Python

计算机视觉应用

Python JISUANJI SHIJUE YINGYONG

主编　刘国华

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内容提要

Python 作为实现计算机视觉编程的第一大语言，简单方便，是一种效率极高的语言。本书分 10 章，首先，介绍了 Python 在计算机视觉中图像基本操作及在传统图像处理方面的编程应用；其次，介绍了深度卷积神经网络基础及 PyTorch 深度学习框架；最后，介绍了 Python 在图像分类、目标检测和语义分割中的典型应用及基于 Python 语言的实现。在每一章的结尾都附有必要的习题，满足教学或自学练习的需要，以便读者加深对本书所述内容的理解。

本书深度适中，内容力求精练，既可作为高等学校计算机科学与技术、电子信息工程、通信与信息工程等专业本科生与研究生教学参考书，也可供从事计算机视觉、人工智能等相关领域工作的科研人员和工程技术人员参考。

图书在版编目(CIP)数据

Python 计算机视觉应用

Python JISUANJI SHIJUE YINGYONG

主　　编：刘国华

出版发行：上海交通大学出版社　　地　　址：上海市番禺路 951 号

邮政编码：200030　　电　　话：021-64071208

印　　制：三河市骏杰印刷有限公司　　经　　销：全国新华书店

开　　本：850 mm×1 168 mm　1/16　　印　　张：15

字　　数：325 千字

版　　次：2024 年 8 月第 1 版　　印　　次：2024 年 8 月第 1 次印刷

书　　号：ISBN 978-7-313-　-

定　　价：69.80 元

前言

21 世纪以来数字图像处理进入了深度学习和大数据时代。在这一时期，深度学习技术的快速发展推动了图像处理的进一步突破。基于卷积神经网络的图像识别、图像生成等任务取得了显著的突破，使得图像处理在人脸识别、自动驾驶、虚拟现实等领域得到广泛应用。同时，大数据和云计算的发展也为图像处理提供了更多的计算和存储资源，加速了图像处理技术，特别是计算机视觉技术的发展。

实现图像理解是计算机视觉的终极目标。随着数字化、网络化、智能化深度融合的趋势不断加强，计算机视觉技术在各个领域得到更广泛的应用。同时，在党的二十大提出的建设现代化产业体系的战略背景下，计算机视觉技术也将成为推动制造业高端化、智能化、绿色化发展的重要手段之一。

Python 作为实现计算机视觉编程的第一大语言，包含图像处理、数学计算和数据挖掘等各领域学科的标准库，操作方便简单，是一种效率极高的语言。相比其他众多的语言，使用 Python 编写时，程序包含的代码行更少。Python 的语法也有助于创建整洁的代码，相比使用其他语言，使用 Python 编写的代码更容易阅读、调试和扩展。Python 是一种面向对象的解释型计算机程序设计语言，其使用具有跨平台的特点，可以在 Linux、macOS 以及 Windows 系统中搭建环境并使用，其编写的代码在不同平台上运行时，几乎不需要做较大的改动，使用者无不受益于它的便捷性。

简而言之，Python 是一种使你在编程时能够保持自己风格的程序设计语言。无须费劲就可以实现你想要的功能，并且编写的程序清晰易懂（和当前流行的其他各种程序设计语言相比更是如此）。

此外，Python 的强大之处在于它的应用领域范围之广，遍及人工智能、科学计算、Web 开发、系统运维、大数据及云计算、金融、游戏开发等。Python 还在科学领域被大量用于学术研究和应用研究。

本书向读者介绍了计算机视觉相关理论，并展示了 Python 语言在计算机视觉中的编程应用，特别介绍了计算机视觉三大任务，包括图像分类、目标检测和语义分割，以实际案例为主，展示了 Python 语言是如何实现从环境的搭建、代码编写到结果输出的，通过理论与实例相结合的方式，真正实现基于 Python 语言的计算机视觉应用；此外本书也介绍了计算机视觉应用中迁移学习、注意力机制、模型压缩等实用技巧。

本书适用于大学二年级以上（具备必要的数学基础）的相关专业的本科生、研究生，工作在人工智能领域一线的工程技术人员，以及对于计算机视觉感兴趣的并且具备必要预备知识的所有读者。

本书由天津工业大学刘国华教授执笔，连海洋、郭长瑞、赵伟、任家伟、李奕钧、吕世杰、赵英杰参与编写工作并进行程序实验，刘国华负责统稿、定稿。在编写过程中，编者参考了大量书籍、论文、资料和网站文献，也引用了其中某些内容，在此对原作者表示衷心的感谢。

由于编者水平有限，书中疏漏之处在所难免，敬请读者批评指正。编者联系邮箱：liuguohua@tiangong.edu.cn。

编　者

2024 年 1 月

目录

第 1 章　图像基本操作

图像处理是指在计算机上使用算法和代码自动操控、分析和解释图像等，广泛应用于诸多学科和领域。本章讲解如何利用 Python 中的图像处理库对图像进行处理，其中包含很多基本的图像操作，并通过大量示例介绍处理图像所需的 Python 工具包，这些工具包在本书的其他章节也被广泛使用。

1.1 软件安装及环境配置

1.1.1 Anaconda 安装

Anaconda 是一个用于科学计算的 Python 发行版本，支持 Linux、Mac 和 Windows 系统，提供了包管理与环境管理的功能，可以很方便地解决 Python 并存、切换及第三方包安装的问题。Anaconda 的优点主要包括如下几点。

(1)包含 Conda。Conda 是一个环境管理器，其功能依靠 Conda 包来实现，该环境管理器与 pip 类似。

(2)可安装工具包。Anaconda 安装时会自动安装一个基本的 Python 程序，并包含相应的工具包，该 Python 的版本与 Anaconda 的版本有关。

(3)可以在不同平台上安装、使用和管理多个不同的 Python 版本。可实现新框架建立或不同版本 Python 的使用，Anaconda 可以同时实现多个 Python 版本的管理。

本节介绍 Windows 系统下 Anaconda 的安装流程。

(1)进入 Anaconda 官网，如图 1-1 所示。选择 Python 3.8 的 64 位版本，然后单击“Download”按钮即可下载。

图 1-1　Anaconda 下载选择页面

(2)下载成功后可进入 Anaconda 的安装界面,单击“Next”按钮,如图 1-2 所示。

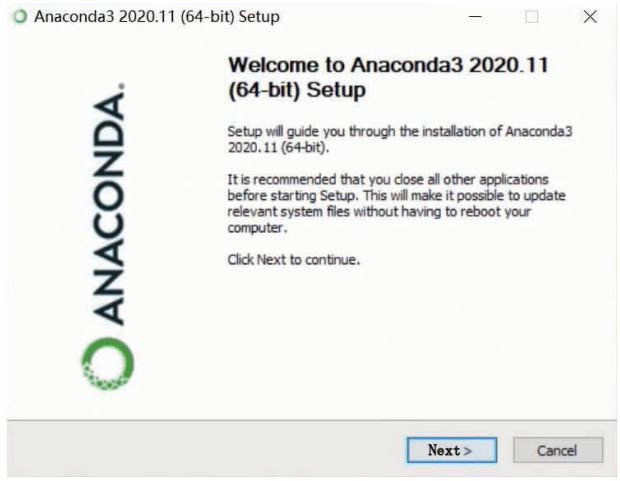


图 1-2　Anaconda 安装步骤 1

(3)在弹出的对话框中单击“I Agree”按钮,代表同意以上内容,如图 1-3 所示。

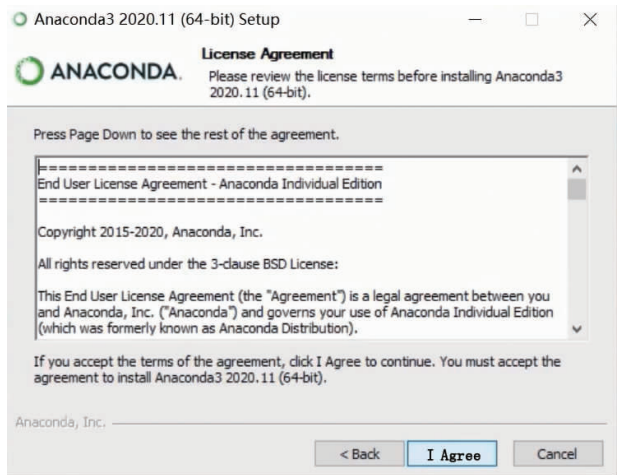


图 1-3　Anaconda 安装步骤 2

(4)在弹出的对话框中进行默认选择,然后单击“Next”按钮,如图 1-4 所示。

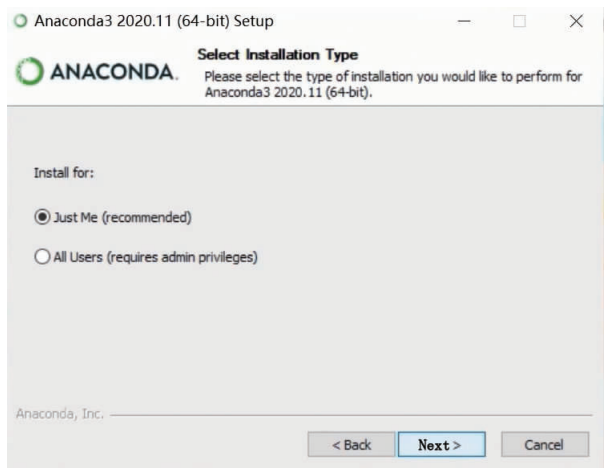


图 1-4　Anaconda 安装步骤 3

(5)在弹出的对话框中选择安装路径,建议安装在 D 盘,如图 1-5 所示;也可根据个人需要选择其他安装路径。然后单击“Next”按钮。

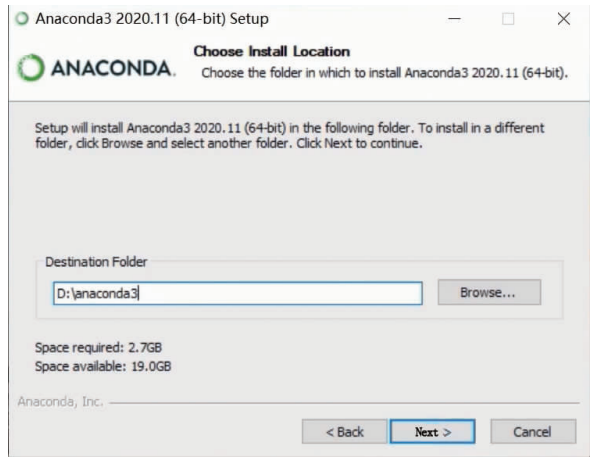


图 1-5 Anaconda 安装步骤 4

(6)在弹出的对话框中单击“Install”按钮开始安装,如图 1-6 所示。

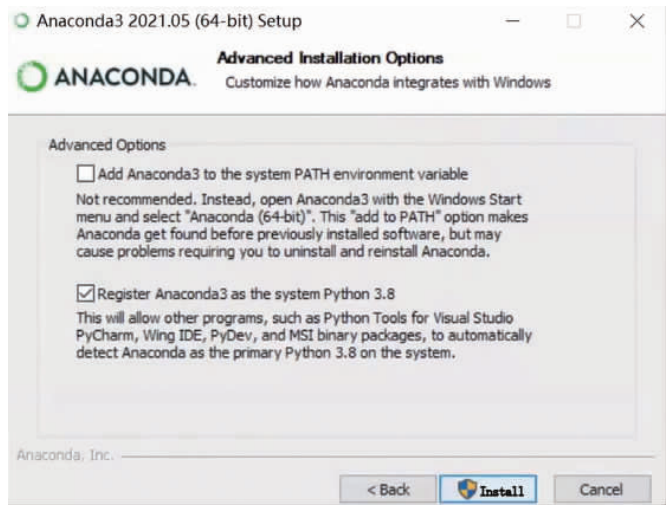


图 1-6 Anaconda 安装步骤 5

(7)安装完毕后,在弹出的对话框中单击“Finish”按钮即可,如图 1-7 所示。

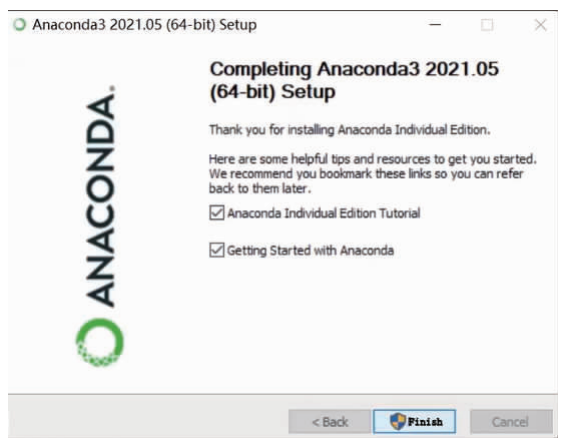


图 1-7 Anaconda 安装步骤 6

(8)检查环境变量。在终端会话中运行 Python,以检查环境变量是否正确。按“Win+R”组合键打开“运行”对话框,输入“cmd”并单击“确定”按钮,然后在弹出的命令提示符窗口中输入“python”并按“Enter”键,当出现图 1-8 所示的界面时,则说明环境变量无误,即 Python 已成功安装。

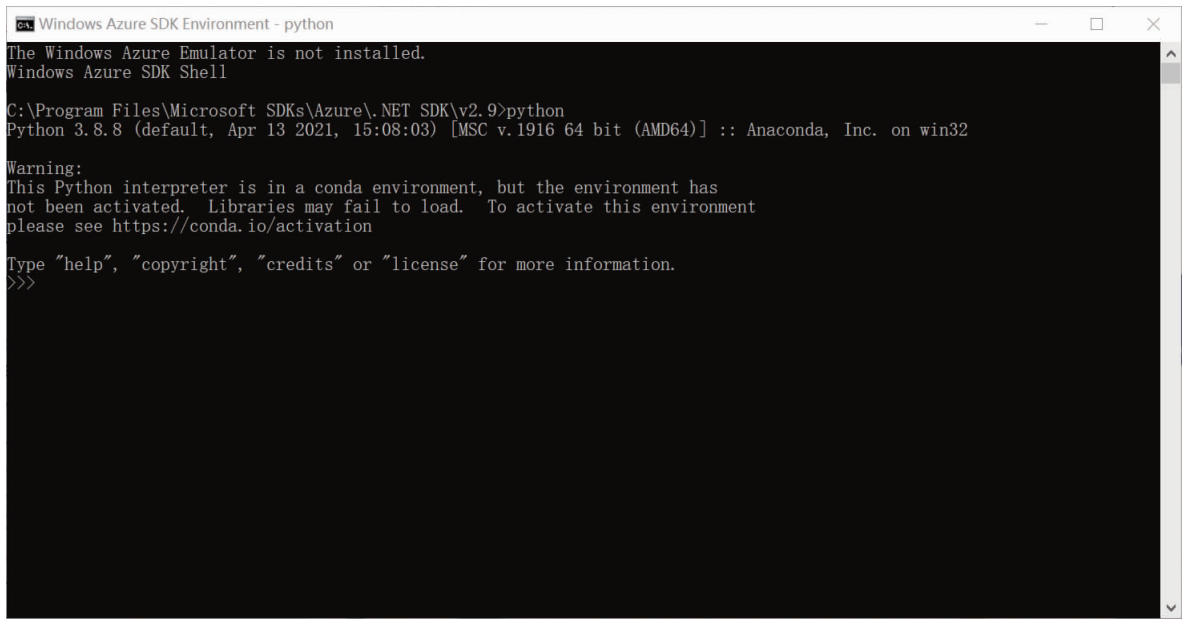


图 1-8 检测 Python 安装

安装 Anaconda 后,理论上已经可以进行 Python 编程了,但是为了提高编写程序的效率,还需要安装集成开发环境(integrated development environment,IDE)。一般选择 PyCharm 作为开发工具。

1.1.2 Pycharm 安装

PyCharm 是一款著名的 Python IDE,是一整套可以帮助用户在使用 Python 语言开发时提高效率的工具,具备基本的调试、语法检查、Project 管理、代码跳转、智能提示、单元测试、版本控制等功能。此外,该 IDE 还提供了一些高级功能,以用于支持 Django 框架下的专业 Web 开发。以 Java 开发为基础,PyCharm 和 IntelliJ IDEA 十分相似;以 Android 开发为基础,则 PyCharm 和 Android Studio 十分相似。

可以直接在官网下载 PyCharm 安装包,如图 1-9 所示。

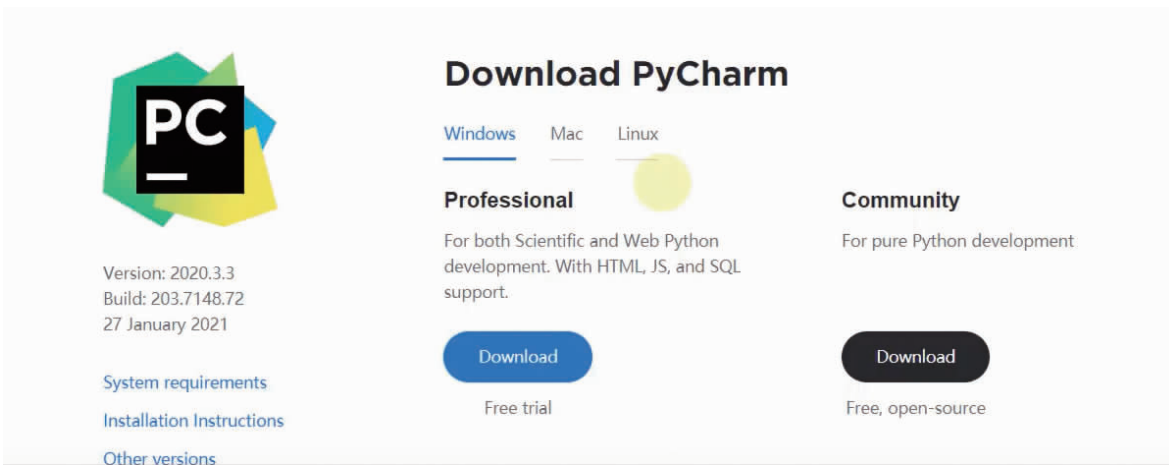


图 1-9 PyCharm 下载选择页面

PyCharm 的安装步骤如下。

(1)打开安装程序,在弹出的对话框中单击“Next”按钮,如图 1-10 所示。

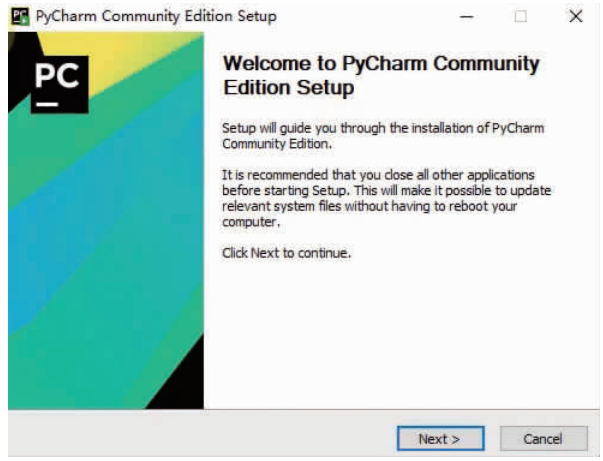


图 1-10 PyCharm 安装步骤 1

(2)在弹出的对话框中选择安装路径,然后单击“Next”按钮,如图 1-11 所示。

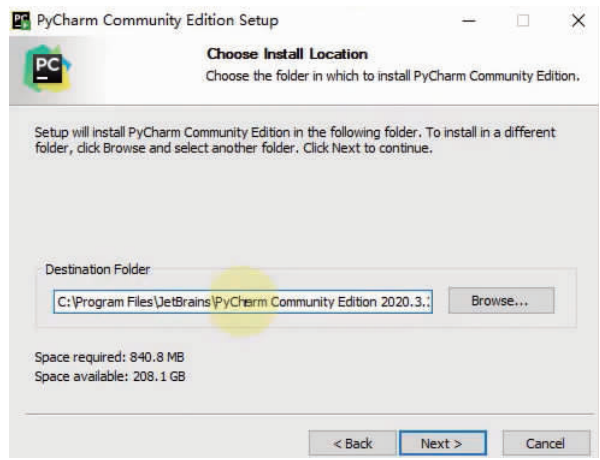


图 1-11 PyCharm 安装步骤 2

(3)在弹出的对话框中选择安装选项,然后单击“Next”按钮,如图 1-12 所示。

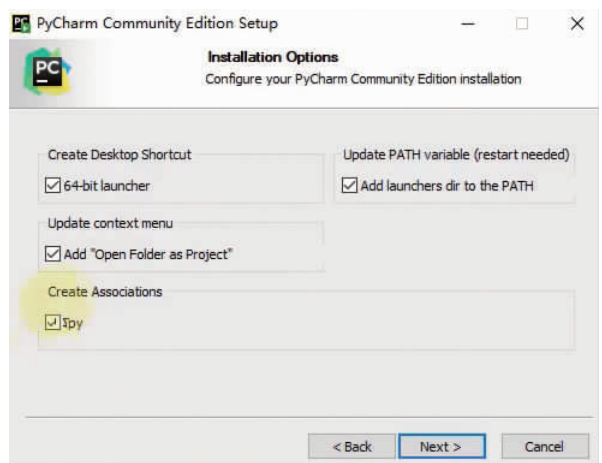


图 1-12 PyCharm 安装步骤 3

(4)在弹出的对话框中单击“Install”按钮开始安装,如图 1-13 所示。

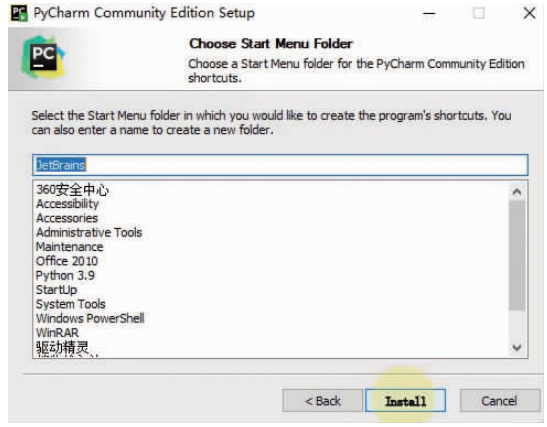


图 1-13　PyCharm 安装步骤 4

(5)安装完毕后,在弹出的对话框中单击“Finish”按钮,如图 1-14 所示。

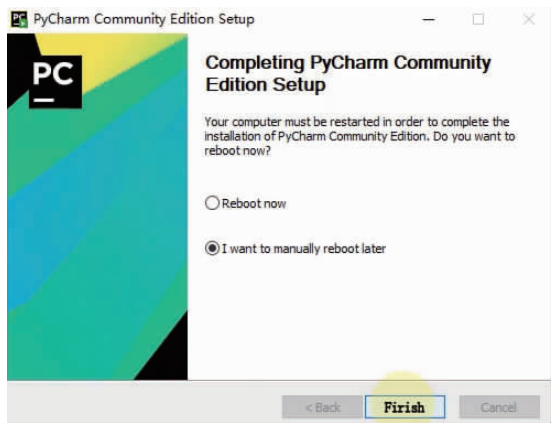


图 1-14　PyCharm 安装步骤 5

(6)安装完成后,打开 PyCharm,单击“Create Project”按钮,即可创建新的 Python 项目,输入项目存放位置及项目名称,单击“Create”按钮,如图 1-15 所示。

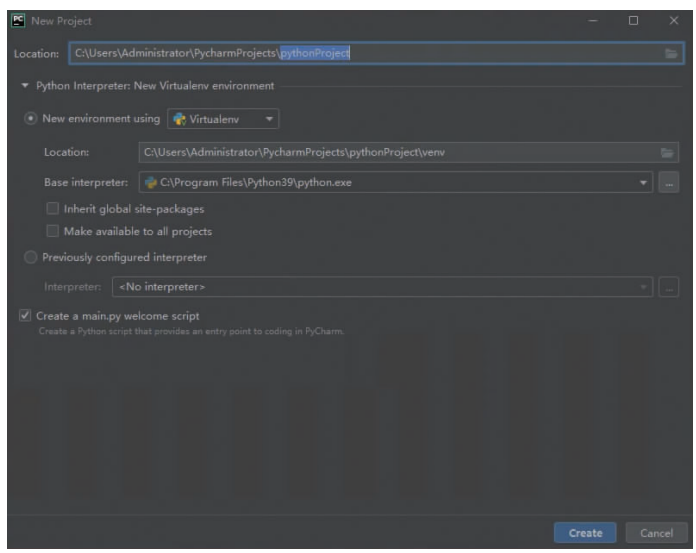


图 1-15　创建 Python 项目

(7)在项目目录上右击,在弹出的快捷菜单中选择“New”→“Python File”选项,即可创建新的Python文件,如图1-16所示。

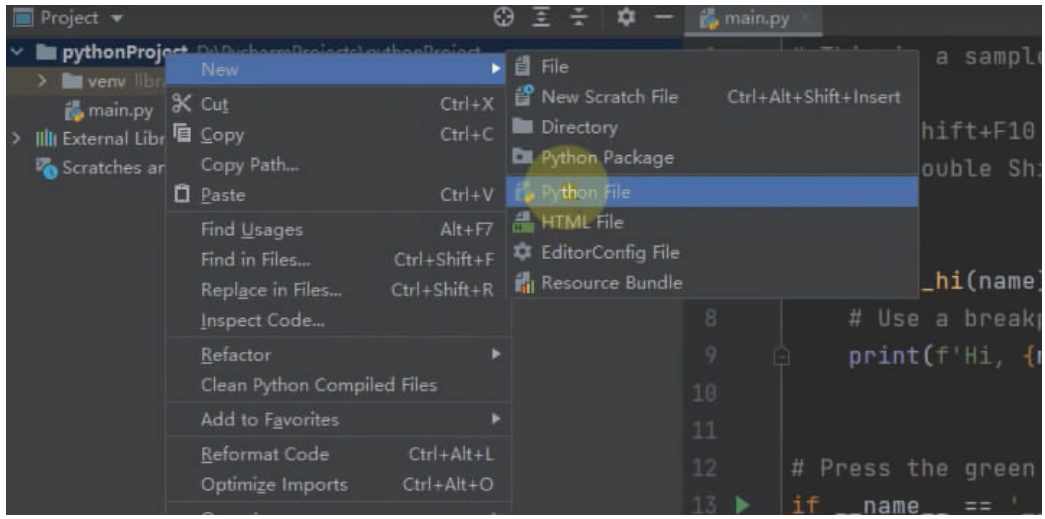


图1-16 创建新的Python文件

1.1.3 在Python中安装图像处理库

本小节主要安装后续处理图像所需要的图像处理库,需要安装的库有PIL(或Pillow)、Matplotlib、NumPy、SciPy、scikit-image等。

在命令提示符窗口中输入pip install+需要安装的第三方库并按“Enter”键,其他图像处理库都用这种方式进行安装即可,如图1-17所示(以numpy为例)。

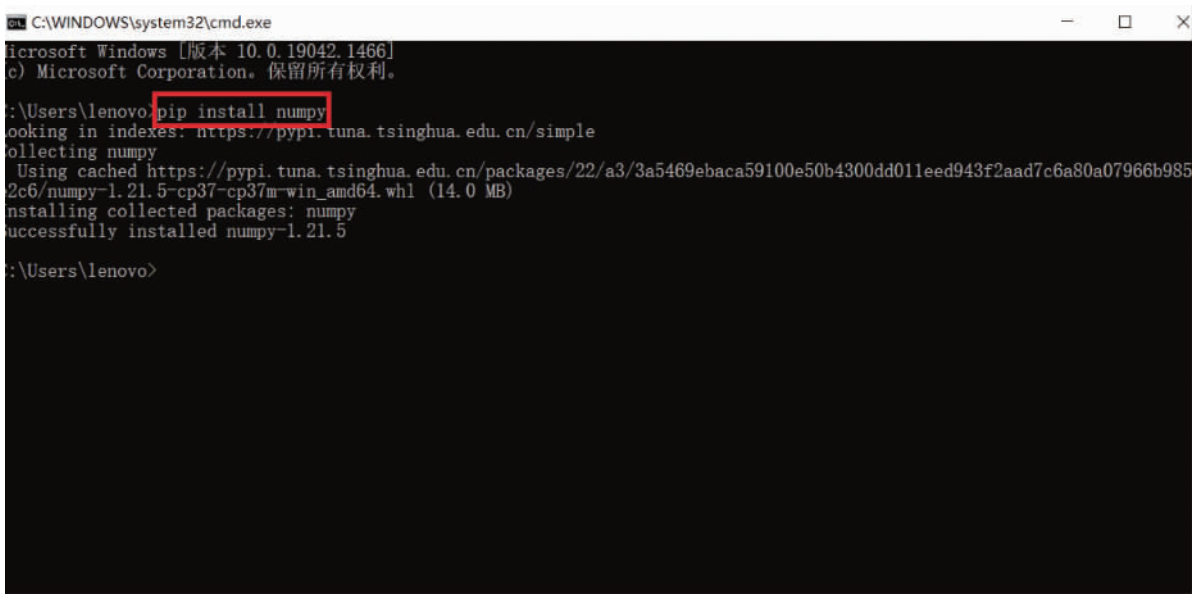


图1-17 安装numpy库

下面补充关于pip的一些更新和卸载的方法。

(1)pip自身的升级代码如下:

```
py -m pip install --upgrade pip
```

(2)pip 安装、卸载、升级代码如下：

```
安装:pip install 包名
卸载:pip uninstall 包名
升级:pip install --upgrade 包名
```

(3)pip 查看已安装包的代码如下：

```
pip list
```

(4)pip 检测需要更新包的代码如下：

```
pip list - outdated
```

(5)pip 查看包的详细信息的代码如下：

```
pip show 包名
```

(6)pip 安装指定版本的包的代码如下：

```
pip install 包名 = =版本号
```

可通过使用==、>=、<=、>、<来指定版本号。例如：

```
pip install numpy= =1.20.3
pip install'matplotlib>3.4'
pip install'matplotlib>3.4.0,<3.4.3'
```

1.2 使用 PIL 处理图像

至此正式进入图像的基本操作的学习，本章中处理图像时所用的原图均为“panda.jpg”，如图 1-18 所示。

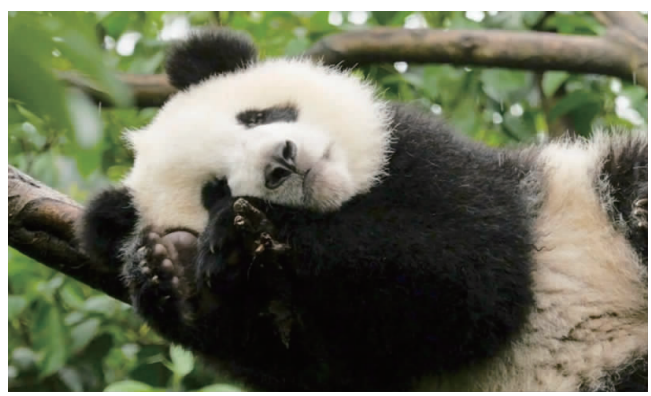


图 1-18 panda.jpg

1.2.1 读取及保存图像

Python 图像处理类库(Python imaging library，PIL)提供了通用的图像处理功能及大量有用的基

本图像操作。例如，图像缩放、裁剪、旋转、颜色转换等。利用 PIL 中的函数，可以从大多数图像格式的文件中读取数据，然后写入最常见的图像格式文件中。PIL 中最重要的模块为 Image。要读取一幅图像，可以使用如下代码：

```
#模块导入
from PIL import Image
#读取图像
im = Image.open('panda.jpg')
```

上述代码的返回值“im”是一个 PIL 图像对象。

显示图像可以使用的代码如下：

```
#显示图像
im.show()
```

程序运行结果如图 1-19 所示。

图 1-19　显 示 图 像

图像作为 PIL.PngImagePlugin.PngImageFile 类的对象加载，可以用宽度、高度和模式等属性来查找，如宽度(像素)×高度(像素)或分辨率，以及图像的模式，具体代码如下：

```
#获得图像信息
print(im.width, im.height, im.mode, im.format, type(im))
```

程序运行结果如下：

```
28 555 RGB JPEG <class'PIL.JpegImagePlugin.JpegImageFile'>
```

图像的颜色转换可以使用 convert()函数来实现。要读取一幅图像，并将其转换成灰度图像，只需要加上 convert('L')，具体代码如下：

```
#读取图像并将其转换为灰度图像
im = Image.open('panda.jpg').convert('L')
im.show()
```

程序运行结果如图 1-20 所示。

图 1-20　读取图像并将其转换为灰度图像

继续添加如下代码，即可将生成的灰度图像保存至 photos 文件夹下并命名为 panda_gray：

```
#保存图像
im.save('photos/panda_gray.jpg')
```

1.2.2　图像区域的复制粘贴

使用 crop()函数可以从一幅图像中裁剪指定区域，具体代码如下：

```
from PIL import Image
im = Image.open('panda.jpg').convert('L')
#设定裁剪的区域
box = (100,100,400,400)
#裁剪指定区域
region = im.crop(box)
#显示图像
region.show()
```

程序运行结果如图 1-21 所示。

图 1-21　裁剪指定的区域

该区域由四元组来指定。四元组的坐标依次是(左,上,右,下)。PIL 中指定坐标系的左上角坐标为(0,0)。可以旋转上面代码中获取的区域,然后使用 paste()语句将该区域放回原图中,具体代码如下:

```
#旋转区域
region = region.transpose(Image.ROTATE_180)
#粘贴区域
new_im = im.paste(region,box)
im.show(new_im)
```

程序运行结果如图 1-22 所示。

图 1-22 图像区域的旋转和粘贴

1.2.3 调整图像尺寸和旋转图像

要调整一幅图像的尺寸,可以调用 resize()函数。该方法的参数是一个元组,用来指定新图像的大小,具体代码如下:

```
from PIL import Image
im = Image.open('panda.jpg')
#调整图像的尺寸并显示图像
out = im.resize((128,128)).show()
```

程序运行结果如图 1-23 所示。

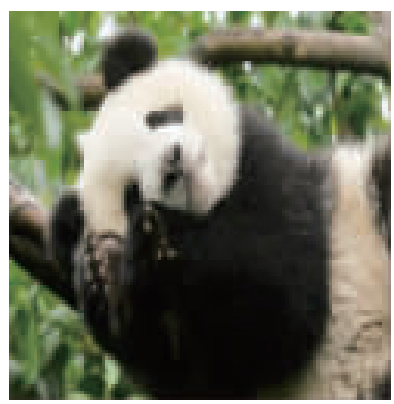


图 1-23 调整图像尺寸

要旋转一幅图像,可以使用逆时针方式表示旋转角度,然后调用 rotate()函数,具体代码如下:

```
from PIL import Image
im = Image.open('panda.jpg')
#旋转图像并显示图像
out = im.rotate(45).show()
```

程序运行结果如图 1-24 所示。

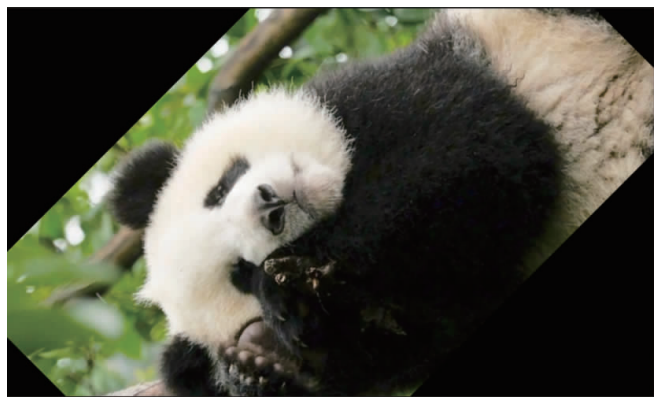


图 1-24　旋 转 图 像

1.2.4　其他图像处理

1. 图像负片

图像负片的原理是将原图像中每个像素的颜色值取反，即将颜色值的最大值(255)减去原来的颜色值，得到新的颜色值。图像负片可以由 point()函数实现，具体代码如下：

```
from PIL import Image
im = Image.open('panda.jpg')
#图像负片变换
im_t = im.point(lambda x: 255 - x)
im_t.show()
```

程序运行结果如图 1-25 所示。

图 1-25　图 像 负 片

2. 几何变换

几何变换是通过将适当的矩阵(通常用齐次坐标表示)与图像矩阵相乘来完成的。由于这些变换

会改变图像的几何方向，因此称这些变换为几何变换。

(1)镜像图像。可以使用 transpose()函数得到在水平或垂直方向上的镜像图像，具体代码如下：

```
from PIL import Image
im = Image.open('panda.jpg')
#图像镜像变换
im.transpose(Image.FLIP_LEFT_RIGHT).show()
```

程序运行结果如图 1-26 所示。

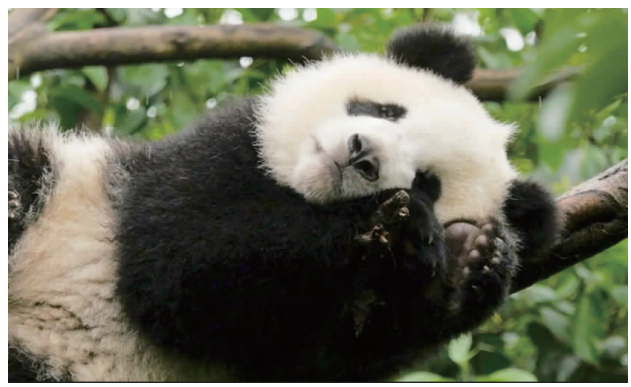


图 1-26 镜像图像

(2)仿射变换。二维仿射变换矩阵可以应用于图像的每个像素(在齐次坐标中)，以进行仿射变换，这种变换通常通过反向映射(扭曲)来实现。

下面是用 transform()函数进行仿射变换的例子。transform()函数中的数据参数是一个六元组(a,b,c,d,e,f)。对于输出图像中的每个像素(x,y)，新值取自输入图像中的位置(ax+by+c,dx+ey+f)，使用最接近的像素进行近似。transform()函数可用于缩放、平移、旋转和剪切原始图像。

```
from PIL import Image
im = Image.open("panda.jpg")
#图像仿射变换
im.transform((int(1.4 * im.width), im.height), Image.AFFINE,data = (1,-0.5,0,0,1,
0)).show()
```

程序运行结果如图 1-27 所示。

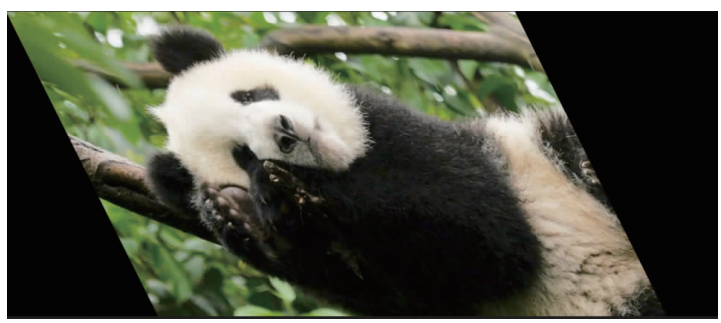


图 1-27 图像的仿射变换

3. 更改像素值

可以使用 putpixel()函数更改图像中的像素值。使用函数向图像中添加噪声可以通过从图像中随

机选择几个像素值，然后将这些像素值的一半设置为黑色，另一半设置为白色，来为图像添加椒盐噪声(salt-and-pepper noise)。添加椒盐噪声的具体代码如下：

```
#模块导入
import numpy as np
from PIL import Image
im = Image.open("panda.jpg")
#设置选择像素点的数量
n = 5000
#随机选择 5000 个像素点
x, y = np.random.randint(0, im.width, n), np.random.randint(0, im.height,n)
for (x,y) in zip(x,y):
#添加椒盐噪声
im.putpixel((x, y), ((0,0,0)
if np.random.rand() < 0.5
else (255,255,255)))
im.show()
```

程序运行结果如图 1-28 所示。

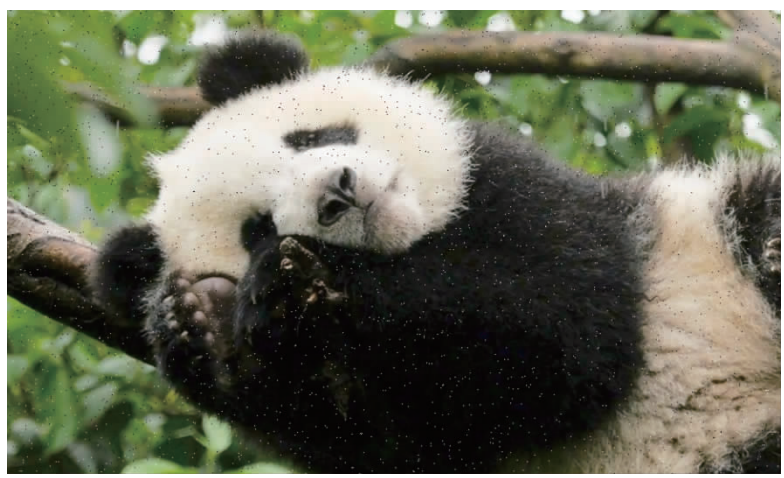


图 1-28　添加椒盐噪声

4. 绘制图形

可以用 PIL.ImageDraw 模块中的函数在图像上绘制线条或其他几何图形。例如，ellipse()函数可用于绘制椭圆，具体代码如下：

```
from PIL import Image, ImageDraw
im = Image.open("panda.jpg")
draw = ImageDraw.Draw(im)
#绘制图形
draw.ellipse((125, 125, 200, 250), fill = (255,255,255,128))
im.show()
```

程序运行结果如图 1-29 所示。

图 1-29　在图像上绘制图形

5. 添加文本

可以使用 PIL. ImageDraw 模块中的 text()函数向图像添加文本，具体代码如下：

```
from PIL import Image, ImageDraw, ImageFont
im = Image.open("panda.jpg")
draw = ImageDraw.Draw(im)
#设置字体
font = ImageFont.truetype("arial.ttf", 48)
#添加文本
draw.text((10, 5), "Welcome to image processing with python", font = font)
im.show()
```

程序运行结果如图 1-30 所示。

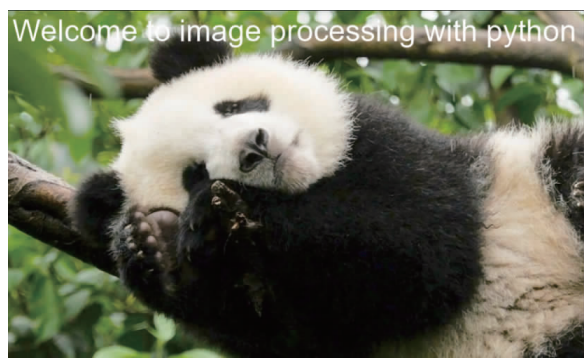


图 1-30　在图像上添加文本

1.3 使用 Matplotlib 处理图像

1.3.1 在图像中绘制点和线

处理数学运算、绘制图表，或在图像上绘制点、直线和曲线时，Matplotlib 是个很好的类库，具有比 PIL 更强大的绘图功能。Matplotlib 中的 PyLab 接口包含很多方便用户创建图像的函数。下面以采用几个点和一条线绘制图像为例进行介绍，具体代码如下：

```
from PIL import Image
from pylab import *
#读取图像到数组中
im = array(Image.open('panda.jpg'))
#绘制图像
imshow(im)
#取一些点
x = [100,100,400,400]
y = [200,500,200,500]
#使用红色星状标记绘制点
plot(x,y,'r*')
#绘制连接前两个点的线
plot(x[:2],y[:2])
#添加标题,显示绘制的图像
title('Plotting: "panda.jpg"')
show()
```

程序运行结果如图 1-31 所示。

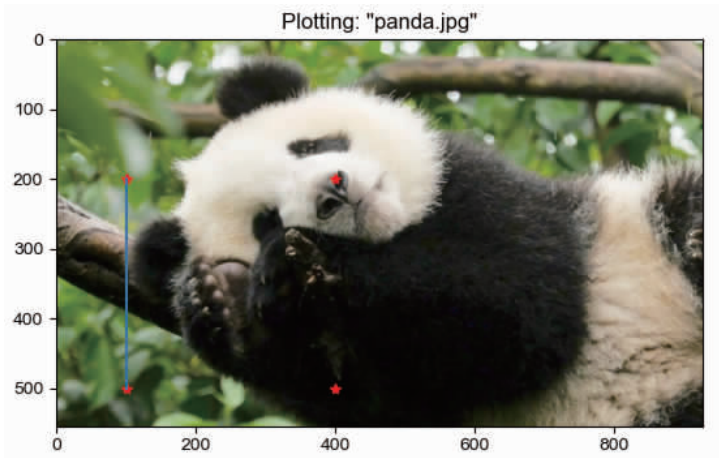


图 1-31　在图像上绘制点和线

上面的代码首先绘制出原始图像,然后在 x 和 y 列表中给定点的 x 坐标和 y 坐标上绘制出红色星状标记点,最后在两个列表表示的前两个点之间绘制一条线段(默认为蓝色)。show()命令首先打开图形用户界面(graphical user interface,GUI),然后新建一个图像窗口。该图形用户界面会循环阻断脚本,然后暂停,直到最后一个图像窗口关闭。在每个脚本中能调用一次 show()命令,而且通常是在脚本的结尾调用。注意,在 PyLab 库中,通常指定图像的左上角为坐标原点。

图像的坐标轴是一个很有用的调试工具,但是,如果想绘制出较美观的图像,坐标轴最好不显示,而添加下列命令可以使坐标轴不显示:

```
axis('off')
```

1.3.2　图像轮廓和直方图

绘制图像的轮廓(或其他二维函数的等轮廓线)是常用的操作。因为绘制轮廓需要对每个坐标[x,

y]的像素值施加同一个阈值，所以首先需要将图像灰度化，绘制轮廓的代码如下：

```
from PIL import Image
from pylab import *
#读取图像到数组中
im = array(Image.open('panda.jpg').convert('L'))
#新建一个图像
figure()
#不使用颜色信息
gray()
#在原点的左上角显示轮廓图像
contour(im, origin='image')
axis('equal')
axis('off')
show()
```

程序运行结果如图 1-32 所示。

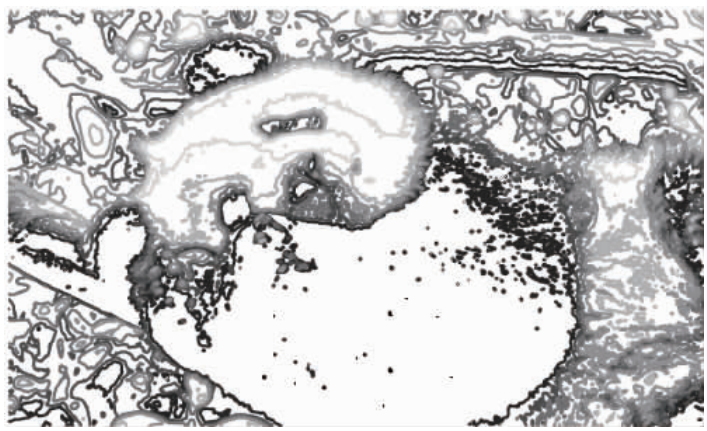


图 1-32　绘制图像的轮廓

图像的直方图用来表征该图像像素值的分布情况。用一定数目的小区间(bin)来指定表征像素值的范围，每个小区间会得到落入该小区间表示范围的像素数目。该(灰度)图像的直方图可以使用 hist()函数绘制，具体代码如下：

```
from PIL import Image
from pylab import *
#读取图像到数组中
im = array(Image.open('panda.jpg').convert('L'))
figure()
#绘制图像的直方图
hist(im.flatten(),128)
show()
```

程序运行结果如图 1-33 所示。

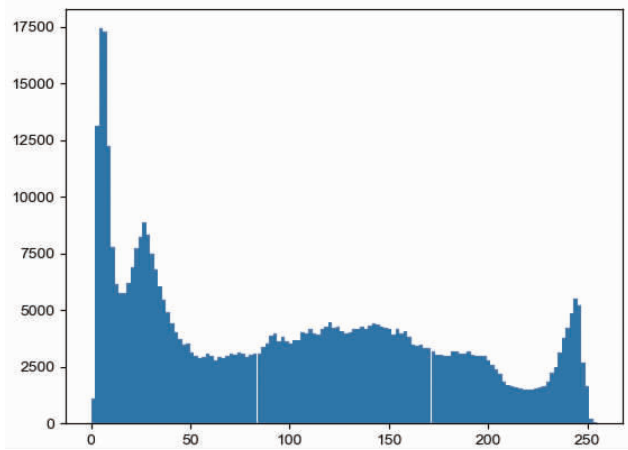


图 1-33　绘制图像的直方图

hist()函数的第二个参数指定小区间的数目。需要注意的是，因为 hist()只接受一维数组作为输入，所以在绘制图像直方图之前必须先对图像进行压平处理。flatten()函数将任意数组按照行优先准则转换成一维数组。

彩色图像的直方图可使用 histogram()函数绘制。histogram()函数可用于计算每个通道像素的直方图(像素值与频率表)，并返回相关联的输出。例如，对于 RGB 图像，输出包含 3×256＝768 个值，具体代码如下：

```
from PIL import Image
from matplotlib import pyplot as plt
im = Image.open('panda.jpg')
#计算每个通道像素的直方图
pl = im.histogram()
#绘制每个通道像素的直方图
plt.bar(range(256), pl[:256], color = 'r', alpha = 0.5)
plt.bar(range(256), pl[256:2 * 256], color = 'g', alpha = 0.4)
plt.bar(range(256), pl[2 * 256:], color = 'b', alpha = 0.3)
plt.show()
```

程序运行结果如图 1-34 所示。

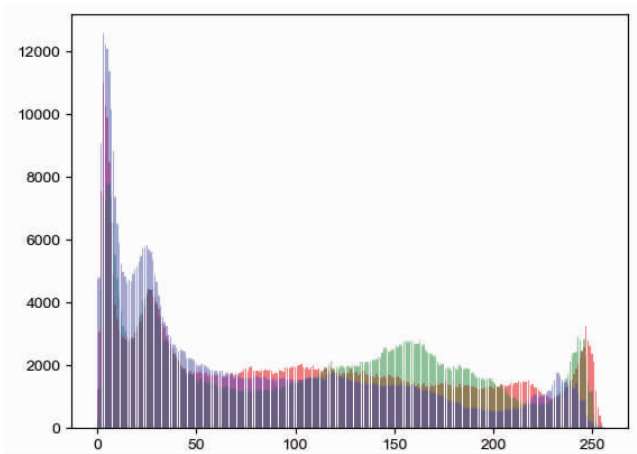


图 1-34　绘制彩色图像的直方图

可以用 split()函数来分离多通道图像的通道，如下面的代码可对 RGB 图像实现 RGB 通道的分离：

```
from PIL import Image
from matplotlib import pyplot as plt
im = Image.open('panda.jpg')
#分离图像的 RGB 通道
ch_r, ch_g, ch_b = im.split()
#分别绘制 RGB 通道图像
plt.figure(figsize=(18,6))
plt.subplot(1,3,1);plt.imshow(ch_r, cmap=plt.cm.Reds); plt.axis('off')
plt.subplot(1,3,2); plt.imshow(ch_g, cmap=plt.cm.Greens); plt.axis('off')
plt.subplot(1,3,3); plt.imshow(ch_b, cmap=plt.cm.Blues); plt.axis('off')
plt.tight_layout()
plt.show()
```

程序运行结果如图 1-35 所示，即 R(红色)、G(绿色)和 B(蓝色)通道创建的三个输出图像(a)、(b)、(c)。

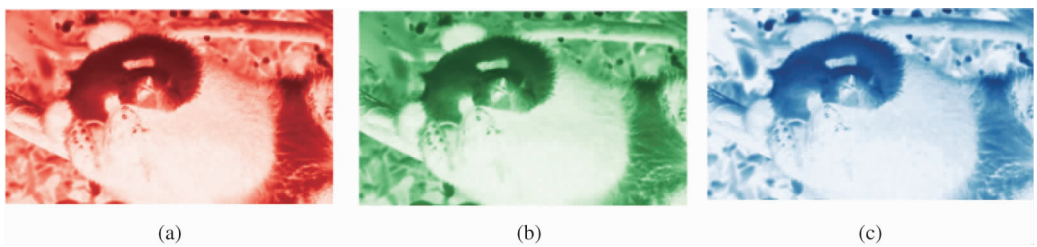


图 1-35　分离图像的 RGB 通道

合并图像的多个通道可以使用 merge()函数，代码如下所示。其中，颜色通道是通过分离 panda 的 RGB 图像，并在红蓝通道交换后合并得到的。

```
#合并多通道图像
im = Image.merge('RGB', (ch_b, ch_g, ch_r))
im.show()
```

程序运行结果如图 1-36 所示，即合并 B、G 和 R 通道而创建的输出图像。

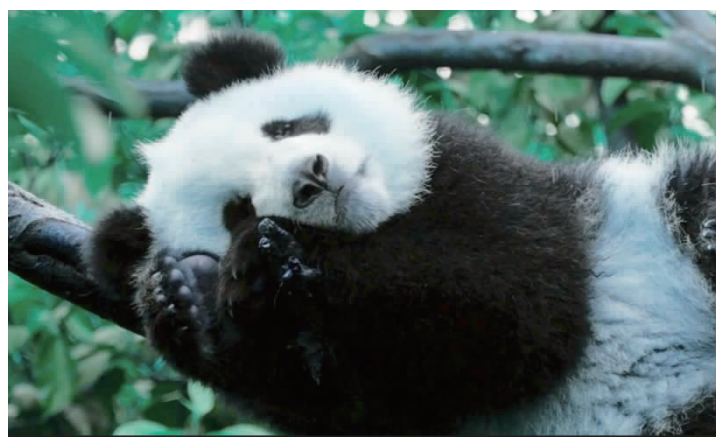


图 1-36　通过合并通道而创建的图像

1.4 使用 NumPy 处理图像

1.4.1 图像的数组化

在先前的例子中，当载入图像时，通过调用 array()函数将图像转换成 NumPy 的数组对象。NumPy 中的数组对象是多维的，可以用来表示向量、矩阵和图像。一个数组对象类似于一个列表(或列表的列表)，但是数组中所有的元素必须具有相同的数据类型。除非创建数组对象时指定数据类型，否则会按照数据的类型自动确定。

以图像数据为例，具体代码如下：

```
from PIL import Image
from numpy import array
#读取图像到数组中
im = array(Image.open('panda.jpg'))
print(im.shape, im.dtype)
im = array(Image.open('panda.jpg').convert('L'),'f')
print(im.shape, im.dtype)
```

程序运行结果如下：

```
(555, 928, 3) uint8
(555, 928) float32
```

每行的第一个元组表示图像数组的行、列、颜色通道，紧接着的字符串表示数组元素的数据类型。因为图像通常被编码成无符号八位整数(uint8)，所以在第一种情况下，载入图像并将其转换到数组中，数组的数据类型为“uint8”。在第二种情况下，对图像进行灰度化处理，并且在创建数组时使用额外的参数“f”；该参数将数据类型转换为浮点型。由于灰度图像没有颜色信息，所以在形状元组中，它只有两个数值。

数组中的元素可以使用下标访问。位于坐标 i,j，以及颜色通道 k 的像素值可以通过下述代码访问：

```
value = im[i,j,k]
```

多个数组元素可以使用数组切片方式访问。切片方式返回的是以指定间隔下标访问该数组的元素值。下面是有关灰度图像的一些例子：

```
im[i,:] = im[j,:]#将第 j 行的数值赋值给第 i 行
im[:,i] = 100#将第 i 列的所有数值设为 100
im[:100,:50].sum()#计算前 100 行、前 50 列所有数值的和
im[50:100,50:100]#第 50～100 行，第 50～100 列(不包括第 100 行和第 100 列)
im[i].mean()#第 i 行所有数值的平均值
im[:,-1]#最后一列
im[-2,:](or im[-2])#倒数第二行
```

1.4.2 灰度变换

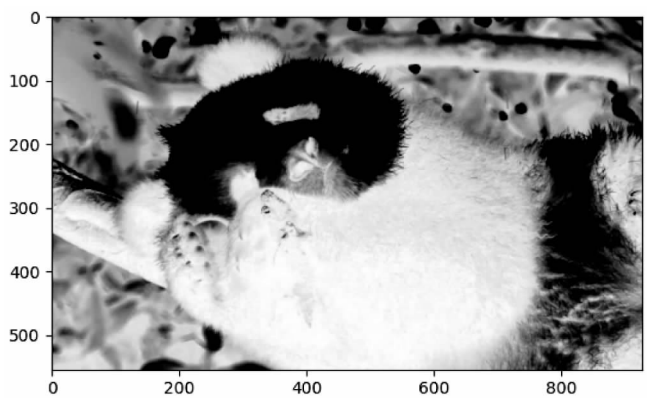
将图像读入 NumPy 数组对象后，可以对它们进行任意数学操作。一个简单的例子就是图像的灰

度变换。具体代码如下：

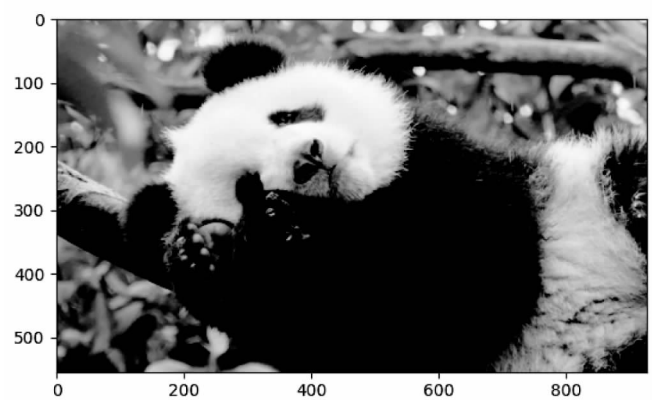
```
from PIL import Image
from pylab import *
from numpy import *
im = array(Image.open('panda.jpg').convert('L'),'f')
#对图像进行反相处理,结果为图(a)
im1 = 255 - im
figure()
imshow(im1,cmap = plt.get_cmap('gray'))
#将图像像素值变换到 100—200 区间,结果为图(b)
im2 = (100.0/255) * im + 100
figure()
imshow(im2,cmap = plt.get_cmap('gray'))
#对图像的像素值求平方后得到的图像,结果为图(c)
im3 = 255.0 * (im/255.0) ** 2
figure()
imshow(im3,cmap = plt.get_cmap('gray'))
show()
```

程序运行结果如图 1-37(a)至图 1-37(c)所示。

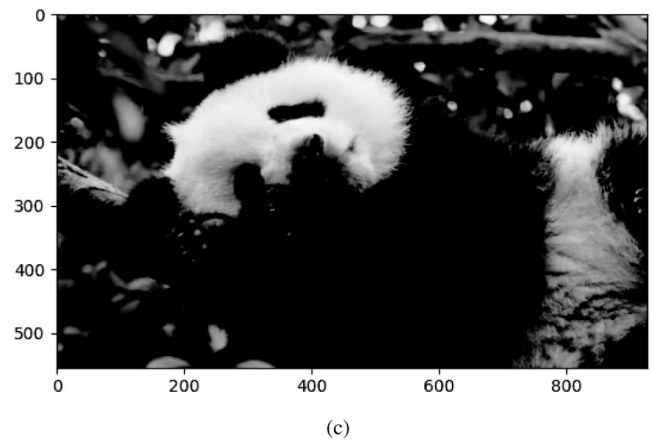
图(a)将灰度图像进行反相处理;图(b)将图像的像素值变换到 100～200;图(c)对图像的像素值求平方,使较暗的像素值变得更小。



(a)



(b)



(c)

图 1-37　图像的灰度变换

(a)对图像进行反相处理;(b)将图像像素值变换到 100～200;(c)对图像的像素值求平方

1.5 使用 SciPy 处理图像

1.5.1 图像模糊

图像模糊可通过将(灰度)图像和一个高斯核进行卷积操作来实现,SciPy 有用来进行滤波操作的 gaussian_filter 模块。该模块使用快速一维分离的方式来计算卷积。具体代码如下:

```
import matplotlib.pyplot as plt
from PIL import Image
from numpy import *
from scipy.ndimage import gaussian_filter
im = array(Image.open('panda.jpg').convert('L'))
#高斯模糊
im2 = gaussian_filter(im,5)
plt.imshow(im2,cmap = plt.get_cmap('gray'))
plt.show()
```

程序运行结果如图 1-38 所示。

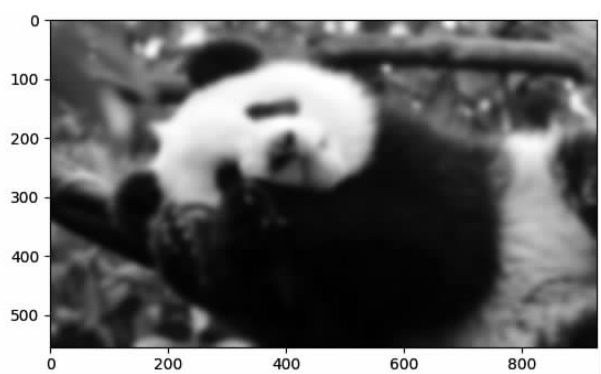
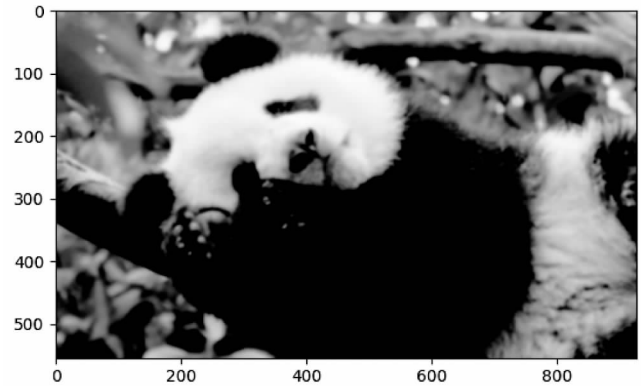
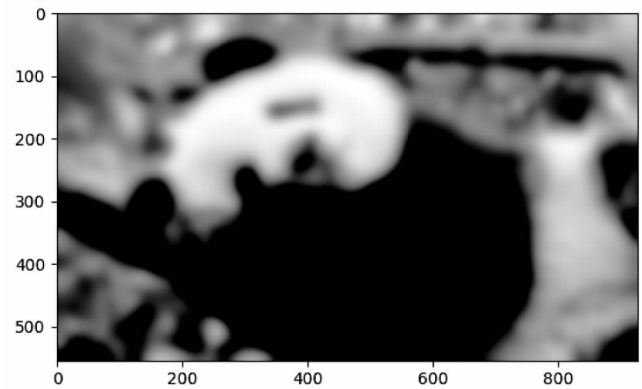


图 1-38　对图像进行高斯模糊

guassian_filter()函数的最后一个参数表示标准差σ。随着σ的增加，一幅图像被模糊的程度越大，处理后的图像细节丢失越多。σ=2 和σ=10 的模糊效果如图 1-39(a)和图 1-39(b)所示。



(a)



(b)

图 1-39　不同 σ 值的模糊效果

(a)σ=2 的模糊效果；(b)σ=10 的模糊效果

若打算模糊一幅彩色图像，则简单地对每一个颜色通道进行高斯模糊即可，具体代码如下：

```
import matplotlib.pyplot as plt
from PIL import Image
from numpy import *
from scipy.ndimage import gaussian_filter
im = array(Image.open('panda.jpg'))
#返回给定形状和类型的新数组，用零填充
im2 = zeros(im.shape)
#对每个颜色通道进行高斯模糊
for i in range(3):
  im2[:,:,i] = gaussian_filter(im[:,:,i],5)
im2 = uint8(im2)
plt.imshow(im2)
plt.show()
```

程序运行结果如图 1-40 所示。

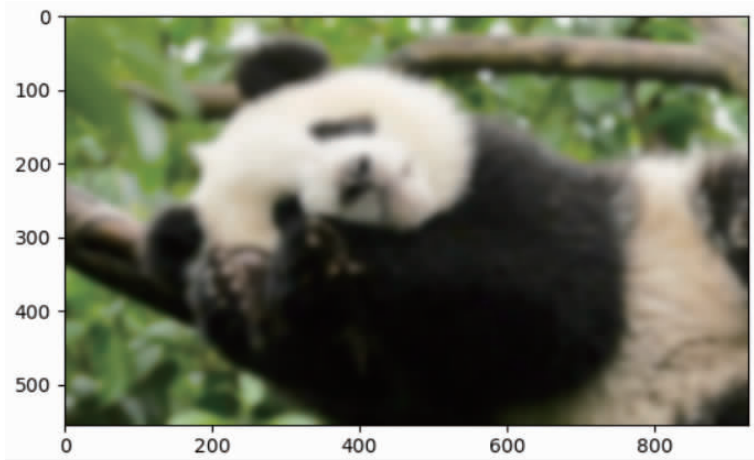


图 1-40　对彩色图像进行高斯模糊

1.5.2　图像导数

图像强度的变化情况是非常重要的信息。强度的变化可以用灰度图像(对于彩色图像,通常对每个颜色通道分别计算导数)的 x 和 y 方向导数进行描述。梯度有两个重要的属性,一个是梯度的大小,它描述了图像强度变化的强弱;另一个是梯度的角度,它描述了图像中在每个点(像素)上强度变化最大的方向。

导数滤波器可以使用 scipy. ndimage 中的 sobel 模块的标准卷积操作来简单地实现,具体代码如下:

```
import matplotlib.pyplot as plt
from PIL import Image
from numpy import *
from scipy.ndimage import sobel
im = array(Image.open('panda.jpg').convert('L'))
#Sobel 导数滤波器
imx = zeros(im.shape)
#计算 x 方向导数
sobel(im,1,imx)
imy = zeros(im.shape)
#计算 y 方向导数
sobel(im,0,imy)
#计算梯度
magnitude = sqrt(imx**2+imy**2)
plt.imshow(imx,cmap=plt.get_cmap('gray'))
plt.show()
```

上述代码使用 Sobel 导数滤波器来计算 x 和 y 的方向导数及梯度大小。sobel()函数的第二个参数表示选择 x 或 y 方向导数,第三个参数保存输出的变量。将上述代码 plt. imshow(imx,cmap=plt. get_cmap('gray'))中的第一个参数分别修改为 imx、imy、magnitude,运行程序可得到 x 方向导数图像、y 方向导数图像及梯度大小图像,如图 1-41(a)~图 1-41(c)所示。

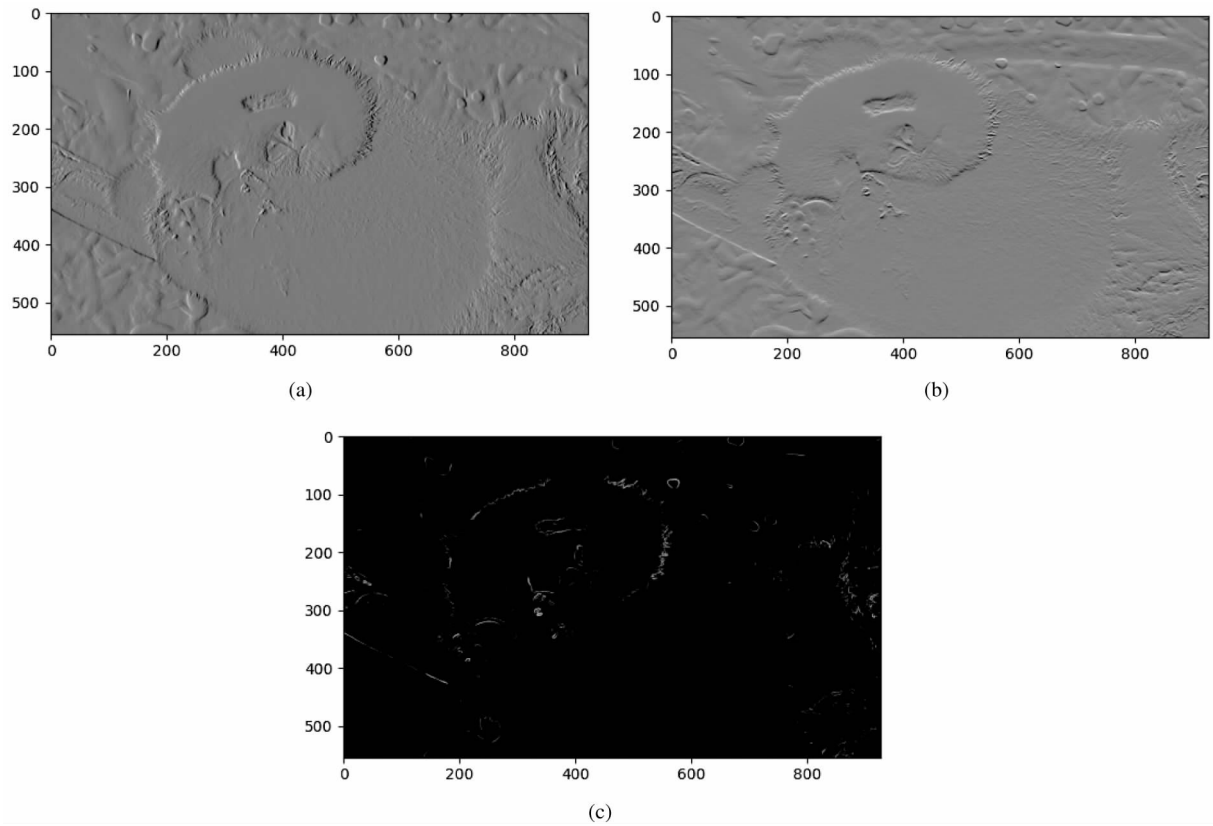


图 1-41 Sobel 滤波器计算导数

(a)x 方向导数图像；(b)y 方向导数图像；(c)梯度大小图像

上述计算图像导数的方法有一些缺陷，在该方法中滤波器的尺度需要随着图像分辨率的变化而变化。为了在图像噪声方面保持稳定，以及在任意尺度上计算导数，可以使用高斯导数滤波器，具体代码如下：

```
import matplotlib.pyplot as plt
from PIL import Image
from numpy import *
from scipy.ndimage import gaussian_filter
im = array(Image.open('panda.jpg').convert('L'))
#标准差
sigma = 5
#Gaussian 导数滤波器
imx = zeros(im.shape)
#计算 x 方向导数
gaussian_filter(im, (sigma,sigma), (0,1), imx)
imy = zeros(im.shape)
#计算 y 方向导数
gaussian_filter(im, (sigma,sigma), (1,0), imy)
plt.imshow(imx,cmap = plt.get_cmap('gray'))
plt.show()
```

上述代码使用 Gaussian 导数滤波器来计算 x 和 y 的方向导数。gaussian_filter 的第二个参数为使

用的标准差，第三个参数指定计算哪种类型的导数。将上述代码 plt. imshow(imx，cmap＝plt. get_cmap('gray'))中的第一个参数分别修改为 imx、imy。程序运行结果如图 1-42(a)和图 1-42(b)所示。

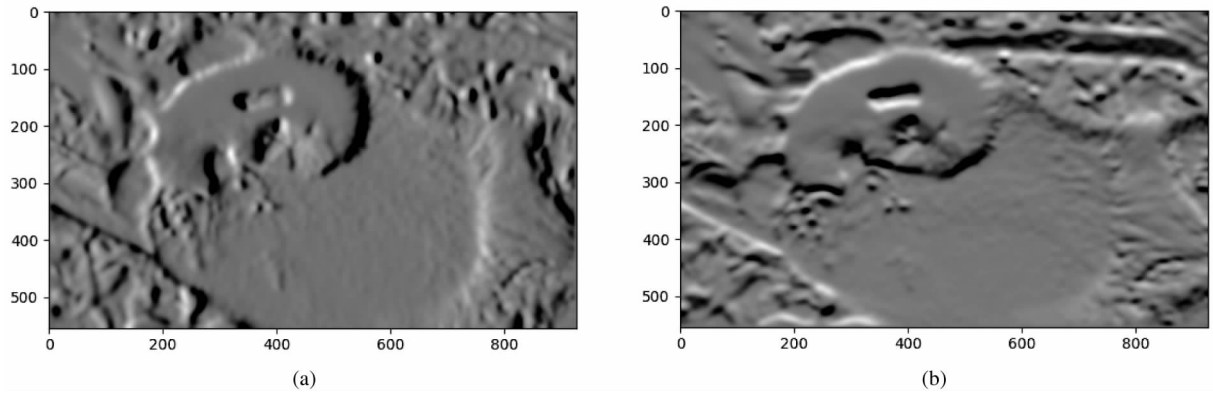


图 1-42　Gaussian 导数滤波器计算导数

(a)σ＝5 的 x 导数图像；(b)σ＝5 的 y 导数图像

σ＝2 和 σ＝10 的 x、y 方向导数图像如图 1-43(a)～图 1-43(d)所示。

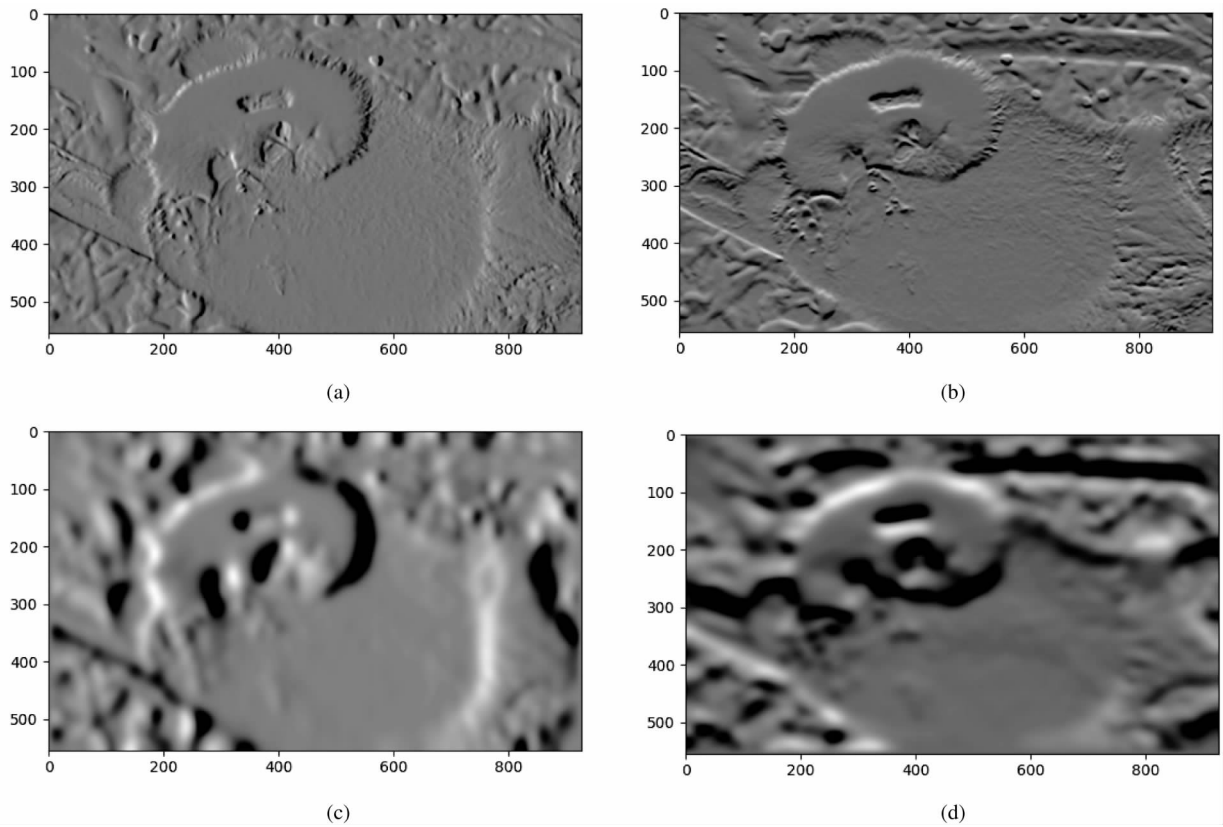


图 1-43　不同 σ 值的 x、y 方向导数图像

(a)σ＝2 的 x 方向导数图像；(b)σ＝2 的 y 方向导数图像；(c)σ＝10 的 x 方向导数图像；(d)σ＝10 的 y 方向导数图像

1.6 使用 scikit-image 处理图像

1.6.1 图像的旋流变换

scikit-image 是一个开源的 Python 图像处理库，它为图像处理任务提供了广泛的算法和工具。该库建立在 NumPy 数组之上，可以轻松地与其他科学计算库集成。scikit-image 提供了许多图像处理功能，包括旋流变换和添噪等。旋流变换(swirl transform)是 scikit-image 文档中定义的非线性变换。如下代码展示了如何使用 swirl()函数来实现变换，其中，strength 是函数的旋流量参数，radius 以像素表示旋流程度，rotation 用来添加旋转角度。

```
from matplotlib.pyplot import imread
from skimage.transform import swirl
from matplotlib import pyplot as plt
#读取图像
im = imread("panda.jpg")
#旋流变换
swirled = swirl(im, rotation=0, strength=15, radius=200)
plt.imshow(swirled)
plt.axis('off')
plt.show()
```

程序运行结果如图 1-44 所示。

图 1-44 图像的旋流变换

1.6.2 图像的添噪

使用 random_noise()函数向图像添加不同类型的噪声。如下代码展示了如何将具有不同方差的高斯噪声添加到图像中。

```
from skimage.util import random_noise
from matplotlib.pyplot import imread
from matplotlib import pyplot as plt
from skimage import img_as_float
```

```
#读取图像并转换为浮点格式
im = img_as_float(imread("panda.jpg"))
plt.figure(figsize=(15,12))
#标准差
sigmas = [0.1, 0.25, 0.5, 1]
#向图像中添加不同的标准差
for i in range(4):
  noisy = random_noise(im, var=sigmas[i]**2)
  plt.subplot(2,2,i+1)
  plt.imshow(noisy)
  plt.axis('off')
  plt.title('Gaussian noise with sigma=' + str(sigmas[i]), size=20)
plt.tight_layout()
plt.show()
```

运行上述代码，添加不同标准差的高斯噪声生成的输出图像如图 1-45(a)～图 1-45(d)所示，从图中可以看到，高斯噪声的标准差越大，输出图像的噪声就越大。

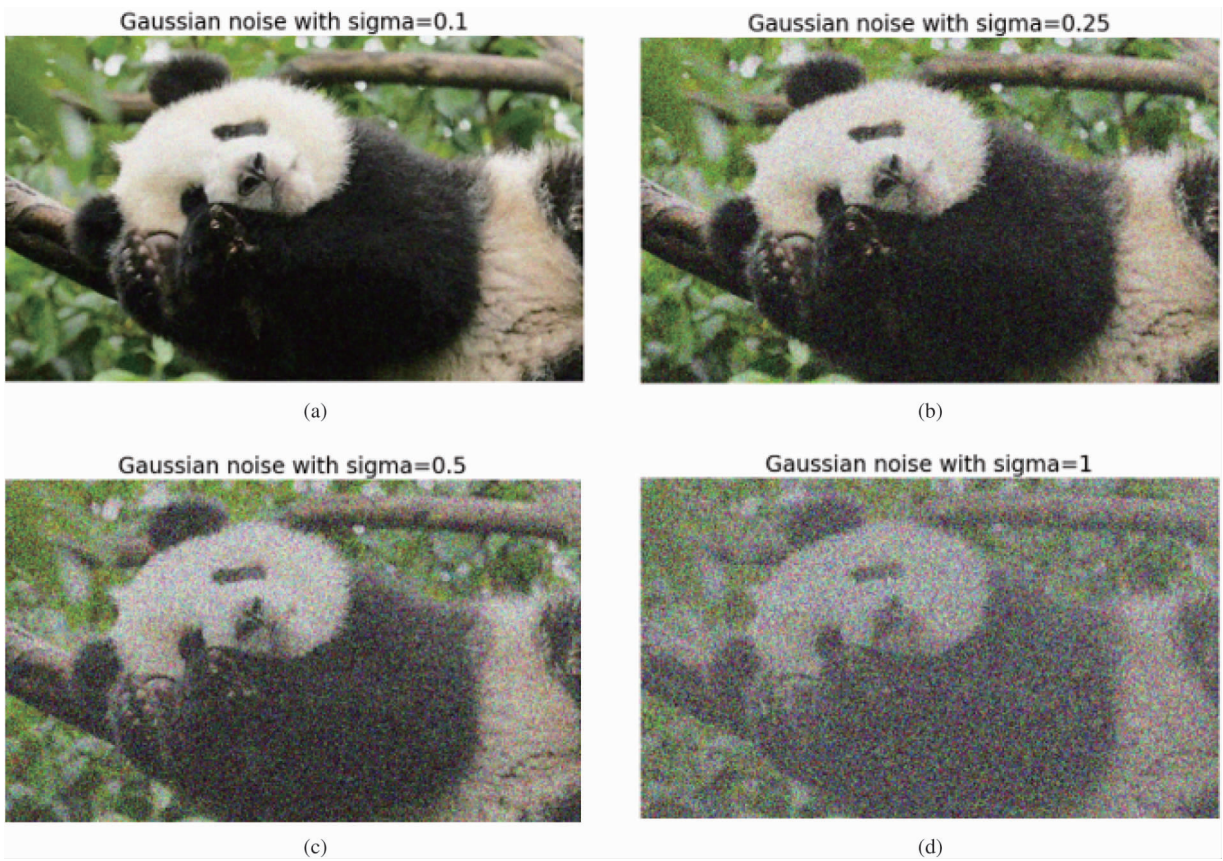


图 1-45　添加不同高斯噪声生成的输出图像

(a)σ=0.1;(b)σ=0.25;(c)σ=0.5;(d)σ=1

本章小结

本章主要介绍了以下内容：Anaconda、PyCharm 及图像处理库的安装；利用 PIL 进行图像的读取、保存、区域的复制和粘贴，图像尺寸的调整，图像旋转以及其他常用的图像处理操作；利用 Matplotlib 在图像中绘制点线、提取图像的轮廓和直方图；利用 NumPy 实现图像的数组化和灰度变换；利用 SciPy 进行图像模糊和利用导数处理图像；利用 scikit-image 实现图像的旋流变换和添噪。

习题

1-1　使用 Python 读取一张图像并裁剪感兴趣的区域，然后将原图进行镜像处理，旋转之前获取的区域，然后使用 paste()函数将该区域放到镜像图片上。

1-2　读取一张图像，用 split()函数分离多通道图像的通道，再使用 merge()函数合并多通道图像的通道，并交换蓝绿通道。

1-3　读取一张图像，然后对其做以下处理：对灰度图像进行反相处理，将图像的像素值变换到 100—200 区间，对图像的像素值求平方。

1-4　读取一张图像，使用 gaussian_filter 模块对其进行 σ 分别为 2、5、10 的处理。

1-5　读取一张图像，使用 swirl()函数进行旋转角度为 90 度、旋流量参数为 15、旋流程度为 500 的旋流变换。

第 2 章　传统图像处理方法

图像处理是计算机视觉领域的关键技术之一，它涵盖了一系列操作，旨在改善图像的质量、提取有用信息、识别对象及理解图像内容。本章将继续深入探讨传统图像处理方法，这些方法是计算机视觉的基础，为许多高级应用提供了重要的理论支撑。

2.1 图 像 增 强

图像增强是图像处理的重要领域，旨在通过算法改善图像质量和信息。这些算法分为基于空间域的方法和基于频率域的方法。基于空间域的方法直接处理像素，包括点运算和邻域处理。点运算是对像素进行灰度值映射，而邻域处理考虑周围像素灰度值，可以是线性或非线性、低通或高通滤波。这些技术有助于增强图像清晰度和突出关键特征，可广泛应用于各种不同领域。本节将介绍一些常见的图像增强方法，包括直方图均衡化、图像的平滑和图像的锐化等。

2.1.1　直方图均衡化

直方图均衡化采用单调的非线性映射，该映射通过重新分配输入图像的像素强度值，使输出图像的强度分布均匀（直方图平坦），从而增强图像的对比度。如下代码演示了如何使用 scikit-image 的曝光模块进行直方图均衡化。直方图均衡化的实现有两种不同的风格：第一种是对整个图像的全局操作；第二种是局部的（自适应的）操作，将图像分割成块，并在每个块上运行直方图均衡化。

```
    import pylab
    from skimage.color import rgb2gray
    from skimage import img_as_ubyte, exposure
    from matplotlib.pyplot import imread
    img = rgb2gray(imread('panda.jpg'))
    #直方图均衡化
    img_eq = exposure.equalize_hist(img)
    #自适应直方图均衡化
    img_adapteq = exposure.equalize_adapthist(img, clip_limit = 0.03)
    pylab.gray()
    images = [img, img_eq, img_adapteq]
    titles = ['original input ', 'after histogram equalization', 'after adaptive histogram
equalization']
    for i in range(3):
        pylab.figure(figsize = (20,10)), plot_image(images[i], titles[i])
    pylab.figure(figsize = (15,5))
```