第 3 章

CHAPTER 3

初识数字图像处理

本章主要通过简单介绍 Python 中常用的第三方图像处理库 Pillow 和图形图像绘制库 Matplotlib 来初步认识数字图像处理。Pillow 库起源于 PIL 库，目前 PIL 库已经停止更新，并且仅支持 Python 2.7 版本，无法满足 Python 3 版本下的图像处理需求。Pillow 库是在 PIL 库的基础上开发的支持 Python 3 版本的升级版图像处理库，然而 Pillow 库不仅“复制”了 PIL 库所有的功能，还在此基础上增加了很多新的图像处理功能。Matplotlib 库是 Python 在图形图像绘制上的主要工具库，也是很多高级数据可视化库的基础。

3.1 Pillow 库的简单使用

Pillow 库作为 Python 下图像处理的常用库，具有以下三大特性。

(1) 支持典型的图像文件格式，能够存取常见格式的图像文件，如 jpeg、png、bmp、gif、tiff 等格式的图像，同时支持不同图像格式之间的相互转换。

(2) 提供了丰富的图像处理功能，如图像归档和图像处理，图像归档包含构建缩略图像、图像的批量处理、生成预览图像等；图像处理包含图像任意的仿射变换、图像裁剪、图像颜色空间变换、直方图均衡化等。

(3) 提供了和其他图形用户界面(Python GUI)工具交互的接口，如 Tkinter、Qt 等，可方便地在 GUI 上显示图像。

以下将从图像生成、图像存取与显示、图像基本属性、图像格式、图像的基本操作和图形处理等方面介绍 Pillow 库的基本使用方法。

Image 类是 Pillow 库中最主要的类，该类被定义在模块 Image 中。使用下列方式可以导入 Image 模块，同时考虑到本节还要用到一个交互的库 ipywidgets，在此将该库一并导入，代码如下：

```
#导入图像处理库 Pillow
from PIL import Image
#导入一个交互的库
from ipywidgets import interact
```

3.1.1 图像生成

一幅数字图像可以由图像尺寸和每个像素的像素值共同表示。数字图像的尺寸可以由宽和高表示,以确定图像中像素的数量,例如,一幅宽为 256 像素、高为 100 像素的图像,总共有 25 600(256×100)像素。根据图像类型的不同,图像中每个像素的像素值是有差异的。对于灰度图像,每个像素都占用 1 字节,表示 256 种灰度级;对于彩色图像,如果使用 RGB 空间表示颜色,则每个像素都占用 3 字节,每字节分别表示红色(R)、绿色(G)和蓝色(B)的强度。Pillow 库中的 Image 类在内部将图像以像素为单位,以逐行存储的方式进行管理,例如,彩色图像可表示为一个由像素构成的序列:RGBRGBRGBRGB…RGB。

Image 类提供了直接从字节串生成图像的静态方法,下面以纯色灰度图、渐变灰度图、灰阶图、彩色图、横条纹图等的生成,展示图像的尺寸和像素等属性与图像的物理存储方式(字节串)的关系,从而加深对数字图像概念的理解。

Pillow 库中 Image 类的 frombytes()静态方法可用于将给定的字节串转换为图像对象。基于此,可以通过构建字节串,然后借助该方法将字节串转换成图像。

Image.frombytes()静态方法的详解如下:

```
Image.frombytes(mode, size, data, decoder_name='raw')
```

该静态方法返回一个 Image 对象,输入参数的含义如下。

(1) mode:设置创建图像的模式,即每个像素包含通道数和通道排列方式。可选的模式有 '1'(二值图像)、'L'(灰度图像)、'RGB'(真彩色图像)等。

(2) size:设置创建图像的尺寸,是一个表示图像宽和高的元组,如(width,height)。

(3) data:设置表示图像原始数据的字节串,字节串的长度为 width×height×len,其中 len 为单像素的字节长度,例如,当图像模式为'L'时,len=1,当图像模式为'RGB'时,len=3。

(4) decoder_name:是可选参数,用于设置使用的解码器,默认值为'raw',表示无解码器,即 data 中存储的是原始像素。

1. 纯色灰度图像生成

灰度图像的尺寸可由图像的宽和高确定,而灰度图像的每个像素值可表示为 1 字节。使用相同数据,生成一幅宽为 256 像素、高为 100 像素的纯色灰度图像和一幅宽为 128 像素、高为 200 像素的纯色图像的示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#生成纯色灰度图像
pixelvalue=2 #设置灰度值
d=[pixelvalue for i in range(256)]*100
print('d=',d)
#转换为字节串
imgd=bytes(d)
print('imgd=',imgd)
#根据图像数据,生成宽为 256 像素、高为 100 像素的灰度图像
img=Image.frombytes('L',(256,100),imgd)
```

```
img.show()
#根据图像数据,生成宽为 128 像素、高为 200 像素的灰度图像
img=Image.frombytes('L',(128,200),imgd)
img.show()
```

运行代码,结果如下：

```
#输出内容过长,已自动对输出内容进行截断
d=[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
imgd= b'\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\
x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\
x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\
x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\
x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\
x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\
x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
```

注意：代码运行结果中如出现“输出内容过长,已自动对输出内容进行截断”,则意味着,由于运行结果太长,对其进行截断展示,即代码运行的结果只展示了一部分。

代码生成的灰度图像如图 3-1 所示。

(a) 256×100的灰度图　　(b) 128×200的灰度图

图 3-1　纯色灰度图像

以上示例代码首先构建了一个长度为 25 600,元素均为 2 的整型列表,然后通过 Python 内置函数 bytes()将列表转换为字节串,产生像素的二进制表示,最后使用 Image.frombytes()方法将生成的字节串按照指定的宽和高生成灰度图像,并使用 show()方法进行显示。在生成灰度图像时,可以通过改变输入参数 size 来改变图像的尺寸,但是像素的总体数量要与字节串的长度相同。

注意:字节串与字符串是两种不同的数据类型,不能在生成图像时使用字符串。具体来讲,字节串是由多字节连接起来的序列,本质上是原始的二进制数据,字节串中一个元素的长度是 1 字节,而字符串是由多个字符连接起来的序列,是一种逻辑上的概念,字符串中一个字符对应的实际字节长度是由字符的编码方式确定的。通过字符串的 encode()方法可以将字符串编码为字节串,通过字节串 decode()方法可以将字节串转换为字符串。

接下来,借助 ipywidgets 模块中的交互式控件,生成指定灰度的纯色图像。具体方法是使用 ipywidgets 中的 interact()函数创建滑块控件,由滑块控件实现灰度的设置,使用滑块滑动的事件完成纯色图像的创建和显示。interact()函数的使用方法如下。

```
interact(func,para)
```

(1) func:滑块滑动事件需要绑定的回调函数,需要自定义。

(2) para:滑块绑定回调函数的参数,要与回调函数 func 中的参数一致。

通过交互式功能生成纯色图像的示例代码如下:

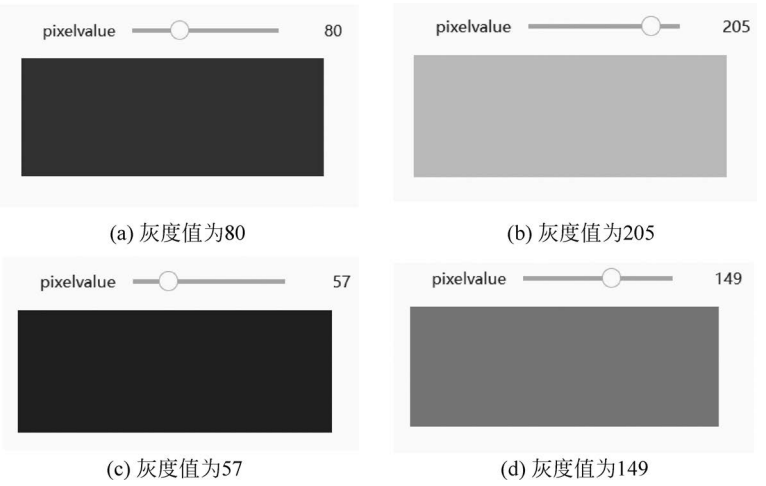
```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#通过交互式功能生成纯色图像
def tmp2(pixelvalue=80):
    #回调函数
    d=[pixelvalue for i in range(256)]*100
    #转换为字节串
    imgd=bytes(d)
    #根据图像数据,生成宽为 256 像素、高为 100 像素的灰度图像
    img=Image.frombytes('L',(256,100),imgd)
    img.show()

#可以利用动态交互工具实时调节不同范围,生成不同的纯色图像
t=interact(tmp2, pixelvalue=(0,255))          #控件
```

代码的运行结果图如图 3-2 所示。

以上示例代码通过交互式控件滑块 interact()实现了可调灰度的纯色图像。interact()中的第 1 个参数 tmp2 表示回调函数,此处回调函数为自定义函数,函数功能为使用字节数组生成纯色灰度图像,而且回调函数的输入参数为灰度图像的灰度值。interact()中的第 2 个参数 pixelvalue=(0,255)限定了灰度的范围,即滑块可以滑动的范围。

2. 条纹图像生成

纯色灰度图像即图像中每个像素的像素值相同,整幅图像灰度一致。与纯色灰度图像



(a) 灰度值为80　(b) 灰度值为205

(c) 灰度值为57　(d) 灰度值为149

图 3-2　利用交互式功能生成纯色图像

不同，条纹图像（也称为渐变灰度图像）中像素的灰度值是不同的，但是灰度值按照某个规律发生改变，如灰度值从 0 逐步增加到 255，灰度变化的步长为 1，从而使图像视觉上呈现渐变的效果。

条纹图像按照条纹的方向可分为竖条纹图像和横条纹图像。竖条纹图像即图像每列的像素灰度值相同，不同列的像素灰度值不同。基于生成纯色灰度图像的原理，由字节数组生成竖条纹图像的示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#生成竖条纹图像
d=[i for i in range(256)]*100
print('d=',d)
#转换为字节串
imgd=bytes(d)
print('imgd=',imgd)
#根据图像数据，生成宽为 256 像素、高为 100 像素的渐变灰度图像
img=Image.frombytes('L',(256,100),imgd)
print(f'图像的格式是{img.format},图像的类型是{img.mode}, 图像的宽和高是{img.size}')
img.show()
```

运行代码，结果如下：

```
#输出内容过长，已自动对输出内容进行截断
d= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
```

```
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176,
177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193,
194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 2
imgd= b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\
x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./0123456789:;
<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\
x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94
\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\
xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\
xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\
xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\
xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\
xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff\x00\x01\x02\x03\x04\x05\x06\
x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\
x1c\x1d\x1e\x1f !"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x
图像的格式是 None,图像的类型是 L, 图像的宽和高是(256, 100)
```

运行代码,产生的渐变灰度图像如图 3-3 所示。

图 3-3 竖条纹图像

从示例代码中可以看出,首先构建了 100 组从 0～255 且步长为 1 的递增数值列表,然后将该整型列表转换成字节串,最后采用字节串生成图像函数得到渐变图像。根据竖条纹图像的示例代码可以看出,由于最初构建了 100 个长度为 256 列表,列表元素从 0～255 变化,从而如果产生 256×100 的图像,图像的每行像素则对应列表元素,从左到右呈现从黑到白(像素值从 0～255 变化),图像每列像素的灰度值都相同。

接下来介绍通过构建字节串的方式生成高度方向上的横条纹图像。横条纹图像即图像的每行像素的灰度值相同,不同行像素的灰度值不同。考虑到字节串列表不能直接用于生成图像,可以使用字节串的 join()方法将字节串列表转换成字节串,示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#生成横条纹图像
d=[ bytes([i]*100) for i in range(256)]
print('d=',d)
#转换为字节串
imgd=b''.join(d)
#print(imgd)
#根据图像数据,生成宽为 256 像素、高为 100 像素的渐变灰度图像
img=Image.frombytes ('L',(100,256),imgd)
img.show()
```

在使用字节列表生成横条纹图像时,使用列表推导的方法生成了图像的 256 行,每行是

长度为 100 的由相同字节构成的字节串，每行字节串中字节的值是行号，考虑到字节串列表不能直接转换成图像，使用字节串的 join()方法将上述构建的字节串列表转换成了字节串，并保存到 imgd 变量中，最后用 Image.frombytes()方法将字节数组生成了横条纹图像。运行代码，结果如下：

```
#输出内容过长，已自动对输出内容进行截断
d=
[b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00', b'\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\
x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01
\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\
x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01
\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\
x01\x01\x01\x01\x01\x01', b'\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\
x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
```

运行代码，效果如图 3-4 所示。

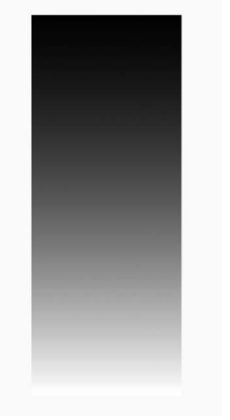


图 3-4　横条纹图像

3. 灰阶图像生成

灰阶图像作为渐变灰度图像的特例，是指同一灰度占据一定宽度的像素（灰阶间隔）。从视觉上看，图像的灰度变化像一个阶梯状。通过设置灰阶间隔，可以得到灰阶图像。具体的灰阶图像的示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#生成灰阶图像
#构建元素重复 4 次的列表
d=([(i//4)*4 for i in range(256)])*100
```

```
print('d=',d)
#转换为字节串
imgd=bytes(d)
#根据图像数据,生成宽为 256 像素、高为 100 像素的渐变灰度图像
img=Image.frombytes('L',(256,100),imgd)
img.show()
```

运行代码,结果如下:

```
#输出内容过长,已自动对输出内容进行截断
d=[0, 0, 0, 0, 4, 4, 4, 4, 8, 8, 8, 8, 12, 12, 12, 12, 16, 16, 16, 16, 20, 20, 20, 20,
24, 24, 24, 24, 28, 28, 28, 28, 32, 32, 32, 32, 36, 36, 36, 36, 40, 40, 40, 40, 44, 44,
44, 44, 48, 48, 48, 48, 52, 52, 52, 52, 56, 56, 56, 56, 60, 60, 60, 60, 64, 64, 64, 64,
68, 68, 68, 68, 72, 72, 72, 72, 76, 76, 76, 76, 80, 80, 80, 80, 84, 84, 84, 84, 88, 88,
88, 88, 92, 92, 92, 92, 96, 96, 96, 96, 100, 100, 100, 100, 104, 104, 104, 104, 108,
108, 108, 108, 112, 112, 112, 112, 116, 116, 116, 116, 120, 120, 120, 120, 124, 124,
124, 124, 128, 128, 128, 128, 132, 132, 132, 132, 136, 136, 136, 136, 140, 140, 140,
140, 144, 144, 144, 144, 148, 148, 148, 148, 152, 152, 152, 152, 156, 156, 156, 156,
160, 160, 160, 160, 164, 164, 164, 164, 168, 168, 168, 168, 172, 172, 172, 172, 176,
176, 176, 176, 180, 180, 180, 180, 184, 184, 184, 184, 188, 188, 188, 188, 192, 192,
192, 192, 196, 196, 196, 196, 200, 200, 200, 200, 204, 204, 204, 204, 208, 208, 208,
208, 212, 212, 212, 212, 216, 216, 216, 216, 220, 220
```

运行代码,生成的灰阶图像如图 3-5 所示。

图 3-5 灰阶图像

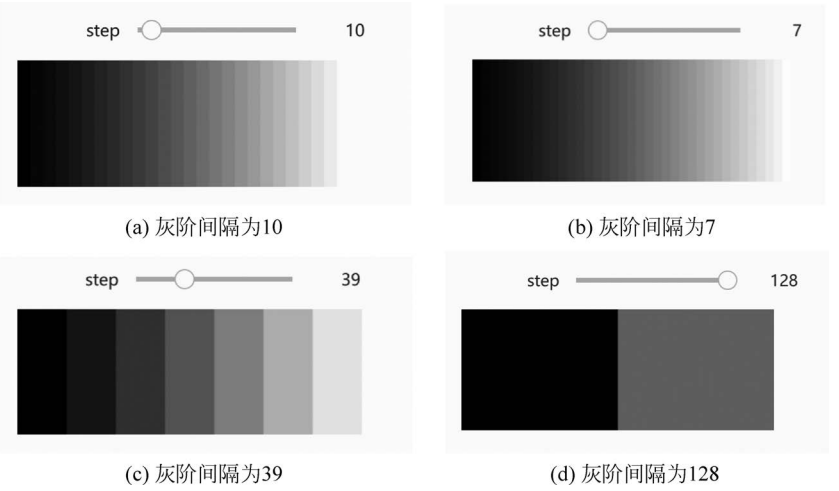
生成灰阶图像的难点在于产生表示第 1 行的初始列表,使列表内元素成阶梯状变化。借助 Python 中的整除运算符//,然后结合前面生成渐变灰度图像的原理,可以生成灰阶图像。

基于生成灰阶图像的方法,结合交互式控件,可以得到可调灰阶间隔的灰阶图像,示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#通过交互式功能生成不同间隔的灰阶,注意观察人眼对灰阶的分辨
def tmp3(step=10):
    k=256
    d=([ (i//step)*step for i in range(0,k)]) *100
    #转换为字节串
    imgd=bytes(d)
    #根据图像数据,生成宽为 256 像素、高为 100 像素的渐变灰度图像
    img=Image.frombytes('L',(k,100),imgd)
    img.show()

#可以利用动态交互工具来实时调节不同范围,得到不同间隔的灰阶图像
t=interact(tmp3, step=(1,128))          #控件
```

从回调函数 tmp3()可以看出,灰阶图像宽度和高度分别为 256、100 像素,灰阶的间隔通过交互界面的滑块调整,并且灰阶间隔的宽度被限定在 1～128。示例代码运行的效果如图 3-6 所示。从此图可以看出,当将滑块调整为不同的数值时,如 10、7、39 和 128,灰阶的间

隔也发生了变化。因为灰阶图像的宽度256是固定的，当设置不同的灰阶间隔时，最后得到的图像的灰度范围不同。灰阶间隔越小，可表示的灰度范围越大，图像中灰度的过渡越平滑；反之，灰阶间隔越大，可表示的灰度范围越小，图像中灰度的过渡越剧烈。



(a) 灰阶间隔为10　(b) 灰阶间隔为7

(c) 灰阶间隔为39　(d) 灰阶间隔为128

图 3-6　不同灰阶间隔的图像

4. 彩色图像生成

彩色图像可以由图像的宽、高及由3字节表示彩色像素的像素值表示。具体来讲，彩色像素中3字节分别表示该像素在红(R)、绿(G)、蓝(B)3个通道上的数值。如果将彩色像素看作由RGB 3个通道的值构成的一个长度为3的一维数组，那么，对于尺寸为$M\times N$的彩色图像，则需要用$M\times N$个表示彩色像素的一维数组构成。在Pillow库中，Image.frombytes()方法对于表示彩色图像的字节串的排列方式与灰度图像一致，即都按照像素为单位逐行进行排列，只不过彩色图像的一个像素是表示RGB的3字节。

根据上述分析，通过构建一个整数数组可生成不同颜色的彩色图像，数组的最内层是3个元素，分别表示彩色像素的RGB通道的数值，数组的最外层有$M\times N$个元素，即对应彩色图像的M行和N列，总共有$M\times N$像素。目前使用列表代替数组，生成彩色图像的示例代码如下：

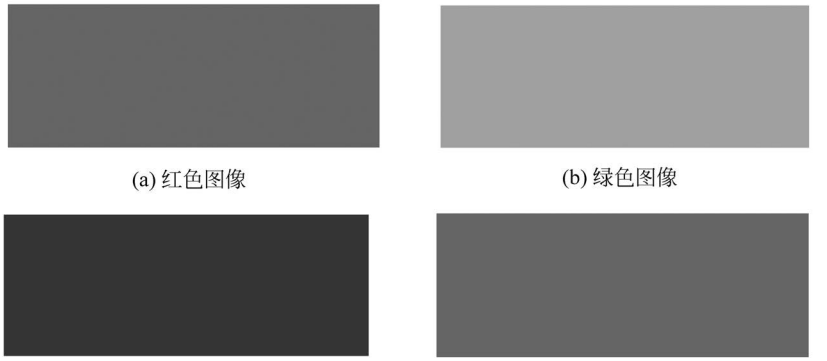
```
#第3章/3.1节-Pillow库的简单使用.ipynb
#生成红色图像
r=[255,0,0]                          #设置R、G、B 3个通道的值
row=100
column=256
d=row*column*r                       #得到红色图对应的数组元素
print('d=',d)
#转换为字节串
imgd=bytes(d)
#根据图像数据，生成宽为256像素、高为100像素的纯红色图像
img=Image.frombytes('RGB',(column,row),imgd)
```

```
img.show()

#生成绿色图像
g=[0,255,0]                     #设置 R、G、B 3 个通道的值
row=100
column=256
d=row*column*g                  #得到有绿色图对应的数组元素
#转换为字节串
imgd=bytes(d)
#根据图像数据,生成宽为 256 像素、高为 100 像素的纯绿色图像
img=Image.frombytes('RGB',(column,row),imgd)
img.show()

#生成蓝色图像
b=[0,0,255]                     #设置 R、G、B 3 个通道的值
row=100
column=256
d=row*column*b                  #得到有蓝色图对应的数组元素
#转换为字节串
imgd=bytes(d)
#根据图像数据,生成宽为 256 像素、高为 100 像素的纯蓝色图像
img=Image.frombytes('RGB',(column,row),imgd)
img.show()

#生成其他彩色图像
rgb=[124,120,155]               #设置 R、G、B 3 个通道的值
row=100
column=256
d=row*column*rgb                #得到有蓝色图对应的数组元素
#转换为字节串
imgd=bytes(d)
#根据图像数据,生成宽为 256 像素、高为 100 像素的纯蓝色图像
img=Image.frombytes('RGB',(column,row),imgd)
img.show()
```

运行代码,结果如下:

```
#输出内容过长,已自动对输出内容进行截断
d=[255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
255, 0, 0, 255, 0, 0, 255, 0
```

该段代码生成了 4 幅尺寸相同的彩色图像，分别是红色图像、绿色图像、蓝色图像和其他颜色图像，如图 3-7 所示。



(a) 红色图像　(b) 绿色图像

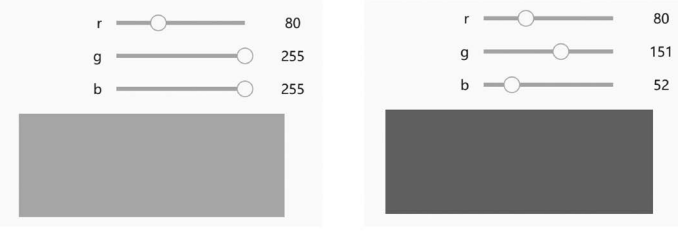
(c) 蓝色图像　(d) 其他颜色图像

图 3-7　生成的彩色图像（见彩插）

接下来基于交互式控制模块，生成色彩可调的彩色图像，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#通过交互式功能生成纯色图像
def tmp4(r=80,g=255,b=255):
    #回调函数
    color=[r,g,b]
    row=100
    column=256
    d=row*column*color
    #转换为字节串
    imgd=bytes(d)
    #根据图像数据，生成宽为 256 像素、高为 100 像素的渐变彩色图像
    img=Image.frombytes('RGB',(column,row),imgd)
    img.show()

#可以利用动态交互工具实时调节不同范围，生成不同的纯色图像
t=interact(tmp4, r=(0,255),g=(0,255),b=(0,255))          #控件
```

与上述使用动态交互工具的方式相似，通过构建滑块来调整 R、G、B 3 个通道的值，从而改变彩色图像的颜色，代码生成的不同颜色的图像如图 3-8 所示。



(a) 亮蓝图像　(b) 深绿图像

图 3-8　色彩可调的彩色图像（见彩插）

以上通过纯色灰度图像、渐变灰度图像、灰阶图像和彩色图像的生成等例子，验证了数字图像中图像尺寸、像素、灰度图像、彩色图像等基本概念，展示了数字图像在计算机内部以字节串表示的方法和结构。

3.1.2 图像存取与显示

在 Pillow 库中，通过 Image 类，可以实现图像的读取(打开)、显示和保存。Image 类中的静态方法 open()用于打开图像，在方法执行后返回 Image 类的实例，表示打开的图像。如果图像文件不存在或打开失败，则会抛出 FileNotFoundError 异常或 IOError 异常。open()方法支持 jpeg、png、bmp、gif、ppm、webp、tiff 等常见的图像文件格式，并可根据文件后缀名和图像内部信息自动选择适宜的读取接口，无须设置图像文件的类型。open()方法的用法如下：

```
Image.open(fp,mode='r')
```

(1) fp：文件名或者打开文件的路径(字符串)，如'./gray1.png'表示打开当前文件夹下的名为 gray1.png 的图像。

(2) mode：可选参数，默认值为'r'，表示以只读方式打开。

(3) 返回值：Image 类实例，即打开的图像对象。

(4) 错误：找不到要打开的文件，或者图像文件不能被识别或打开。

在图像打开之后，一般需要对图像进行显示。Image 对象提供了用于图像显示的 show()方法。只要打开图像得到 Image 类的图像实例，就可以直接调用 show()方法进行图像显示。show()方法执行时会调用系统内部默认的图像显示程序显示当前图像。使用 Pillow 库打开图像并进行显示的示例代码如下：

```
#打开和显示图像
img=Image.open('../images/1.jpg')
img.show() #Image 对象的 show()方法进行图像的显示
```

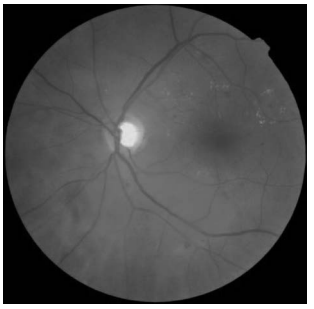


图 3-9 眼底原图

需要显示的图像和代码在同一根目录下的不同文件夹下，在此通过相对路径的文件读取方式来打开图像 1.jpg。在准备好待打开图像文件的路径后，使用 Image.open()函数直接打开图像，返回值是 Image 类的实例对象 img，接着借助图像对象的 show()方法来显示图像。示例代码的运行结果如图 3-9 所示。

此外，Image 类还提供了图像保存方法 Image.save()，可以对打开或创建的 Image 对象进行保存，存储为图像文件。当使用图像保存方法 save()保存图像时，如果不指定图像文件的保存格式，系统则会根据图像存储路径中的文件后缀名确定图像存储格式；如果指定图像格式，则尝试以指定格式存储图像。方法 save()的用法如下：

```
Image.save(fp,format=None)
```

(1) fp：图像的存储路径，包含图像的名称和后缀，以字符串类型表示。

(2) format：可选参数，用于指定保存图像的格式。如果默认，则使用的格式由 fp 中图像文件名的后缀决定。

(3) 返回值：无。

使用 Pillow 存储图像的示例代码如下：

```
#保存图像，img 为上个例子打开的眼底图像返回的 Image 对象
img.save('./images/1.png')
```

注意：Image.save()通常可用于进行图像格式的转换，但不是所有格式的图像都可以通过 Image.save()保存成其他格式的图像。例如，如果原图像的文件格式为 PNG 格式，通过 Image.save()函数将该图像保存成 JPG 格式，则系统会出错。原因在于 PNG 和 JPG 保存图像的模式不一致，PNG 采用四通道 RGBA 模式保存图像，即红色、绿色、蓝色和 Alpha 透明色，而 JPG 采用三通道 RGB 模式保存图像。

另外，Image 类还提供了图像实例的模式转换方法 Image.convert()，由此可实现不同图像模式间的转换。Image.convert()的用法如下：

```
Image.convert(mode)
```

(1) mode：要转换成的图像模式。Pillow 库支持的图像模式如表 3-1 所示。

(2) 返回值：转换后的图像。

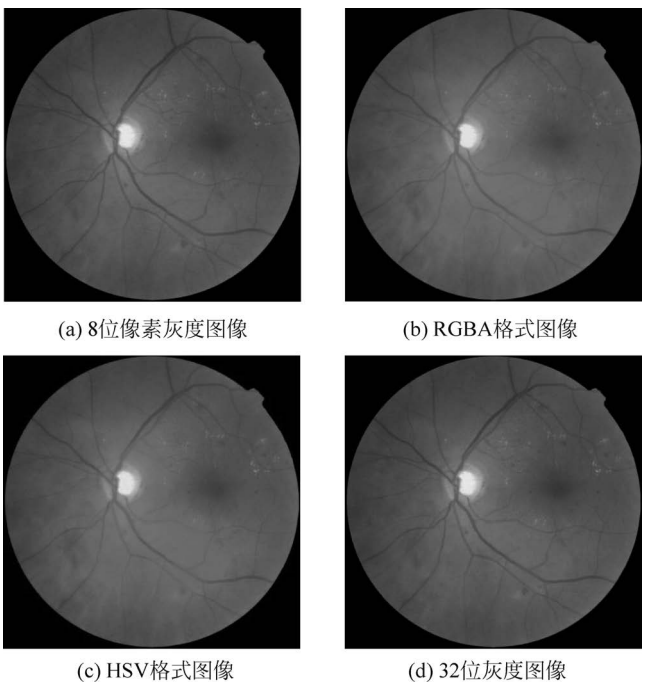
表 3-1　图像模式

mode	描　　述
1	1 位像素(可取 0 或者 1，0 表示黑，1 表示白)，单通道
L	8 位像素(取值范围为 0～255)，单通道，灰度图
P	8 位像素，可使用颜色映射表映射成任何颜色模式，单通道
RGB	3×8 位像素，真彩图，三通道，每个通道的取值范围为 0～255
RGBA	4×8 位像素，真彩色＋透明通道，四通道
CMYK	4×8 位像素，四通道，适用于打印图片
YCbCr	3×8 位像素，彩色视频格式，三通道
LAB	3×8 位像素，一个要素是亮度(L)，a 和 b 是两种颜色通道。a 包括的颜色是从深绿色(低亮度值)到灰色(中亮度值)再到亮粉红色(高亮度值)；b 是从亮蓝色(低亮度值)到灰色(中亮度值)再到黄色(高亮度值)。三通道
HSV	3×8 位像素，色相、饱和度和值颜色空间，三通道
I	32 位有符号整数像素，单通道
F	32 位浮点像素，单通道

使用 Pillow 对前面打开的图像进行模式转换,示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#将 RGB 图像转换为灰度图像模式
grayimg=img.convert('L')
grayimg.show()
#将 RGB 图像转换为 RGBA 图像模式
RGBAimg=img.convert('RGBA')
RGBAimg.show()
#将 RGB 图像转换为 HSV 图像模式
HSVimg=img.convert('HSV')
HSVimg.show()
#将 RGB 图像转换为 32 位灰度图像模式
Iimg=img.convert('I')
Iimg.show()
```

运行代码,生成的灰度图如图 3-10 所示,通过 Image.convert()方法可以将 RGB 图像转换成其他模式的图像,其中函数的参数决定要转换成的图像的模式,可以根据要求选择合适的转换模式。



(a) 8位像素灰度图像　(b) RGBA格式图像

(c) HSV格式图像　(d) 32位灰度图像

图 3-10　不同图像模式的眼底图像(见彩插)

3.1.3　图像属性查询

图像具有一些基本属性,如图像的格式、图像的尺寸、图像模式、图像是否为只读图像等相关信息,而了解图像的这些基本属性对于认识和处理图像具有非常重要的作用,如图像基

本处理中可以通过图像尺寸的调整进而改变其大小、改变图像的格式进行保存、进行图像模式转换等。这些操作都需要事先了解图像的基本属性，进而对其进行合理且正确的操作。

Pillow 库提供了一些图像基本属性查询方法，如 Image. size、Image. format、Image . readonly、Image. mode 等，具体的基本属性查询方法和作用如表 3-2 所示。

表 3-2　图像基本属性查询方法和作用

序　　号	名　　称	描　　述
1	Image. width	查看图像的宽
2	Image. height	查看图像的高
3	Image. size	查看图像尺寸(宽和高)
4	Image. format	查看图像格式
5	Image. readonly	查看图像是否为只读
6	Image. info	查看图像的相关基本信息

基于表 3-2 提供的属性查询方法的基本用法，可以快速查询图像的基本属性，图像属性查询的使用，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#打开图像
img=Image.open('../images/1.webp')
img.show()
#查看图像基本属性
print(f'图像的宽是{img.width},图像的高是{img.height}, 图像的尺寸是{img.size}')
print(f'图像的格式是{img.format},图像的类型是{img.mode}, 图像是否为只读是{img.
readonly}')
print(f'图像的基本信息是{img.info}')
```

打开并显示的图像如图 3-11 所示。

图 3-11　RGB 彩色图像

运行代码，结果如下：

```
图像的宽是 1000,图像的高是 517, 图像的尺寸是(1000, 517)
图像的格式是 WEBP,图像的类型是 RGB, 图像是否为只读是 0
图像的基本信息是{'loop': 1, 'background': (255, 255, 255, 255), 'timestamp': 0, 'duration': 0}
```

从运行结果可以看出，可以采用 Image. width 和 Image. height 属性分别获取图像的宽度和高度，也可以直接使用 Image. size 属性同时读取图像的宽度和高度，该属性将图像的

宽和高以元组形式返回。另外,Image.readonly()方法的返回值为 0 或 1,分别对应是否只读,Image.info()方法的返回值为字典形式,包括一些图像的额外信息。

3.1.4 图像处理初步

Pillow 库提供了简单的图像处理功能,只需调用相应的 API,就能完成常见的图像处理任务,如缩放、旋转、裁切、混合等操作,非常适合作为学习图像处理的入门。

1. 图像缩放

图像缩放是图像处理的基本操作之一,Image 对象提供了 resize()方法,可对图像实现指定尺寸的缩小和放大。resize()方法的用法如下:

```
Image.resize(size,resample=image.BICUBIC,box=None,reducing_gap=None)
```

(1) size:图像缩放后的尺寸,一般使用元组表示,即缩放后的尺寸(width,height)。

(2) resample:可选参数,是指图像的重采样方法,默认为 Image.BICUBIC,也可选其他滤波方式。总共支持 4 种重采样方法,分别是 Image.BICUBIC(双三次插值法)、PIL.Image.NEAREST(最近邻插值法)、PIL.Image.BILINEAR(双线性插值法)、PIL.Image.LANCZOS(下采样过滤插值法)。

(3) box:可选参数,表示对指定图像区域进行缩放,box 为长度为 4 的元组(左、上、右、下),表示缩放图像区域。需要注意的是,选取的缩放区域必须在原始图像的范围内,不能超出原始图像区域,否则会报错。当该参数采用默认值时,表示对整个原图进行缩放。

(4) redcuing_gap:可选参数,主要用于优化缩放后图像的效果,一般为浮点参数值。通常取值 3.0 和 5.0。

(5) 返回值:Image 类实例,即缩放后的图像对象。

基于 Pillow 库中 Image 对象提供的 resize()方法实现对整幅图像的缩放,示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#图像缩放
img=Image.open('../images/1.webp')
img.show()
#对整图进行缩小,缩小后的大小为(500,254)
reimg=img.resize((500,254))
reimg.show()
#对整图进行放大,放大后的大小为(2000,1034)
amimg=img.resize((2000,1034))
amimg.show()
print(f'原始图像的尺寸是{img.size},缩小后图像的尺寸是{reimg.size},放大后图像的尺寸是{amimg.size}')
```

示例代码的运行结果如下:

```
原始图像的尺寸是(1000, 517),缩小后图像的尺寸是(500, 254),放大后图像的尺寸是(2000, 1034)
```

代码运行产生的效果如图 3-12 所示，结合上述的图像尺寸的变化和效果图可以看出，通过直接改变参数 size 可以直接调整图像的大小，其他参数选取默认值，即图像重采样方式选择双三次插值法(Image. BICUBIC)、参数 box 和 redcuing_gap 都为 None。

注意：一般在使用 Image. resize()函数来调整图像尺寸的时候，图像的缩放要保证图像的高、宽比保持不变，从而可以使缩放后的图像不出现变形。

(a) 1000×517原始图像

(b) 500×254缩小图像

(c) 2000×1034放大图像

图 3-12　整幅图像缩放效果

接下来，通过设置 resize()方法不同的参数实现部分图像区域的缩放，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#对局部图像进行缩放
img=Image.open('../images/1.webp')
img.show()
#截取原始图像的部分区域(0,0,200,200)
oriimage=img.resize((200,200),resample=Image.LANCZOS,box=(0,0,200,200))
oriimage.show()
#将局部图像缩小为(100,100)
reimage=img.resize((100,100),resample=Image.LANCZOS,box=(0,0,200,200))
reimage.show()
#将局部图像放大为(500,500)
amimage=img.resize((500,500),resample=Image.LANCZOS,box=(0,0,200,200))
amimage.show()
```

```
print(f'原始局部图像的尺寸是{oriimage.size},缩小后局部图像的尺寸是{reimage.size},
放大后局部图像的尺寸是{amimage.size}')
```

运行代码,结果如下:

```
原始局部图像的尺寸是(200, 200),缩小后局部图像的尺寸是(100, 100),放大后局部图像的尺寸
是(500, 500)
```

代码的运行效果如图 3-13 所示,图 3-13(a)为原始的整幅图像,图 3-13(b)为局部原始图像(待缩放的区域),图 3-13(c)为局部图像缩小为 100×100 后的图像,图 3-13(d)为局部图像放大为 500×500 后的图像。具体图像缩放前后的尺寸信息如上面的运行结果所示,其中待缩放的区域为原始图像的(0,0,200,200)的图像区域,即左上角尺寸为 200×200 的图像区域,只需设置 Image.resize()函数中的参数 box;图像的重采样方式也设置为下采样过滤插值法(PIL.Image.LANCZOS)。

(a) 1000×517原始图像　(b) 200×200局部图像

(c) 100×100局部缩小图像　(d) 500×500局部放大图像

图 3-13　局部图像缩放效果

除了可以通过 Image.resize()函数实现图像的缩放之外,Pillow 库还提供了另外一种方式,可对图像按指定尺寸进行缩小,即缩略图(Thumbnail Image)。缩略图是对原始图像进行缩小处理,得到一个指定尺寸的图像,通过创建原始图像的缩略图可以提高图像的展示效果。

Image 对象创建缩略图的方法为 thumbnail(),该方法的具体用法如下:

```
Image.thumbnail(size,resample)
```

(1) size:图像缩小后的尺寸,一般使用元组表示,即缩小后的尺寸(width,height)。

(2) resample:可选参数,是指图像重采样滤波器,取值方式可参考 Image.resize()函数中的 resample 参数的设置。

通过原始图像构建缩略图的示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#创建图像的缩略图
img=Image.open('../images/1.webp')
print("原始图尺寸",img.size)
img.thumbnail((100,52))
print("缩略图尺寸",img.size)
img=Image.open('../images/1.webp')
print("原始图尺寸",img.size)
img.thumbnail((500,100))
print("缩略图尺寸",img.size)
```

运行代码，结果如下：

```
原始图尺寸 (1000, 517)
缩略图尺寸 (100, 52)
原始图尺寸 (1000, 517)
缩略图尺寸 (193, 100)
```

通过示例代码的运行结果可以看出原始图像尺寸为(1000,517)，通过设置不同的缩小尺寸，可以得到尺寸不同的缩略图，但是值得注意的是，缩略图尺寸的大小可能与指定的尺寸大小不一致，这是由于 Pillow 库在对图像的宽和高进行缩小时会按照原始的比例进行缩小。如果指定的缩小尺寸和原始图像的尺寸不一致，则系统会强制按照取小的原则等比例地缩小图像，得到和原始图像长宽比一致的缩略图。如第 2 次指定缩略图大小为(500,100)，其缩略图的指定宽高比为 5，其和原始图像的宽高比 1.93 不一致，则函数会根据指定缩略图的高 100 和原始图像的宽高比 1.93 来计算缩略图的高，即 100×1.93＝193。

2. 通道的分离和合并

数字图像是由一系列的像素构成的，每个像素是数字图像的基本单元，而每个像素值可以用一个数字表示(单通道图像，如灰度图像)，也可以用多个数字表示(如三通道的 RGB 图像)。以 RGB 图像为例，其图像的每个像素有 3 个通道：R 通道、G 通道和 B 通道，每个通道表示不同颜色的数值。一般而言，图像通道的分离和合并也就是图像颜色的分离和合并。

在图像处理中，如果需要单独处理多通道图像中某种颜色通道的分量，则需要将该颜色通道的数据从三通道的数据中心分离出来，然后进行处理，从而可以减少数据所占的内存，加快图像处理速度。与此同时，当处理完多个通道后，需要对所有处理完的通道进行合并，重新生成新的多通道图像。

Pillow 库中的 Image 对象提供了 split()和 merge()方法，用于图像通道的分离和合并，split()方法用于将 RGB 彩色图像分离成 R、G、B 3 个单通道图像，而 merge()方法用于实现图像通道的合并，可以是单个图像合并，也可以是两个或以上图像的合并。下面分别对两种方法的用法进行介绍。

1) Image. split()

(1) 无输入参数，对图像实例 Image 沿通道分离。

(2) 返回值为一个由 R、G、B 3 个单通道灰度图像组成的元组。

2) Image.merge(mode,bands)

(1) mode: 指定输出图像的模式,如 RGB、HSV 等。

(2) bands: 需要合并的通道图像,一般用元组或者列表表示。例如,要合并 R、G、B 3 个单通道图像,则 bands 取值为(R,G,B)。

(3) 返回值为合并后的 Image 图像实例。

通过程序实例进一步理解上述两种方法的使用,示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#RGB 图像分离
img=Image.open('../images/1.webp')
r,g,b=img.split()
print('红色通道图像尺寸=',r.size,'绿色通道图像尺寸=',g.size,'蓝色通道图像尺寸=',
b.size)
#将 3 个通道以灰度显示
r.show()
g.show()
b.show()
```

RGB 图像通道分离示例代码的运行结果如下:

```
红色通道图像尺寸=(1000, 517) 绿色通道图像尺寸=(1000, 517) 蓝色通道图像尺寸=(1000, 517)
```

上述示例代码将打开的 RGB 图像的 3 个通道使用 split()方法进行分离,生成 R、G、B 3 个单通道图像,如图 3-14 所示。结合上述运行结果和生成的 3 个单通道图像可知,使用图

(a) R通道图像　　(b) G通道图像

(c) B通道图像

图 3-14　图像分离

像分离 split()方法对 RGB 图像进行分离时不改变图像的尺寸，会返回尺寸一致的 R、G、B 3 个单通道图像，每个通道以灰度模式存储。

图像的合并一般归为两类：同一图像不同通道的合并和不同图像不同通道的合并。同一图像的不同通道合并只需将要合并的通道按顺序排列进行合并，不需要考虑所要合并图像的尺寸。不同图像的不同通道合并不但需要考虑通道排序，还要保证所要合并的通道图像的尺寸一致。下面通过两个程序示例分别详述图像合并的两类。

同一图像不同通道的合并，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#RGB 图像所分离的单通道的合并
img=Image.open('../images/1.webp')
img.show()
r,g,b=img.split()                          #RGB 图像分离
bgrimg=Image.merge("RGB", (b, g, r))       #合并顺序 b、g、r，与原图像颜色不一致
bgrimg.show()
rgbimg=Image.merge("RGB", (r, g, b))       #合并顺序 r、g、b，生成原图
rgbimg.show()
brgrimg=Image.merge("RGB", (b, r, g))      #合并顺序 b、r、g，与原图像颜色不一致
brgrimg.show()
```

程序的运行结果如图 3-15 所示，图 3-15(a)为原始的 RGB 图像，图 3-15(b)、(c)、(d)分别为 RGB 原始图像分离出的 R、G、B 3 个通道按照不同顺序合并的效果。原始图像中 3 个通道的排序为 R、G、B，在图像合并时，采用了 3 种不同的通道顺序进行图像合并，调整了通道的顺序，得到 3 种不同的合并图像。对比合并图像和原始图像，可以看出，当合并通道的顺序和原始通道的顺序相同时，得到的合并通道图像和原始图像一样。当合并通道的顺序

图 3-15　图像的不同通道合并(见彩插)

和原始通道的顺序不同时,得到的合并通道图像和原始图像不一样,这表明在一幅图像中图像的通道顺序是重要的,不可随意交换。

不同图像的不同通道合并,示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#尺寸一致的不同图像的合并
img1=Image.open('../images/1.webp')
img1.show()
r1,g1,b1=img1.split()                          #RGB 图像分离
img2=Image.open('../images/1.jpg')
#将 img2 的尺寸缩放到与 img1 相同
resimg2=img2.resize(img1.size)
resimg2.show()
r2,g2,b2=resimg2.split()                       #RGB 图像分离
brgrimg=Image.merge("RGB", (b1, r2, g1))       #不同图像的合并
brgrimg.show()
brgrimg=Image.merge("RGB", (r1, b1, g2))       #不同图像的合并
brgrimg.show()
```

示例代码产生的结果如下所示,从得到的图像分离的单通道图像尺寸可知图像的分离不改变图像通道尺寸,只改变图像的通道数。

```
红色通道图像尺寸=(1000, 517) 绿色通道图像尺寸=(1000, 517) 蓝色通道图像尺寸=(1000, 517)
```

示例代码运行生成的图像如图 3-16 所示,其中图 3-16(a)和图 3-16(b)分别为原始的待合并的两幅图像,图 3-16(c)和图 3-16(d)是两幅原始图像的不同通道图像按照不同的方式

(a) RGB卡通图像　(b) RGB眼球图像

(c) 合并图像　(d) 合并图像

图 3-16　不同图像合并效果(见彩插)

合并得到的结果。从合并图像可以看出，通道图像的选择及通道图像的排序的不同，最后合并得到的图像也不同。根据每个 RGB 图像包含 3 个通道计算得到两幅图合并后可产生 120(6×5×4)种(通道图像不能重复使用)不同的合并结果。

3. 图像几何变换

图像几何变换又称为图像空间变换，它对图像中像素的坐标进行变换，将像素映射到图像中的新坐标。图像的几何变换改变了像素的空间位置，建立了一种原图像像素与变换后图像像素之间的映射关系。

Pillow 库的 Image 对象提供了常见的图像几何变换操作方法，如 transpose()、rotate()等。transpose()方法可以实现图像沿水平或者竖直方向的翻转，而 rotate()方法则可实现图像沿任意角度的旋转。这两种函数的具体用法如下：

```
Image.transpose(method)
```

(1) 参数 method 表示图像要按照哪种方式翻转，method 可以取的参数值如下。

Image. FLIP_LEFT_RIGHT：左右水平翻转；

Image. FLIP_TOP_BOTTOM：上下垂直翻转；

Image. ROTATE_90：图像逆时针旋转 90°；

Image. ROTATE_180：图像旋转 180°；

Image. ROTATE_270：图像逆时针旋转 270°；

Image. TRANSPOSE：图像转置；

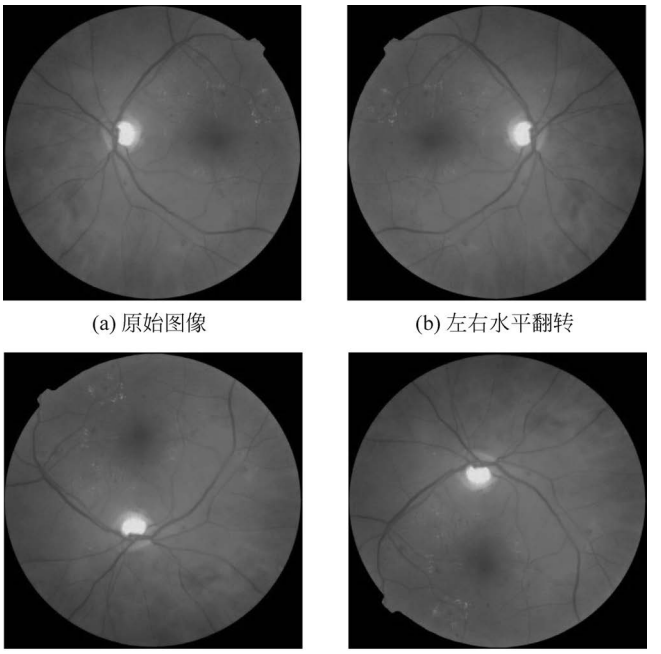
Image. TRANSVERSE：图像横向翻转。

(2) 返回值为翻转后的图像。

Image. transpose()函数的示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#图像翻转
img=Image.open('../images/1.jpg')
img.show()
#返回一个新的 Image 对象
im_out=img.transpose(Image.FLIP_LEFT_RIGHT)       #左右水平翻转
im_out.show()
im_out=img.transpose(Image.ROTATE_90)             #逆时针旋转 90°
im_out.show()
im_out=img.transpose(Image.TRANSPOSE)             #图像转置
im_out.show()
```

代码的运行效果如图 3-17 所示。图 3-17(a)为原始图像，图 3-17(b)为左右水平翻转后的图像，图 3-17(c)为逆时针旋转 90°之后的图像，图 3-17(d)为转置后的图像。从得到的几何变换后的图像效果来看，左右水平翻转图像是以原始图像中心竖直线为对称轴进行左右翻转后得到的图像，而逆时针旋转 90°的图像则是以原始图像的中心为旋转中心进行逆时针旋转 90°得到的图像，图像转置是将图像的 x 坐标和 y 坐标互换，图像的大小会随之改变，即将高度和宽度互换后得到的图像。



(a) 原始图像　(b) 左右水平翻转

(c) 逆时针旋转90° 图像　(d) 转置图像

图 3-17　图像翻转

```
Image.rotate(angle,resample=PIL.Image.NEAREST,expand=0,center=None,translate=
None,fillcolor=None)
```

(1) angle：图像逆时针旋转的角度。

(2) resample：重采样方法，可选参数，默认为最近邻插值法 PIL.Image.NEAREST。

(3) expand：是否对图像进行扩展，可选参数，默认值为 False 或者省略，表示输出图像和输入图像的尺寸一致。如果参数为 True，则表示扩大输出图像的尺寸，使其足够大，能容纳整个旋转图像。

(4) center：图像旋转的中心，取值为二元组，如(30,20)，表示以左上角为原点，以向右 30 像素和向下 20 像素的位置为旋转中心，可选参数，默认为以原图像中心进行旋转。

(5) translate：旋转后图像的平移量，取值为二元组，默认为不进行平移。

(6) fillcolor：图像旋转之后对图像之外的区域填充的颜色，可选参数。

(7) 返回值，为旋转后的图像实例。

Image.rotate()函数的示例代码如下：

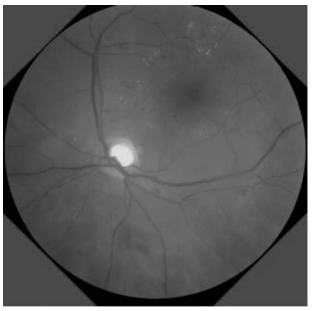
```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#图像任意角度旋转
img=Image.open('../images/1.jpg')
im_out=img.rotate(45,translate=(0,-25),fillcolor="green")
im_out.show()
```

```
im_out=img.rotate(45,translate=(0,45),fillcolor="blue")
im_out.show()
```

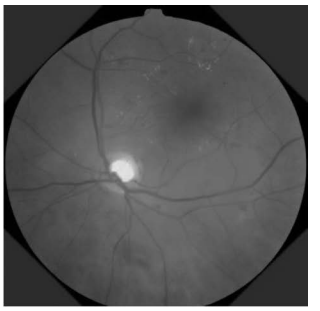
图像旋转之后的效果如图 3-18 所示，图 3-18(a)是图像按照逆时针方向旋转 45°后再向上平移 25 像素的结果，并且图像旋转后之外的区域选择绿色填充，图 3-18(b)是图像按照逆时针方向旋转 45°后再向下平移 45 像素的结果，并且图像旋转后之外的区域选择蓝色填充。

4. 图形和文字的绘制

绘制图形和文字是在图像上添加图形或文字，常用于标注感兴趣的区域等，在深度学习的目标检测中得到了广泛应用，如图 3-19 所示。



(a) 旋转图像（向上平移）



(b) 旋转图像（向下平移）

图 3-18　图像旋转(见彩插)

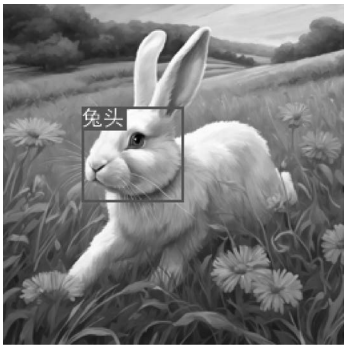


图 3-19　绘制图形和文字

Pillow 库中的 ImageDraw 提供了在图像上的绘图功能，ImageFont 提供了加载字体的方法。在图像上绘制图形和文本前，先要创建绘图对象和设置文本字体，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
#绘制图形和文字
from PIL import Image, ImageDraw,ImageFont

img=Image.open('../images/lena.png')
draw=ImageDraw.Draw(img)

#font=ImageFont.load_default()
font=ImageFont.truetype('simhei',16)
draw.font=font
box=(105,90,180,195)              #x1,y1,x2,y2
#在图像上画矩形框
draw.rectangle(xy=box, fill=None ,outline='red',width=2)
label='人脸'
_,_, w, h=font.getbbox(label)     #text width, height
draw.rectangle((box[0], box[1], box[0] + w + 1,box[1] + h + 1),fill='red',)
draw.text((box[0], box[1] ), label, fill='white')
#img.save('./boxface.png')
img.show()
```

上述示例代码,使用 ImageDraw.Draw()函数在打开的图像上创建了一个画布对象,然后使用 ImageFont.turetype()函数加载了名称为 simhei(黑体)的字体,并将字体大小设置为 16,最后将创建的字体设置为画布对象的默认字体,代码的运行效果如图 3-19 所示。

画布对象(Draw)支持多种几何形状的绘制,表 3-3 列出了几种形状的绘制方法名称和描述。

表 3-3 绘制方法名称和描述

序 号	方 法 名 称	描 述
1	arc(xy,start,end,fill=None,width=0)	绘制弧线
2	chord(xy,start,end,fill=None,outline=None,width=1)	绘制带有弦的弧线
3	ellipse(xy,fill=None,outline=None,width=1)	绘制椭圆
4	line(xy,fill=None,width=0,joint=None)	绘制线段
5	pieslice(xy,start,end,fill=None,outline=None,width=1)	绘制扇形
6	point(xy,fill=None)	绘制点
7	polygon(xy,fill=None,outline=None,width=1)	绘制多边形
8	rectangle(xy,fill=None,outline=None,width=1)	绘制矩形
9	text(xy,text,fill=None,font=None,anchor=None,spacing=4,align='left',direction=None,features=None,language=None,stroke_width=0,stroke_fill=None,embedded_color=False)	绘制文本

表 3-3 中绘制方法的使用示例代码如下:

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
img=Image.open('../images/lena.png')
draw=ImageDraw.Draw(img)
#绘制弧线
draw.arc([10,10,50,50],30,150,fill='blue')
#绘制带有弦的弧线
draw.chord([60,10,100,50],30,150,fill='yellow',outline='blue')
#绘制椭圆
draw.ellipse([110,10,160,50],outline='red',width=2)
#绘制线段
draw.line([170,30,230,30],fill='green',width=3)
#绘制扇形
draw.pieslice([10,60,60,100],30,150,fill='pink',outline='green')
#绘制点
draw.point([80,80],fill='#fff')
#绘制多边形
draw.polygon([120,70,110,90,130,90],fill='green',outline='red')
#绘制矩形
draw.rectangle([150,70,180,90], outline='white', width=2)
#绘制文本
draw.text([120,100], text="HELLO", outline='white', width=2)
#显示绘制结果
img.show()
```

上述示例代码，在打开的图像上使用不同的绘图方法在画布对象所在的图像上进行图形和文本的绘制，绘制效果如图 3-20 所示。

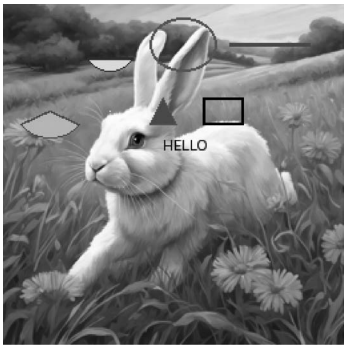


图 3-20　不同绘图方法的效果

3.1.5　Tkinter 显示图像

Tkinter 作为 Python 自带的图形用户界面(GUI)库，具有无须安装、控件丰富和使用简单等优点，能够为程序快捷地开发图形用户界面。在 Tkinter 中显示图像时，需要将图像转换为 PhotoImage 或 BitmapImage 对象。PhotoImage 表示图像对象，只支持少量的图像文件格式，如 PGM、PPM、GIF 和 PNG，而对 TIF、JPEG、WEBP 等其他常用图像格式不支持。BitmapImage 表示位图图像，只支持 XBM 格式的位图。这就使使用上述两个 Tkinter 内置的图像对象在显示图像时受到很大的制约。

Pillow 库提供了一个 ImageTk 类，能够将 Image 对象转换为 Tkinter 库中 PhotoImage 和 BitmapImage 对象，从而可以在 Tkinter 的标签(Label)控件和画布(Canvas)控件中进行图像的显示。此外，在图像显示前还可以借助 Pillow 库对图像进行简单处理。

对于 PNG 等格式的图像，可以使用 Tkinter 直接显示，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
import tkinter as tk
#创建 Tkinter 窗口
mainwindow=tk.Tk()

#将图像转换为 Tkinter 可用的图像对象
tkimg=tk.PhotoImage(master=mainwindow,file='../images/lena.png')

#创建 Label 对象，并将图像绘制到 Label 上
label=tk.Label(mainwindow,image=tkimg)

#将 Label 添加到主窗体
label.grid()
mainwindow.mainloop()
```

在上述示例代码中，先创建了一个主窗体 mainwindow，然后使用 PhotoImage 类创建

了一个图像对象 tkimg，参数 master 表示创建图像对象所归属的主对象，参数 file 表示图像的路径，随后将 tkimg 作为参数 image 添加到 Label 控件，最后将 Label 控件添加到主窗体并显示。代码的运行结果如图 3-21 所示，图像文件'../images/lena.png'显示在 Tkinter 窗体中。

图 3-21 PhotoImage 显示

注意：上述代码需要在本地开发环境中运行，在线开发环境不能使用 Tkinter 库。

当需要显示 PhotoImage 类不支持的格式或需要对图像进行简单处理时，就需要借助 Pillow 中的 ImageTk 作为媒介转换为 Tkinter 中的 PhotoImage 和 BitmapImage 对象，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
import tkinter as tk
from PIL import ImageTk,Image

#创建 Tkinter 窗口
mainwindow=tk.Tk()

#打开图像文件,并缩放尺寸
image=Image.open('../images/lena.tif').resize((256,256))

#将图像转换为 Tkinter 可用的图像对象
tkimg=ImageTk.PhotoImage(image)

label=tk.Label(mainwindow,image=tkimg)
label.grid() #放入主窗体
mainwindow.mainloop()
```

在上述示例代码中，先使用 Pillow 库中的 Image 对象打开图像，并进行缩放处理，然后使用 ImageTk.PhotoImage()方法将 Image 对象转换为 Tkinter 支持的 PhotoImage 对象，最后运行程序，显示效果与上一个示例相同。

对于 BitmapImage 所支持的 XBM 图像格式目前更少见，借助 Pillow 库中的图像模式转换方法 convert()和 ImageTk.BitmapImage()方法，可以方便地生成 Tkinter 中的 BitmapImage 对象，示例代码如下：

```
#第 3 章/3.1 节-Pillow 库的简单使用.ipynb
import tkinter as tk
from PIL import ImageTk,Image

#创建 Tkinter 窗口
mainwindow=tk.Tk()
```

```
#打开图像文件,缩放并转换为 bitmap
image=Image.open('../images/lena.tif').resize((256,256)).convert('1')

#将图像转换为 Tkinter 可用的图像对象
tkimg=ImageTk.BitmapImage(image)

#创建 Label 对象,并将图像绘制到 Label 上
label=tk.Label(mainwindow,image=tkimg)
label.grid() #放入主窗体
mainwindow.mainloop()
```

在上述示例代码中,在打开图像后立刻进行图像缩放处理,并将图像模式转换为位图,然后使用 ImageTk.BitmapImage()方法将 Image 对象转换为 Tkinter 中的 BitmapImage 对象,最后在 Label 控件中进行显示,运行结果图 3-22 所示。

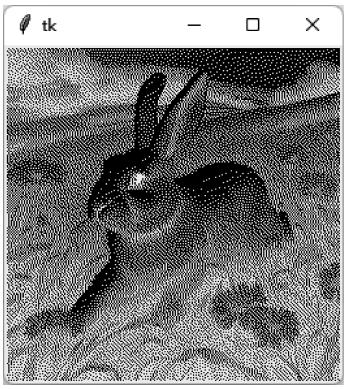


图 3-22　BitmapImage 显示

借助 Pillow 库的 ImageTk 类,在 Tkinter 的界面中可以灵活地进行图像的显示,对于 Tkinter 与图像处理更详细的讨论,将在最后一章中进行全面介绍。

3.2　Matplotlib 库的简单使用

Matplotlib 库是 Python 下绘制图形图像的非常重要的库,常用于数据可视化。Matplotlib 库支持跨平台运行,并且含有丰富的二维绘图接口和部分三维绘图接口,是数据分析中不可缺少的重要工具包之一。相较于用 Pillow 展示图像时需要生成 Image 对象,Matploblib 可直接对表示图像的数组进行绘制,方便随时观察图像,此外,在图像的点运算中需要绘制变换曲线,也需要借助 Matploblib 的折线图绘制功能。

Matplotlib 库有 3 个主要功能:图像绘制、折线图绘制及显示绘制结果。在进行功能展示之前,统一导入本节需要用到的库,其导入代码如下:

```
%matplotlib inline
import matplotlib.pyplot as plt                    #导入 pyplot 对象,并重命名为 plt
```

```
#导入图像处理库 Pillow 中的 Image 类
from PIL import Image
#导入一个交互的库
from ipywidgets import interact
#导入 NumPy 库
import numpy as np
```

3.2.1 图像绘制

Matplotlib 库提供了 pyplot.imshow()函数用于显示图像,其常用于绘制二维的灰度图像或彩色图像,也可以绘制数组、热力图或地图等。pyplot.imshow()函数的用法如下:

```
pyplot.imshow(img,cmap=None,norm=None,aspect=None,interpolation=None,alpha=
None,vmin=None,vmax=None,origin=None,extent=None,shape=None,filternorm=1,
filterrad=4.0,imlim=None,resample=None,url=None,data=None)
```

(1) img:输入图像,既可以是 Pillow 中的 Image 对象,也可以是以下形状的 NumPy 数组。

(M,N):标量数据的图像。数值通过归一化和色图映射到颜色。具体映射方式可以使用 norm、cmap、vmin 和 vmax 等参数进行定义。

(M,N,3):RGB 值的图像。RGB 值可以是 0～1 的浮点数或 0～255 的 uint8 类型的整数。

(M,N,4):RGBA 值的图像,包括透明度。RGBA 值可以是 0～1 的浮点数或 0～255 的 uint8 类型的整数。

其中,前两个维度(M,N)定义了图像的高和宽。

(2) cmap:颜色映射表,用于控制图像中不同标量数值所对应的颜色,只作用于灰度图像。可选参数,可以选择内置的颜色映射,如 gray、hot、jet 等,也可以自定义颜色映射,默认为 RGB(A)颜色空间,默认值为 None。

(3) norm:归一化方式。用于将标量数据缩放到[0,1]范围内。在默认情况下,使用线性缩放,将最低值映射为 0,将最高值映射为 1。如果给定,则可以是 Normalize 的实例或其子类。

(4) vmin 和 vmax:当使用标量数据且没有明确的 norm 参数时,vmin 和 vmax 定义了图像在显示时像素值的有效范围,小于 vmin 的像素值会被截断为 vmin,大于 vmax 的像素值会被截断为 vmax。在默认情况下,色图覆盖所提供数据的完整值范围。如果给定了 norm 实例,则忽略此参数。

(5) aspect:图像的宽高比。对于图像来讲,这个参数特别重要,因为它决定了数据像素是否为正方形。可以取值'equal'(保持 1∶1 的宽高比,使像素为正方形)或'auto'(保持 Axes 固定,根据数据调整宽高比)。

(6) interpolation:图像在显示过程中需要缩放时所采用的插值方法。可以选择'nearest'、'bilinear'、'bicubic'等插值方法。

(7) alpha:图像透明度,范围在 0(透明)～1(不透明)。如果 alpha 是一个数组,则透明度逐像素应用,并且 alpha 必须具有与 img 相同的形状。

(8) origin：坐标轴原点的位置。数组中[0,0]索引在 Axes 左上角或左下角的位置。默认值为'upper',通常用于矩阵和图像。注意,对于'lower',垂直轴指向上方；对于'upper',垂直轴指向下。

(9) extent：图像的数据坐标边界框。默认值由以下条件确定：像素在数据坐标中具有单位大小,其中心位于整数坐标上,中心坐标在水平方向上从 0 到列数－1,垂直方向上从 0 到行数－1,可以设置为[xmin,xmax,ymin,ymax]。

(10) filternorm：图像缩放过滤器的参数。如果设置了 filternorm,则过滤器会对整数值进行归一化和修正舍入误差。

(11) filterrad：具有半径参数的滤波器的过滤器半径。适用于插值方法为'sinc'、'lanczos'或'blackman'的情况。

(12) resample：当为 True 时,使用完整的重采样方法；当为 False 时,只有输出图像大于输入图像时才进行重采样。

(13) url：创建图像的 URL。

注意：plt.imshow()函数只对图像进行绘制处理、产生绘制结果等,并不进行真正意义上的图像显示,一般需要配合 plt.show()函数显示绘制的图像。

1. Image 对象的绘制

对 Pillow 库中的 Image 对象,可直接使用 plt.imshow()函数绘制并使用 plt.show()函数显示图像,示例代码如下：

```
#图像绘制并显示
img=Image.open('../images/1.jpg')
plt.imshow(img)                    #绘制图像
plt.show()                         #显示图像
```

代码的运行结果如图 3-23 所示,打开的图像通过 plt.imshow()函数执行了图像绘制,并通过 plt.show()函数进行显示。从运行结果可以看出,使用 plt.show()函数进行图像显示时,Matplotlib 会自行将图像缩放到适宜尺寸,并为图像添加坐标轴,显示图像的实际尺寸,方便观察图像。

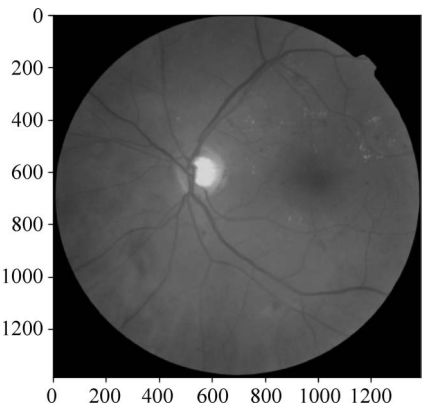


图 3-23　图像绘制效果

注意：plt.imshow()函数进行图像显示是按照 RGB 顺序控制图像色彩的,对于单通道的灰度图像会使用默认的颜色映射表,将单通道的图像显示为伪彩色图像,如果要使用 plt.imshow()函数正确地显示灰度图像,则必须加入参数 cmap='gray'。

下面通过设置 plt.imshow()函数中的 cmap 参数来绘制并显示灰度图,其示例代码如下:

```
#imshow()函数用于绘制灰度图
grayimg=img.convert('L')
plt.imshow(grayimg)
plt.show()                              #为什么显示的不是灰度图
plt.imshow(grayimg,cmap='gray')         #将 cmap 参数设置为'gray'
plt.show()                              #正确显示灰度图像
```

示例代码的运行结果如图 3-24 所示,图 3-24(a)为没有设置 cmap 参数时绘制的灰度图显示效果。从此图可以看出,显示的图像不是灰度图,而是伪彩图。图 3-24(b)为设置 cmap='gray'参数时的效果。从此图中可以看出,设置 cmap='gray'后,能够正确显示灰度图。

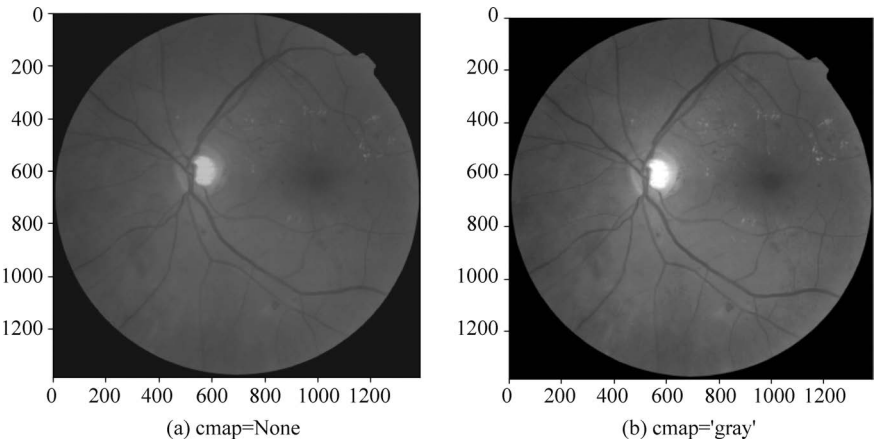


图 3-24 灰度图绘制效果(见彩插)

2. NumPy 数组的绘制

一幅尺寸为 $M\times N$ 像素的数字图像,如果是灰度图像,则可以表示为 $M\times N$ 二维数组;如果是彩色图像,则可以用 $M\times N\times 3$ 或 $M\times N\times 4$ 的三维数组。对于表示数字图像的二维数组,数字图像中的各像素值按照一定的顺序存放在二维数组中,即数字图像左上角的像素为第(0,0)像素,数字图像右下角的像素为第$(M-1,N-1)$像素,数字图像的第(i,j)像素存放到二维数组第 $i-1$ 行第 $j-1$ 列位置(二维数组元素索引从(0,0)开始)。

下面将建立二维数组,并使用 plt.imshow()函数绘制二维数组,示例代码如下:

```
#第 3 章/3.2 节-Matplotlib 库的简单使用.ipynb
#绘制二维数组
imgar=np.random.randint(0,256,(100,256),dtype='uint8')  #去掉 dtype='uint8'的参
#数,观察结果
#print(f'数组的高宽为{imgar.shape}')
plt.imshow(imgar)                   #未设置 cmap 参数,不能正确地显示为灰度图
plt.show()
plt.imshow(imgar,cmap='gray')       #设置 cmap 参数,能够正确表示灰度图
plt.show()
```

示例代码使用 NumPy 中的方法构建了一个 100 行 256 列且数组元素值取值 0～255 的二维随机数组，并通过 plt.imshow()函数进行数组绘制。上述构建的二维数组，每个数组元素为单个数值表示，因此二维数组可表示为单通道图像，即灰度图。示例代码的运行结果如下：

```
数组的高宽为(100, 256)
```

对应 cmap 的不同取值，获得的数字图像如图 3-25 所示。图 3-25(a)为默认 cmap 参数，二维数组表示的图像显示为伪彩色图。图 3-25(b)为设置 cmap＝'gray'的二维数组表示的图像，此图像显示为灰度图。对比图 3-25(a)和图 3-25(b)，可以看出，在使用 plt.imshow()函数绘制灰度图时，需要设置参数 cmap＝'gray'，这样才能正确显示灰度图像。

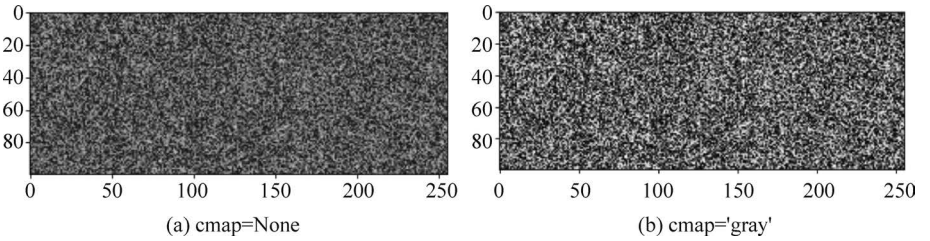


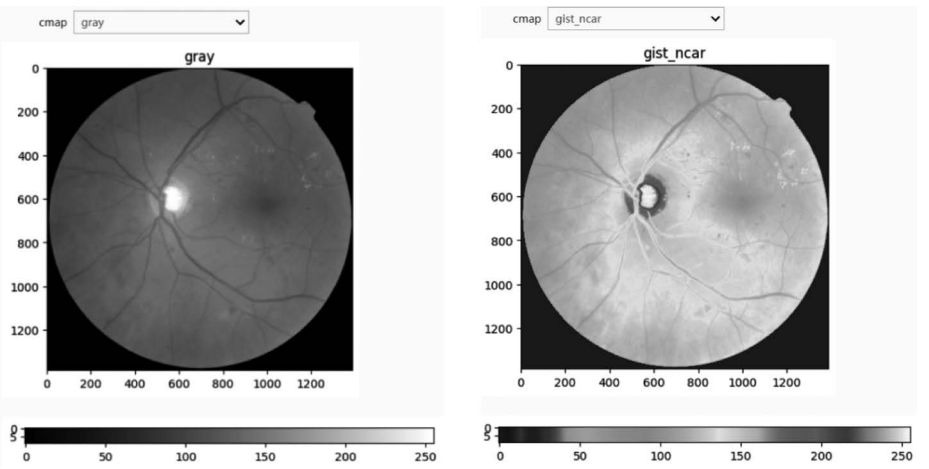
图 3-25　二维数组对应图像(见彩插)

在使用 plt.imshow()函数绘制图像时，cmap 参数决定了绘制图像的颜色映射规则。在 Matploblib 中内置了多种颜色映射规则，下面通过动态交互工具设置可选 cmap 参数，实现单通道的图像的伪彩色显示效果，示例代码如下：

```
#第 3 章/3.2 节-Matplotlib 库的简单使用.ipynb
#使用动态工具调整 cmap 参数
#Matplotlib 提供的颜色映射名称
colormaps=['Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_
r', 'BuPu', 'BuPu_r', 'CMRmap',
'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_
r', 'OrRd', 'OrRd_r', 'Oranges',
'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2',
'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu',
'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_
r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r',
 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1',
'Set1_r', 'Set2', 'Set2_r', 'Set3',
 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_
r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r',
 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_
r', 'bone', 'bone_r', 'brg', 'brg_r',
 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r',
'copper', 'copper_r', 'cubehelix',
```

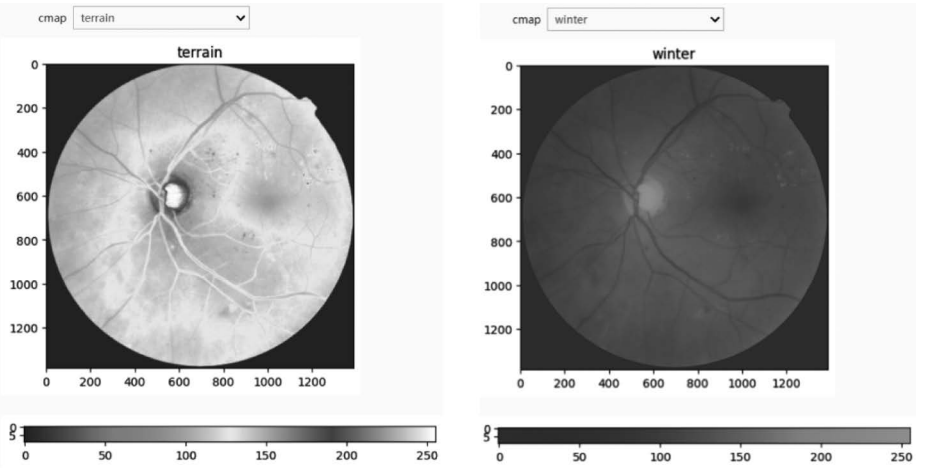
```
'cubehelix_r', 'flag', 'flag_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_
gray_r', 'gist_heat', 'gist_heat_r',
 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_
stern_r', 'gist_yarg', 'gist_yarg_r',
 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r',
'hsv', 'hsv_r', 'inferno', 'inferno_r',
 'jet', 'jet_r', 'magma', 'magma_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean',
'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r',
 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'seismic', 'seismic_r', 'spring', 'spring_
r', 'summer', 'summer_r', 'tab10', 'tab10_r',
 'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_
r', 'turbo', 'turbo_r', 'twilight', 'twilight_r',
 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'winter', 'winter_r']
img=Image.open('./images/1.jpg').convert('L')
def tmp(cmap='gray'):
   plt.imshow(img,cmap=cmap)
   plt.title(cmap)
   plt.show()                    #为什么显示的不是灰度图
   plt.imshow([list(range(256))]*10,cmap=cmap)
   plt.show()
#显示颜色映射控件
t=interact(tmp, cmap=colormaps)
```

示例代码的运行结果如图 3-26 所示，通过设置 cmap 不同的取值，使用 plt.imshow()函数绘制图像，可以得到不同伪彩色显示的图像。可以根据需要调整 cmap 参数，得到高对比度的感兴趣区域的图像，便于更好地分析和处理图像。例如设置 cmap='gist_ncar'，可以使眼球中心区域的对比度提高。



(a) cmap='gray'　　(b) cmap='gist_ncar'

图 3-26　不同 cmap 取值的图像绘制效果(见彩插)



(c) cmap='terrain'　　(d) cmap='winter'

图 3-26　(续)

对于形状为 $M\times N\times 3$ 或 $M\times N\times 4$ 的三维数组，与灰度图像相似，数组的前两维表示图像的高和宽，最后一维表示图像的通道维，其值表示图像的通道数。plt.imshow()函数会把形状为 $M\times N\times 3$ 的数组按照 RGB 颜色模式显示为彩色图像，将形状为 $M\times N\times 4$ 的数组按照 RGBA 的颜色模式显示为带有透明通道的彩色图像。此外，plt.imshow()对数组元素的数据类型和大小是有要求的，如果数组元素的类型是浮点数，则其值必须为 0～1；如果数组元素的类型是整数，则其值必须为 0～255。

下面将建立三维数组，并使用 plt.imshow()函数将三维数组绘制为彩色图像，示例代码如下：

```
#第 3 章/3.2 节-Matplotlib 库的简单使用.ipynb
#创建一个三维数组，元素类型为 uint8 类型、元素值为 0、尺寸为 100×256×3
imgar=np.zeros((100,256,3),dtype='uint8')
#将数组中表示红色通道的元素修改为纯红色
imgar[...,0]=255
plt.imshow(imgar)                                    #将数组显示为 RGB 图像
plt.show()
#创建一个三维数组，元素类型为 float32 类型、元素值为 0、尺寸为 100×256×4
imgar=np.zeros((100,256,4),dtype='float32')
#将数组中表示红色通道的元素修改为纯红色
imgar[...,0]=1.0
imgar[...,3]=0.3
#将数组中表示透明通道的元素修改为半透明
plt.imshow(imgar)                                    #将数组显示为 RGBA 图像
plt.show()
```

示例代码使用 NumPy 分别创建了两个元素值为 0 的表示 RGB 和 RGBA 图像的数组。第 1 个数组的元素是整型，将红色通道赋值为 255，表示纯红色的图像，效果如图 3-27(a)所

示；第 2 个数组的元素为浮点数，将红色通道赋值为 1.0，表示纯红色，将透明通道赋值为 0.3，表示透明度较高，效果如图 3-27(b)所示。

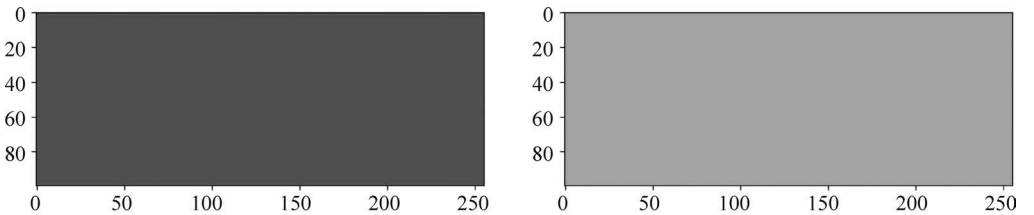


图 3-27　三维数组的彩色显示(见彩插)

将数组显示为图像，表明了图像与数组具有密切的关系，借助于 NumPy 强大的数组运算功能，能够完成丰富的图像处理任务。在第 4 章中将会详细介绍与图像处理相关的 NumPy 数组运算。

3.2.2　图形绘制

Matplotlib 中的 pyplot 模块提供了可以用来绘制各种图形的函数，具体如表 3-4 所示。

表 3-4　绘图类型

函　数	描　述	函　数	描　述
bar	绘制条形图	polar	绘制极坐标图
barh	绘制水平条形图	scatter	绘制 x 与 y 的散点图
hist	绘制直方图	stackplot	绘制堆叠图
his2d	绘制二维直方图	stem	绘制二维离散数据
pie	绘制饼图	step	绘制阶梯图
plot	绘制折线图	quiver	绘制一个二维箭头

1. 折线图

通常情况下，pyplot.plot()函数作为绘制二维图形的基本函数使用较多，pyplot.plot()函数的用法如下：

```
pyplot.plot(x,y,fmt,data=None,**kwargs)
```

(1) x,y：表示所要绘制点或线的节点，x 为 x 轴数据，y 为 y 轴数据，数据都可以为列表或者 NumPy 数组。

(2) fmt：可选参数，是一个定义图的基本属性的字符串，由颜色、线的样式和数据上的标记点 3 部分构成，格式为 fmt ='[color][marker][line] '，用于定义所要绘制点或线的基本格式。

(3) ** kwargs：可选参数，用于在二维平面上设置指定属性，绘图常用的属性有以下几个。

① color(c)：指定折线的颜色，默认为蓝色。可以使用颜色名称(如'red'、'green')来指

定颜色。

② linestyle(ls)：指定折线的样式，默认为实线('-')。常用的样式包括实线('-')、虚线('--')、点线(':')和点画线('-.')。

③ linewidth(lw)：指定折线的宽度，默认为 1。

④ marker：指定折线上数据点的标记，默认不显示数据点。常用的标记包括圆形('o')、方形('s')、三角形('^')。

⑤ markersize：指定标记的大小，默认为 6。

⑥ label：用于给折线添加标签，可以在图例中显示。

⑦ alpha：指定折线的透明度，默认为 1，取值范围为 0(完全透明)到 1(完全不透明)。

下面通过绘制曲线图来介绍 pyplot.plot()函数的使用，示例代码如下：

```
#第 3 章/3.2 节-Matplotlib 库的简单使用.ipynb
#绘制曲线图
x=np.linspace(-3, 3, 30)                    #x 轴取值范围
y=x**2                                      #纵坐标取值
plt.plot(x,y)                               #曲线绘制
plt.show()                                  #曲线显示
```

代码的运行效果如图 3-28 所示，其中绘制的曲线方程为 $y=x^2$，并且 x 的取值范围为[−3,3]。在此使用了 plt.plot()函数来绘制曲线，并使用 plt.show()函数显示曲线。需要注意的是，绘制完曲线后，必须调用 plot.show()函数显示曲线，这样才能将绘制的曲线展示出来。

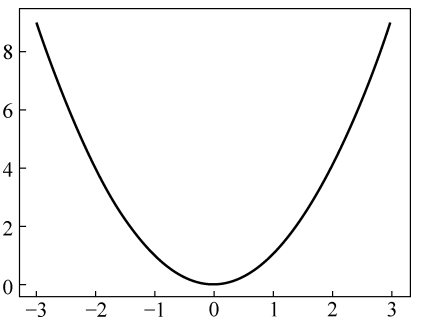


图 3-28　曲线图

2. 图形对象

Matplotlib 库的 pyplot 模块提供了图形对象 figure。可以通过调用 pyplot 模块中的 figure()函数来实例化 figure 对象。figure()函数的用法如下：

```
figure(num=None,figsize=None,dpi=None,facecolor=None,edgecolor=None,frameon=True)
```

(1) num：图形编号或名称，数字为编号，字符串为名称。

(2) figsize：指定 figure 对象的宽和高，单位为英寸。

(3) dpi：指定绘制图形对象的分辨率，即每英寸有多少像素，默认值为 80。

(4) facecolor：绘制图形背景的颜色。

(5) edgecolor：绘制图形边框的颜色。

(6) frameon：是否显示边框。

通过创建 figure 对象进行图形绘制，示例代码如下：

```
#第 3 章/3.2 节-Matplotlib 库的简单使用.ipynb
#通过创建 figure 对象进行图形绘制
```

```
x=np.linspace(-3, 3, 30)                    #x 轴取值范围
y=x **2                                     #纵坐标取值
#创建画布,大小为 5×5,图形背景为蓝色
plt.figure(1,figsize=(5,5),facecolor='blue')
plt.plot(x,y)
plt.show()
```

示例代码的运行结果如图 3-29 所示。与图 3-28 相比,图 3-29 设置了 figure 对象的尺寸和背景颜色。

除可以绘制单个图形图像外,figure 对象还可以创建多个子图,通过 subplot()函数将 figure 对象划分为 n 个区域,通过 n 次调用 subplot()函数创建 n 个子图。subplot()函数的用法如下:

```
subplot(nrows,ncols,index)
```

(1) nrows:subplot 的行数,取值为整数。

(2) ncols:subplot 的列数,取值为整数。

(3) index:子图索引,用来选定具体第几幅子图。初始值为 1,表示左上角子图。

例如,subplot(235)表示在当前的 figure 对象创建一个 2 行 3 列的绘图区域,如图 3-30 所示,同时选择在第 5 个位置上绘制子图。

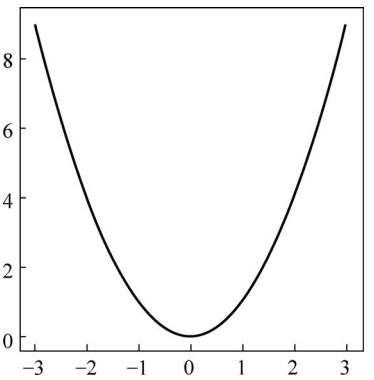


图 3-29 figure 对象创建

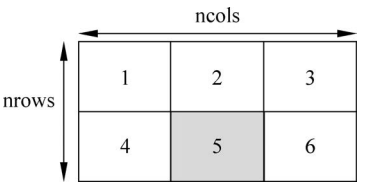


图 3-30 subplot 绘图示意图

示例代码如下:

```
#第 3 章/3.2 节-Matplotlib 库的简单使用.ipynb
#创建多个子图
x=np.linspace(-3, 3, 30)                    #x 轴取值范围
y1=x **2                                    #纵坐标取值
y2=x **3                                    #纵坐标取值
plt.figure()                                #创建画布
plt.subplot(121)                            #创建 1 行 2 列 2 个子图,并准备绘制第 1 个子图
plt.plot(x,y1)                              #图形绘制
plt.subplot(122)                            #绘制第 2 个子图
plt.plot(x,y2)
plt.show()
```

代码的运行结果如图 3-31 所示，绘制的两个子图按照 1 行 2 列排序，第 1 个子图绘制函数曲线 $y_1=x^2$，第 2 个子图绘制函数曲线 $y_2=x^3$。

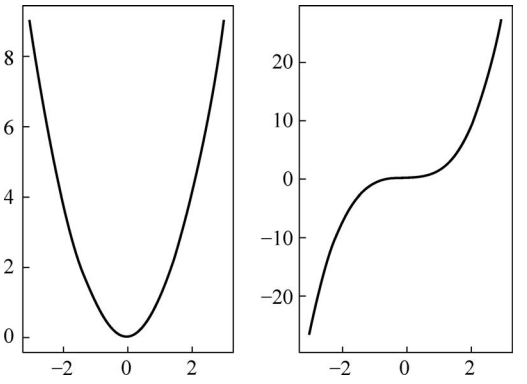


图 3-31　子图绘制结果

3. 绘图属性

Matplotlib 库提供了丰富的图形绘制属性，例如绘图符号、绘图线型、绘图颜色、图例等。

Matplotlib 库支持的绘图线的类型(linestyle)如表 3-5 所示，包含实线、点虚线、破折线和点画线。

表 3-5　绘图线型

类　　型	简　　写	描　　述
'solid'	'-'	实线
'dotted'	':'	点虚线
'dashed'	'--'	破折线
'dashdot'	'-.'	点画线

Matplotlib 库支持的绘图线的颜色(color)如表 3-6 所示，包含红色、绿色、蓝色、青色、品红、黄色、黑色和白色。

表 3-6　绘图线颜色

颜色标记	描　　述	颜色标记	描　　述
'r'	红色	'm'	品红
'g'	绿色	'y'	黄色
'b'	蓝色	'k'	黑色
'c'	青色	'w'	白色

Matplotlib 库支持的绘图标记符(marker)如表 3-7 所示，在绘制图形时，通过添加标记符，可以提高图形的可视化效果。

注意：plt.plot()函数的 fmt 参数格式'[color][marker][line]'中的 color 可设置为表 3-6 中的值，marker 可设置为表 3-7 中的值，line 可设置为表 3-5 中的值。

表 3-7 绘图标记符

标记符号	描述	标记符号	描述
'.'	圆点	'2'	向上 Y 形
'o'	圆圈	'3'	向左 Y 形
'x'	叉号	'4'	向右 Y 形
'D'	钻石形	'\/'	向下三角形
'H'	六角形	'/\'	向上三角形
's'	方形	'<'	向左三角形
'+'	加号	'>'	向右三角形
'1'	向下 Y 形		

legend()函数用于在绘图区域放置图例,图例可以展示每条数据曲线的名称。legend()函数的用法如下:

```
legend(handles,labels,loc)
```

(1) handles:所有线型的实例,为一个序列。

(2) labels:指定标签的名称,参数用字符串表示。

(3) loc:指定图例位置的参数,参数可用字符串或整数表示,loc 参数的表示方法如表 3-8 所示。

表 3-8 loc 参数

位置	字符串表示	整数表示
自适应	best	0
右上方	upper right	1
左上方	upper left	2
左下方	lower left	3
右下方	lower right	4
右侧	right	5
居中靠左	center left	6
居中靠右	center right	7
底部居中	lower center	8
上部居中	upper center	9
中部	center	10

下面通过代码来理解绘图属性和图例的使用方法和效果,具体示例代码如下:

```
#第 3 章/3.2 节-Matplotlib 库的简单使用.ipynb
#绘制含有多条曲线的图形,并设置图形属性
x=np.linspace(-3, 3, 10)                    #x 轴取值范围
y1=x**2                                     #曲线 1
y2=x**3                                     #曲线 2
plt.plot(x, y1, color='r', marker='o', ls='--', lw=1, label='y=x**2')   #曲线 1,颜色为
#红色,点标记为圆圈,线型为虚线,线宽为 1,曲线标签为 x**2
```

```
plt.plot(x, y2, color='b', marker='s', ls='-', lw=2, label='y=x**3')   #曲线 2,颜色为
#蓝色,点标记为方形,线型为实线,线宽为 1,曲线标签为 x**3
plt.legend(loc='upper left')                    #图例放置在左上方
plt.show()
```

示例代码的运行效果如图 3-32 所示,示例代码绘制了两条曲线,并且曲线的配置不同。曲线 y_1 的颜色为红色,标记符为圆圈,曲线为虚线,线的宽度为 1,并且曲线标签为 $y=x$ ** 2。曲线 y_2 的颜色为蓝色,标记符为圆圈,曲线为实线,线的宽度为 2,并且曲线标签为 $y=x$ ** 3。

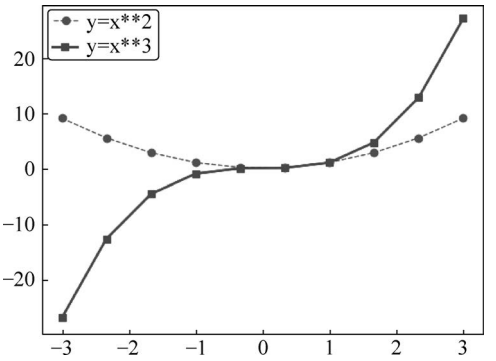


图 3-32　多曲线绘图

3.3　本章小结

本章主要介绍了图像处理库 Pillow 和绘图库 Matplotlib 的简单使用方法,一方面初步认识了图像处理,另一方面熟悉了图像的存取和显示方法,为接下来的图像处理做好准备。在学习使用 Pillow 库时,通过介绍和使用库函数,简单地介绍了图像生成、图像存取与现实、图像属性查询和图像操作等图像处理方法。在学习 Matplotlib 库时,以图像绘制、图形绘制及绘图属性为例,初步介绍了图像的绘制、显示方法,以及简单的折线图形绘制方法。