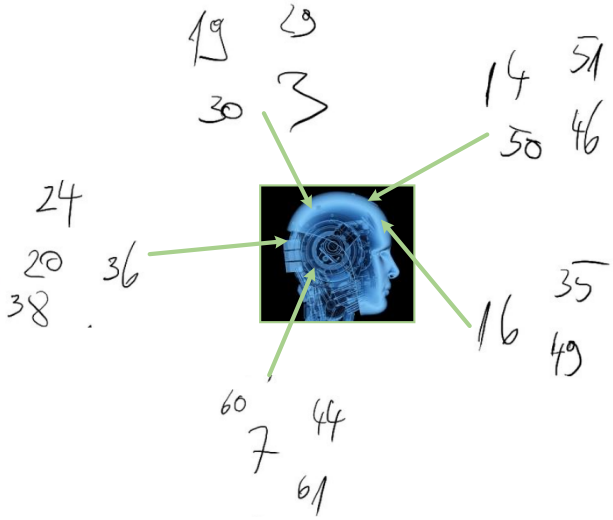


图 4-1 待学习的手写数字识别 AI

4.2 任务描述

上网下载通用的标准手写数字图像数据集，查看手写数字图像集对大小、组成结构、单个样本的图像显示以及单个样本的数值显示，并把样本分成训练数据集和测试数据集，对训练与测试数据集中对样本数据进行预处理，转换成能用来进行深度学习对数值数据集，对训练与测试数据集中对标签（label）数据进行预处理，以满足深度学习模型对格式需要。

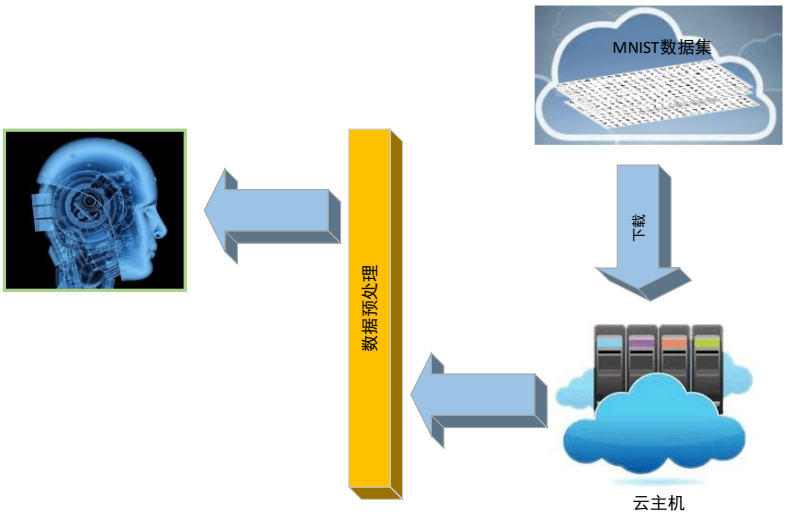


图 4-2 手写数字图像训练知识库准备过程

4.3 任务分解

按照任务要求，我们对手写数字识别大脑的创建过程描述如图 4-3 所示。

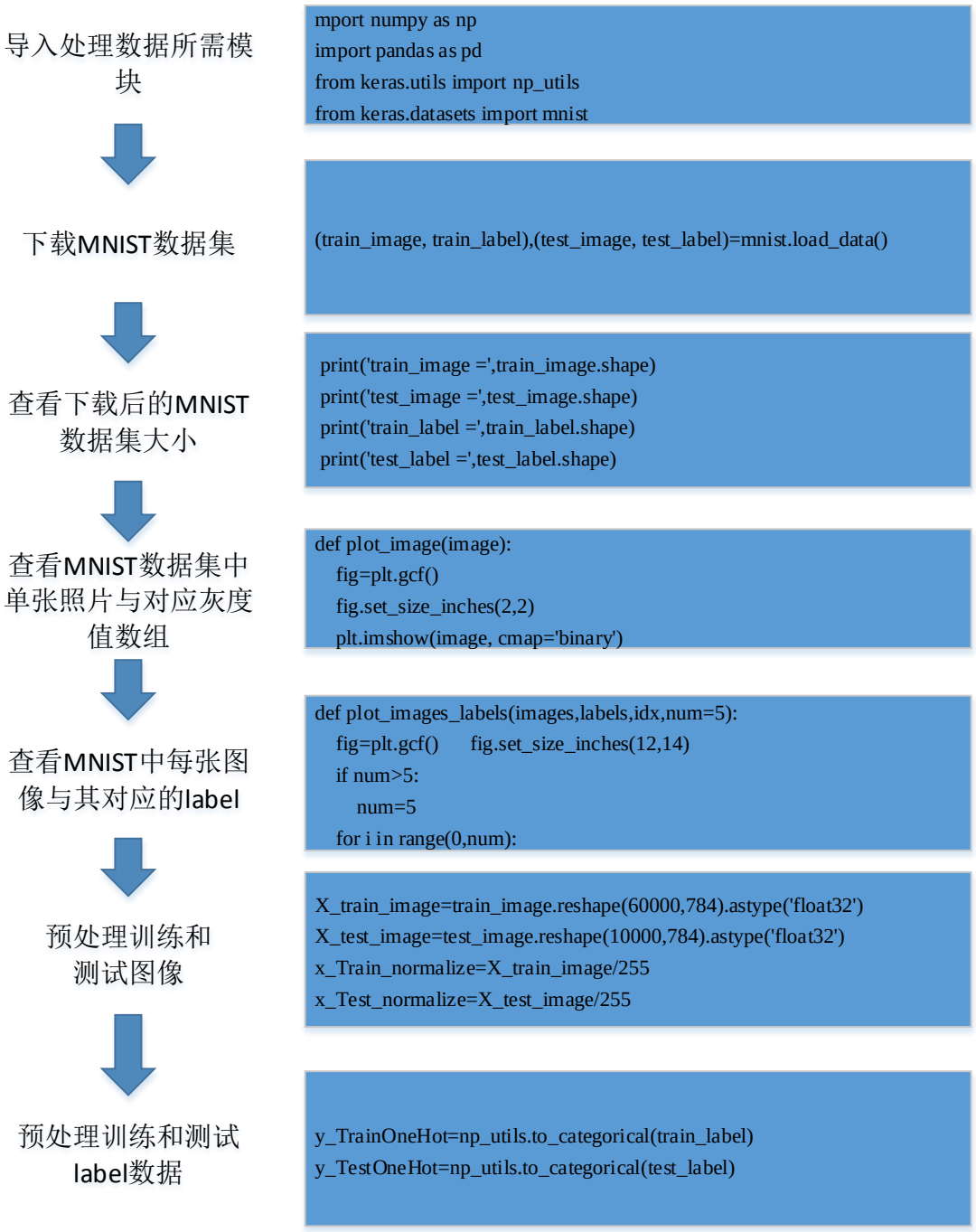


图 4-3 准备手写数字图像训练所有知识库

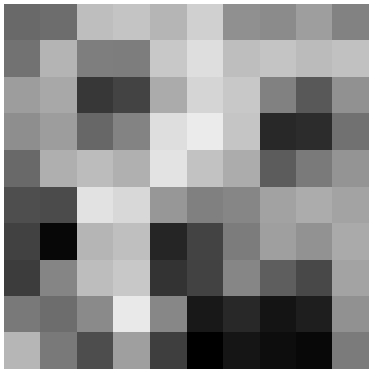
4.4 知识储备

4.4.1 灰度图像

灰度图像我们常见的黑白图像，但是，黑白图像并不是我们所说的非黑即白，只有两个颜色。在计算机图像领域中黑白图像只有黑白两种颜色，灰度图像在黑

色与白色之间还有许多级的颜色深度。灰度使用黑色调表示物体,即用黑色为基准色，不同的饱和度的黑色来显示图像。 每个灰度对象都具有从 0%（白色）到 100%(黑色)的亮度值。使用黑白或灰度扫描仪生成的图像通常以灰度显示。

我们知道，在计算机里面，图像是用像素点来表示的，比如说我们一张 10x10 的灰度照片，如下图 4-？所示，图像是一张 10x10 的灰度图像，我们在计算机中保存这张图像是以一个二维数组来表示的，数组中表示像素点的灰度取值区间在[0,255]之间。0 表示最黑，255 表示最白。



105	109	190	196	181	208	144	139	158	130
114	180	126	125	200	222	190	196	187	193
157	168	55	67	171	213	200	129	88	145
142	157	103	131	222	235	197	40	44	113
105	175	188	176	227	194	172	92	122	148
78	75	226	216	151	128	134	162	172	163
65	7	181	191	37	67	124	159	146	170
60	133	189	200	51	66	134	94	72	163
121	109	138	233	135	24	40	20	30	145
182	121	77	159	62	0	21	13	8	123

图 4-4 灰度图像与像素的灰度值

一副图像实际上是有很多个像素点组成的，每个像素点就是一个颜色，上图 4-1 灰度图，我们可以分割成 10x10 的 100 个区域，每个区域就是一个颜色。如下图所示，最左上角的像素点的灰度值为 105，该区域对应的 105 所代表的灰度颜色。右下角倒数第三个为 13，越接近 0 表示越黑，所以可以看出该像素点的的灰度颜色很黑。因此，平时我们用图像处理软件如 Photoshop 处理图像，实际上就是改变了图像像素点的值，相应的显示的颜色也会发生变化。

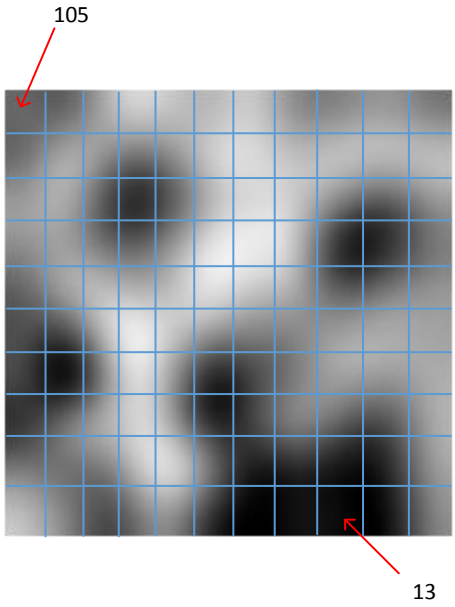


图 4-5 灰度图像各像素点对应的灰度值

4.4.2 MNIST 手写数字数据集

MNIST 数据集来自美国国家标准与技术研究所，National Institute of Standards and Technology (NIST)。训练集 (training set) 由来自 250 个不同人手写的数字构成，其中 50% 是高中学生，50% 来自人口普查局 (the Census Bureau) 的工作人员。测试集(test set) 也是同样比例的手写数字数据。

MNIST 数据集可在 http://yann.lecun.com/exdb/mnist/ 获取，它包含了四个部分：

- Training set images: train-images-idx3-ubyte.gz (9.9 MB，解压后 47 MB，包含 60,000 个样本)
- Training set labels: train-labels-idx1-ubyte.gz (29 KB，解压后 60 KB，包含 60,000 个标签)
- Test set images: t10k-images-idx3-ubyte.gz (1.6 MB，解压后 7.8 MB，包含 10,000 个样本)
- Test set labels: t10k-labels-idx1-ubyte.gz (5KB，解压后 10 KB，包含 10,000 个标签)

MNIST 数据集中所有的图片都是灰度图像，每张照片是一个手写数字，如下图所示：

4.5 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：导入处理数据所需要的模块，包括 utils 模块，numpy 模块，pandas 模块，keras 模块中的 utils、datasets 等模块，以方便后续的处理。

第二步：下载 MNIST 数据集，利用 keras 提供的 load_data()函数来下载 keras 提供的 MNIST 数据集。

第三步：查看下载后的 MNIST 数据集大小，并对每个部分的数据大小进行查看。

第四步：导入 matplotlib 模块进行进行图像绘制，查看 MNIST 数据集中单张照片与对应灰度值数组，进一步了解灰度数字图像。

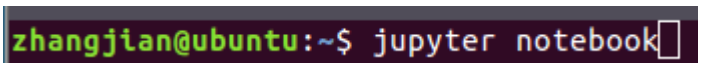
第五步：进行多张图像绘制，并查看 MNIST 中每张图像与其对应的 label 之间的关系。

第六步：预处理训练和测试图像，将二维图像进行一维拉伸，并对灰度值进行归一化处理。

第七步：预处理训练和测试 label 数据，由于将来是要进行分类预测和识别，因此，将 label 数据进行 One Hot Encoding 编码。

4.5.1 创建 jupyter notebook 项目

（1）打开 jupyter notebook



（2）在 python3 下新建一个 notebook 项目，命名为 rask4-1，如下图 4-7 所示。

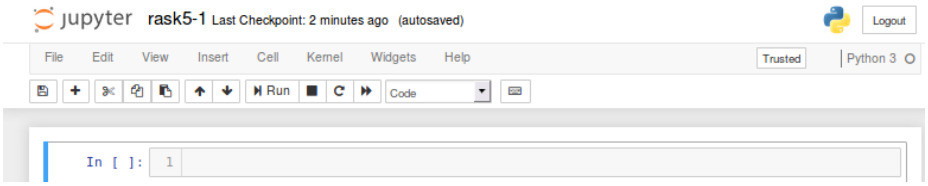


图 4-7 新建 notebook 项目示意图

4.5.2 导入处理数据集所需要的模块

（1）在 jupyter notebook 中输入图 4-8 中显示的代码，并确认代码无错误。

```
In [3]: import numpy as np   #导入numpy模块并命名为np
        import pandas as pd  #d导入pandas模块并命名为pd
        from keras.utils import np_utils   #导入keras下的utils模块并命名为np_utils
        from keras.datasets import mnist   #导入keras下的datasets模块下的minist数据库模块
        np.random.seed(10)                  #设置随机数种子10，以便后续产生随机数
        print('loading module has finished')
```

图 4-8 导入处理数据集所需模块代码

（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 4-9 所示：

```
In [3]: import numpy as np   #导入numpy模块并命名为np
        import pandas as pd  #d导入pandas模块并命名为pd
        from keras.utils import np_utils   #导入keras下的utils模块并命名为np_utils
        from keras.datasets import mnist   #导入keras下的datasets模块下的minist数据库模块
        np.random.seed(10)                  #设置随机数种子10，以便后续产生随机数
        print('loading module has finished')

        loading module has finished

In [ ]:
```

图 4-9　导入处理数据集所需模块代码运行结果图

（3）代码解析

- import numpy as np

导入 numpy 模块，Numpy 是 python 语言的扩展链接库，支持多维数组与矩阵运算。

- import pandas as pd

导入 pandas 模块，pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。pandas 提供了大量能使我们快速便捷地处理数据的函数和方法。

- from keras.utils import np_utils

utils 模块提供了在 keras 框架下的一系列有用工具，后面我们会用到 utils 模块中的 to_categorical（）和 normalize（）函数等。更多详细的 utils 模块介绍可参考 keras 中文文档中的 utils 介绍，https://keras-cn.readthedocs.io/en/latest/utils/。

- from keras.datasets import mnist

导入 keras 中的 dataset 模块，keras 中提供了一些常用的数据集共用户使用，例如 CIFAR10 小图片分类数据集、CIFAR100 小图片分类数据库、IMDB 影评倾向分类、路透社新闻主题分类、MNIST 手写数字识别、Fashion-MNIST 数据集、Boston 房屋价格回归数据集等。

- np.random.seed(10)

设置 seed 可以产生的随机数据范围，当我们设置相同的 seed，每次生成的随机数相同。如果不设置 seed，则每次会生成不同的随机数，例如，我们输入如下图所示：

```
>>> import numpy
>>> numpy.random.seed(10)
>>> numpy.random.rand(4)
array([0.77132064, 0.02075195, 0.63364823, 0.74880388])
>>> numpy.random.seed(10)
>>> numpy.random.rand(4)
array([0.77132064, 0.02075195, 0.63364823, 0.74880388])
>>>
```

图 4-10　相同的随机数种子产生相同的随机数

当我们设置相同的随机数种子时，每次产生的随机数是相同的，不同的随机数种子产生的随机数不同，如下所示：

```
>>> numpy.random.seed(10)
>>> numpy.random.rand(4)
array([0.77132064, 0.02075195, 0.63364823, 0.74880388])
>>> numpy.random.seed(0)
>>> numpy.random.rand(4)
array([0.5488135 , 0.71518937, 0.60276338, 0.54488318])
>>>
```

图 4-11 不同的随机数种子产生不同的随机数

4.5.3 下载 MNIST 数据集

（1）在 jupyter notebook 中输入图 4-12 中显示的代码，并确认代码无错误。

```
In [6]: (train_image, train_label),(test_image, test_label)=mnist.load_data() #下载mnist数据集到train_image, train_label,test_image, test_label
```

图 4-12 导入处理数据集所需模块代码

（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 4-9 所示，第一次运行时，程序会检查用户目录下是否已经有 MNIST 数据集文件，如果还没有，程序会自动下载该文件。由

于要下载文件，该程序的运行时间可能会比较长。

图 4-13 下载 MNIST 数据集

（3）代码解析

➢ (train_image, train_label),(test_image, test_label)=mnist.load_data()

mnist 数据库有 60,000 个用于训练的 28*28 的灰度手写数字图片，10,000 个测试图片。代码下载数据集后，会将数据保存在四个集合中，分别为：

- train_image：保存训练数字图像，共 60000 个
- train_label：保存训练数字图像的正确数字，共 60000 个
- test_image：保存测试数字图像，共 10000 个
- test_label：保存测试数字图像的正确数字，共 10000 个

（4）keras 下 mnist 数据库使用介绍

- 使用方法

```
1  from keras.datasets import mnist
2  (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

图 4-14 keras 下 mninst 数据集的使用方法

- 参数

path：如果你在本机上已有此数据集（位于'~/.keras/datasets/'+path），则载入。否则数据将下载到该目录下

- 返回值

两个 Tuple,(X_train, y_train), (X_test, y_test)，其中

X_train 和 X_test：是形如（nb_samples, 28, 28）的灰度图片数据，数据类型是无符号 8 位整形（uint8）。

y_train 和 y_test：是形如（nb_samples,）标签数据，标签的范围是 0~9。

- 数据库保存位置

数据库将会被下载到'~/.keras/datasets/'+path

➢ (train_image, train_label),(test_image, test_label)=mnist.load_data()

代码下载数据集后，会将数据保存在四个集合中，分别为：

- train_image：保存训练数字图像，共 60000 个
- train_label：保存训练数字图像的正确数字，共 60000 个
- test_image：保存测试数字图像，共 10000 个
- test_label：保存测试数字图像的正确数字，共 10000 个

4.5.4 查看 MNIST 数据集

（1）在 jupyter notebook 中输入图 4-15 中显示的代码，并确认代码无错误。

```
In [8]: print('train_image =',train_image.shape)  #查看train_image集合中的大小
        print('test_image =',test_image.shape)    #查看test_image集合中的大小
        print('train_label =',train_label.shape)  #查看train_label集合中的大小
        print('test_label =',test_label.shape)    #查看test_label集合中的大小
```

图 4-15 查看数据集所需模块代码

（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 4-16 所示。

```
train_image = (60000, 28, 28)
test_image = (10000, 28, 28)
train_label = (60000,)
test_label = (10000,)
```

图 4-16 查看数据集代码运行结果

（3）代码解析

- print('train_image=',train_image.shape)

本段代码分别输出四个集合中数据的维度，从结果中可以看出，train_image 集合的输出结果为（60000,28,28），显示这是个三维的数据，可以理解为一共有 60000 张图片，每张图片的大小都是 28x28 大小。train_label 集合的输出结果为（60000,），显示该集合是一个一维数组，可以理解为 60000 个数字，对应 60000 张图像的真实数字，如图 4-17 所示。

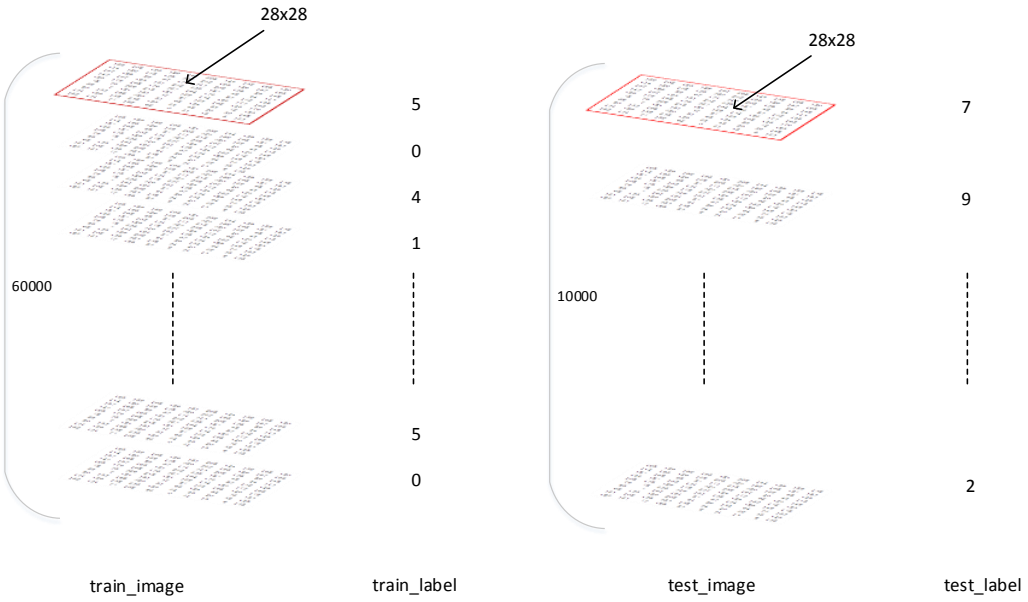


图 4-17 下载的 mnist 数据各部分大小

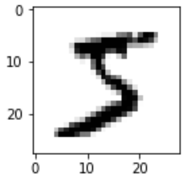
4.5.5 查看 MNIST 数据集中单张照片与对应灰度值数组

（1）在 jupyter notebook 中输入图 4-15 中显示的代码，并确认代码无错误。

```
In [7]: import matplotlib.pyplot as plt
        def plot_image(image):
            fig=plt.gcf()
            fig.set_size_inches(2,2)
            plt.imshow(image, cmap='binary')
            plt.show()
        plot_image(train_image[0])
        train_image[0]
```

图 4-15 查看数据集所需模块代码

（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 4-16 所示。




```
Out[7]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
                 18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
                253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,

                ……………………………………

               [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
                195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
                 11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0]], dtype=uint8)
```

图 4-16 查看数字照片与对应灰度值数组

（3）代码解析

- import matplotlib.pyplot as plt

导入 matplotlib.pyplot 模块并命名成 plt。Matplotlib 是一个 Python 的 2D 绘图库，通过 Matplotlib，仅需要几行代码，便可以生成绘图，直方图，功率谱，条形图，错误图，散点图等。pyplot 是 Matplotlib 软件包中子包，提供了一个类似 MATLAB 的绘图框架,支持 Python 语言。

- def plot_image(image):

定义名为 plot_image 的函数，其输入参数是 image

- fig=plt.gcf()

- fig.set_size_inches(2,2)

获取当前的图形对象，并将图像设置为 2*2 的大小。

- plt.imshow(image, cmap='binary')

使用 plt.imshow 显示图形，传入的图形是 image，大小是 28*28。

Cmap 参数,由其文档可知，在 colormap 类别上，有如下分类：

- **perceptual uniform sequential colormaps：**感知均匀的序列化 colormap
- **gray：**0-255 级灰度，0：黑色，1：白色，黑底白字；
- **gray_r：**翻转 gray 的显示，如果 gray 将图像显示为黑底白字，gray_r 会将其显示为白底黑字；
- **diverging colormaps：**两端发散的色图 colormaps；

- plt.show()

图像绘制。

- plot_image(train_image[0])

查看训练数据集中 image 部分的第一张图片。

- train_image[0]

查看训练数据集中第一张照片的灰度值数组。

在 MNIST 数据集中， Image 是一副 28*28 的灰度图片，数组中每个单元的数值在 0~255 之间。其中 0 表示白色，255 表示黑色。

图 4-17　MNIST 数据集中一张图片与对应的灰度值矩阵

4.5.6 查看 MNIST 数据集中多张照片与对应数字类别

（1）在 jupyter notebook 中输入图 4-18 中显示的代码，并确认代码无错误。

```
In [10]: def plot_images_labels(images,labels,idx,num=5):
             fig=plt.gcf()
             fig.set_size_inches(12,14)
             if num>5:
                 num=5
             for i in range(0,num):
                 ax=plt.subplot(1,5,1+i)
                 ax.imshow(images[idx],cmap='binary')
                 title='label='+str(labels[idx])
                 ax.set_title(title,fontsize=10)
                 ax.set_xticks([])
                 ax.set_yticks([])
                 idx+=1
             plt.show()
         plot_images_labels(train_image,train_label,0,5)
```

图 4-18 查看数据集多张照片与对应数字类别

（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 4-19 所示。

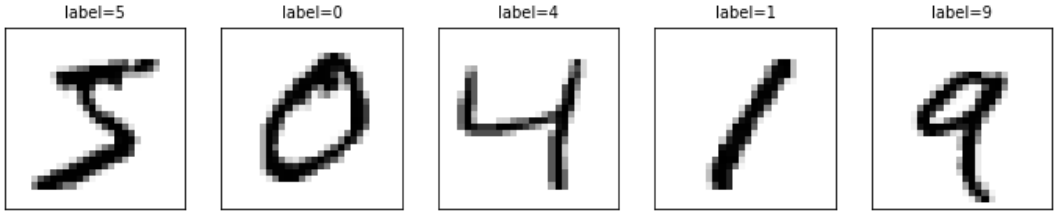


图 4-19 查看数据集中多张照片与对应数字类别

（3）代码解析

- def plot_images_labels(images,label,index,num):

定义名为 plot_images_labels 的函数，其输入参数是 images(要显示的图像数组)，label（对应的 label 数组），index（要显示的图像位于训练集或测试集中的起始位置），num(要显示的图像数量)。

- fig=plt.gcf()
- fig.set_size_inches(12,14)

调用 gcf 创建一个(12,14)绘图对象，并且使它成为当前的绘图对象。

- if num>5
- num=5

最多显示不超过 5 张图像，如果超过 5 张，则只选 5 张。

- for i in range(0,num)

循环处理。

- ax=subplot(1,5,1+i)

在绘图对象中创建 1 行 5 列子图形，并选择第 1+i 个来放置图像

- ax.imshow(images[index],cmap='binary')

在子图形中放置 images[index]图像，并设置成灰度形式

- title="label="+label[index]
- ax.set_title(title,fontsize=10)
- ax.set_xtick([])
- ax.set_ytick([])

设置子图形的标题（title）内容和字体大小、X 轴和 y 轴内容为空

- plt.show()

显示绘制的图像。

- plot_images_labels(train_image,train_label,0,5)

显示训练图像中从第 0 副开始的连续 5 副数字图像。

4.5.7 预处理训练和测试图像

（1）在 jupyter notebook 中输入图 4-20 中显示的代码，并确认代码无错误。

```
In [12]: X_train_image=train_image.reshape(60000,784).astype('float32')
         X_test_image=test_image.reshape(10000,784).astype('float32')
         print('X_train_image:',X_train_image.shape)
         print('X_test_image:',X_test_image.shape)
         X_train_image[0]
         x_Train_normalize=X_train_image/255
         x_Test_normalize=X_test_image/255
         x_Train_normalize [0]
```

图 4-20 预处理训练和测试图像代码

（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 4-21 所示。

```
X_train_image: (60000, 784)
X_test_image: (10000, 784)
```

```
Out[15]: array([0.        , 0.        , 0.        , 0.19215687, 0.93333334,
       0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.9843137 , 0.3647059 ,
       0.32156864, 0.32156864, 0.21960784, 0.15294118, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.07058824, 0.85882354, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.7764706 , 0.7137255 ,
       0.96862745, 0.94509804, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.3137255 , 0.6117647 , 0.41960785, 0.99215686, 0.99215686,
       0.8039216 , 0.04313726, 0.        , 0.16862746, 0.6039216 ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.05490196,
       0.00392157, 0.6039216 , 0.99215686, 0.3529412 , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ],
      dtype=float32)
```

……

图 4-21 预处理训练和测试图像结果

（3）代码解析

- X_train_image=train_image.reshape(60000,784).astype(‘float’)
- X_test_image=test_image.reshape(10000,784).astype(‘float’)

将三维的训练图像集和测试图像转换成二维的训练图像集和测试图像集，即将 60000*28*28 的数据集转换成 60000*784 的数据集，并将数据由整形转换成浮点类型。

- print（’X_train_image:’,X_train.shape）
- print（’X_test_image:’,X_train.shape）

输出结果为：

```
X_train_image: (60000, 784)
X_test_image: (10000, 784)
```

图 4-22 维度转换后的结果

显示转换后数字图像集成为了一个二维数组。

如图所示，转换后的第一幅图像变成一维数组共 784 个元素。

- x_Train_normalize=X_train_image/255
- x_Test_normalize=X_test_image/255

将图像的数据进行归一化处理，使得所有的数值都在 0~1 区间内，这样可以提高后续训练模型的准确性，较少图像由于光照不同带来的影响。因为图像的像素值都是在 0~255 之间，所以除以 255 进行归一化处理。

- x_Train_normalize [0,200:300]

输出结果为：

```
Out[15]: array([0.        , 0.        , 0.        , 0.19215687, 0.93333334,
                0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
                0.99215686, 0.99215686, 0.99215686, 0.9843137 , 0.3647059 ,
                0.32156864, 0.32156864, 0.21960784, 0.15294118, 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.07058824, 0.85882354, 0.99215686, 0.99215686,
                0.99215686, 0.99215686, 0.99215686, 0.7764706 , 0.7137255 ,
                0.96862745, 0.94509804, 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.3137255 , 0.6117647 , 0.41960785, 0.99215686, 0.99215686,
                0.8039216 , 0.04313726, 0.        , 0.16862746, 0.6039216 ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.05490196,
                0.00392157, 0.6039216 , 0.99215686, 0.3529412 , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ],
               dtype=float32)
```

图 4-23 对图像数据进行预处理后的数值

显示归一化处理后第一幅图像的数据。由结果中可见，所有的数据都是除以0~1 之间。经过处理，将所有的图像全部转换为一维的数据，取值区间在[0,1]之间，如下图 4-24 所示。

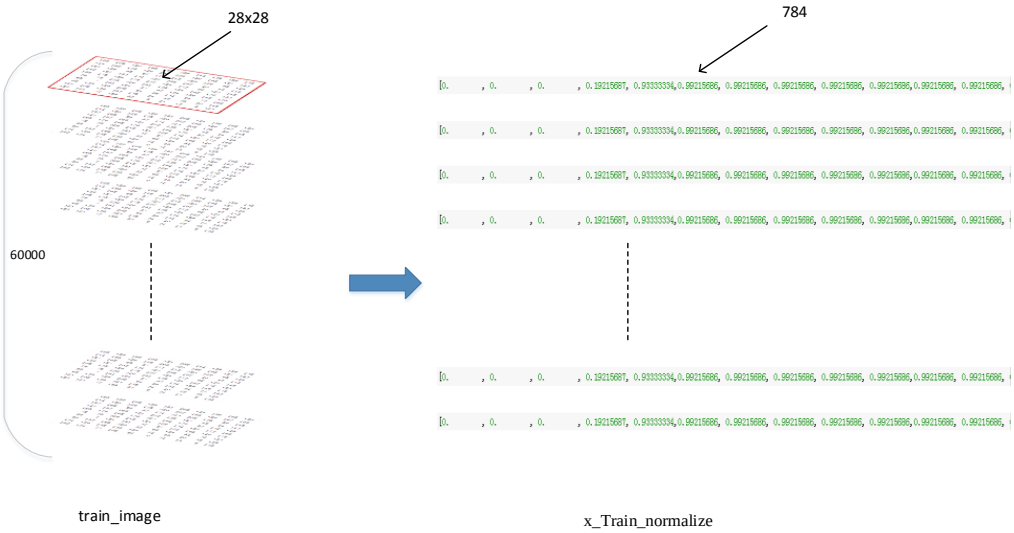


图 4-24 预处理训练图像数据

4.5.8 预处理训练和测试的 label 数据

（1）在 jupyter notebook 中输入图 4-25 中显示的代码，并确认代码无错误。

```
In [16]: print(train_label[0:10])
         y_TrainOneHot=np_utils.to_categorical(train_label)
         y_TestOneHot=np_utils.to_categorical(test_label)
         y_TrainOneHot[:10]
```

图 4-25 预处理训练和测试 label 数据代码

（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 4-26 所示。

```
          [5 0 4 1 9 2 1 3 1 4]

Out[16]: array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
                [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
                [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
                [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
                [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
                [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]], dtype=float32)
```

图 4-26 预处理训练和测试 label 数据结果

（3）代码解析

- print(train_label[:10])

显示训练 label 集中前 10 个数据。

输出结果：

```
[5 0 4 1 9 2 1 3 1 4]
```

由结果可以看到，所有的数字都在 0~9 之间，并且是 int 类型的数

- y_TrainOneHot=np_utils.to_categorical(train_label)
- y_TestOneHot=np_utils.to_categorical(test_label)

利用 np_utils.to_categorical()函数对 label 数据进行 One-Hot-Encoding 转换。

np_utils.to_categorical（）函数:多类分类问题与二类分类问题类似，需要将类别变量（categorical function）的输出标签转化为数值变量。在多分类问题中我们将转化为虚拟变量（dummy variable）：即用 one hot encoding 方法将输出标签的向量（vector）转化为只在出现对应标签的那一列为 1，其余为 0 的布尔矩阵。

- y_TrainOneHot[:10]

显示 One-Hot-Encoding 转换后训练 label 集中前十个数据。

输出结果：

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]], dtype=float32)
```

图 4-27 One-Hot-Encoding 转换后 label 的变化结果

由输出结果可以看到，每个数据是由 9 个 0 和 1 个 1 组成的。1 的位置对应原来 label 数据的数值大小。如第一个数是 5，那么转换后的第一行数据中，第 5 个位置是 1，其它位置都是 0,如下图所示。

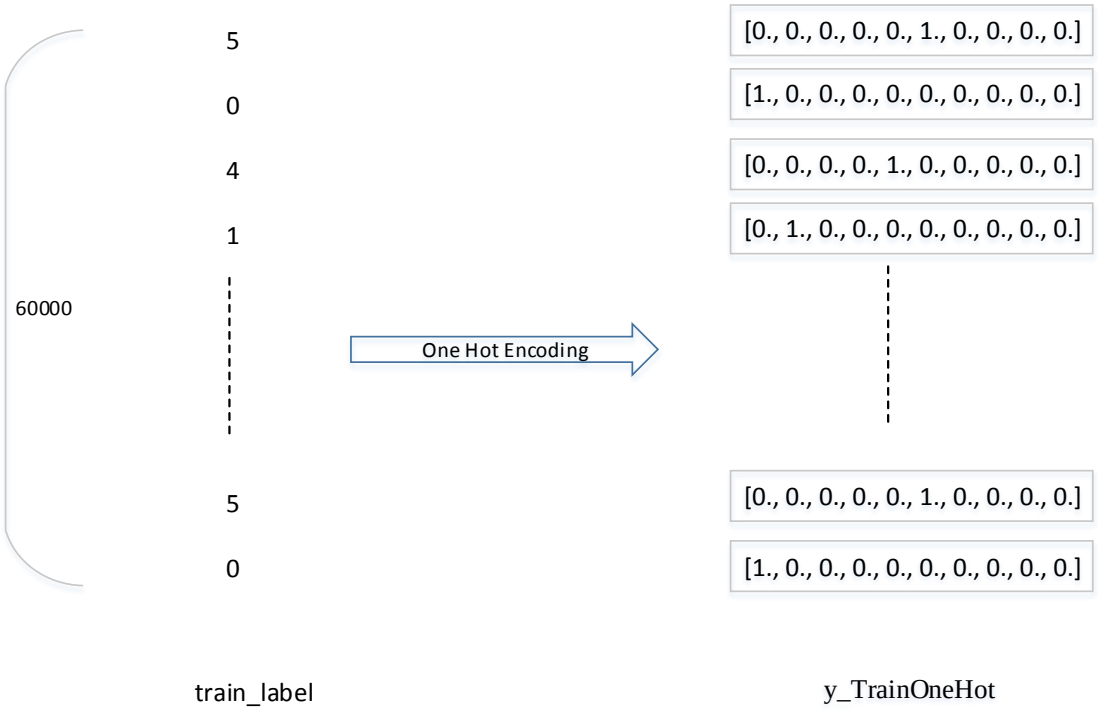


图 4-28 预处理训练 label 数据示例图

4.6 项目代码完整示例

```
### rask4-1.py
#准备MNIST数据集
#
#
#*************** 第一步：导入处理数据所需要的模块**********************
import numpy as np   #导入numpy模块并命名为np
import pandas as pd  #d导入pandas模块并命名为pd
from keras.utils import np_utils   #导入keras下的utils模块并命名为np_utils
from keras.datasets import mnist   #导入keras下的datasets模块下的minist数据库模块
np.random.seed(10)                  #设置随机数种子10，以便后续产生随机数
print('loading module has finished')

#*************** 第二步：下载MNIST数据集**********************
##下载mnist数据集到train_image，train_label,test_image，test_label
(train_image, train_label),(test_image, test_label)=mnist.load_data()

#*************** 第三步：查看下载后的数据集大小**********************
print('train_image =',train_image.shape)  #查看train_image集合中的大小
print('test_image =',test_image.shape)     #查看test_image集合中的大小
print('train_label =',train_label.shape)  #查看train_label集合中的大小
print('test_label =',test_label.shape)     #查看test_label集合中的大小

#****** 第四步：导入matplotlib模块进行图像绘制，查看单张照片与灰度值****
import matplotlib.pyplot as plt    #导入matplotlib.pyplot模块并命名成plt
def plot_image(image):    #定义名为plot_image的函数，其输入参数是image
    fig=plt.gcf()         #获取当前的图形对象
    fig.set_size_inches(2,2)     #将图像设置为2*2的大小
    plt.imshow(image, cmap='binary')   #使用plt.imshow显示图形，传入的图形是image
    plt.show()               #绘制图像
plot_image(train_image[0])    #查看训练数据集中image部分的第一张图片
train_image[0]                #查看训练数据集中第一张照片的灰度值数组
```

```
32
33  #******* 第五步：进行多张图像绘制，查看图像与其对应的label**************
34  #定义名为plot_images_labels的函数，其输入参数是
35  #images(要显示的图像数组)，
36  #label（对应的label数组），
37  #index（要显示的图像位于训练集或测试集中的起始位置），
38  #num(要显示的图像数量)，默认值为5。
39  def plot_images_labels(images,labels,idx,num=5):
40      fig=plt.gcf()      #获取当前的图形对象
41      fig.set_size_inches(12,14)    将图像设置为12*14的大小
42      if num>5:             #设置最大显示图像数量为5
43          num=5
44      for i in range(0,num):       #遍历所有图像进行显示
45          ax=plt.subplot(1,5,1+i)       #获取子图形对象
46          ax.imshow(images[idx],cmap='binary')    #将图像传给对象
47          title='label='+str(labels[idx])         #添加对应图像的label值到title
48          ax.set_title(title,fontsize=10)          #设置title
49          ax.set_xticks([])                        #设置x轴
50          ax.set_yticks([])                        #设置y轴
51          idx+=1
52      plt.show()     #显示所有图像
53  #显示训练图像中从第0副开始的连续5副数字图像
54  plot_images_labels(train_image,train_label,0,5)
55
56  #******* 第六步：预处理训练和测试图像***************
57  ##将三维的训练图像集和测试图像转换成二维的训练图像集和测试图像集
58  X_train_image=train_image.reshape(60000,784).astype('float32')
59  X_test_image=test_image.reshape(10000,784).astype('float32')
60  #显示转换后数字图像集大小
61  print('X_train_image:',X_train_image.shape)
62  print('X_test_image:',X_test_image.shape)
63  #将图像的数据进行归一化处理
64  x_Train_normalize=X_train_image/255
65  x_Test_normalize=X_test_image/255
66  x_Train_normalize [0,200:300] #显示归一化后训练图像第一张的第200-300像素的数值
67
68  #******* 第七步：预处理训练和测试label数据***************
69  print(train_label[0:10])  #显示训练label集中前10个数据
70  #对label数据进行One-Hot-Encoding转换
71  y_TrainOneHot=np_utils.to_categorical(train_label)
72  y_TestOneHot=np_utils.to_categorical(test_label)
73  y_TrainOneHot[:10]#显示One-Hot-Encoding转换后训练label集中前十个数据
```

4.7 小结与应用

本章主要介绍了 MNIST 数据集的情况，以及如何下载 MNIST 数据集并对 MNIST 数据集进行训练前的数据预处理，同时，本章还初步介绍了数字图像处理的概念。图像是人类获取和交换信息的主要来源，因此，图像处理的应用领域必然涉及到人类生活和工作的方方面面。随着人类活动范围的不断扩大，图像处理的应用领域也将随之不断扩大。

（1）航天和航空技术方面

航天和航空技术方面的应用数字图像处理技术在航天和航空技术方面的应用，除了 JPL 对月球、火星照片的处理之外，另一方面的应用是在飞机遥感和卫星遥感技术中。许多国家每天派出很多侦察飞机对地球上有兴趣的地区进行大量的空中摄影。

（2）生物医学工程方面

数字图像处理在生物医学工程方面的应用十分广泛，而且很有成效。除了上面介绍的 CT 技术之外，还有一类是对医用显微图像的处理分析，如红细胞、白细胞分类，染色体分析，癌细胞识别等。此外，在 X 光肺部图像增晰、超声波图像处理、心电图分析、立体定向放射治疗等医学诊断方面都广泛地应用图像处理技术。

（3）通信工程方面

当前通信的主要发展方向是声音、文字、图像和数据结合的多媒体通信。具体地讲是将电话、电视和计算机以三网合一的方式在数字通信网上传输。

（4）军事公安方面

在军事方面图像处理和识别主要用于导弹的精确末制导，各种侦察照片的判读，具有图像传输、存储和显示的军事自动化指挥系统，飞机、坦克和军舰模拟训练系统等；公安业务图片的判读分析，指纹识别，人脸鉴别，不完整图片的复原，以及交通监控、事故分析等。目前已投入运行的高速公路不停车自动收费系统中的车辆和车牌的自动识别都是图像处理技术成功应用的例子。

（5）机器人视觉

机器视觉作为智能机器人的重要感觉器官，主要进行三维景物理解和识别，是目前处于研究之中的开放课题。机器视觉主要用于军事侦察、危险环境的自主机器人，邮政、医院和家庭服务的智能机器人，装配线工件识别、定位，太空机器人的自动操作等。

随着数字图像处理技术与人工智能技术的不断发展，其应用领域也在不断扩展。

习题

1.简述数字图像处理的主要应用领域？

2.修改本章 4.5.6 节中的代码，使程序最多能显示 25 张照片以及所对应的 label 数据。

3.编写代码，使得该代码能够读取一张本地照片，并以灰度数字图像的形式显示出来。