
HUAWEI
从零到一
Python图像处理
云享.书库 | 第三期
Eastmount
华为云社区云享专家
武汉大学在读博士
贵州财经学院教师
Python图像处理技术全面总结
上百个丰富的编程实战案例
初学者能从零到壹快速上手
华为云开发者社区

华为云开发者社区

《从零到一 · Python 图像处理及识别》

作者 杨秀璋（Eastmount）

作者简介

杨秀璋（Eastmount），华为云享专家，华为云开发者社区年度十佳博主，贵州财经大学信息学院教师，武汉大学国家网络安全学院在读博士。本科及硕士毕业于北京理工大学，长期从事 Web 数据挖掘、知识图谱、人工智能、网络空间安全等方向研究，并致力于将大数据和人工智能技术运用于少数民族文字、语音及古籍抢救。现已出版专著 6 部，发表论文 30 余篇，主持课题 6 项，含 1 项贵州省优秀自然科学基金（贵州舆情知识图谱研究），并参与国家级课题多项。推崇开源精神，近十年在华为云、Github、公众号（娜璋 AI 安全之家）等分享原创博客 600 余篇，开源项目 100 余个，全网累计阅读量超过 1000 万人次，

粉丝 20 余万。此外，与国内外顶尖互联网公司的技术工程师们具有长期的合作关系，多次受邀在省直机关单位和国内外会议上做技术报告分享，长期参加 CTF、大数据安全等比赛，多次带领学生参加“互联网+”“创新创业”“图书情报”比赛并获奖。希望大家都能在华为云分享博客，有所收获。感恩遇见，不负青春！

华为云开发者社区

华为开发者社区公众号

本书作者公众号

目录

第一部分 图像处理基础知识

第 01 篇 图像处理基础知识和 OpenCV 配置

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

第一篇文章作为整个系列的引子，将介绍什么是图像处理，图像处理的基础知识，Python 语言的优势和 OpenCV 基础概述。接下来，让我们开启整个系列的学习吧！

1.什么是图像处理

数字图像处理（Digital Image Processing）又称为计算机图像处理（Computer Image Processing），旨在将图像信号转换成数字信号并利用计算机对其进行处理的过程。常见的图像处理方法如图 1-1 所示。

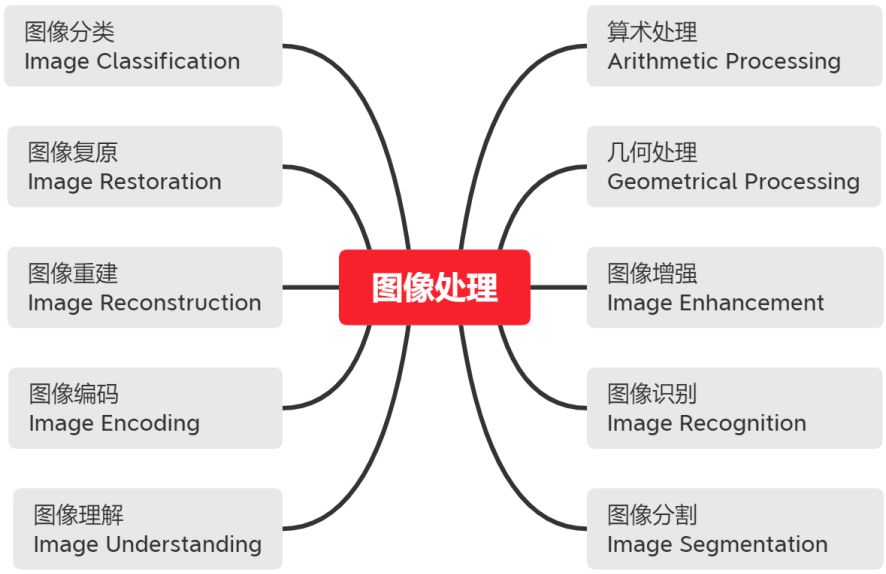


图 1-1 图像处理相关的应用方法

数字图像处理作为一门学科大约形成于 20 世纪 60 年代初期。早期的图像处理的目的是改善图像的质量，常用的处理方法包括图像增强、复原、编码、压缩等。随着图像处理技术的深入发展，从 70 年代中期开始，计算机技术和人工智能、思维科学研究迅速发展，数字图像处理向更高、更深层次发展。人们已开始研究如何用计算机系统解释图像，实现类似人类视觉系统理解外部世界，这被称为图像理解或计算机视觉。现如今，图像处理取得了不少重要的研究成果，其在许多领域（如通信、气象、生物、医学、物理、经济、文化等）已经得到广泛的应用[1-3]。

2.图像处理基础

图像都是由像素（pixel）构成的，像素表示为图像中的小方格，这些小方格都有一个明确的位置和被分配的色彩数值，而这些小方格的颜色和位置就决定该

图像所呈现出来的样子。像素是图像中的最小单位，每一个点阵图像包含了一定量的像素，这些像素决定图像在屏幕上所呈现的大小。图 1-2 表示一张由像素组成的叮当猫。

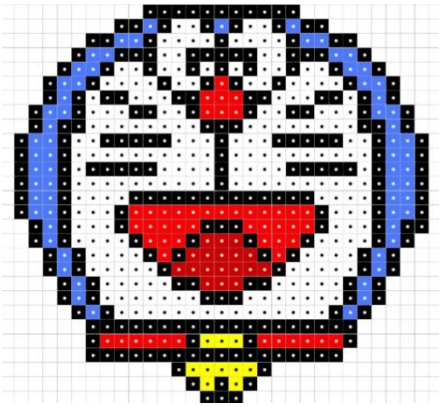


图 1-2 像素图像

图像通常分为二值图像、灰度图像和彩色图像，图 3-2 展示了图像处理经典“Lena”图的各种图像。

图 1-3 二值图像、灰度图像和彩色图像

（1）二值图像

二值图像又称为黑白图像，图像中任何一个点非黑即白，要么为白色（像素为 255），要么为黑色（像素为 0）。将灰度图像转换为二值图像的过程，常通过依次遍历判断实现，如果像素大于 127 则设置为 255，否则设置为 0。如图

1-4 所示，一幅二值图像对应的矩阵。

图 1-4 二值图像对应矩阵

（2）灰度图像

灰度图像是指每个像素的信息由一个量化的灰度级来描述的图像，没有彩色信息，如图 1-5 所示。改变像素矩阵的 RGB 值可以实现将彩色图转变为灰度图。常见的方法是将灰度划分为 256 种不同的颜色，将原来的 RGB(R,G,B)中的 R、G、B 统一替换为 Gray，形成新的颜色 RGB(Gray,Gray,Gray)，即灰度图。将彩色图像转换为灰度图是图像处理的最基本预处理操作。后面的文章将详细介绍不同灰度转换方法的实现过程。

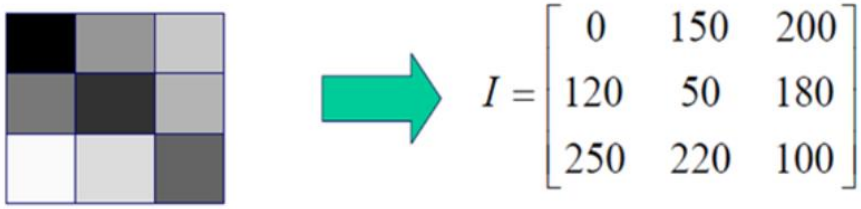


图 1-5 灰度图像对应矩阵

（3）彩色图像

彩色图像是 RGB 图像，RGB 表示红、绿、蓝三原色，计算机里所有颜色都是三原色不同比例组成的，即三色通道。RGB (Red 红色，Green 绿色，Blue 蓝色)，是根据人眼识别的颜色而定义的空间，可用于表示大部分颜色，也

是图像处理中最基本、最常用、面向硬件的颜色空间，是一种光混合的体系。图 1-6 展示了图像中某一点像素（205,89,68）所对应三原色像的素值，其中 R 表示红色分量、G 表示绿色分量、B 表示蓝色分量[4]。

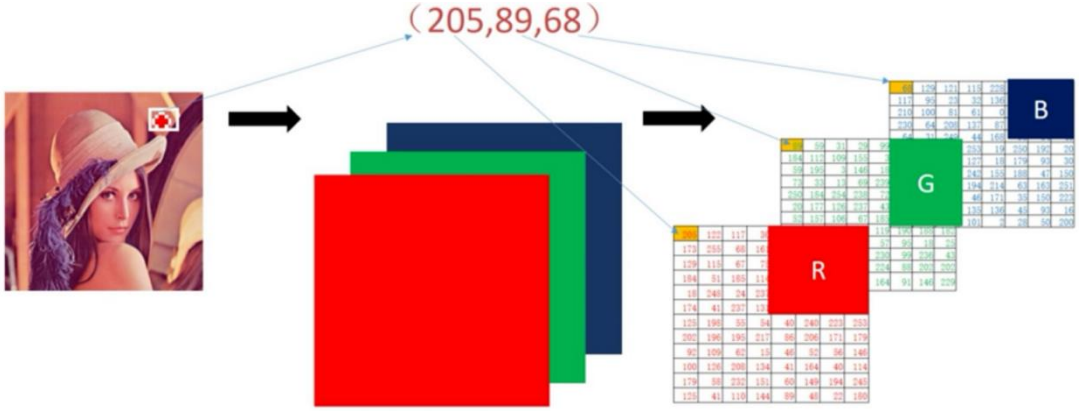


图 1-6 彩色图像组成原理

3.Python 语言

Python 是 Guido van Rossum 在 1989 年开发的一门语言，它既是解释性编程语言，又是面向对象的语言，其操作性和可移植性高，被广泛应用于数据挖掘、图像处理、人工智能领域。Python 具有语言清晰、容易学习、高效率的数据结构、丰富且功能强大的第三方包等优势。同时，Python 语言含有高效率的数据结构，它和其他的面向对象编程语言一样，具有参数、列表表达式、函数、流程控制（循环与分支）、类、对象等功能。Python 优雅的语法以及解释性的本质，使其成为一种能在多种功能、多种平台上撰写脚本及快速开发的理想语言[5]。Python 的具体优势如下：

- 语法清晰，代码友好，易读性好
- 应用广泛，具有大量的第三方库支持，包括机器学习、人工智能等

- Python 可移植性强，易于操作各种存储数据的文本文件和数据库
- Python 是一门面向对象语言，支持开源思想

本系列主要通过 Python 调用 OpenCV、Matplotlib、Numpy、Sklearn 等第三方包实现图像处理，其优雅清晰的语法结构减少了读者的负担，从而大大增强程序的质量。该系列文章采用 Python3.7 版本实现，并贯穿整个系列的所有代码。同时结合官方的 Python 解释器进行详细介绍，作者认为官方的解释器能让读者更好地学习基础语法知识，更好地掌握图像处理算法的精髓，从而为后续自己的需求应用提供最大的帮助。图 1-7 是官方下载地址[6]。

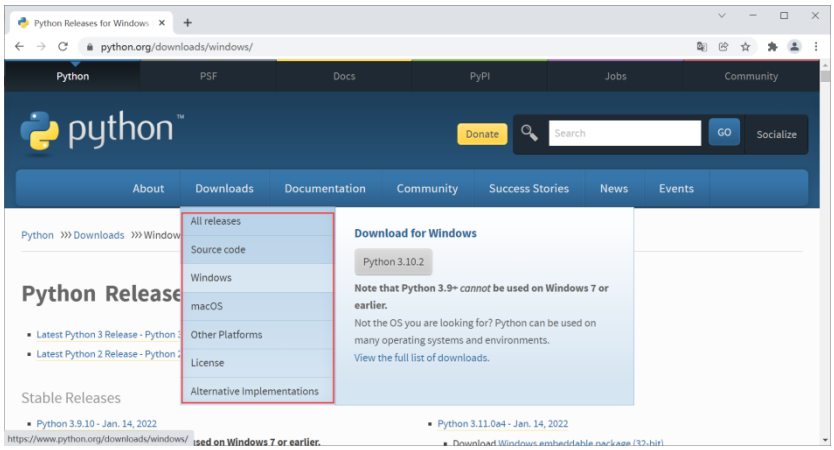


图 1-7 Python 官方下载地址

读者可以结合本机的操作系统选择适合的版本，比如作者选择了 Windows 操作系统下的“Python 3.7.4”版本。

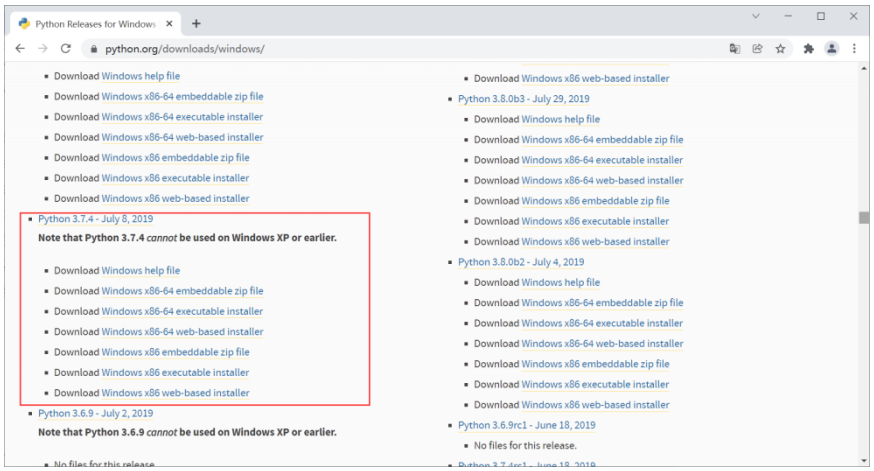


图 1-8 下载指定版本

接着双击“python-3.7.4-amd64.exe”软件进行安装，安装过程选择指定路径，按照 Python 安装向导，点击“Next”按钮即可。

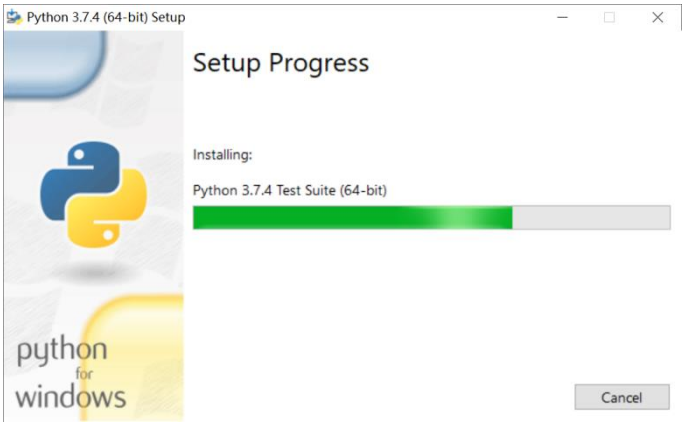


图 1-9 安装过程

Python 提供了集成开发环境（Python Integrated Development Environment，IDLE）供读者编写脚本文件，图 1-10 展示了 IDLE 显示 print("Hello World")代码的效果。

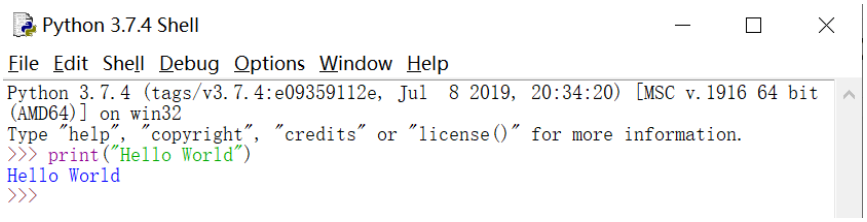


图 1-10 Python 集成开发环境

在 IDLE 界面中点击“File”中“New File”新建文件，并另存为 py 文件，如“1-1-test.py”，然后编写相关代码并点击运行“Run Module F5”按钮，即可运行 Python 脚本文件。

```
# -*- coding:utf-8 -*-

print(2)

print("Hello World")


x = 2

y = 4

z = x + y

print(x)

print(y)

print(z)

print(x, y, z)
```

输出结果如图 1-11 所示。

```
# -*- coding:utf-8 -*-
print(2)
print("Hello World")

x = 2
y = 4
z = x + y
print(x)
print(y)
print(z)
print(x, y, z)
```

```
2
Hello World
2
4
6
2 4 6
>>>
```

图 1-11 输出结果

这里作者仅简单介绍了 Python 的优势和安装过程，如果您是一名初学者，建议好好学习 Python 的基础知识，包括基础语法、数据类型、条件语句、循环语句、函数定义、类定义、面向对象等知识。后续图像处理文章遇到相关知识点，作者也会简要介绍。

4.OpenCV 安装和基础

OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库，可以运行在 Linux、Windows、Android 和 Mac 操作系统上。它是一个由 C/C++语言编写而成的轻量级并且高效的库，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法[7]。其官方地址为：https://opencv.org/。

该系列文章主要使用 Python 调用 OpenCV2 库函数进行图像处理操作，首先告知读者如何在 Python 编程环境下安装 OpenCV 库。OpenCV 安装主要通过 pip 指令进行。如图 1-12 所示，在命令提示符 CMD 环境下，通过 cd 命令进入 Python 安装目录的 Scripts 文件夹下，再调用下列命令安装。

- pip install opencv-python

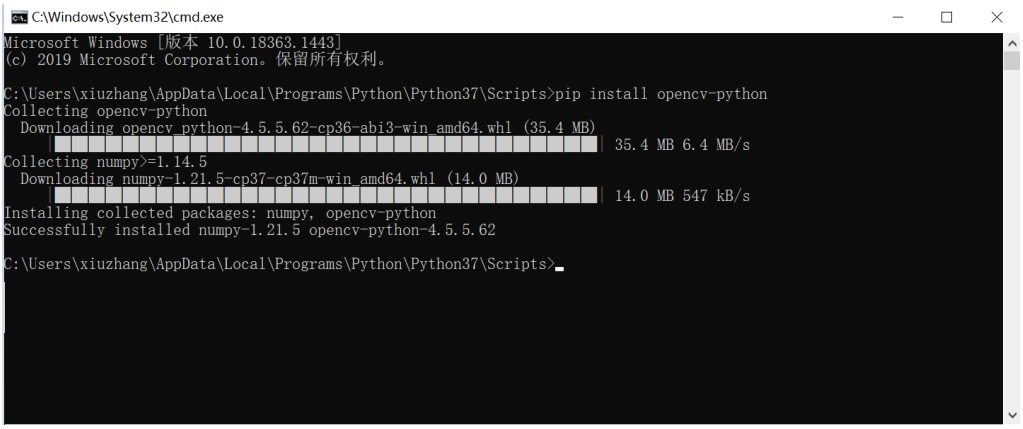


图 1-12 安装 OpenCV 扩展包

当 OpenCV 扩展包安装成功后，在 Python 中输入“import cv2”语句导入该扩展包，测试安装是否成功，如果没有异常报错即安装成功，如图 1-13 所示。

```
>>> import cv2
>>>
```

图 1-13 导入 OpenCV 扩展包

最后，补充一个简答的图像显示案例，算是这篇文章的结尾。接下来，我们将正式进入“Python 图像处理及识别”系列，将从各个方面详细介绍图像处理的相关知识。

```
# -*- coding:utf-8 -*-

import cv2



#读取图片

img = cv2.imread("huawei.png")
```

```
#显示图像
cv2.imshow("Demo", img)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

显示效果如图 1-14 所示。

图 1-14 OpenCV 显示图像

5.总结

写到这里，这篇文章就介绍结束。本文主要分享了什么是图像处理、图像处理基础知识、Python 语言和 OpenCV 基础知识。通过这篇文章，初学者可以学会如何安装相关的程序，开启整个系列的学习。同时，大家在学习该系列文章时，一定要自己动手实现所有代码和案例，这才能提升您的编程能力。

参考文献：

[1] 冈萨雷斯. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社, 2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 百度百科. 图像处理[EB/OL].(2021-12-15). https://baike.baidu.com/item/图像处理/294902.

[4] Eastmount. [Python 图像处理] 一.图像处理基础知识及 OpenCV 入门函数[EB/OL]. (2018-08-16). https://blog.csdn.net/Eastmount/article/details/81748802.

[5] 杨秀璋, 颜娜. Python 网络数据爬取及分析从入门到精通（分析篇）[M]. 北京：北京航天航空大学出版社, 2018.

[6] Python 官网. Welcome to Python.org[EB/OL]. (2022-11-10). https://www.python.org.

[7] 毛星云, 冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社, 2015.

第 02 篇 OpenCV 入门详解——显示读取修改及保存图像

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍了图像处理基础知识，这篇文章将详细讲解 OpenCV 入门知识，包括 OpenCV 常见数据类型、显示图像、读取像素、修改像素、创建图像、复制图像、保存图像等内容。

1.OpenCV 常见数据类型

OpenCV 是一个轻量级高效的跨平台计算机视觉库，实现了图像处理和计算机视觉方面的多种通用算法。所谓的图像可以理解为一个数组，图像处理就是对数组的处理。首先，本文将介绍 OpenCV 中常见的数据类型，包括点 Point 类、颜色 Scalar 类、尺寸 Size 类、矩形 Rect 类、矩阵 Mat 类[1-2]。

（1）点 Point

表示二维坐标系中的点，含 x 和 y。其示例如下：

```
#OpenCV 示例
Point p; p.x=1, p.y=2;
```

```
Point p=Point(1, 2);


#Python 示例
points_list = [(160, 160), (136, 160)]
```

（2）颜色 Scalar

包含四个元素的数组，设置像素值 RGB 三通道，第四个参数可忽略。其示例如下：

```
#OpenCV 示例  BGR 三分量
Scalar(b, g, r);


#Python 示例
(0, 0, 255)
```

（3）尺寸 Size

它和 Point 相似，主要成员包括 height 和 width。其示例如下：

```
#OpenCV 示例
Size(5, 5);
Size_(_Tp _width, _Tp _height);


#Python 示例
```

```
width, height = img.shape
```

（4）矩形 Rect

Rect 类称为矩形类，包含 Point 类的成员 x 和 y（代表矩形左上角的坐标）和 Size 类的成员 width 和 height（代表矩形的大小）。其示例如下：

```
#OpenCV 示例
Rect rect = rect1 & rect2;      #求两矩形交集
Rect rect = rect1 | rect2;      #求两矩形并集
Rect rectShift = rect + point;  #矩形平移
Rect rect = rect1 + size;       #矩形缩放


#Python 示例
cv2.rectangle(img, (20,20), (150,250), (255,0,0), 2)
```

（5）矩阵 Mat

通用的矩阵类，用来创建和操作多维矩阵。其示例如下：

```
#OpenCV 示例
Mat M(3,2, CV_8UC3, Scalar(0,0,255));


#Python 示例
np.zeros((256,256,3), np.uint8)
```

2.OpenCV 读取与显示图像

在 OpenCV2 中，图像的读取和显示是最简单的两句代码，它们通过 imread()和 imshow()函数实现[3]。OpenCV 读取图像的 imread()函数原型如下，它将从指定的文件加载图像并返回矩阵，如果无法读取图像（因为缺少文件、权限不正确、格式不支持或图像无效等），则返回空矩阵（Mat::data==NULL）。

retval = imread(filename[, flags])

- filename 表示需要载入的图片路径名，其支持 Windows 位图、JPEG 文件、PNG 图片、便携文件格式、Sun rasters 光栅文件、TIFF 文件、HDR 文件等。
- flags 为 int 类型，表示载入标识，它指定一个加载图像的颜色类型，默认值为 1。其中 cv2.IMREAD_UNCHANGED 表示读入完整图像或图像不可变，包括 alpha 通道；cv2.IMREAD_GRAYSCALE 表示读入灰度图像；cv2.IMREAD_COLOR 表示读入彩色图像，默认参数，忽略 alpha 通道。

OpenCV 中显示图像调用 imshow()函数，它将在指定窗口中显示一幅图像，窗口会自动调整为图像大小，其原型如下所示：

imshow(winname, mat)

- winname 表示窗口的名称
- mat 表示要显示的图像

下面是第一个示例程序，主要用于读取和加载经典的“Lena”图像。

```
# -*- coding:utf-8 -*-

# By：Eastmount

import cv2


#读取图片

img = cv2.imread("Lena.png")


#显示图像

cv2.imshow("Demo", img)


#等待显示

cv2.waitKey(0)

cv2.destroyAllWindows()
```

输出结果如图 2-1 所示：

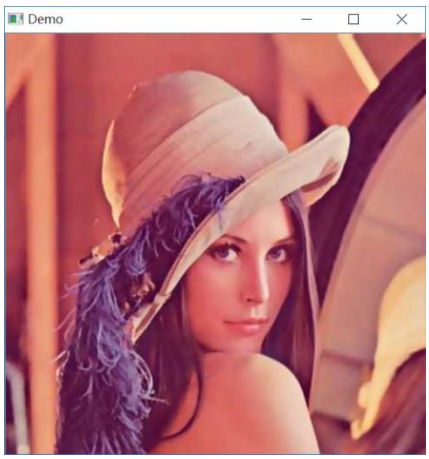


图 2-1 图像显示

需要注意，在图像显示过程中，如果代码中没有 waitKey(0)函数，其运行结果可能会出现错误，加载一幅灰色的图像，如图 2-2 所示。

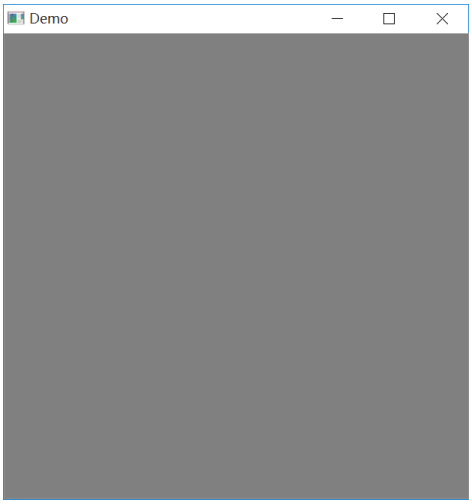


图 2-2 图像显示错误

因此，在显示图像过程中，通常还会调用两个操作窗口的函数，它们分别是 waitKey()和 destroyAllWindows()。

retval = waitKey([, delay])

- 键盘绑定函数，共一个参数 delay，表示等待的毫秒数，看键盘是否有输入，返回值为 ASCII 值。如果其参数为 0，则表示无限期的等待键盘输入；参数大于 0 表示等待 delay 毫秒；参数小于 0 表示等待键盘单击。

destroyAllWindows()

- 该函数可以轻易删除所有建立的窗口。如果你想删除特定的窗口可以使用 cv2.destroyWindow()，并在括号内输入要删除的窗口名。

同时，可以设置加载图像后无限期等待，直到输入指定的按键才退出窗口，如下面的代码需要输入 ESC 才退出。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2


#读取图片
img = cv2.imread("Lena.png")


#显示图像
cv2.imshow("Demo", img)


#无限期等待输入
k=cv2.waitKey(0)


#如果输入 ESC 按键退出
if k==27:
    cv2.destroyAllWindows()
```

此外，在对比实验中，我们通常需要显示多张图片，此时可以调用 NumPy 和 Matplotlib 库辅助完成[4]，具体实现过程如下所示。

- NumPy（Numeric Python）是 Python 提供的数值计算扩展包，拥有高效的处理函数和数值编程工具，主要用于科学计算，如矩阵数据类型、线性代数、矢量处理等。

- Matplotlib 是 Python 强大的数据可视化工具和 2D 绘图库，常用于创建海量类型的 2D 图表和一些基本的 3D 图表，类似于 MATLAB 和 R 语言。Matplotlib 提供了一整套和 Matlab 相似的命令 API，十分适合交互式地进行制图，而且也可以方便地将它作为绘图控件，嵌入 GUI 应用程序中。Matplotlib 是一名神经生物学家 John D. Hunter 博士于 2007 年创建，函数设计上参考了 Matlab，现在在 Python 的各个科学计算领域都得到了广泛应用。

该程序是调用 cv2.imread()函数分别读取四张图片，并转换为 RGB 颜色空间，接着通过 for 循环分别设置各子图对应的图像、标题及坐标轴名称，其中 plt.subplot(2,2)表示生成 2×2 张子图。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图像
img1 = cv2.imread('lena.png')
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)


img2 = cv2.imread('xluo.png')
```

```
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)


img3 = cv2.imread('flower.png')
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)


img4 = cv2.imread('huawei.png')
img4 = cv2.cvtColor(img4, cv2.COLOR_BGR2RGB)


#显示四张图像
titles = ['lena', 'people', 'flower', 'huawei']
images = [img1, img2, img3, img4]
for i in range(4):
    plt.subplot(2, 2, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出如图 2-3 所示，它显示了四幅图像。在图像处理对比中，同时对比多种算法的处理效果是非常重要的手段之一。

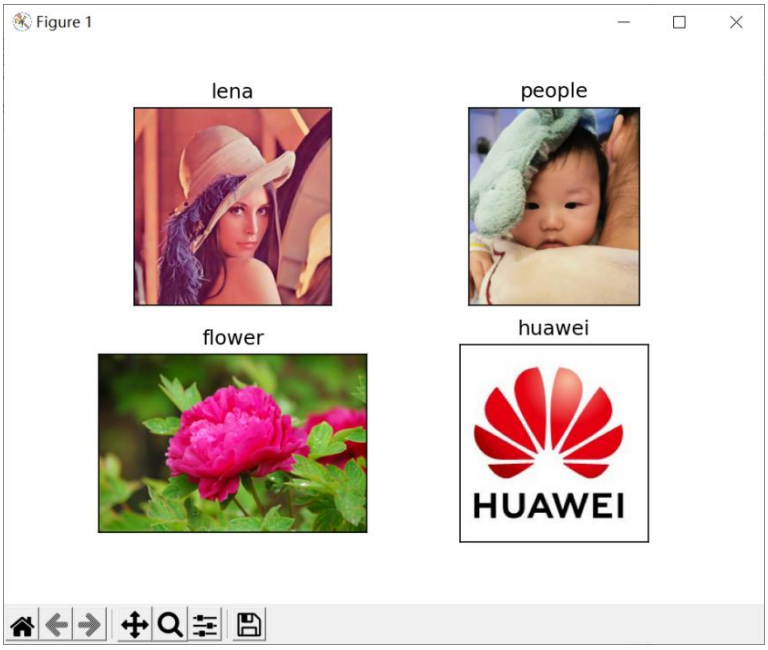


图 2-3 Matplotlib 显示四张子图

3.OpenCV 像素处理

OpenCV 中读取图像的像素值可以直接通过遍历图像的位置实现，如果是灰度图像则返回其灰度值，如果是彩色图像则返回蓝色（B）、绿色（G）、红色（G）三个分量值。其示例如下：

- **灰度图像：返回值 = 图像[位置参数]**

示例：test=img[88,42]

- **彩色图像：返回值 = 图像[位置元素, 0 | 1 | 2]获取 BGR 三个通道像素**

示例：blue=img[88,142,0] green=img[88,142,1] red=img[88,142,2]

当需要修改图像中的像素时，则定位指定像素并直接赋新像素值即可，彩色

图像需要依次给三个分量赋值。如下列代码所示。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2


#读取图片
img = cv2.imread("Lena.png")


#读取像素
test = img[88,142]
print("读取的像素值:", test)


#修改像素
img[88,142] = [255, 255, 255]
print("修改后的像素值:", test)


#分别获取 BGR 通道像素
blue = img[88,142,0]
print("蓝色分量", blue)
green = img[88,142,1]
print("绿色分量", green)
```

```
red = img[88,142,2]
print("红色分量", red)


#显示图像
cv2.imshow("Demo", img)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

读取的像素值及修改后的像素值结果如图 2-4 所示。

```
>>>
读取的像素值: [108 103 195]
修改后的像素值: [255 255 255]
蓝色分量 255
绿色分量 255
红色分量 255
>>>
```

图 2-4 读取与修改像素值的结果

下面代码是将 100 到 200 行、150 到 250 列的像素区域设置为白色的效果。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
```

```
#读取图片
img = cv2.imread("Lena.png")

#该区域设置为白色
img[100:200, 150:250] = [255,255,255]

#显示图像
cv2.imshow("Demo", img)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图 2-5 是最终显示的效果图，它将 img[100:200, 150:250] 区域显示为白色。

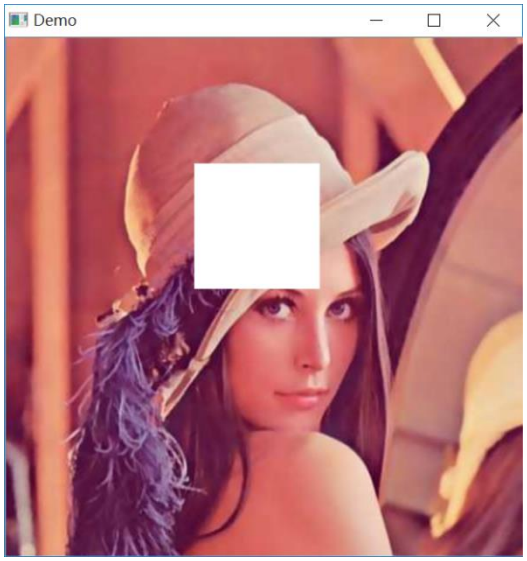


图 2-5 读取与修改像素值的结果

4.NumPy 像素处理

前面是直接读取和修改图像像素的方法，下面讲解通过 NumPy 库读取像素和修改像素的方法。NumPy 是 Python 提供的数值计算扩展包，拥有高效的处理函数和数值编程工具，Array 是 NumPy 库中最基础的数据结构，表示数组。NumPy 可以很方便地创建各种不同类型的多维数组，并且执行一些基础操作。

在图像处理中，NumPy 读取像素调用 item()函数实现，修改像素调用 itemset()实现，其原型如下所示[5]。使用 Numpy 进行像素读取，调用方式如下：

- **返回值 = 图像.item(位置参数)**

例如：blue = img.item(78, 100, 0)

使用 Numpy 的 itemset 函数修改像素，调用方式如下：

❖ **图像.itemset(位置，新值)**

例如：img.itemset((88,99), 255)

最终实现代码如下所示。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy

#读取图片
img = cv2.imread("Lena.png")
print(type(img))

#Numpy 读取像素
print(img.item(78, 100, 0))
print(img.item(78, 100, 1))
print(img.item(78, 100, 2))

#Numpy 修改像素
img.itemset((78, 100, 0), 100)
img.itemset((78, 100, 1), 100)
img.itemset((78, 100, 2), 100)
```

```
print(img.item(78, 100, 0))
print(img.item(78, 100, 1))
print(img.item(78, 100, 2))
```

输出结果如下所示，原始图像 BGR 像素值为 88、84、196，修改后的像素值为 100、100、100。

```
<class 'numpy.ndarray'>
88
84
196
100
100
100
```

5.OpenCV 创建图像

由于在 OpenCV2 中没有 CreateImage 函数，如果需要创建图像，则需要使用 Numpy 库函数实现。如下述代码，调用 np.zeros()函数创建空图像，创建的新图像使用 Numpy 数组的属性来表示图像的尺寸和通道信息，其中参数 img.shape 表示原始图像的形状，np.uint8 表示类型。

```
emptyImage = np.zeros(img.shape, np.uint8)
```

例如 img.shape 为(500, 300, 3)，它表示 500×300 像素的图像，3 表

示这是一个 RGB 图像。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取图片
img = cv2.imread("Lena.png")

#创建空图像
emptyImage = np.zeros(img.shape, np.uint8)

#显示图像
cv2.imshow("Demo", emptyImage)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

显示结果如图 2-6 所示，这是一幅新创建的“空白”图像。

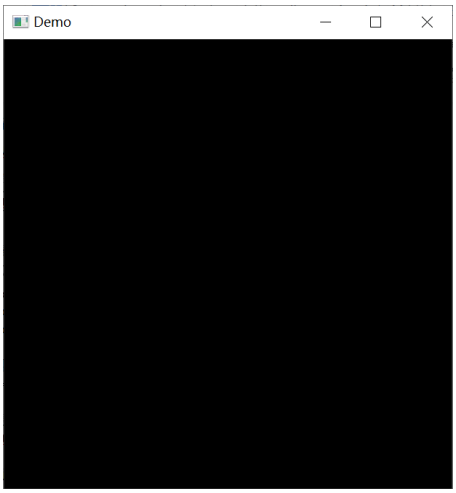


图 2-6 创建图像

6.OpenCV 复制图像

复制原有图像来获取一幅新图像，可以调用 copy()函数实现。

```
emptyImage2 = img.copy()
```

下述代码实现了图像的创建和复制功能。

```
# -*- coding:utf-8 -*-

# By：Eastmount

import cv2

import numpy as np



#读取图片

img = cv2.imread("Lena.png")



#创建空图像
```

```
emptyImage = np.zeros(img.shape, np.uint8)


#复制图像
emptyImage2 = img.copy()


#显示图像
cv2.imshow("Demo1", img)
cv2.imshow("Demo2", emptyImage)
cv2.imshow("Demo3", emptyImage2)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

最终输出结果如图 2-7 所示，Demo1 表示原始图像，Demo2 表示创建的空白图像，Demo3 表示复制的图像。

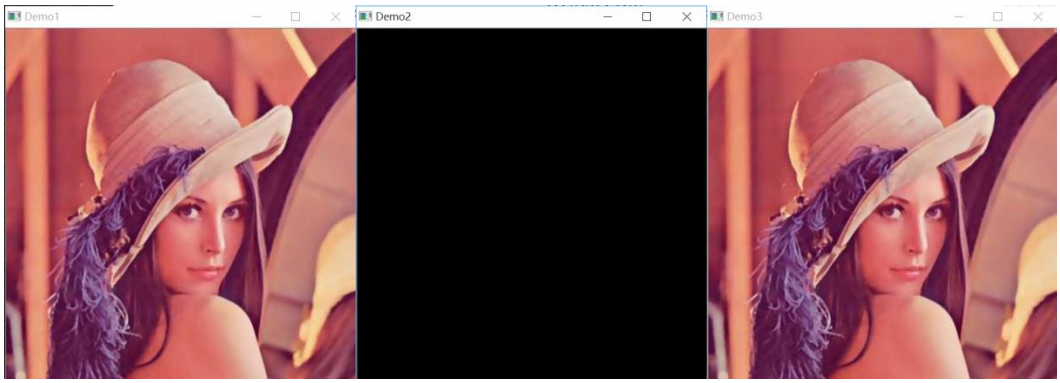


图 2-7 复制图像

7.OpenCV 保存图像

在 OpenCV 中，输出图像到文件使用的函数为 imwrite()，其函数原型如下：

retval = imwrite(filename, img[, params])

- filename 表示要保存的路径及文件名
- img 表示图像矩阵
- params 表示特定格式保存的参数编码，默认值为空。对于 JPEG 图片，该参数（cv2.IMWRITE_JPEG_QUALITY）表示图像的质量，用 0-100 的整数表示，默认值为 95。对于 PNG 图片，该参数（cv2.IMWRITE_PNG_COMPRESSION）表示的是压缩级别，从 0 到 9，压缩级别越高，图像尺寸越小，默认级别为 3。对于 PPM、PGM、PBM 图片，该参数表示一个二进制格式的标志（cv2.IMWRITE_PXM_BINARY）[2]。注意，该类型为 Long，必须转换成 int。

下面是一个调用 imwrite()函数输出图像到指定的文件的代码。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
#读取图像
img = cv2.imread("Lena.png")

#显示图像
cv2.imshow("Demo", img)

#保存图像
cv2.imwrite("dst1.jpg", img,
[int(cv2.IMWRITE_JPEG_QUALITY), 5])
cv2.imwrite("dst2.jpg", img,
[int(cv2.IMWRITE_JPEG_QUALITY), 100])
cv2.imwrite("dst3.png", img,
[int(cv2.IMWRITE_PNG_COMPRESSION), 0])
cv2.imwrite("dst4.png", img,
[int(cv2.IMWRITE_PNG_COMPRESSION), 9])

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

原始图像“Lena.jpg”为 222KB，调用 imwrite()函数输出保存的图像

共四张，如图 2-8 所示，其中“dst1.jpg”被压缩，大小为 4.90KB，“dst2.jpg”图像大小为 99.1KB，“dst3.png”大小为 499KB，“dst4.png”大小为 193KB。

图 2-8 保存图像

8.总结

写到这里，这篇文章就介绍结束。本文详细介绍了 OpenCV 图像处理的基础用法，包括图像读取、显示、像素处理、创建、复制、写入和保存。通过这篇文章，初学者将学会基本的图像处理操作，代码比较简单，但希望大家一定要自己动手实现所有代码和案例，只有不断实践才能提升。

参考文献：

[1] 冈萨雷斯. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社, 2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] Eastmount. [Python 图像处理] 一.图像处理基础知识及 OpenCV 入门函数

[EB/OL]. (2018-08-16). https://blog.csdn.net/Eastmount/article/details/81748802.

[4] 杨秀璋, 颜娜. Python 网络数据爬取及分析从入门到精通（分析篇）[M]. 北京：北京航天航空大学出版社, 2018.

[5] Eastmount. [Python 图像处理] 二.OpenCV+Numpy 库读取与修改像素 [EB/OL]. (2018-08-28). https://blog.csdn.net/eastmount/article/details/82120114.

第 03 篇 OpenCV 绘制各类几何图形

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍了 OpenCV 图像处理基础知识，这篇文章将介绍如何使用 OpenCV 绘制各类几何图形，包括 cv2.line()、v2.circle()、cv2.rectangle()、cv2.ellipse()、cv2.polylines()、cv2.putText()函数[1-3]。这将帮助我们了解基础图形绘制的基础用法，同时能在此基础上实现画图软件等应用拓展。

1.绘制直线

在 OpenCV 中，绘制直线需要获取直线的起点和终点坐标，调用 cv2.line()函数实现该功能。该函数原型如下所示：

- img = line(img, pt1, pt2, color[, thickness[, lineType[, shift]]])
  - img 表示需要绘制的那幅图像
  - pt1 表示线段第一个点的坐标
  - pt2 表示线段第二个点的坐标
  - color 表示线条颜色，需要传入一个 RGB 元组，如(255,0,0)代表蓝色

- thickness 表示线条粗细
- lineType 表示线条的类型
- shift 表示点坐标中的小数位数

下面的代码是绘制一条直线，通过 np.zeros()创建一幅黑色图像，接着调用 cv2.line()绘制直线，参数包括起始坐标和颜色、粗细。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#创建黑色图像
img = np.zeros((256,256,3), np.uint8)

#绘制直线
cv2.line(img, (0,0), (255,255), (55,255,155), 5)

#显示图像
cv2.imshow("line", img)

#等待显示
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

输出结果如图 3-1 所示，从坐标（0,0）到（255,255）绘制一条直线，其直线颜色为(55,255,155)，粗细为 5。

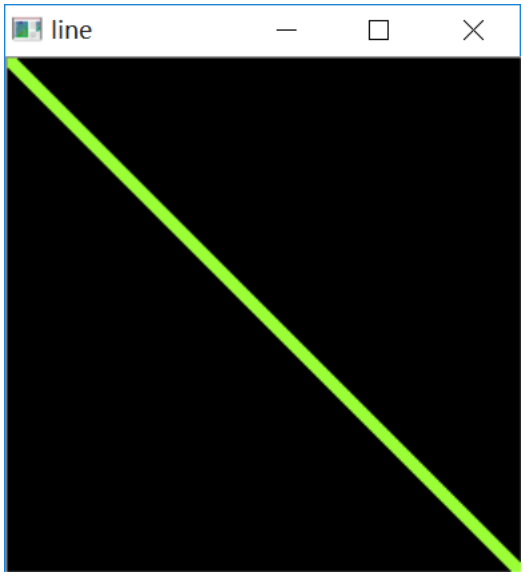


图 3-1 绘制直线

2.绘制矩形

在 OpenCV 中，绘制矩形通过 cv2.rectangle()函数实现，该函数原型如下所示：

- img = rectangle(img, pt1, pt2, color[, thickness[, lineType[, shift]]])
  - img 表示需要绘制的那幅图像
  - pt1 表示矩形的左上角位置坐标
  - pt2 表示矩形的右下角位置坐标
  - color 表示矩形的颜色

- thickness 表示边框的粗细
- lineType 表示线条的类型
- shift 表示点坐标中的小数位数

下面的代码是绘制一个矩形，通过 np.zeros()创建一幅黑色图像，接着调用 cv2.rectangle()绘制矩形。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#创建黑色图像
img = np.zeros((256,256,3), np.uint8)

#绘制矩形
cv2.rectangle(img, (20,20), (150,250), (255,0,0), 2)

#显示图像
cv2.imshow("rectangle", img)

#等待显示
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

输出结果如图 3-2 所示，从左上角坐标为（20,20），右下角坐标为（150,250），绘制的矩形颜色为蓝色（255,0,0），粗细为 2。

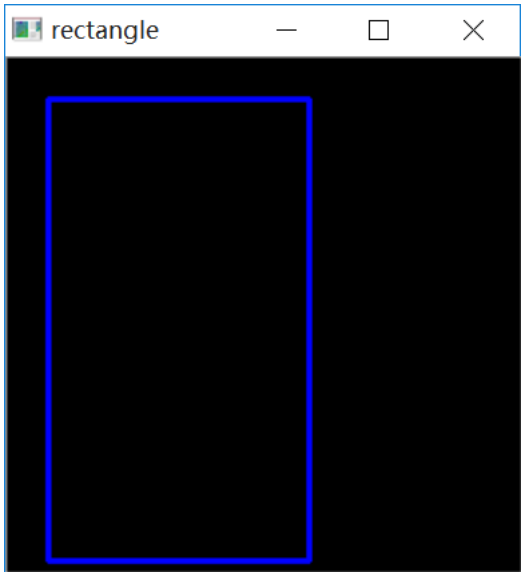


图 3-18 绘制矩形

3.绘制圆形

在 OpenCV 中，绘制矩形通过 cv2.rectangle()函数实现，该函数原型如下所示：

- img = circle(img, center, radius, color[, thickness[, lineType[, shift]]])
  - img 表示需要绘制圆的图像
  - center 表示圆心坐标
  - radius 表示圆的半径

- color 表示圆的颜色
- thickness 如果为正值，表示圆轮廓的厚度；负厚度表示要绘制一个填充圆
- lineType 表示圆的边界类型
- shift 表示中心坐标和半径值中的小数位数

下面代码是绘制一个圆形。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#创建黑色图像
img = np.zeros((256,256,3), np.uint8)

#绘制圆形
cv2.circle(img, (100,100), 50, (255,255,0), 4)

#显示图像
cv2.imshow("circle", img)

#等待显示
```

```
cv2.waitKey(0)

cv2.destroyAllWindows()
```

输出结果如图 3-3 所示，它在圆形为（100,100）的位置，绘制了一个半径为 50，颜色为（255,255,0）、粗细为 4 的圆。

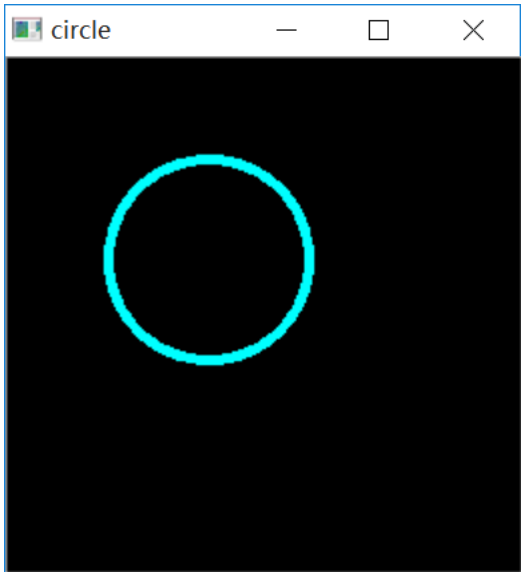


图 3-3 绘制圆形

注意，如果将粗细设置为“-1”，则绘制的圆为实心，如图 3-4 所示。

- cv2.circle(img, (100,100), 50, (255,255,0), -1)

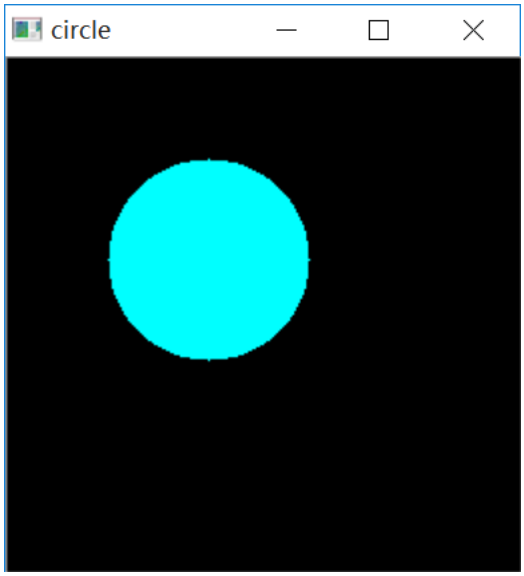


图 3-4 绘制实心圆形

4.绘制椭圆

在 OpenCV 中，绘制椭圆比较复杂，要多输入几个参数，如中心点的位置坐标，长轴和短轴的长度，椭圆沿逆时针方向旋转的角度等。cv2.ellipse()函数原型如下所示：

- img = ellipse(img, center, axes, angle, startAngle, endAngle, color[, thickness[, lineType[, shift]]])
  - img 表示需要绘制椭圆的图像
  - center 表示椭圆圆心坐标
  - axes 表示轴的长度（短半径和长半径）
  - angle 表示偏转的角度（逆时针旋转）
  - startAngle 表示圆弧起始角的角度（逆时针旋转）
  - endAngle 表示圆弧终结角的角度（逆时针旋转）
  - color 表示线条的颜色
  - thickness 如果为正值，表示椭圆轮廓的厚度；负值表示要绘制一个填充椭圆
  - lineType 表示圆的边界类型
  - shift 表示中心坐标和轴值中的小数位数

下面是绘制一个椭圆的代码。

```
# -*- coding:utf-8 -*-
```

```
# By：Eastmount
import cv2
import numpy as np

#创建黑色图像
img = np.zeros((256,256,3), np.uint8)

#绘制椭圆
#椭圆中心(120,100) 长轴和短轴为(100,50)
#偏转角度为 20
#圆弧起始角的角度 0 圆弧终结角的角度 360
#颜色(255,0,255) 线条粗细 2
cv2.ellipse(img, (120, 100), (100, 50), 20, 0, 360, (255, 0,
255), 2)

#显示图像
cv2.imshow("ellipse", img)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 3-5 所示，其椭圆中心为（120,100），长轴为 100，短轴为 50，偏转角度为 20，圆弧起始角的角度为 0，圆弧终结角的角度为 360，表示一个完整的椭圆。绘制的颜色为(255,0,255)，粗细为 2。

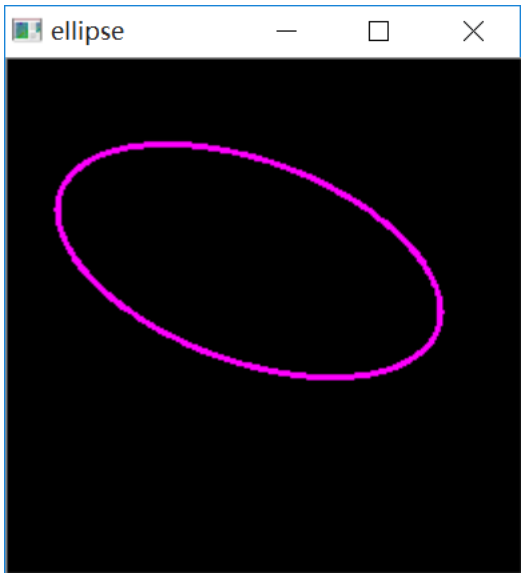


图 3-5 绘制椭圆

5.绘制多边形

在 OpenCV 中，调用 cv2.polylines()函数绘制多边形，它需要指定每个顶点的坐标，通过这些点构建多边形，其函数原型如下所示：

- img = polylines(img, pts, isClosed, color[, thickness[, lineType[, shift]]])
  - img 表示需要绘制的图像
  - center 表示多边形曲线阵列
  - isClosed 表示绘制的多边形是否闭合，False 表示不闭合
  - color 表示线条的颜色

- thickness 表示线条粗细
- lineType 表示边界类型
- shift 表示顶点坐标中的小数位数

下面是绘制一个多边形的代码。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#创建黑色图像
img = np.zeros((256,256,3), np.uint8)

#绘制多边形
pts = np.array([[10,80], [120,80], [120,200], [30,250]])
cv2.polylines(img, [pts], True, (255, 255, 255), 5)

#显示图像
cv2.imshow("ellipse", img)

#等待显示
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

输出结果如图 3-6 所示，绘制的多边形为白色的闭合图形。

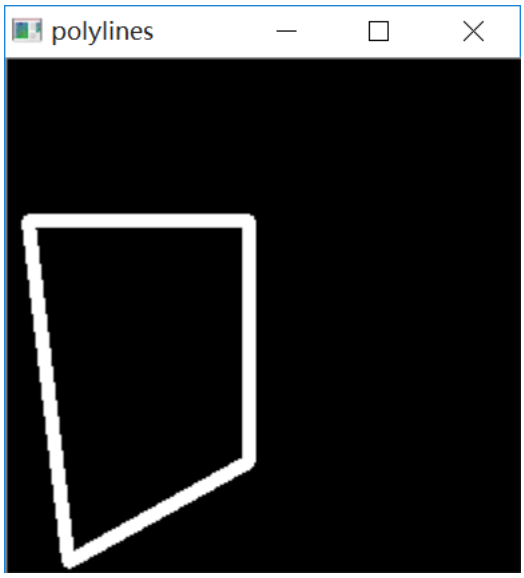


图 3-6 绘制多边形

下面的代码是绘制一个五角星多边形。

```
# -*- coding:utf-8 -*-

# By：Eastmount

import cv2

import numpy as np


#创建黑色图像

img = np.zeros((512,512,3), np.uint8)


#绘制多边形

pts = np.array([[50, 190], [380, 420], [255, 50], [120, 420],
```

```
[450, 190]])
cv2.polylines(img, [pts], True, (0, 255, 255), 10)


#显示图像
cv2.imshow("ellipse", img)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 3-7 所示，它将五个顶点左边分别连接起来，构成了一个黄色的五角星。

图 3-7 绘制五角星

6.绘制文字

在 OpenCV 中，调用 cv2.putText()函数添加对应的文字，其函数原型如下所示：

- img = putText(img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])
  - img 表示要绘制的图像
  - text 表示要绘制的文字
  - org 表示要绘制的位置，图像中文本字符串的左下角

- fontFace 表示字体类型，具体查看 see cv::HersheyFonts
- fontScale 表示字体的大小，计算为比例因子乘以字体特定的基本大小
- color 表示字体的颜色
- thickness 表示字体的粗细
- lineType 表示边界类型
- bottomLeftOrigin 如果为真，则图像数据原点位于左下角，否则它在左上角

下面是绘制文字的代码。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#创建黑色图像
img = np.zeros((256,256,3), np.uint8)

#绘制文字
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img, 'I love Python!I love Huawei!',
                (10, 100), font, 0.5, (255, 255, 0), 2)
```

```
#显示图像
cv2.imshow("polylines", img)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 3-8 所示，绘制的文字为“I love Python!I love Huawei!”。

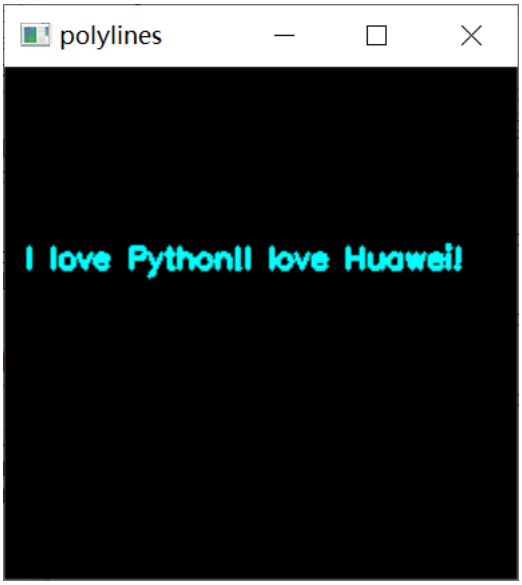


图 3-8 绘制文字

7.总结

本文详细介绍了 OpenCV 绘制几何图形的方法，利用 cv2.line()、v2.circle() 、 cv2.rectangle() 、 cv2.ellipse() 、 cv2.polylines() 、

cv2.putText()函数实现。初学者通过这篇文章将了解基础图形绘制的基础用法，为后续应用提供帮助。同时，建议读者结合这篇文章实现一个画图软件，您可以吗?

参考文献：

[1] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社, 2015.

[2] Eastmount. [计算机图形学课程] 一.MFC 基本绘图函数使用方法[EB/OL]. (2016-11-16). https://blog.csdn.net/Eastmount/article/details/53180524.

[3] 我 i 智能. Python 下 opencv 使用笔记（二）（简单几何图像绘制）[EB/OL]. (2015-07-07). https://blog.csdn.net/on2way/article/details/46793911.

第 04 篇 图像算术与逻辑运算详解

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍如何使用 OpenCV 绘制各类几何图形。这篇文章将详细讲解图像算法运算与逻辑运算，包括图像加法、图像减法、图像与运算、图像或运算、图像非运算与图像异或运算。让我们来对比下这些运算在图像中能实现什么样的效果。

1.图像加法运算

图像加法运算主要有两种方法。第一种是调用 Numpy 库实现，目标图像像素为两张图像的像素之和；第二种是通过 OpenCV 调用 add()函数实现。第二种方法的函数原型如下：

dst = add(src1, src2[, dst[, mask[, dtype]]])

- src1 表示第一张图像的像素矩阵
- src2 表示第二张图像的像素矩阵
- dst 表示输出的图像，必须和输入图像具有相同的大小和通道数

- mask 表示可选操作掩码（8 位单通道数组），用于指定要更改的输出数组的元素。
- dtype 表示输出数组的可选深度

注意，当两幅图像的像素值相加结果小于等于 255 时，则输出图像直接赋值该结果，如 120+48 赋值为 168；如果相加值大于 255，则输出图像的像素结果设置为 255,如(255+64) 赋值为 255。下面的代码实现了图像加法运算。

```
#coding:utf-8
# By：Eastmount
import cv2
import numpy as np

#读取图片
img = cv2.imread("luo.png")

#图像各像素加 100
m = np.ones(img.shape, dtype="uint8")*100

#OpenCV 加法运算
result = cv2.add(img, m)

#显示图像
```

```
cv2.imshow("original", img)
cv2.imshow("result", result)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出如图 4-1 所示，左边为“小珞珞”的原始图像，右边为像素值增加 100 像素后的图像，输出图像显示更偏白。

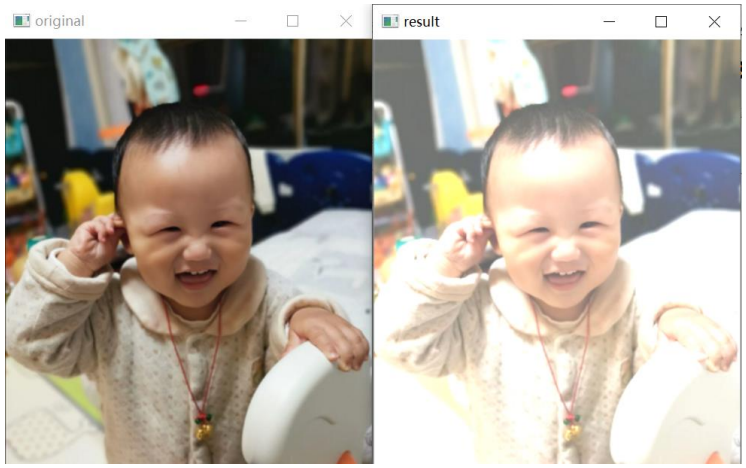


图 4-1 图像加法运算

2.图像减法运算

图像减法运算主要调用 subtract()函数实现，其原型如下所示：

```
dst = subtract(src1, src2[, dst[, mask[, dtype]]])
```

- src1 表示第一张图像的像素矩阵
- src2 表示第二张图像的像素矩阵

- dst 表示输出的图像，必须和输入图像具有相同的大小和通道数
- mask 表示可选操作掩码（8 位单通道数组），用于指定要更改的输出数组的元素。
- dtype 表示输出数组的可选深度

具体实现代码如下所示：

```
#coding:utf-8
# By：Eastmount
import cv2
import numpy as np

#读取图片
img = cv2.imread("luo.png")

#图像各像素减 50
m = np.ones(img.shape, dtype="uint8")*50

#OpenCV 减法运算
result = cv2.subtract(img, m)

#显示图像
cv2.imshow("original", img)
```

```
cv2.imshow("result", result)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出如图 4-2 所示，左边为原始图像，右边为像素值减少 50 像素后的图像，输出图像显示更偏暗。

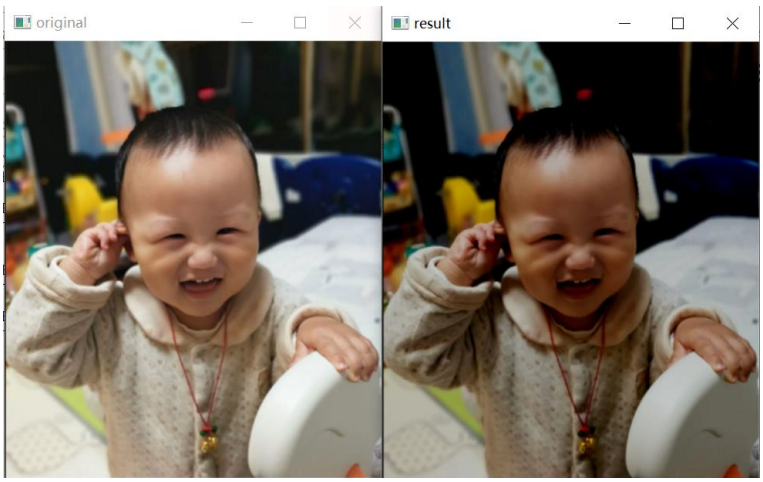


图 4-2 图像减法运算

3.图像与运算

与运算是计算机中一种基本的逻辑运算方式，符号表示为“&”，其运算规则为：

- 0&0=0
- 0&1=0
- 1&0=0

- 1&1=1

图像的与运算是指两张图像（灰度图像或彩色图像均可）的每个像素值进行二进制“与”操作，实现图像裁剪。

dst = bitwise_and(src1, src2[, dst[, mask]])

- src1 表示第一张图像的像素矩阵
- src2 表示第二张图像的像素矩阵
- dst 表示输出的图像，必须和输入图像具有相同的大小和通道数
- mask 表示可选操作掩码（8 位单通道数组），用于指定要更改的输出数组的元素。

下面代码是通过图像与运算实现图像剪裁的功能。

```
#coding:utf-8
# By：Eastmount
import cv2
import numpy as np


#读取图片
img = cv2.imread("luo.png", cv2.IMREAD_GRAYSCALE)


#获取图像宽和高
rows, cols = img.shape[:2]
print(rows, cols)
```

```
#画圆形
circle = np.zeros((rows, cols), dtype="uint8")
cv2.circle(circle, (int(rows/2),int(cols/2)), 100, 255, -1)
print(circle.shape)
print(img.size, circle.size)

#OpenCV 图像与运算
result = cv2.bitwise_and(img, circle)

#显示图像
cv2.imshow("original", img)
cv2.imshow("circle", circle)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出如图 4-3 所示，原始图像与圆形进行与运算之后，提取了其中心轮廓。同时输出图像的形状为 377×326。注意，两张图像的大小和类型必须一致。

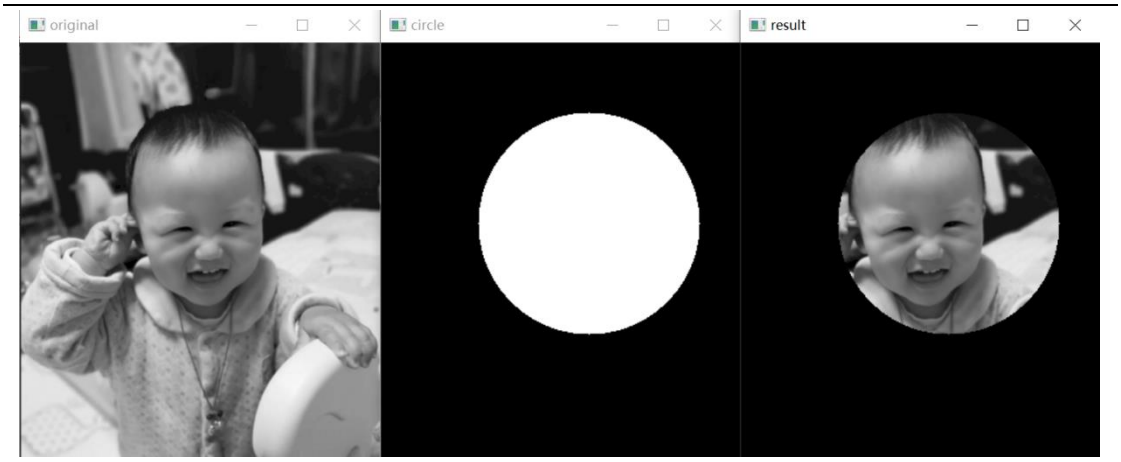


图 4-3 图像与运算

4.图像或运算

逻辑或运算是指如果一个操作数或多个操作数为 true，则逻辑或运算符返回布尔值 true；只有全部操作数为 false，结果才是 false。图像的或运算是指两张图像（灰度图像或彩色图像均可）的每个像素值进行二进制“或”操作，实现图像裁剪。其函数原型如下所示：

```
dst = bitwise_or(src1, src2[, dst[, mask]])
```

- src1 表示第一张图像的像素矩阵
- src2 表示第二张图像的像素矩阵
- dst 表示输出的图像，必须和输入图像具有相同的大小和通道数
- mask 表示可选操作掩码（8 位单通道数组），用于指定要更改的输出数组的元素。

下面代码是通过图像或运算实现图像剪裁的功能。

```
#coding:utf-8
```

```
# By：Eastmount

import cv2

import numpy as np


#读取图片

img = cv2.imread("luo.png", cv2.IMREAD_GRAYSCALE)


#获取图像宽和高

rows, cols = img.shape[:2]


#画圆形

circle = np.zeros((rows, cols), dtype="uint8")

cv2.circle(circle, (int(rows/2),int(cols/2)), 100, 255, -1)


#OpenCV 图像或运算

result = cv2.bitwise_or(img, circle)


#显示图像

cv2.imshow("original", img)

cv2.imshow("circle", circle)

cv2.imshow("result", result)
```

```
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出如图 4-4 所示，原始图像与圆形进行或运算之后，提取了图像除中心原形之外的像素值。

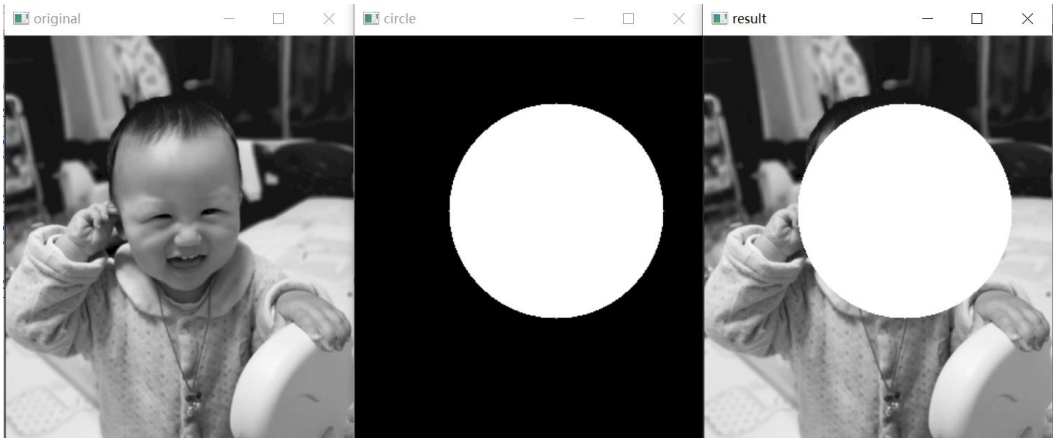


图 4-4 图像或运算

5.图像非运算

图像非运算就是图像的像素反色处理，它将原始图像的黑色像素点转换为白色像素点，白色像素点则转换为黑色像素点，其函数原型如下：

```
dst = bitwise_not(src1, src2[, dst[, mask]])
```

- src1 表示第一张图像的像素矩阵
- src2 表示第二张图像的像素矩阵
- dst 表示输出的图像，必须和输入图像具有相同的大小和通道数

- mask 表示可选操作掩码（8 位单通道数组），用于指定要更改的输出数组的元素。

图像非运算的实现代码如下所示。

```
#coding:utf-8
import cv2
import numpy as np

#读取图片
img = cv2.imread("Lena.png", cv2.IMREAD_GRAYSCALE)

#OpenCV 图像非运算
result = cv2.bitwise_not(img)

#显示图像
cv2.imshow("original", img)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

原始图像非运算之后输出如图 4-5 所示。

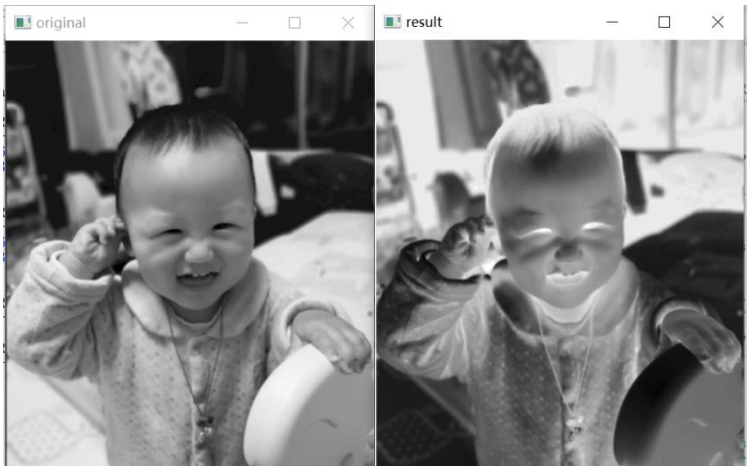


图 4-5 图像非运算

6.图像异或运算

逻辑异或运算（xor）是一个数学运算符，数学符号为“⊕”，计算机符号为“xor”，其运算法则为：如果 a、b 两个值不相同，则异或结果为 1；如果 a、b 两个值相同，异或结果为 0。

图像的异或运算是指两张图像（灰度图像或彩色图像均可）的每个像素值进行二进制“异或”操作，实现图像裁剪。其函数原型如下所示：

```
dst = bitwise_xor(src1, src2[, dst[, mask]])
```

- src1 表示第一张图像的像素矩阵
- src2 表示第二张图像的像素矩阵
- dst 表示输出的图像，必须和输入图像具有相同的大小和通道数
- mask 表示可选操作掩码（8 位单通道数组），用于指定要更改的输出数组的元素。

图像异或运算的实现代码如下所示。

```
#coding:utf-8
# By：Eastmount
import cv2
import numpy as np

#读取图片
img = cv2.imread("luo.png", cv2.IMREAD_GRAYSCALE)

#获取图像宽和高
rows, cols = img.shape[:2]

#画圆形
circle = np.zeros((rows, cols), dtype="uint8")
cv2.circle(circle, (int(rows/2),int(cols/2)), 100, 255, -1)

#OpenCV 图像异或运算
result = cv2.bitwise_xor(img, circle)

#显示图像
cv2.imshow("original", img)
cv2.imshow("circle", circle)
```

```
cv2.imshow("result", result)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

原始图像与圆形进行异或运算之后输出如图 4-6 所示。

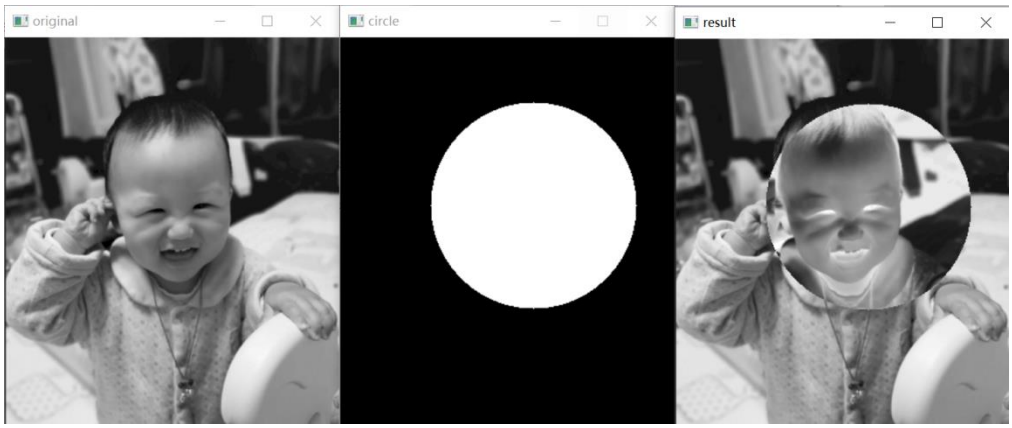


图 4-6 图像异或运算

7.总结

本文详细介绍了图像处理的算术运算与逻辑运算，包括图像加法、图像减法、图像与运算、图像或运算、图像非运算与图像异或运算，并以“小珞珞”图像为案例进行讲解，希望对您有所帮助。

第 05 篇 图像融合处理和 ROI 区域绘制

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像算法运算与逻辑运算，包括图像加法、图像减法、图像与运算、图像或运算、图像非运算与图像异或运算。这篇文章将详细讲解图像融合处理和 ROI 区域绘制，同时补充图像属性、通道和类型转换。

1.图像融合

图像融合通常是指多张图像的信息进行融合，从而获得信息更丰富的结果，能够帮助人们观察或计算机处理。图 5-1 是将两张不清晰的图像融合得到更清晰的效果图。

图 5-1 图像融合原理

图像融合是在图像加法的基础上增加了系数和亮度调节量，它与图像的主要

区别如下[1-3]：

- 图像加法：目标图像 = 图像 1 + 图像 2
- 图像融合：目标图像 = 图像 1 × 系数 1 + 图像 2 × 系数 2 + 亮度调节量

在 OpenCV 中，图像融合主要调用 addWeighted()函数实现，其原型如下。需要注意的是，两张融合图像的像素大小必须一致，参数 gamma 不能省略。

```
dst = cv2.addWeighted(scr1, alpha, src2, beta, gamma)
dst = src1 * alpha + src2 * beta + gamma
```

下面的代码是将两张图片进行图像融合，两张图片的系数均为 1。

```
#coding:utf-8
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图片
src1 = cv2.imread('lena.png')
src2 = cv2.imread('luo.png')


#图像融合
```

```
result = cv2.addWeighted(src1, 1, src2, 1, 0)


#显示图像
cv2.imshow("src1", src1)
cv2.imshow("src2", src2)
cv2.imshow("result", result)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 5-2 所示，它将 src1 图像和 src2 图像按比例系数进行了融合，生成目标结果图 result。

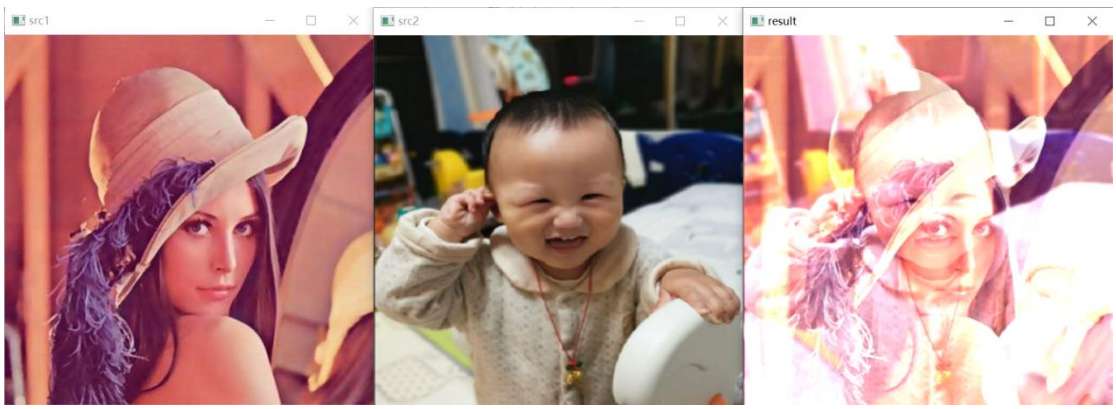


图 5-2 图像融合效果图

同样可以设置不同的融合比例，图 5-3 是下面核心函数的效果图。

- cv2.addWeighted(src1, 0.6, src2, 0.8, 10)

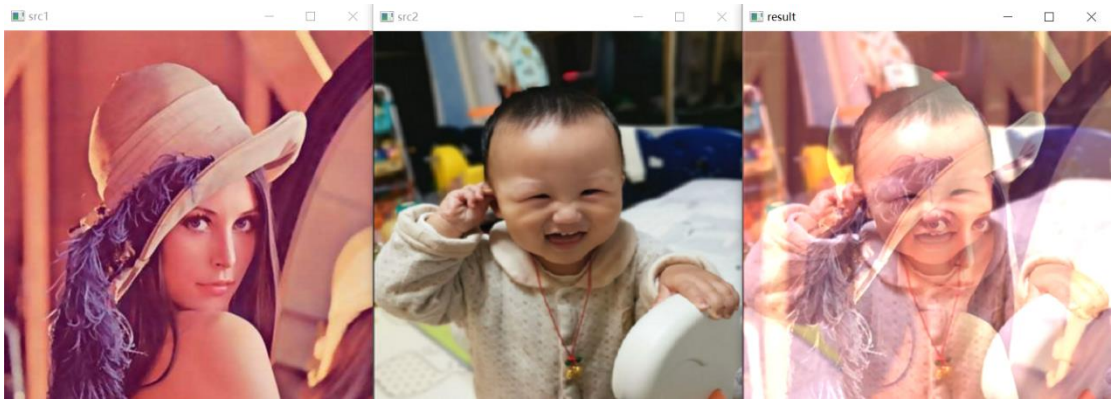


图 5-3 图像融合效果

2.图像 ROI 区域定位

ROI（Region of Interest）表示感兴趣区域，是指从被处理图像以方框、圆形、椭圆、不规则多边形等方式勾勒出需要处理的区域。可以通过各种算子（Operator）和函数求得感兴趣 ROI 区域，被广泛应用于热点地图、人脸识别、图像分割等领域。如图 5-4 获取 Lena 图的脸部轮廓[4]。

图 5-4 获取图像 ROI 区域

通过像素矩阵可以直接获取 ROI 区域，如 img[200:400, 200:400]。下面的代码是获取脸部 ROI 区域并显示。

```
# -*- coding:utf-8 -*-
# By：Eastmount
```

```
import cv2
import numpy as np

#读取图片
img = cv2.imread("lena.png")

#定义 200×200 矩阵 3 对应 BGR
face = np.ones((200, 200, 3))

#显示原始图像
cv2.imshow("Demo", img)

#显示 ROI 区域
face = img[150:350, 150:350]
cv2.imshow("face", face)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 5-5 所示，它将 Lena 原图的脸部提取出来。

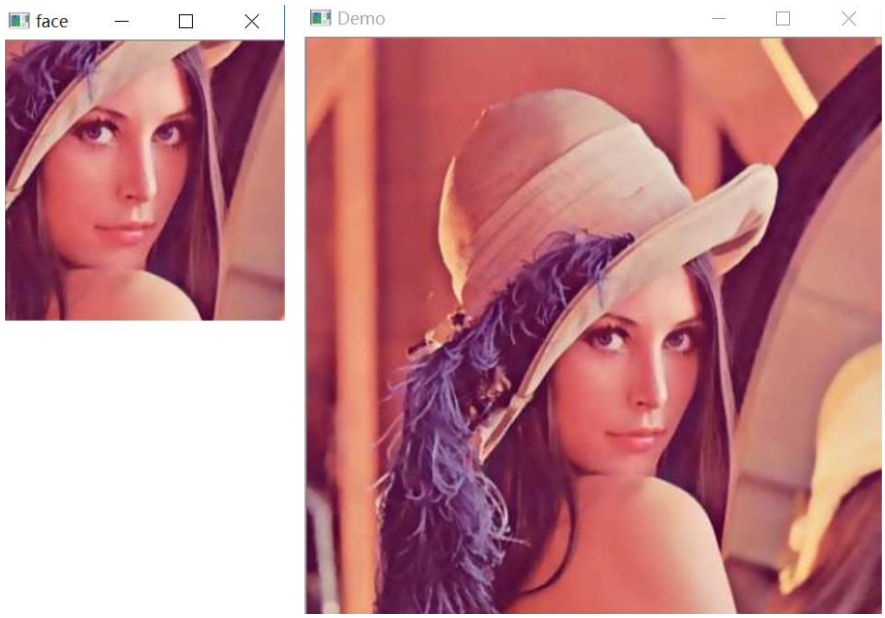


图 5-5 图像脸部提取

同样，如果想将提取的 ROI 区域融合至其他图片，则使用赋值语句即可。下面代码是将提取的 Lena 头部轮廓融合至一幅新的图像中。

```
# -*- coding:utf-8 -*-

# By：Eastmount

import cv2

import numpy as np


#读取图片

img = cv2.imread("Lena.png")

test = cv2.imread("luo.png",)


#定义 150×150 矩阵 3 对应 BGR
```

```
face = np.ones((150, 150, 3))


#显示原始图像
cv2.imshow("Demo", img)


#显示 ROI 区域
face = img[200:350, 200:350]
test[250:400, 250:400] = face
cv2.imshow("Result", test)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行结果如图 5-6 所示，它将提取的 150×150 脸部轮廓融合至新的图像 [250:400, 250:400] 区域。

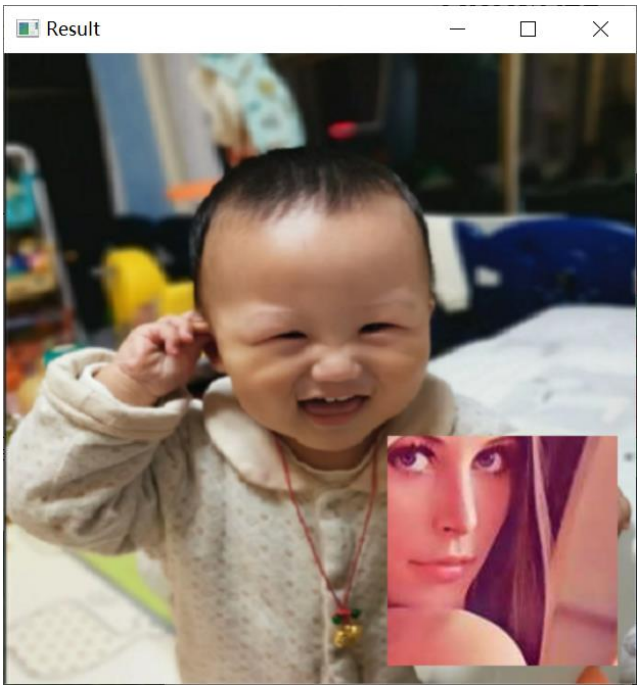


图 5-6 图像 ROI 区域融合

3.图像属性

前面一篇文章中我们已经看到了 size、shape 等关键字。这篇文章就对图像中最常见的三个属性进行介绍，它们分别是图像形状（shape）、像素大小（size）和图像类型（dtype）。

（1）shape

通过 shape 关键字获取图像的形状，返回包含行数、列数、通道数的元组。其中灰度图像返回行数和列数，彩色图像返回行数、列数和通道数。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
```

```
import numpy


#读取图片
img = cv2.imread("luo.png")


#获取图像形状
print(img.shape)


#显示图像
cv2.imshow("Demo", img)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

最终输出结果如图 5-7 所示，（412，412，3），它表示该图像共 412 行、412 列像素，包括 3 个通道。

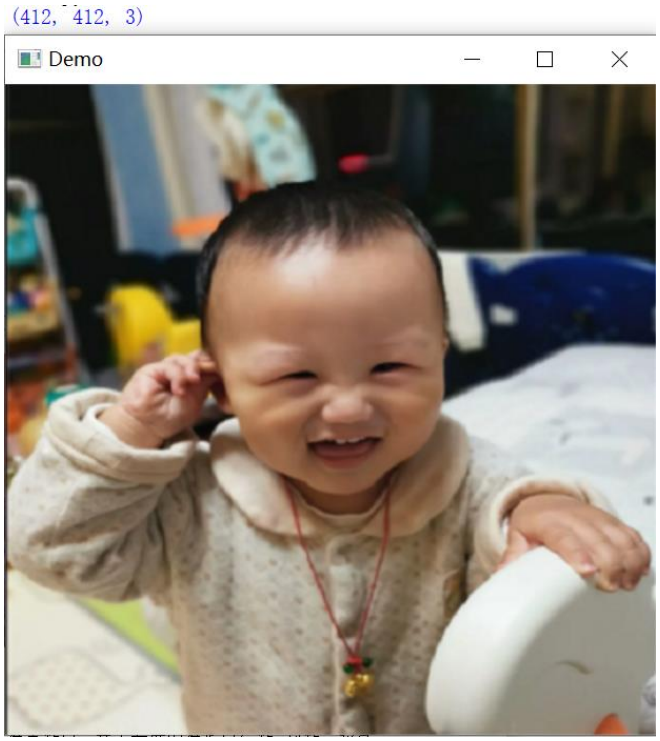


图 5-7 输出图像形状

（2）size

通过 size 关键字获取图像的像素数目，其中灰度图像返回行数×列数，彩色图像返回行数×列数×通道数。下述代码就是获取“luo.png”图像的大小。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy

#读取图片
img = cv2.imread("luo.png")
```

```
#获取图像形状

print(img.shape)


#获取像素数目

print(img.size)
```

输出结果如下所示，包含 510468 个像素，即为 413×412×3。

- (412, 412, 3)
- 509232

（3）dtype

通过 dtype 关键字获取图像的数据类型，通常返回 uint8。

```
# -*- coding:utf-8 -*-

# By：Eastmount

import cv2

import numpy


#读取图片

img = cv2.imread("Lena.png")


#获取图像形状

print(img.shape)
```

```
#获取像素数目
print(img.size)


#获取图像数据类型
print(img.dtype)
```

4.图像通道分离及合并

OpenCV 通过 split()函数和 merge()函数实现对图像通道的处理，包括通道分离和通道合并。

（1）split()函数

OpenCV 读取的彩色图像由蓝色（B）、绿色（G）、红色（R）三原色组成，每一种颜色可以认为是一个通道分量[4]，如图 5-8 所示。

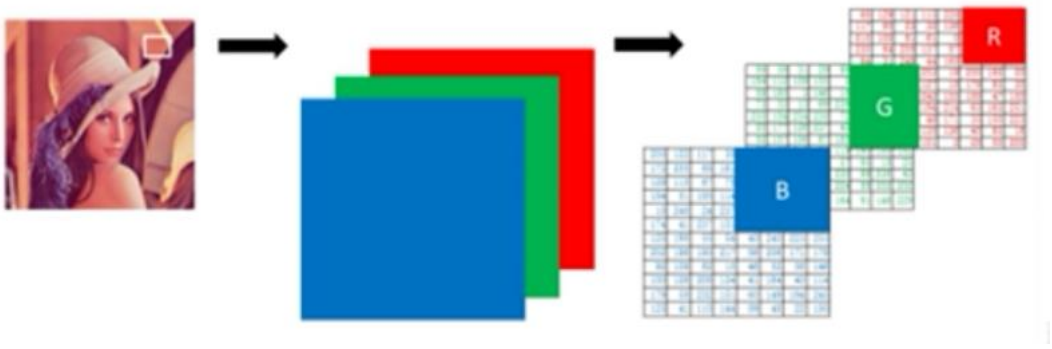


图 5-8 彩色图像 BGR 通道

split()函数用于将一个多通道数组分量成三个单通道，其函数原型如下所示：

- mv = split(m[, mv])
  - m 表示输入的多通道数组

- mv 表示输出的数组或 vector 容器

下面的代码是获取彩色“小珞珞”图像三个颜色通道并分别显示。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy

#读取图片
img = cv2.imread("luo.png")

#拆分通道
b, g, r = cv2.split(img)

#显示原始图像
cv2.imshow("B", b)
cv2.imshow("G", g)
cv2.imshow("R", r)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

显示结果如图 5-9 所示，它展示了 B、G、R 三个通道的颜色分量。

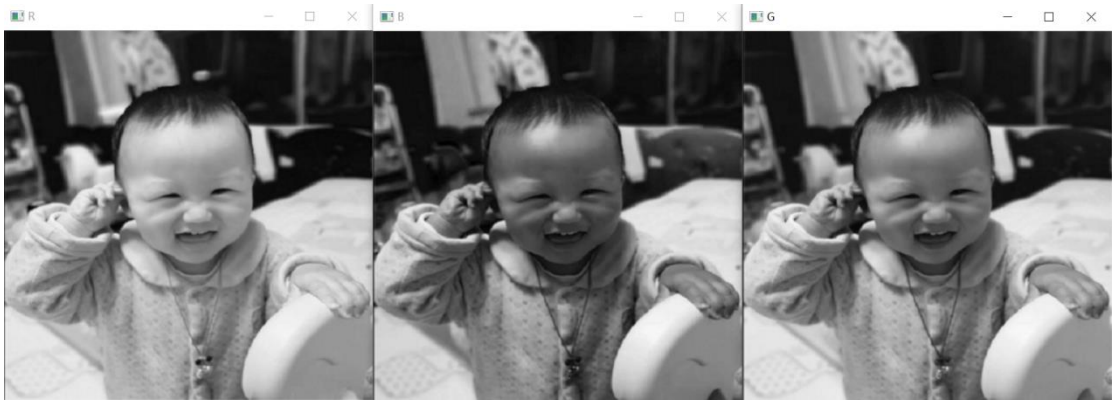


图 5-9 显示彩色图像 BGR 通道图像

同时，可以获取不同通道颜色，核心代码为：

```
b = cv2.split(a)[0]
g = cv2.split(a)[1]
r = cv2.split(a)[2]
```

（2）merge()函数

该函数是 split()函数的逆向操作，将多个数组合成一个通道的数组，从而实现图像通道的合并，其函数原型如下：

- dst = merge(mv[, dst])
  - mv 表示输入的需要合并的数组，所有矩阵必须有相同的大小和深度
  - dst 表示输出具有与 mv 相同大小和深度的数组

实现图像三个颜色通道融合的代码如下：

```
# -*- coding:utf-8 -*-
# By：Eastmount
```

```
import cv2
import numpy as np


#读取图片
img = cv2.imread("luo.png")


#拆分通道
b, g, r = cv2.split(img)


#合并通道
m = cv2.merge([b, g, r])
cv2.imshow("Merge", m)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

显示结果如图 5-10 所示，它将拆分的 B、G、R 三个通道的颜色分量进行了合并，接着显示合并后的图像。

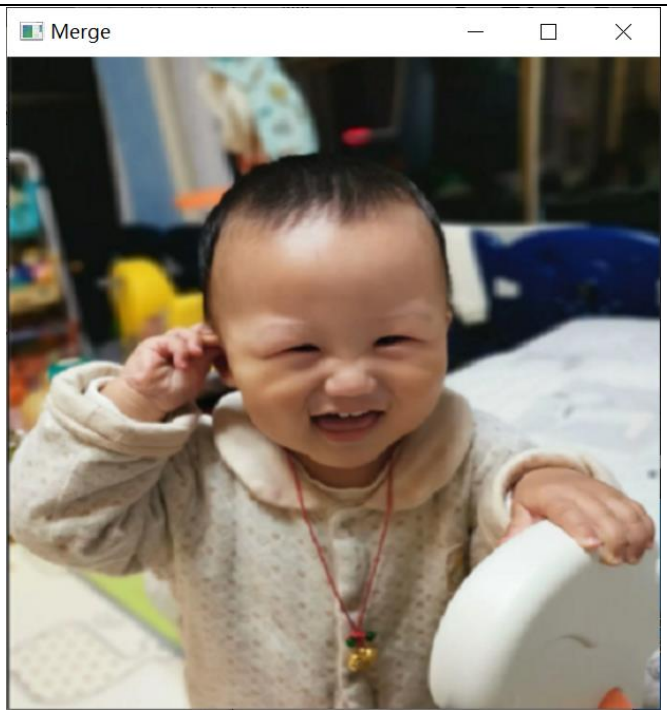


图 5-10 合并图像

同时，可以调用该函数提取图像的不同颜色，比如提取 B 颜色通道，G、B 通道设置为 0。代码如下所示：

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np


#读取图片
img = cv2.imread("luo.png")
rows, cols, chn = img.shape
```

```
#拆分通道
b = cv2.split(img)[0]


#设置 g、r 通道为 0
g = np.zeros((rows,cols), dtype=img.dtype)
r = np.zeros((rows,cols), dtype=img.dtype)


#合并通道
m = cv2.merge([b, g, r])
cv2.imshow("Merge", m)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

此时显示的图像为蓝色通道，如图 5-11 所示，其他颜色的通道方法也类似。

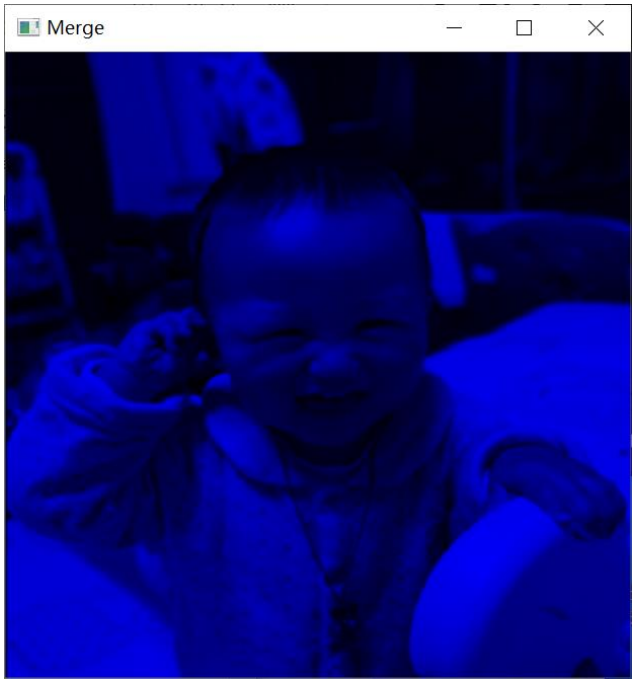


图 5-11 显示合并蓝色通道的图像

5.图像类型转换

在日常生活中，我们看到的大多数彩色图像都是 RGB 类型，但是在图像处理过程中，常常需要用到灰度图像、二值图像、HSV、HSI 等颜色。图像类型转换是指将一种类型转换为另一种类型，比如彩色图像转换为灰度图像、BGR 图像转换为 RGB 图像。OpenCV 提供了 200 多种不同类型之间的转换，其中最常用的包括 3 类，如下：

- cv2.COLOR_BGR2GRAY
- cv2.COLOR_BGR2RGB
- cv2.COLOR_GRAY2BGR

OpenCV 提供了 cvtColor()函数实现这些功能。其函数原型如下所示：

- dst = cv2.cvtColor(src, code[, dst[, dstCn]])
  - src 表示输入图像，需要进行颜色空间变换的原图像
  - dst 表示输出图像，其大小和深度与 src 一致
  - code 表示转换的代码或标识
  - dstCn 表示目标图像通道数，其值为 0 时，则有 src 和 code 决定

该函数的作用是将一个图像从一个颜色空间转换到另一个颜色空间，其中，RGB 是指 Red、Green 和 Blue，一副图像由这三个通道（channel）构成；Gray 表示只有灰度值一个通道；HSV 包含 Hue（色调）、Saturation（饱和度）和 Value（亮度）三个通道。在 OpenCV 中，常见的颜色空间转换标识包括 CV_BGR2BGRA、CV_RGB2GRAY、CV_GRAY2RGB、CV_BGR2HSV、CV_BGR2XYZ、CV_BGR2HLS[3]。

下面是调用 cvtColor()函数将图像进行灰度化处理的代码。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图片
src = cv2.imread('luo.png')
```

```
#图像类型转换
result = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 5-12 所示，它将左边的彩色图像转换为右边的灰度图像，更多灰度转化算法将在后面的文章详细介绍。

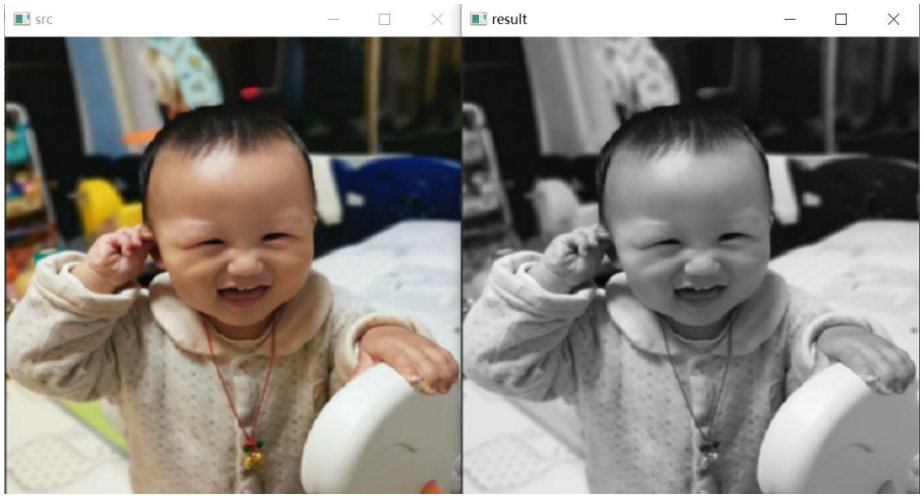


图 5-12 图像灰度化处理

同样，可以调用下列核心代码将彩色图像转换为 HSV 颜色空间，如图 5-

13 所示。

- grayImage = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)

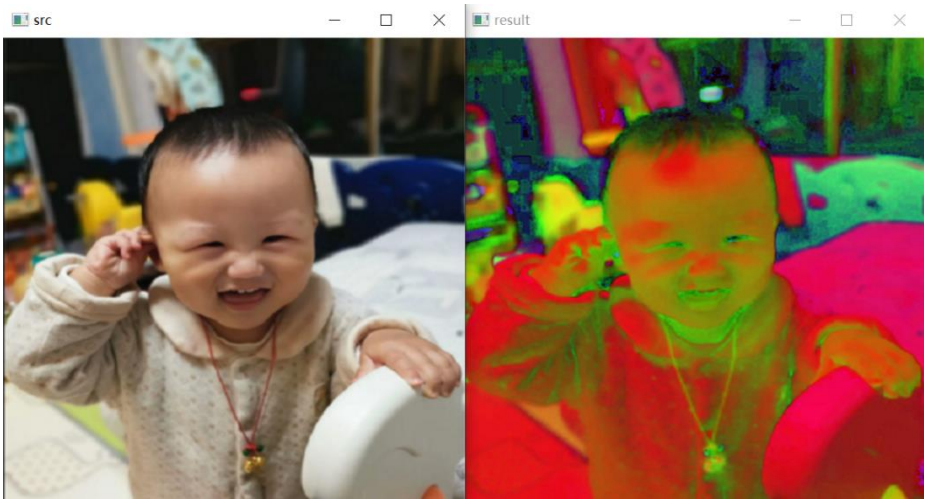


图 5-13 图像 HSV 转换

下面代码对比了九种常见的颜色空间，包括 BGR、RGB、GRAY、HSV、YCrCb、HLS、XYZ、LAB 和 YUV，并循环显示处理后的图像。

```
# -*- coding:utf-8 -*-

# By：Eastmount

import cv2

import numpy as np

import matplotlib.pyplot as plt


#读取原始图像

img_BGR = cv2.imread('luo.png')


#BGR 转换为 RGB
```

```
img_RGB = cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2RGB)

#灰度化处理
img_GRAY = cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2GRAY)

#BGR 转 HSV
img_HSV = cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2HSV)

#BGR 转 YCrCb
img_YCrCb = cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2YCrCb)

#BGR 转 HLS
img_HLS = cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2HLS)

#BGR 转 XYZ
img_XYZ = cv2.cvtColor(img_BGR,
```

```
cv2.COLOR_BGR2XYZ)


#BGR 转 LAB
img_LAB = cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2LAB)


#BGR 转 YUV
img_YUV = cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2YUV)


#调用 matplotlib 显示处理结果
titles = ['BGR', 'RGB', 'GRAY', 'HSV', 'YCrCb', 'HLS', 'XYZ',
'LAB', 'YUV']
images = [img_BGR, img_RGB, img_GRAY, img_HSV,
img_YCrCb,
          img_HLS, img_XYZ, img_LAB, img_YUV]
for i in range(9):
    plt.subplot(3, 3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 5-14 所示：

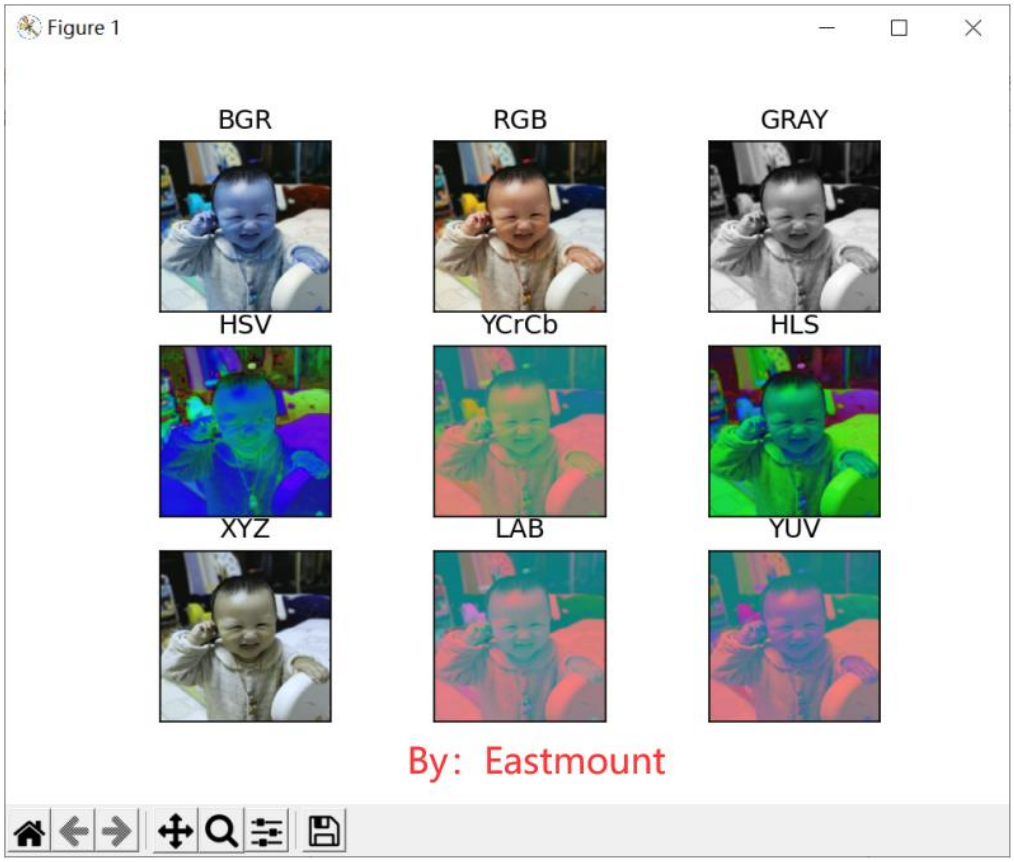


图 5-14 图像九种颜色空间相互转换

6.总结

本章主要讲解 Python 和 OpenCV 的图像基础处理，从读取显示图像到读取修改像素，从创建、复制、保存图像到获取图像属性合通道，再详细讲解了图像算数与逻辑运算，包括图像加法、减法、与运算、或运算、异或运算、非运算，最后讲解了图像融合和获取图像 ROI 区域及图像类型转换。本章知识为后续的图像处理、图像识别、图像变换打下扎实基础。

参考文献：

[1] 冈萨雷斯. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社, 2013.

[2] 毛星云, 冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社, 2015.

[3]Eastmount. [Python 图像处理] 五.图像融合、加法运算及图像类型转换[EB/OL]. (2018-09-03). https://blog.csdn.net/Eastmount/article/details/82347501.

[4] Eastmount. [Python 图像处理] 三.获取图像属性、兴趣 ROI 区域及通道处理 [EB/OL]. (2018-08-29). https://blog.csdn.net/Eastmount/article/details/82177300.

第 06 篇 图像几何变换之平移缩放旋转

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像融合处理和 ROI 区域绘制。这篇文章将详细讲解图像几何变换，包括图像平移、图像缩放和图像旋转。

1.图像几何变换

图像几何变换不改变图像的像素值，在图像平面上进行像素变换。适当的几何变换可以最大程度地消除由于成像角度、透视关系乃至镜头自身原因所造成的几何失真所产生的负面影响。几何变换常常作为图像处理应用的预处理步骤，是图像归一化的核心工作之一[1]。

一个几何变换需要两部分运算：

- **空间变换**：包括平移、缩放、旋转和正平行投影等，需要用它来表示输出图像与输入图像之间的像素映射关系。
- **灰度插值算法**：按照这种变换关系进行计算，输出图像的像素可能被映射到输入图像的非整数坐标上[2]。

图像几何变换在变换过程中会建立一种原图像像素与变换后图像像素之间

的映射关系，通过这种关系，能够从一方的像素计算出另一方的像素的坐标位置。通常将图像坐标映射到输出的过程称作向前映射，反之，将输出图像映射到输入的过程称作向后映射。向后映射在实践中使用较多，原因是能够避免使用向前映射中出现映射不完全和映射重叠的问题。

图 6-1 展示了图像放大的示例，右边图中只有(0,0)、(0,2)、(2,0)、(2,2)四个坐标根据映射关系在原图像中找到了相对应的像素，其余的 12 个坐标没有有效值[3]。

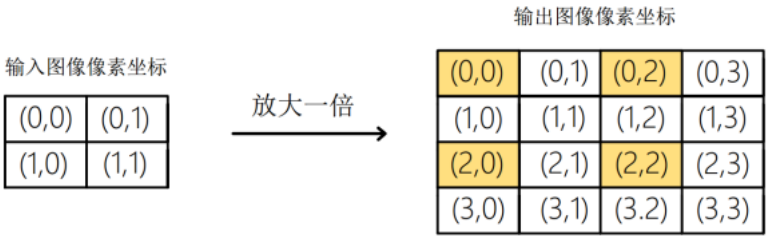


图 6-1 图像放大处理

对于数字图像而言，像素的坐标是离散型非负整数，但是在进行变换的过程中有可能产生浮点坐标值。这在图像处理中是一个无效的坐标。为了解决这个问题需要用到插值算法。常见算法如下：

- 最近邻插值
- 双线性插值
- 双立方插值

图像变换是建立在矩阵运算基础上，通过矩阵运算可以很快找到对应关系。在这篇文章中，我们将介绍常见的图像几何变换，包括图形平移、图像缩放、图像旋转、图像镜像、图像仿射、图像透视等。

2.图像平移

图像平移是将图像中的所有像素点按照给定的平移量进行水平或垂直方向上的移动。假设原始像素的位置坐标为（x0，y0），经过平移量（△x，△y）后，坐标变为（x1, y1），如图 6-2 所示[3-5]。

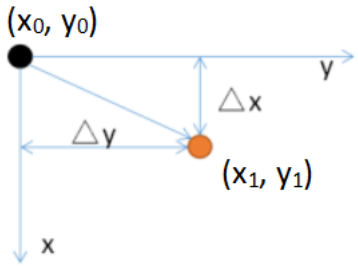


图 6-2 图像平移运算

用数学式子表示为公式（6-1）。

$$\begin{aligned} x_1 &= x_0 + \Delta x \\ y_1 &= y_0 + \Delta y \end{aligned} \tag{6-1}$$

用矩阵表示如公式（6-2）所示：

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix} \tag{6-2}$$

式子中，矩阵称为平移变换矩阵或因子，△x 和△y 称为平移量。图像平移首先定义平移矩阵 M，再调用 warpAffine()函数实现平移，核心函数如下：

```
M = np.float32([[1, 0, x], [0, 1, y]])
```

- M 表示平移矩阵，其中 x 表示水平平移量，y 表示垂直平移量

```
shifted = cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])
```

- src 表示原始图像
- M 表示平移矩阵
- dsize 表示变换后的输出图像的尺寸大小
- dst 为输出图像，其大小为 dsize，类型与 src 相同
- flag 表示插值方法的组合和可选值
- borderValue 表示像素外推法，当 borderMode = BORDER_TRANSPARENT 时，表示目标图像中的像素不会修改源图像中的“异常值”。
- borderValue 用于边界不变的情况，默认情况下为 0

下面代码是图像平移的一个简单案例，它定义了图像平移矩阵 M，然后调用 warpAffine()函数将原始图像垂直向下平移了 50 个像素，水平向右平移了 100 个像素。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np


#读取图片
src = cv2.imread('scenery.png')


#图像平移矩阵
```

```
M = np.float32([[1, 0, 100], [0, 1, 50]])


#获取原始图像列数和行数

rows, cols = src.shape[:2]


#图像平移

result = cv2.warpAffine(src, M, (cols, rows))


#显示图像

cv2.imshow("original", src)

cv2.imshow("result", result)


#等待显示

cv2.waitKey(0)

cv2.destroyAllWindows()
```

输出结果如图 6-3 所示：

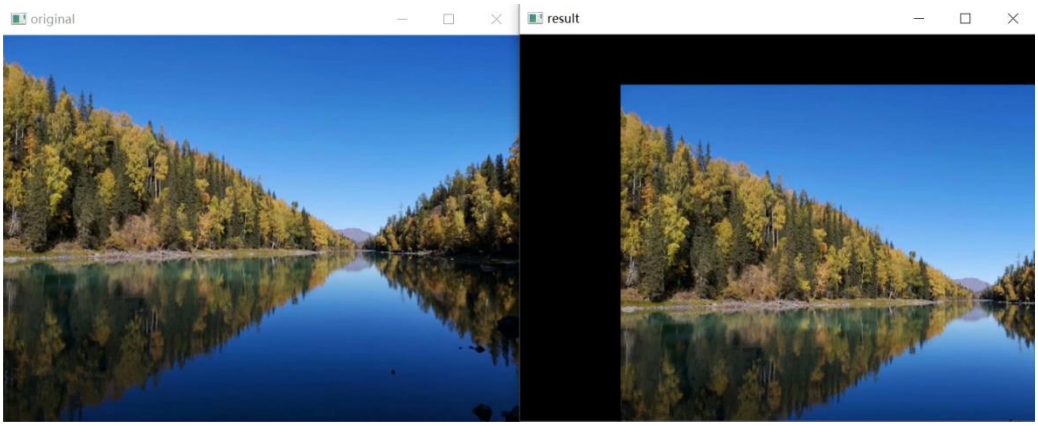


图 6-3 图像平移变换

下面一个案例是将图像分别向下、向上、向右、向左平移，再调用 matplotlib 绘图库依次绘制的过程。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图片
img = cv2.imread('scenery.png')
image = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)


#图像平移
#垂直方向 向下平移 100
```

```
M = np.float32([[1, 0, 0], [0, 1, 100]])
img1 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#垂直方向 向上平移 100
M = np.float32([[1, 0, 0], [0, 1, -100]])
img2 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#水平方向 向右平移 100
M = np.float32([[1, 0, 100], [0, 1, 0]])
img3 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#水平方向 向左平移 100
M = np.float32([[1, 0, -100], [0, 1, 0]])
img4 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#循环显示图形
titles = [ 'Image1', 'Image2', 'Image3', 'Image4']
```

```
images = [img1, img2, img3, img4]
for i in range(4):
    plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 6-4 所示，它从四个方向都进行了平移，并且调用 subplot() 函数将四个子图绘制在一起。

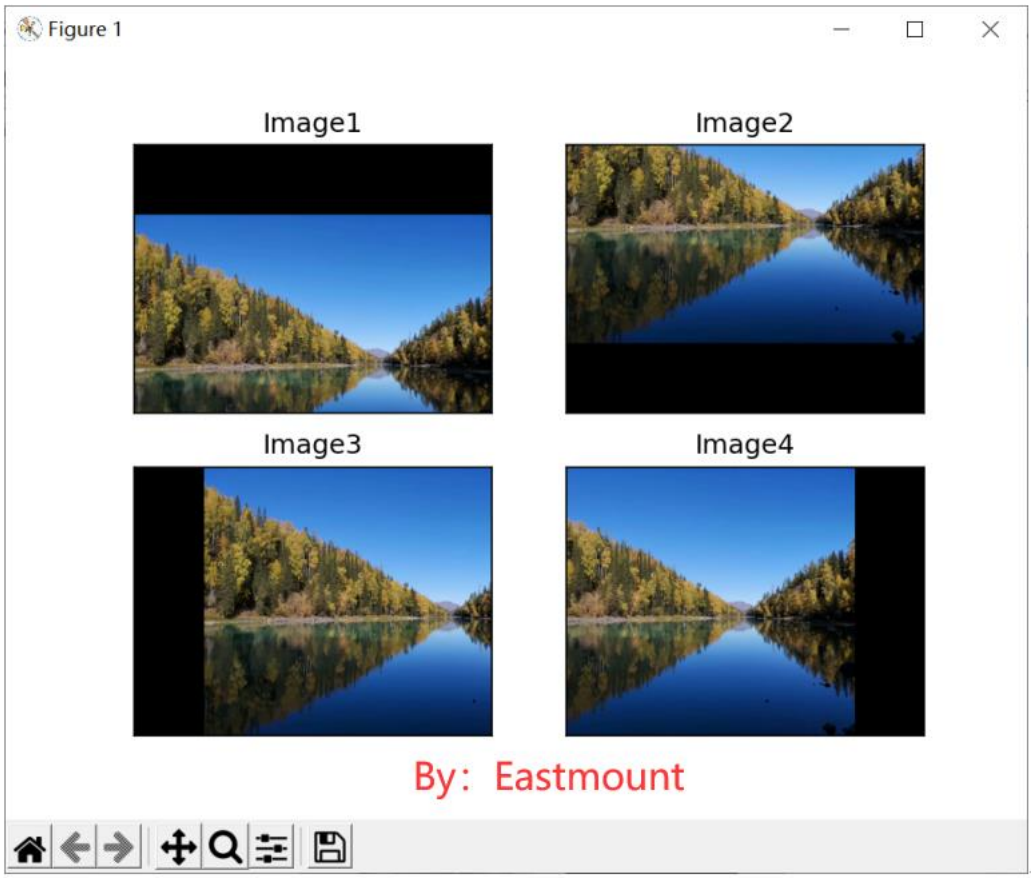


图 6-4 图像四个方向的平移

3.图像缩放

图像缩放（image scaling）是指对数字图像的大小进行调整的过程。在Python 中，图像缩放主要调用 resize()函数实现，函数原型如下：

result = cv2.resize(src, dsize[, result[. fx[, fy[, interpolation]]]])

- src 表示原始图像
- dsize 表示图像缩放的大小
- result 表示图像结果
- fx 表示图像 x 轴方向缩放大小的倍数
- fy 表示图像 y 轴方向缩放大小的倍数
- interpolation 表示变换方法。CV_INTER_NN 表示最近邻插值；CV_INTER_LINEAR 表示双线性插值（缺省使用）；CV_INTER_AREA 表示使用像素关系重采样，当图像缩小时，该方法可以避免波纹出现，当图像放大时，类似于 CV_INTER_NN；CV_INTER_CUBIC 表示立方插值

常见的图像缩放两种方式如下所示，第一种方式是将原图像设置为(160, 160)像素大小，第二种方式是将原始图像缩小为 0.5 倍。

- result = cv2.resize(src, (160,160))
- result = cv2.resize(src, None, fx=0.5, fy=0.5)

设(x_1, y_1)是缩放后的坐标，(x_0, y_0)是缩放前的坐标，sx、sy 为缩放因子，

则图像缩放的计算公式（6-3）所示：

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6-3}$$

下面是 Python 实现图像缩放的代码，它将所读取的风景图像进行缩小。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取图片
src = cv2.imread('scenery.png')

#图像缩放
result = cv2.resize(src, (200,100))
print(result.shape)

#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)

#等待显示
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 6-5 所示，图像缩小为(100, 200, 3)像素。注意，代码中调用函数 cv2.resize(src, (200,100)) 设置新图像大小 dsize 的列数为 200，行数为 100。

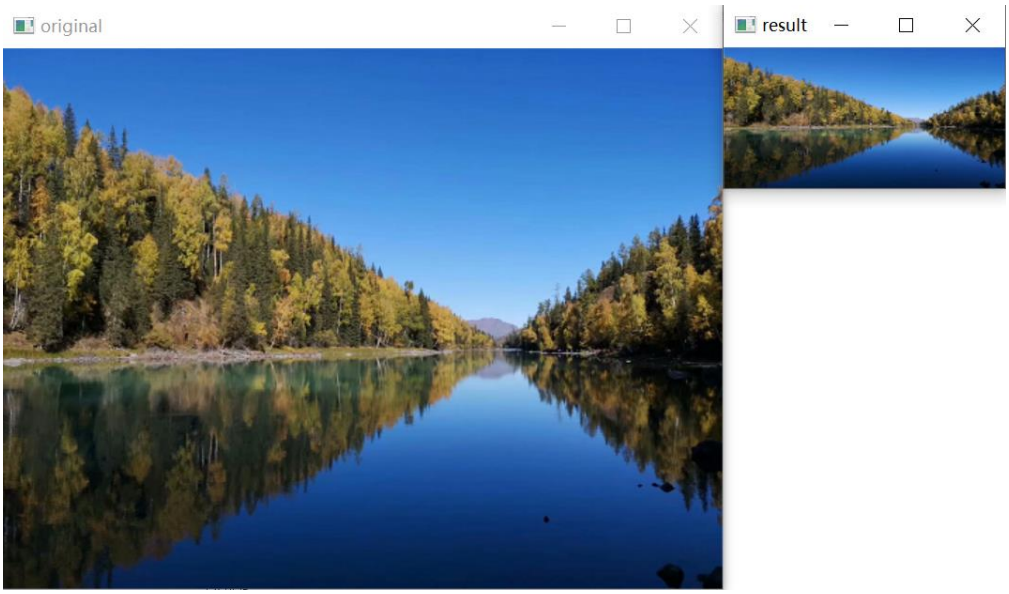


图 6-5 图像缩放

下面讲解另一种图像缩放变换的方法，通过原始图像像素乘以缩放系数进行图像变换，代码如下：

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
#读取图片
src = cv2.imread('scenery.png')
rows, cols = src.shape[:2]
print(rows, cols)


#图像缩放  dsize(列,行)
result = cv2.resize(src, (int(cols*0.6), int(rows*1.2)))


#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

获取图片“scenery.png”的元素像素值，其 rows 值为 384，cols 值为 512，接着进行宽度缩小 0.6 倍、高度放大 1.2 倍的处理，运行前后对比效果如图 6-6 所示。

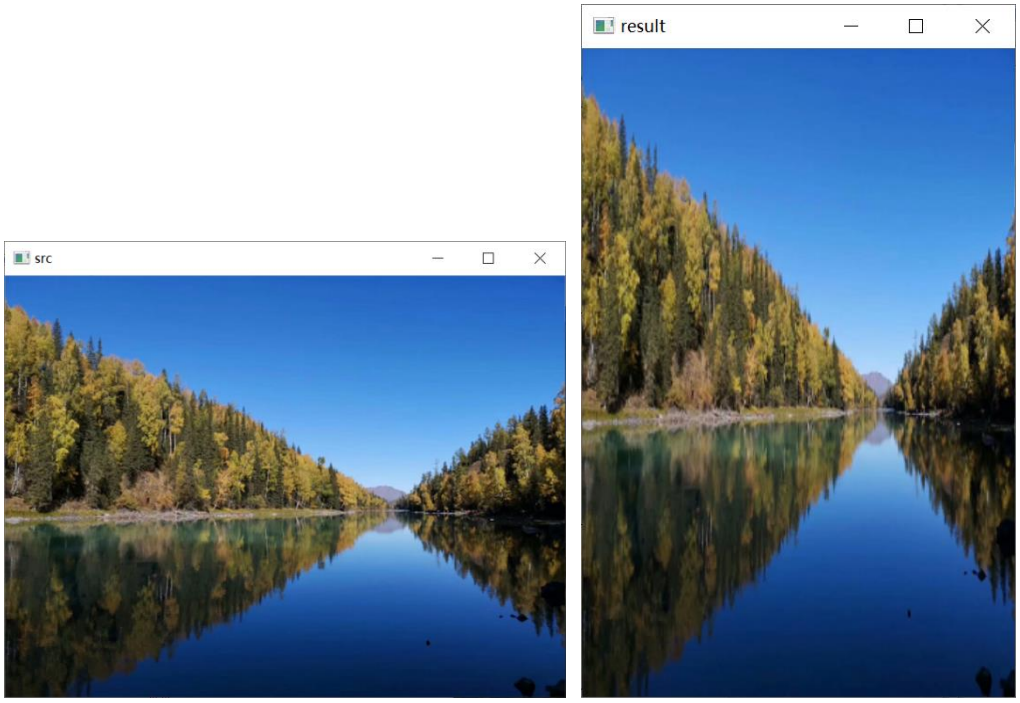


图 6-6 图像按倍数缩放

最后讲解调用(fx,fy)参数设置缩放倍数的方法，对原始图像进行放大或缩小操作。下面代码是 fx 和 fy 方向缩小至原始图像 0.3 倍的操作。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np


#读取图片
src = cv2.imread('scenery.png')
rows, cols = src.shape[:2]
print(rows, cols)
```

```
#图像缩放
result = cv2.resize(src, None, fx=0.3, fy=0.3)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出的结果如图 6-7 所示，这是按比例 0.3×0.3 缩小的。

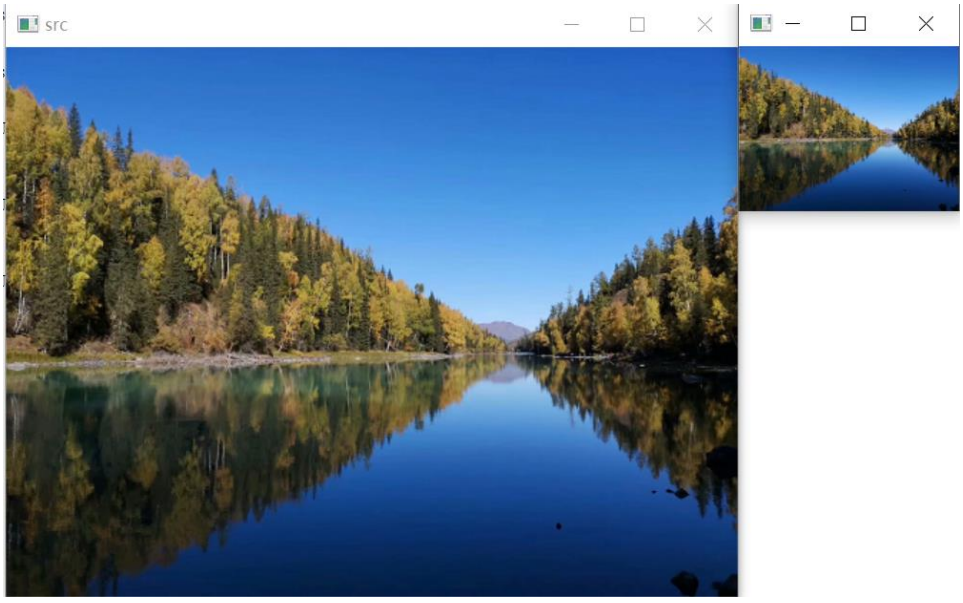


图 6-7 图像缩放第三种方法

4.图像旋转

图像旋转是指图像以某一点为中心旋转一定的角度，形成一幅新的图像的过程。图像旋转变换会有一个旋转中心，这个旋转中心一般为图像的中心，旋转之后图像的大小一般会发生改变。图 6-8 表示原始图像的坐标(x_0, y_0)旋转至(x_1, y_1)的过程。

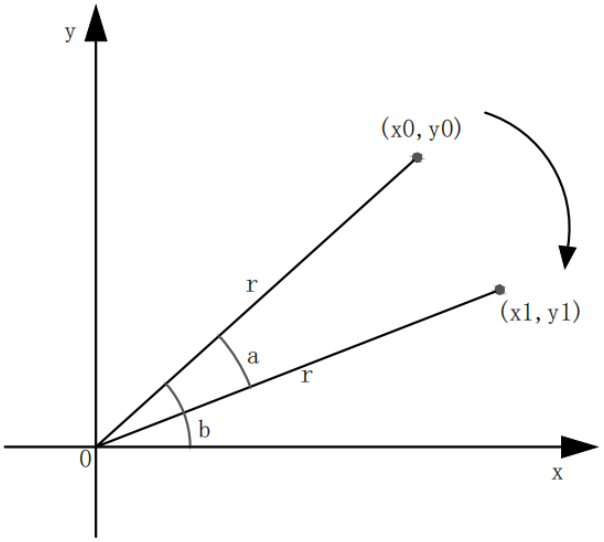


图 6-8 图像旋转变换

旋转公式如(6-4)所示，其中(m,n)是旋转中心，a 是旋转的角度，(left,top)是旋转后图像的左上角坐标。

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -m & n & 1 \end{bmatrix} \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ left & top & 1 \end{bmatrix} \quad (6-4)$$

图像旋转变换主要调用 getRotationMatrix2D()函数和 warpAffine()函数实现，绕图像的中心旋转，函数原型如下：

```
M = cv2.getRotationMatrix2D(center, angle, scale)
```

- center 表示旋转中心点，通常设置为(cols/2, rows/2)
- angle 表示旋转角度，正值表示逆时针旋转，坐标原点被定为左上角
- scale 表示比例因子

rotated = cv2.warpAffine(src, M, (cols, rows))

- src 表示原始图像
- M 表示旋转参数，即 getRotationMatrix2D()函数定义的结果
- (cols, rows)表示原始图像的宽度和高度

实现代码如下所示：

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取图片
src = cv2.imread('scenery.png')

#源图像的高、宽 以及通道数
rows, cols, channel = src.shape

#绕图像的中心旋转
```

```
#函数参数：旋转中心 旋转度数 scale
M = cv2.getRotationMatrix2D((cols/2, rows/2), 30, 1)


#函数参数：原始图像 旋转参数 元素图像宽高
rotated = cv2.warpAffine(src, M, (cols, rows))


#显示图像
cv2.imshow("src", src)
cv2.imshow("rotated", rotated)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

显示效果如图 6-9 所示，绕图像中心点逆时针旋转 30 度。

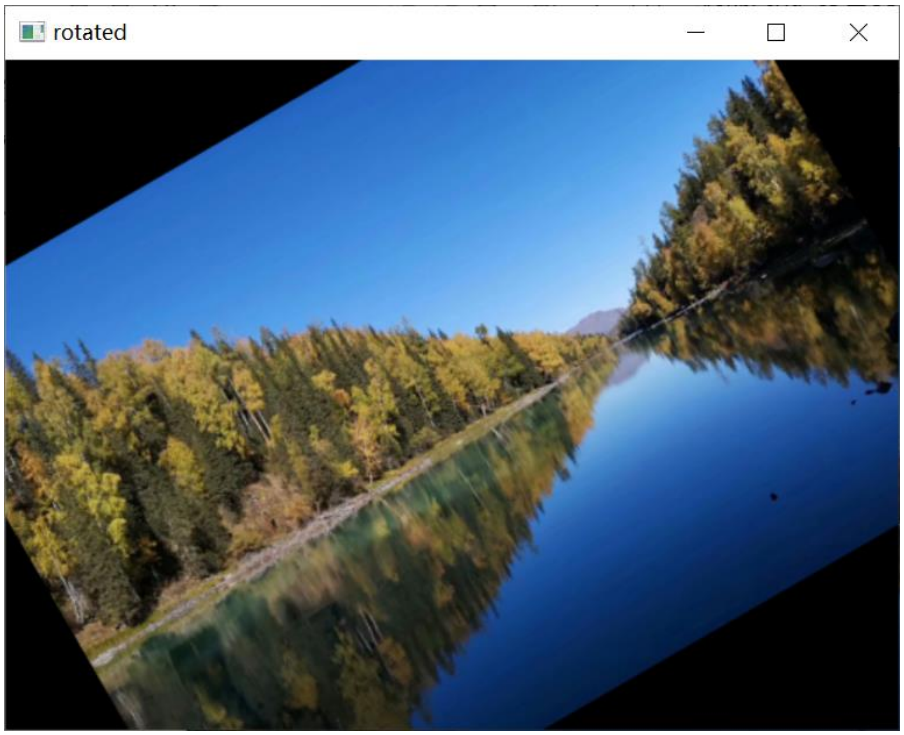


图 6-9 图像逆时针旋转 30 度

5.总结

本章主要讲解 Python 和 OpenCV 的图像几何变换，详细介绍了图像平移、图像缩放和图像旋转，这些知识点也是我们 PC 端或手机端图像处理应用常见的算法，读者可以尝试结合这些应用完成一套图像处理软件。

参考文献：

[1] 冈萨雷斯. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社, 2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社, 2015.

[4]Eastmount. [Python 图像处理] 六.图像缩放、图像旋转、图像翻转与图像平移 [EB/OL]. (2018-09-06). https://blog.csdn.net/Eastmount/article/details/82454335.

[5]Eastmount. [数字图像处理] 六.MFC 空间几何变换之图像平移、镜像、旋转、缩放详解 [EB/OL]. (2015-06-04). https://blog.csdn.net/Eastmount/article/details/46345299.

第 07 篇 图像几何变换之镜像仿射透视

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像几何变换，包括图像平移、图像缩放和图像旋转。这篇文章将继续讲解图像几何变换，包括图像镜像、图像仿射和图像透视。

1.图像镜像

图像镜像是图像旋转变换的一种特殊情况，通常包括垂直方向和水平方向的镜像。水平镜像通常是以原图像的垂直中轴为中心，将图像分为左右两部分进行堆成变换。如图 7-1 所示：

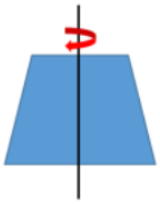


图 7-1 图像水平镜像

垂直镜像通常是以原图像的水平中轴线为中心，将图像划分为上下两部分进行堆成变换的过程，示意图如图 7-2 所示。

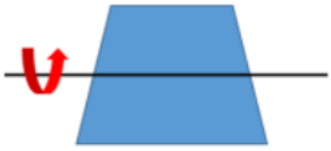


图 7-2 图像垂直镜像

在 Python 中主要调用 OpenCV 的 flip()函数实现图像镜像变换，函数原型如下：

dst = cv2.flip(src, flipCode)

- src 表示原始图像
- flipCode 表示翻转方向，如果 flipCode 为 0，则以 X 轴为对称轴翻转，如果 fliipCode>0 则以 Y 轴为对称轴翻转，如果 flipCode<0 则在 X 轴、Y 轴方向同时翻转。

下面代码是实现三个方向的翻转。

```
# -*- coding:utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图片
img = cv2.imread('scenery.png')
src = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

```
#图像翻转
img1 = cv2.flip(src, 0)     #参数=0 以 X 轴为对称轴翻转
img2 = cv2.flip(src, 1)     #参数>0 以 Y 轴为对称轴翻转
img3 = cv2.flip(src, -1)    #参数<0 以 X 轴和 Y 轴翻转

#显示图形
titles = ['Source', 'Image1', 'Image2', 'Image3']
images = [src, img1, img2, img3]
for i in range(4):
    plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 7-3 所示，图中“Source”为原始图像，“Image1”为以 X 轴为对称轴翻转或垂直镜像，“Image2”为以 Y 轴为对称轴翻转或水平镜像，“Images3”为以 X 轴和 Y 轴翻转。

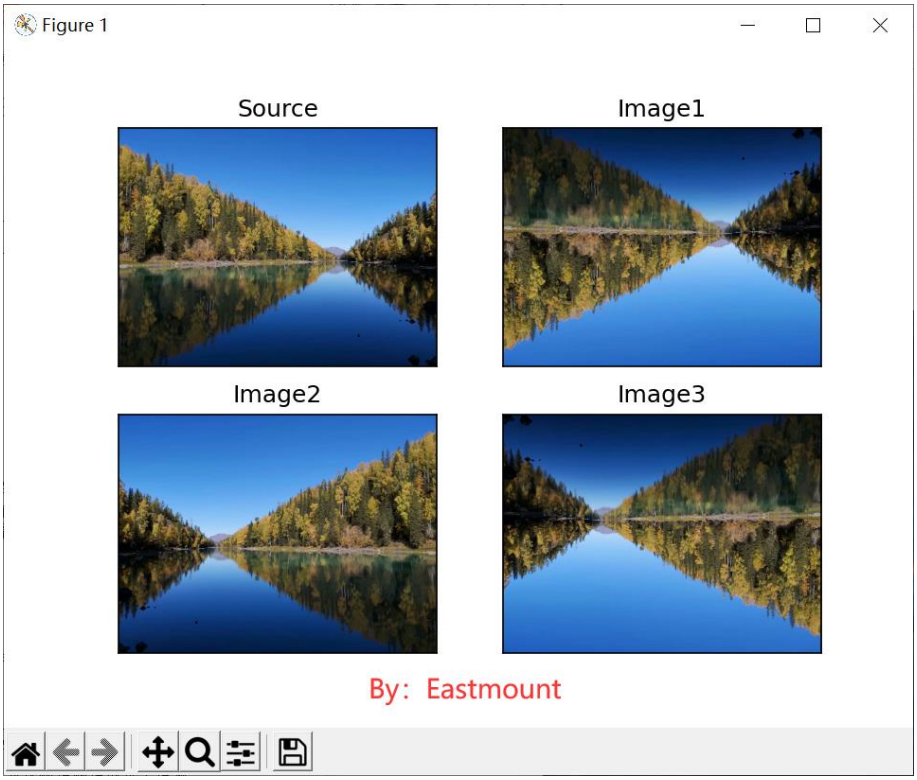


图 7-3 图像镜像变换效果图

2.图像仿射

图像仿射变换又称为图像仿射映射，是指在几何中，一个向量空间进行一次线性变换并接上一个平移，变换为另一个向量空间。通常图像的旋转加上拉升就是图像仿射变换，仿射变换需要一个 M 矩阵实现，但是由于仿射变换比较复杂，很难找到这个 M 矩阵，OpenCV 提供了根据变换前后三个点的对应关系来自动求解 M 的函数：

- cv2.getAffineTransform(pos1,pos2)

其中 pos1 和 pos2 表示变换前后的对应位置关系，输出的结果为仿射矩阵 M，接着使用函数 cv2.warpAffine()实现图像仿射变换。图 7-4 是仿射变换

的前后效果图。

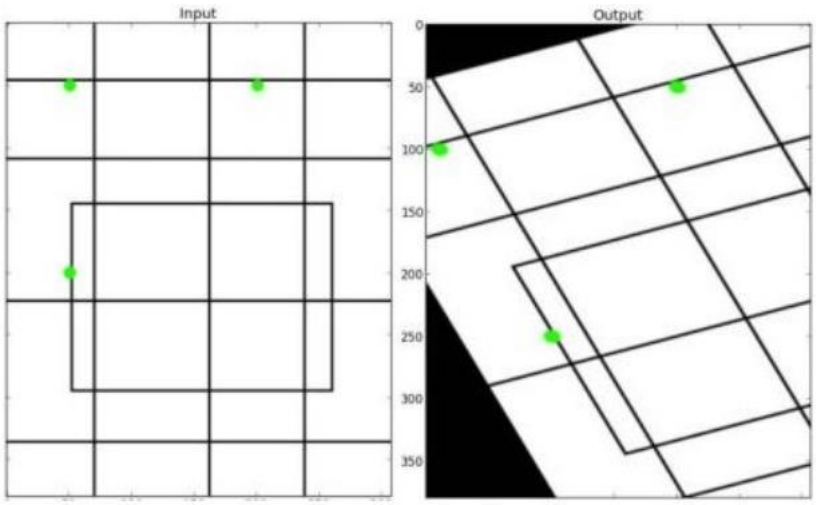


图 7-4 图像仿射变换示例图

图像仿射变换的函数原型如下：

M = cv2.getAffineTransform(pos1,pos2)

- pos1 表示变换前的位置
- pos2 表示变换后的位置

cv2.warpAffine(src, M, (cols, rows))

- src 表示原始图像
- M 表示仿射变换矩阵
- (rows,cols)表示变换后的图像大小，rows 表示行数，cols 表示列数

实现代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
src = cv2.imread('scenery.png')

#获取图像大小
rows, cols = src.shape[:2]

#设置图像仿射变换矩阵
pos1 = np.float32([[50,50], [200,50], [50,200]])
pos2 = np.float32([[10,100], [200,50], [100,250]])
M = cv2.getAffineTransform(pos1, pos2)

#图像仿射变换
result = cv2.warpAffine(src, M, (cols, rows))

#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)
```

```
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 7-5 所示：

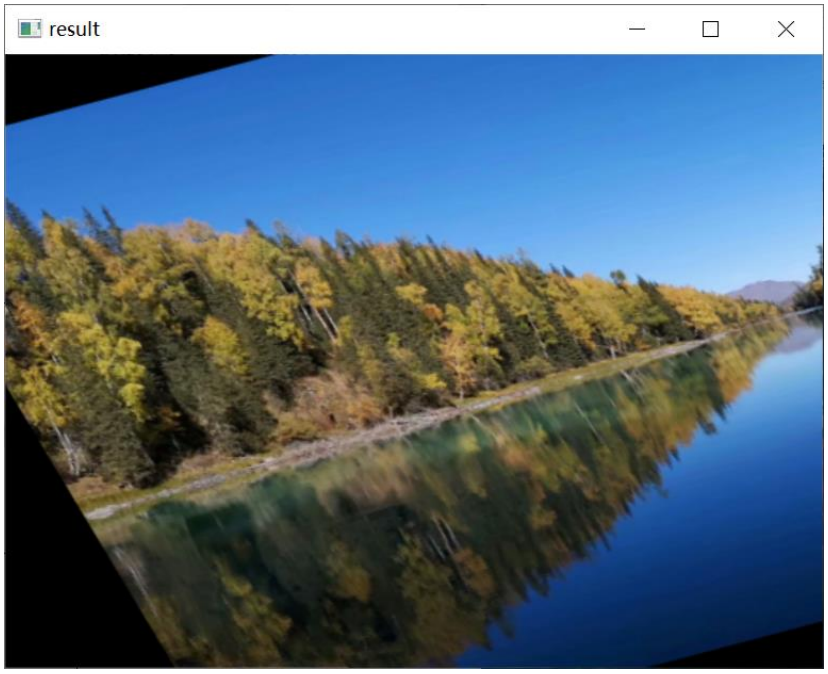


图 7-5 图像仿射变换

3.图像透视

图像透视变换（Perspective Transformation）的本质是将图像投影到一个新的视平面，同理 OpenCV 通过函数 cv2.getPerspectiveTransform(pos1,pos2)构造矩阵 M，其中 pos1 和 pos2 分别表示变换前后的 4 个点对应位置。得到 M 后在通过函数 cv2.warpPerspective(src,M,(cols,rows))进行透视变换。

图像透视变换的函数原型如下：

M = cv2.getPerspectiveTransform(pos1, pos2)

- pos1 表示透视变换前的 4 个点对应位置
- pos2 表示透视变换后的 4 个点对应位置

cv2.warpPerspective(src,M,(cols,rows))

- src 表示原始图像
- M 表示透视变换矩阵
- (rows,cols)表示变换后的图像大小，rows 表示行数，cols 表示列数

假设现在存在一张 A4 纸图像，现在需要通过调用图像透视变换校正图像。

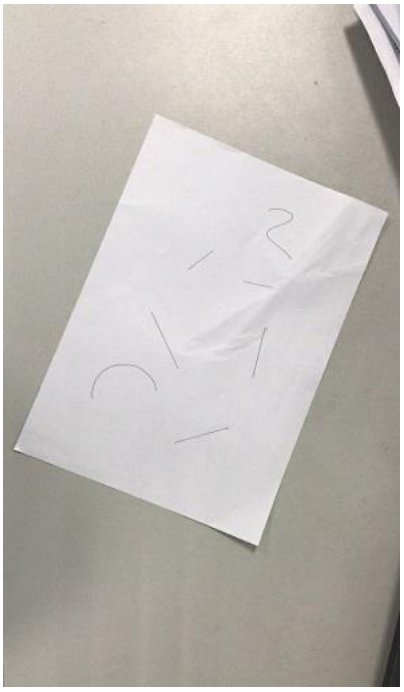


图 7-6 歪曲图像

图像透视变换的校正代码如下所示，代码中 pos1 表示透视变换前 A4 纸的

四个顶点，pos2 表示透视变换后 A4 纸的四个顶点。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
src = cv2.imread('transform.jpg')

#获取图像大小
rows, cols = src.shape[:2]

#设置图像透视变换矩阵
pos1 = np.float32([[114, 82], [287, 156], [8, 322], [216, 333]])
pos2 = np.float32([[0, 0], [188, 0], [0, 262], [188, 262]])
M = cv2.getPerspectiveTransform(pos1, pos2)

#图像透视变换
result = cv2.warpPerspective(src, M, (190, 272))
```

```
#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

最终输出结果如图 7-7 所示，它将图形校正显示。

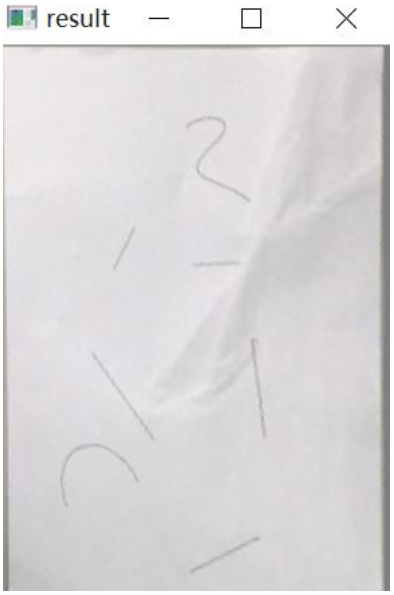


图 7-7 图像透视变换校正

4.总结

本章主要讲解 Python 和 OpenCV 的图像几何变换，详细介绍了图像镜像、图像仿射和图像透视，包括歪曲图像纠正的案例，希望大家喜欢。此外，这

些知识点也是我们 PC 端或手机端图像处理应用常见的算法，读者可以尝试结合这些应用完成一套图像处理软件。

第 08 篇 图像量化处理

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像几何变换，包括图像镜像、图像仿射和图像透视。这篇文章将介绍图像量化处理，即将图像像素点对应亮度的连续变化区间转换为单个特定值的过程。

1.图像量化处理原理

量化（Quantization）旨在将图像像素点对应亮度的连续变化区间转换为单个特定值的过程，即将原始灰度图像的空间坐标幅度值离散化。量化等级越多，图像层次越丰富，灰度分辨率越高，图像的质量也越好；量化等级越少，图像层次欠丰富，灰度分辨率越低，会出现图像轮廓分层的现象，降低了图像的质量。图 8-1 是将图像的连续灰度值转换为 0 至 255 的灰度级的过程[1-3]。

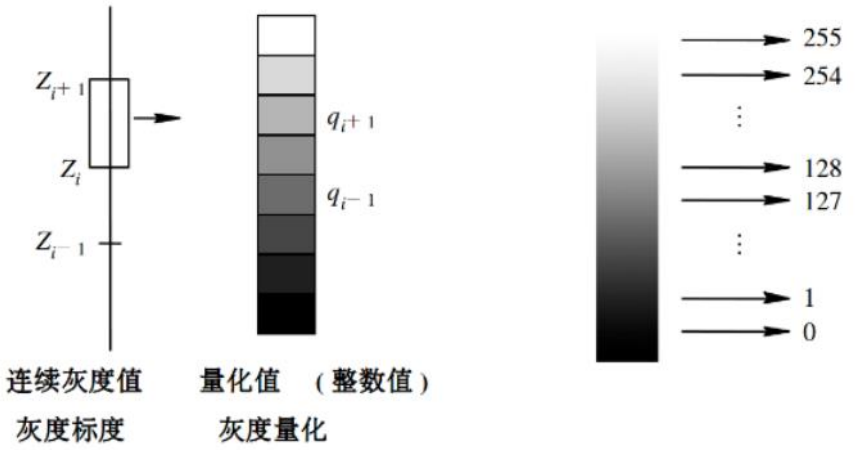


图 8-1 图像量化处理过程

如果量化等级为 2，则将使用两种灰度级表示原始图片的像素（0~255），灰度值小于 128 的取 0，大于等于 128 的取 128；如果量化等级为 4，则将使用四种灰度级表示原始图片的像素，新图像将分层为四种颜色，0~64 区间取 0，64~128 区间取 64，128~192 区间取 128，192~255 区间取 192，依次类推。

图 8-2 是对比不同量化等级的“Lena”图。其中（a）的量化等级为 256，（b）的量化等级为 64，（c）的量化等级为 16，（d）的量化等级为 8，（e）的量化等级为 4，（f）的量化等级为 2。

图 8-2 量化处理对比图

2.图像量化实现

图像量化的实现过程是建立一张临时图片，接着循环遍历原始图像中所有像素点，判断每个像素点应该属于的量化等级，最后将临时图像显示。下面的代码将灰度图像转换为两种量化等级[4]。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('lena-hd.png')
```

```
#获取图像高度和宽度
height = img.shape[0]
width = img.shape[1]

#创建一幅图像
new_img = np.zeros((height, width, 3), np.uint8)

#图像量化操作 量化等级为 2
for i in range(height):
    for j in range(width):
        for k in range(3): #对应 BGR 三分量
            if img[i, j][k] < 128:
                gray = 0
            else:
                gray = 128
            new_img[i, j][k] = np.uint8(gray)

#显示图像
cv2.imshow("src", img)
cv2.imshow("", new_img)
```

```
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 8-3 所示，它将灰度图像划分为两种量化等级。

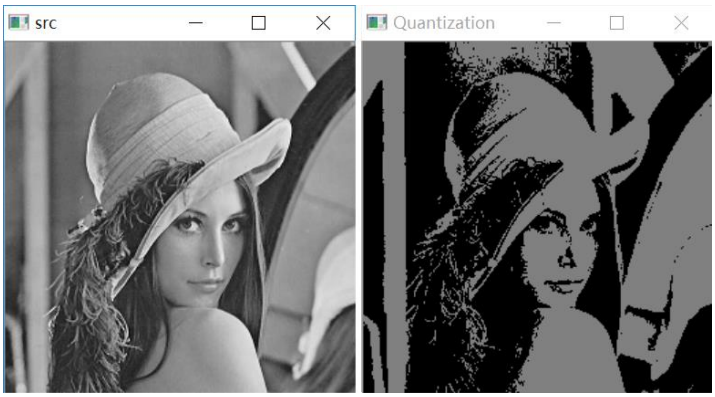


图 8-3 图像量化处理

3.图像量化等级对比

下面的代码分别比较了量化等级为 2、4、8 的量化处理效果[5]。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
```

```
img = cv2.imread('lena-hd.png')


#获取图像高度和宽度
height = img.shape[0]
width = img.shape[1]


#创建一幅图像
new_img1 = np.zeros((height, width, 3), np.uint8)
new_img2 = np.zeros((height, width, 3), np.uint8)
new_img3 = np.zeros((height, width, 3), np.uint8)


#图像量化等级为 2 的量化处理
for i in range(height):
    for j in range(width):
        for k in range(3): #对应 BGR 三分量
            if img[i, j][k] < 128:
                gray = 0
            else:
                gray = 128
            new_img1[i, j][k] = np.uint8(gray)
```

```
#图像量化等级为 4 的量化处理
for i in range(height):
    for j in range(width):
        for k in range(3): #对应 BGR 三分量
            if img[i, j][k] < 64:
                gray = 0
            elif img[i, j][k] < 128:
                gray = 64
            elif img[i, j][k] < 192:
                gray = 128
            else:
                gray = 192
            new_img2[i, j][k] = np.uint8(gray)

#图像量化等级为 8 的量化处理
for i in range(height):
    for j in range(width):
        for k in range(3): #对应 BGR 三分量
            if img[i, j][k] < 32:
                gray = 0
            elif img[i, j][k] < 64:
```

```
                gray = 32
            elif img[i, j][k] < 96:
                gray = 64
            elif img[i, j][k] < 128:
                gray = 96
            elif img[i, j][k] < 160:
                gray = 128
            elif img[i, j][k] < 192:
                gray = 160
            elif img[i, j][k] < 224:
                gray = 192
            else:
                gray = 224
            new_img3[i, j][k] = np.uint8(gray)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图像
titles = ['(a) 原始图像', '(b) 量化-L2', '(c) 量化-L4', '(d) 量化-
L8']
```

```
images = [img, new_img1, new_img2, new_img3]
for i in range(4):
    plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray'),
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 8-4 所示，该代码调用 matplotlib.pyplot 库绘制了四幅图像，其中（a）表示原始图像，（b）表示等级为 2 的量化处理，（c）表示等级为 4 的量化处理，（d）表示等级为 8 的量化处理。

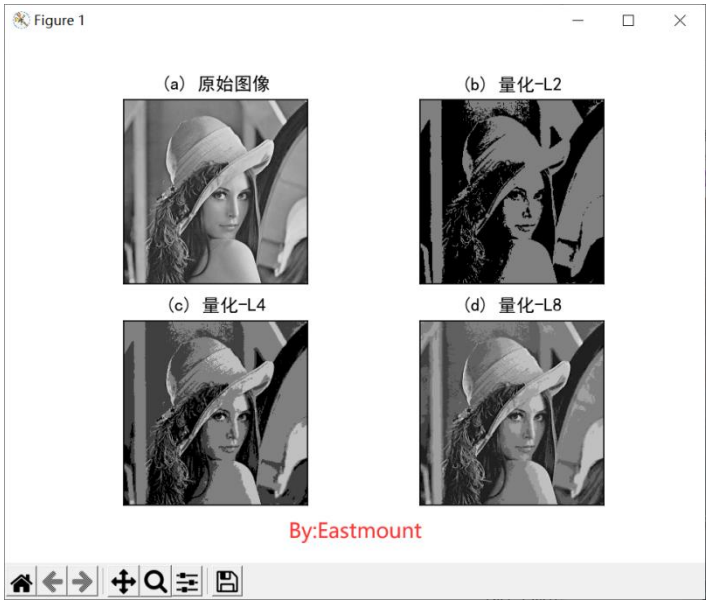


图 8-4 图像量化处理对比图

4.K-Means 聚类实现量化处理

除了通过对像素进行统计比较量化处理，还可以根据像素之间的相似性进行

聚类处理。这里补充一个基于 K-Means 聚类算法的量化处理过程，它能够将彩色图像 RGB 像素点进行颜色分割和颜色量化。此外，该部分只是带领读者简单认识该方法，更多 K-Means 聚类的知识将在图像分割文章中进行详细叙述[6]。

```
# coding: utf-8
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像
img = cv2.imread('luo.png')

#图像二维像素转换为一维
data = img.reshape((-1,3))
data = np.float32(data)

#定义中心（type,max_iter,epsilon)
criteria = (cv2.TERM_CRITERIA_EPS +
            cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
```

```
#设置标签
flags = cv2.KMEANS_RANDOM_CENTERS


#K-Means 聚类 聚集成 4 类
compactness, labels, centers = cv2.kmeans(data, 8, None,
criteria, 10, flags)



#图像转换回 uint8 二维类型
centers = np.uint8(centers)
res = centers[labels.flatten()]
dst = res.reshape((img.shape))


#图像转换为 RGB 显示
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)



#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']
```

```
#显示图像
titles = ['原始图像', '聚类量化 K=8']
images = [img, dst]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray'),
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 8-5 所示，它通过 K-Means 聚类算法将彩色人物图像的灰度聚集成八种颜色。核心代码如下：

- cv2.kmeans(data, 8, None, criteria, 10, flags)

图 8-5 K-Means 量化处理彩色图像

5.总结

本文主要讲解了图像的量化处理，从基本概念到操作，再到扩展进行全方位讲解，并且补充了基于 K-Means 聚类算法的量化处理案例。该部分的知识点能够将生活中的图像转换为数字图像，更好地为后续的图像处理提供帮助。

参考资料：

[1] 冈萨雷斯著. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] yunfung. 数字图像基础之图像取样和量化(Image Sampling and Quantization) [EB/OL]. (2017-04-23). https://www.cnblogs.com/yunfung/p/6753337.html.

[3] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[4] Eastmount. [Python 图像处理] 三十.图像量化及采样处理万字详细总结[EB/OL]. (2020-11-10). https://blog.csdn.net/Eastmount/article/details/109605161.

[5] Eastmount. [数字图像处理] 三.MFC 实现图像灰度、采样和量化功能详解[EB/OL]. (2015-05-28). https://blog.csdn.net/eastmount/article/details/46010637.

[6] 杨秀璋，颜娜. Python 网络数据爬取及分析从入门到精通（分析篇）[M]. 北京：北京航天航空大学出版社, 2018.

第 09 篇 图像采样处理

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像量化处理。这篇文章将详细讲解图像采样处理，包括原理知识、代码实现和局部马赛克处理。

1.图像采样处理原理

图像采样（Image Sampling）处理是将一幅连续图像在空间上分割成 M×N 个网格，每个网格用一个亮度值或灰度值来表示，其示意图如图 9-1 所示。

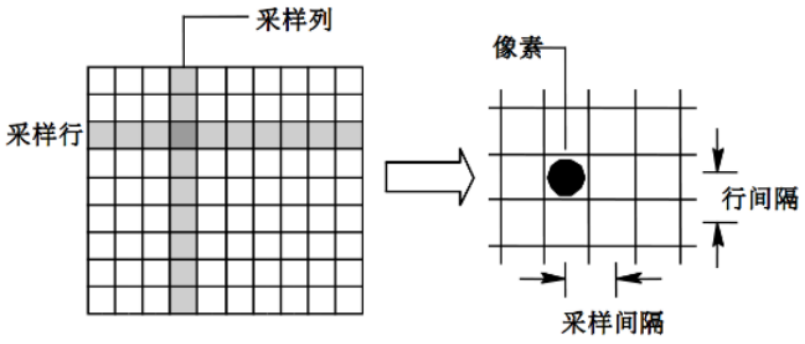


图 9-1 图像采样原理图

图像采样的间隔越大，所得图像像素数越少，空间分辨率越低，图像质量越

差，甚至出现马赛克效应；相反，图像采样的间隔越小，所得图像像素数越多，空间分辨率越高，图像质量越好，但数据量会相应的增大。图 9-2 展示了不同采样间隔的“Lena”图，其中图(a)为原始图像，图(b)为 128×128 的图像采样效果，图(c)为 64×64 的图像采样效果，图(d)为 32×32 的图像采样效果，图(e)为 16×16 的图像采样效果，图(f)为 8×8 的图像采样效果[1-3]。

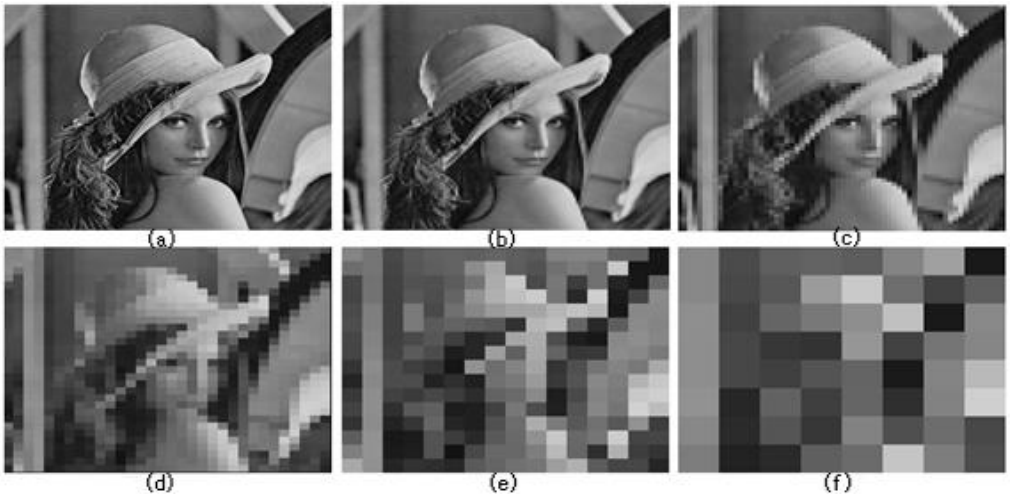


图 9-2 采样处理对比图

2.图像采样实现

下面讲述 Python 图像采样处理相关代码操作。其核心流程是建立一张临时图片，设置需要采样的区域大小（如 16×16），接着循环遍历原始图像中所有像素点，采样区域内的像素点赋值相同（如左上角像素点的灰度值），最终实现图像采样处理。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('lena-hd.png')


#获取图像高度和宽度
height = img.shape[0]
width = img.shape[1]


#采样转换成 16*16 区域
numHeight = int(height/16)
numWidth = int(width/16)


#创建一幅图像
new_img = np.zeros((height, width, 3), np.uint8)


#图像循环采样 16*16 区域
for i in range(16):
    #获取 Y 坐标
    y = i*numHeight
```

```
    for j in range(16):
        #获取 X 坐标
        x = j*numWidth
        #获取填充颜色 左上角像素点
        b = img[y, x][0]
        g = img[y, x][1]
        r = img[y, x][2]

        #循环设置小区域采样
        for n in range(numHeight):
            for m in range(numWidth):
                new_img[y+n, x+m][0] = np.uint8(b)
                new_img[y+n, x+m][1] = np.uint8(g)
                new_img[y+n, x+m][2] = np.uint8(r)

#显示图像
cv2.imshow("src", img)
cv2.imshow("Sampling", new_img)

#等待显示
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

其输出结果如图 9-3 所示，它将灰度图像采样成 16×16 的区域。

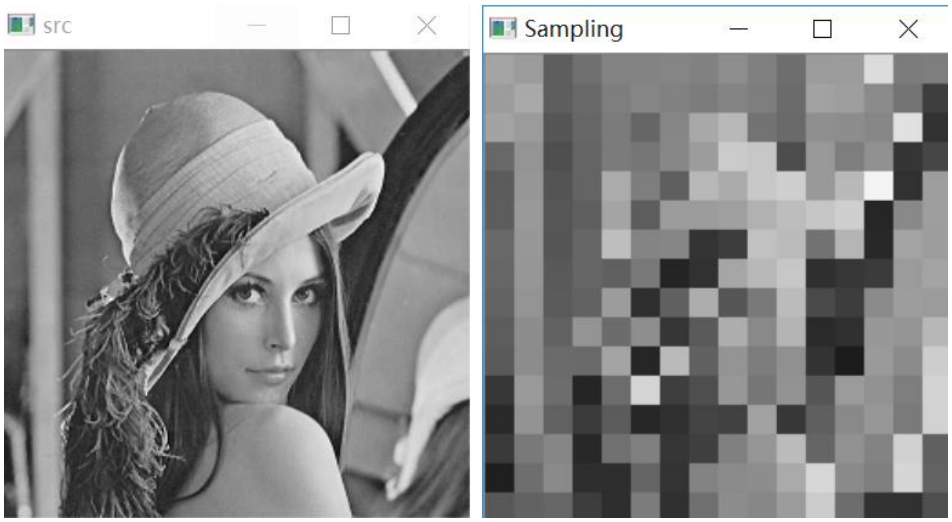


图 9-3 图像 16×16 采样处理

同样，可以对彩色图像进行采样处理，下面的代码将“小珞珞”的图像采样处理成 8×8 的马赛克区域。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('luo.png')
```

```
#获取图像高度和宽度
height = img.shape[0]
width = img.shape[1]

#采样转换成 8×8 区域
numHeight = int(height/8)
numwidth = int(width/8)

#创建一幅图像
new_img = np.zeros((height, width, 3), np.uint8)

#图像循环采样 8*8 区域
for i in range(8):
    #获取 Y 坐标
    y = i*numHeight
    for j in range(8):
        #获取 X 坐标
        x = j*numwidth
        #获取填充颜色 左上角像素点
        b = img[y, x][0]
        g = img[y, x][1]
```

```
            r = img[y, x][2]


            #循环设置小区域采样
            for n in range(numHeight):
                for m in range(numwidth):
                    new_img[y+n, x+m][0] = np.uint8(b)
                    new_img[y+n, x+m][1] = np.uint8(g)
                    new_img[y+n, x+m][2] = np.uint8(r)


#显示图像
cv2.imshow("src", img)
cv2.imshow("Sampling", new_img)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 9-4 所示，它将彩色图像采样成 8×8 的区域。

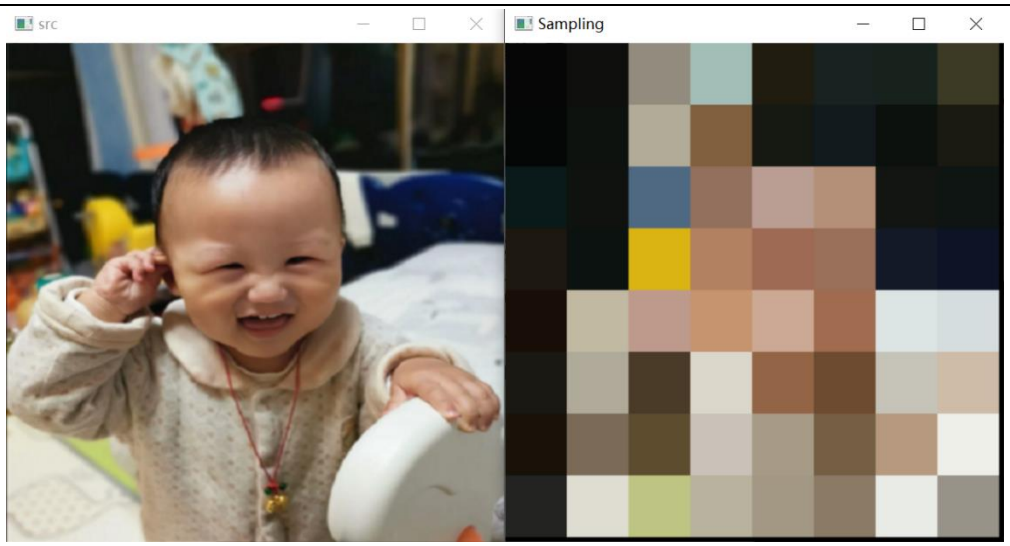


图 9-4 彩色图像 8×8 采样处理

但上述代码存在一个问题，当图像的长度和宽度不能被采样区域整除时，输出图像的最右边和最下边的区域没有被采样处理。这里推荐读者做个求余运算，将不能整除部分的区域也进行相应的采样处理。

3.图像局部采样处理

前面讲述的代码是对整幅图像进行采样处理，那么如何对图像的局部区域进行马赛克处理呢？下面的代码就实现了该功能。当鼠标按下时，它能够给鼠标拖动的区域打上马赛克，并按下“s”键保存图像至本地。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
#读取原始图像
im = cv2.imread('luo.png', 1)


#设置鼠标左键开启
en = False


#鼠标事件
def draw(event, x, y, flags, param):
    global en
    #鼠标左键按下开启 en 值
    if event==cv2.EVENT_LBUTTONDOWN:
        en = True
    #鼠标左键按下并且移动
    elif event==cv2.EVENT_MOUSEMOVE and
flags==cv2.EVENT_LBUTTONDOWN:
        #调用函数打马赛克
        if en:
            drawMask(y,x)
        #鼠标左键弹起结束操作
        elif event==cv2.EVENT_LBUTTONUP:
```

```
                en = False


#图像局部采样操作
def drawMask(x, y, size=10):
    #size*size 采样处理
    m = int(x / size * size)
    n = int(y / size * size)
    print(m, n)
    #10*10 区域设置为同一像素值
    for i in range(size):
        for j in range(size):
            im[m+i][n+j] = im[m][n]


#打开对话框
cv2.namedWindow('image')


#调用 draw 函数设置鼠标操作
cv2.setMouseCallback('image', draw)


#循环处理
while(1):
```

```
        cv2.imshow('image', im)
        #按 ESC 键退出
        if cv2.waitKey(10)&0xFF==27:
            break
        #按 s 键保存图片
        elif cv2.waitKey(10)&0xFF==115:
            cv2.imwrite('sava.png', im)

#退出窗口
cv2.destroyAllWindows()
```

其输出结果如图 9-5 所示，它将人物的脸部进行马赛克处理。

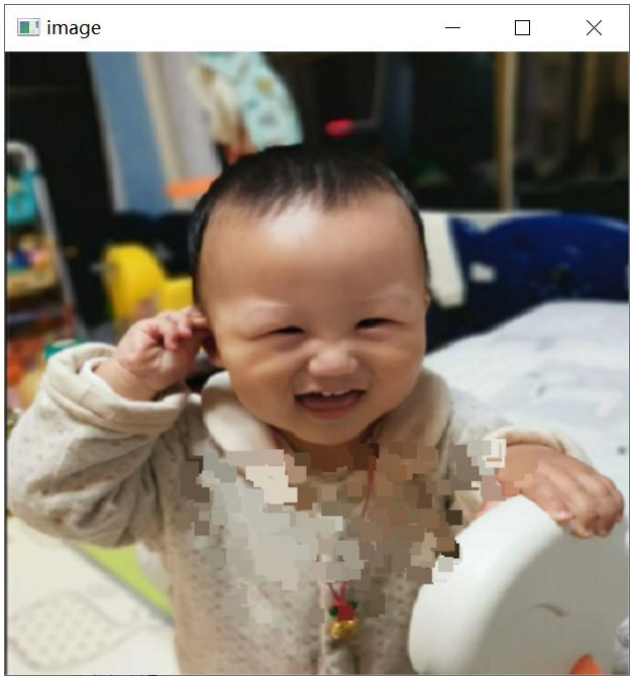


图 9-5 局部马赛克处理

4.总结

本文主要讲解了图像的采样处理，从基本概念到操作，再到扩展进行全方位讲解，并且补充了局部马赛克采样处理案例。该部分的知识点能够将生活中的图像转换为数字图像，更好地为后续的图像处理提供帮助。

参考文献：

[1] 冈萨雷斯著. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] Eastmount. [Python 图像处理] 三十.图像量化及采样处理万字详细总结[EB/OL]. (2020-11-10). https://blog.csdn.net/Eastmount/article/details/109605161.

[3] Eastmount. [数字图像处理] 三.MFC 实现图像灰度、采样和量化功能详解[EB/OL]. (2015-05-28). https://blog.csdn.net/eastmount/article/details/46010637.

第 10 篇 图像金字塔之图像向上取样和向下取样

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像采样处理。这篇文章将详细讲解图像金字塔，包括图像向上取样和向下取样，希望对大家有所帮助。

1.图像金字塔原理

上一篇文章讲解的图像采样处理可以降低图像的大小，本文将补充图像金字塔知识，了解专门用于图像向上采样和向下采样的 pyrUp()和 pyrDown()函数。

图像金字塔是指由一组图像且不同分别率的子图集合，它是图像多尺度表达的一种，以多分辨率来解释图像的结构，主要用于图像的分割或压缩。一幅图像的金字塔是一系列以金字塔形状排列的分辨率逐步降低，且来源于同一张原始图的图像集合。如图 10-1 所示，它包括了四层图像，将这一层一层的图像比喻成金字塔。图像金字塔可以通过梯次向下采样获得，直到达到某个终止条件才停止采样，在向下采样中，层级越高，则图像越小，分辨率越低[1-2]。

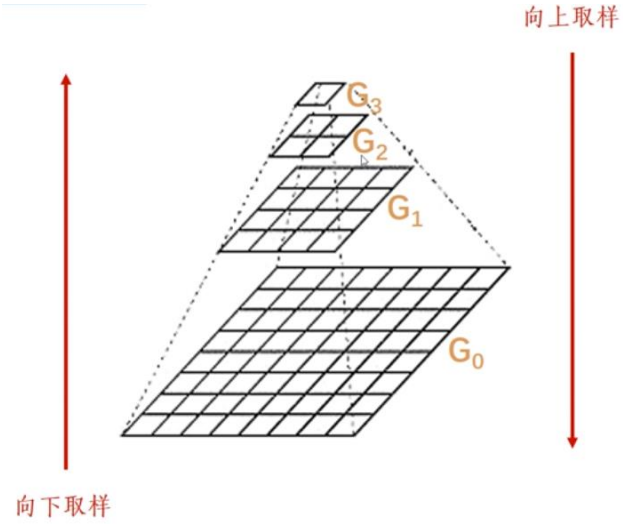


图 10-1 图像金字塔原理

生成图像金字塔主要包括两种方式：

- 向下取样
- 向上取样

在图中，将图像 G_0 转换为 G_1、G_2、G_3，图像分辨率不断降低的过程称为向下取样；将 G_3 转换为 G_2、G_1、G_0，图像分辨率不断增大的过程称为向上取样。

2.图像向上取样

在图像向上取样是由小图像不断放图像的过程。它将图像在每个方向上扩大为原图像的 2 倍，新增的行和列均用 0 来填充，并使用与“向下取样”相同的卷积核乘以 4，再与放大后的图像进行卷积运算，以获得“新增像素”的新值。如图 10-2 所示，它在原始像素 45、123、89、149 之间各新增了一行和一列值为 0 的像素。

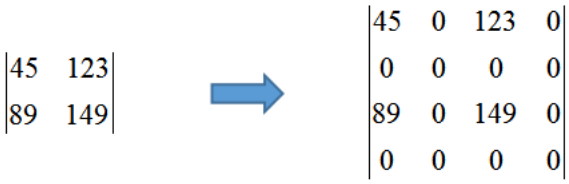


图 10-2 图像向上取样新增像素

在 OpenCV 中，向上取样使用的函数为 pyrUp()，其原型如下所示：

- dst = pyrUp(src[, dst[, dstsize[, borderType]]])
  - src 表示输入图像，
  - dst 表示输出图像，和输入图像具有一样的尺寸和类型
  - dstsize 表示输出图像的大小，默认值为 Size()
  - borderType 表示像素外推方法，详见 cv::bordertypes

向上取样的代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('lena-small.png')


#图像向上取样
```

```
r = cv2.pyrUp(img)


#显示图像
cv2.imshow('original', img)
cv2.imshow('PyrUp', r)
cv2.waitKey()
cv2.destroyAllWindows()
```

输出结果如图 10-3 所示，它将原始图像扩大为原图像的四倍。

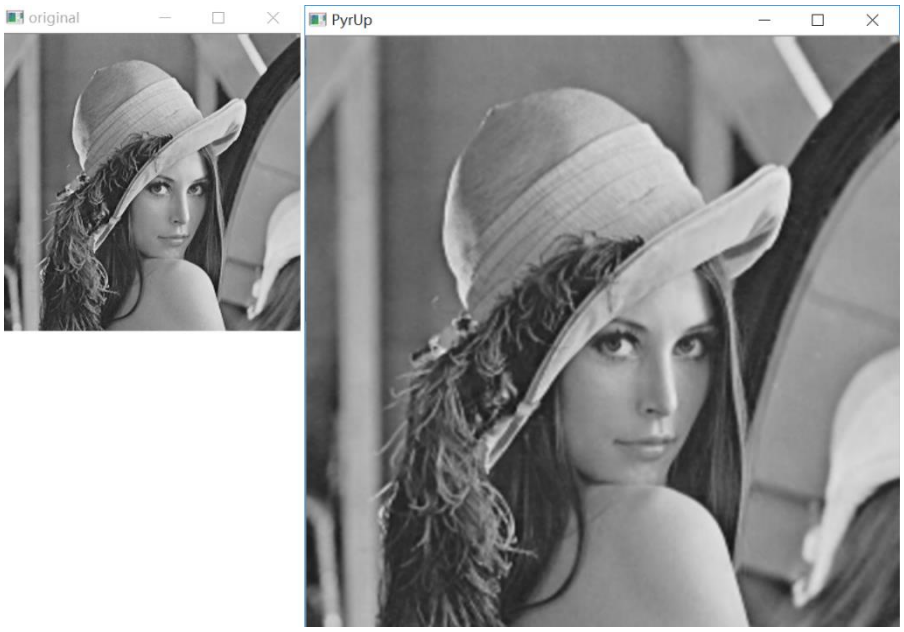


图 10-3 向上取样

多次向上取样的代码如下。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('lena-small.png')


#图像向上取样
r1 = cv2.pyrUp(img)
r2 = cv2.pyrUp(r1)
r3 = cv2.pyrUp(r2)


#显示图像
cv2.imshow('original', img)
cv2.imshow('PyrUp1', r1)
cv2.imshow('PyrUp2', r2)
cv2.imshow('PyrUp3', r3)
cv2.waitKey()
cv2.destroyAllWindows()
```

输出结果如图 10-4 所示，每次向上取样均为上次图像的四倍，但图像的清晰度会降低。

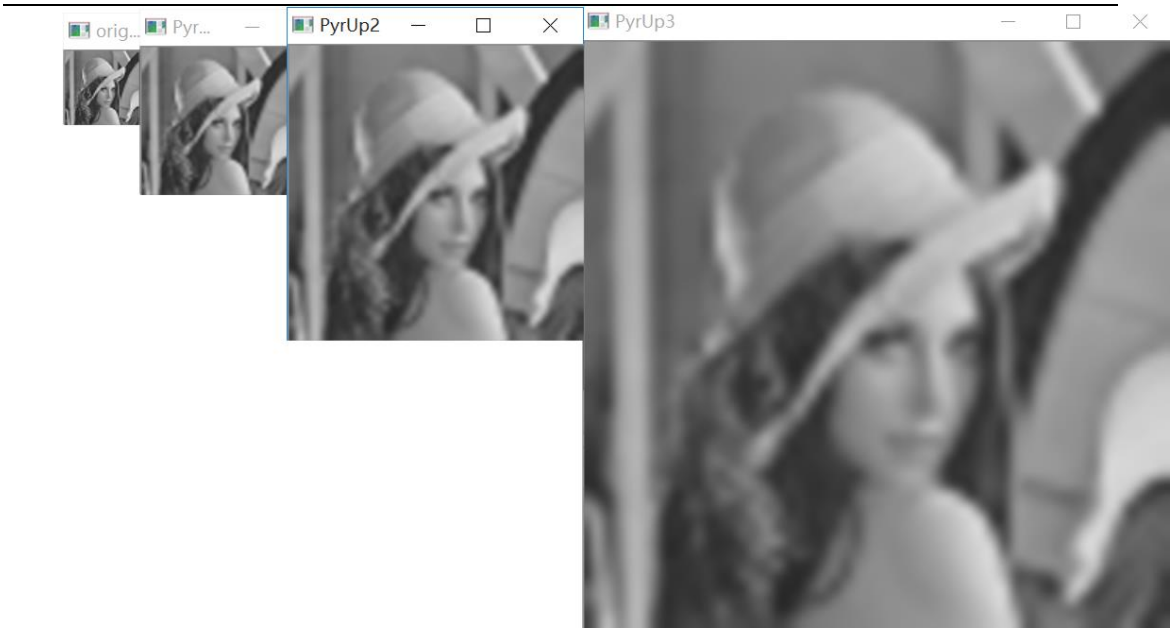


图 10-4 多次向上取样

3.图像向下取样

在图像向下取样中，使用最多的是高斯金字塔。它将对图像 G_i 进行高斯核卷积，并删除原图中所有的偶数行和列，最终缩小图像。其中，高斯核卷积运算就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其他像素值（权重不同）经过加权平均后得到。常见的 3×3 和 5×5 高斯核如下：

$$K(3,3)=\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \tag{10-1}$$

$$K(5,5)=\frac{1}{273}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix} \tag{10-2}$$

高斯核卷积让临近中心的像素点具有更高的重要度，对周围像素计算加权平均值，如图 10-5 所示，其中心位置权重最高为 0.4。

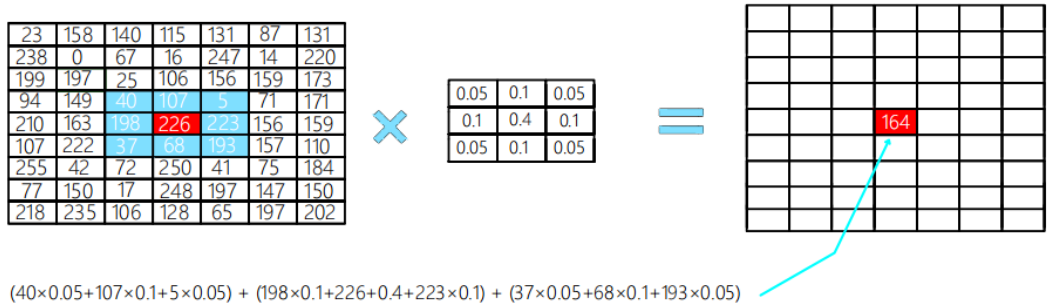


图 10-5 高斯滤波处理过程

在 OpenCV 中，向下取样使用的函数为 pyrDown()，其原型如下所示：

- dst = pyrDown(src[, dst[, dstsize[, borderType]]])
  - src 表示输入图像，
  - dst 表示输出图像，和输入图像具有一样的尺寸和类型
  - dstsize 表示输出图像的大小，默认值为 Size()
  - borderType 表示像素外推方法，详见 cv::bordertypes

向下取样的代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
```

```
img = cv2.imread('nv.png')


#图像向下取样
r = cv2.pyrDown(img)


#显示图像
cv2.imshow('original', img)
cv2.imshow('PyrDown', r)
cv2.waitKey()
cv2.destroyAllWindows()
```

输出结果如图 10-6 所示，它将原始图像压缩成原图的四分之一。

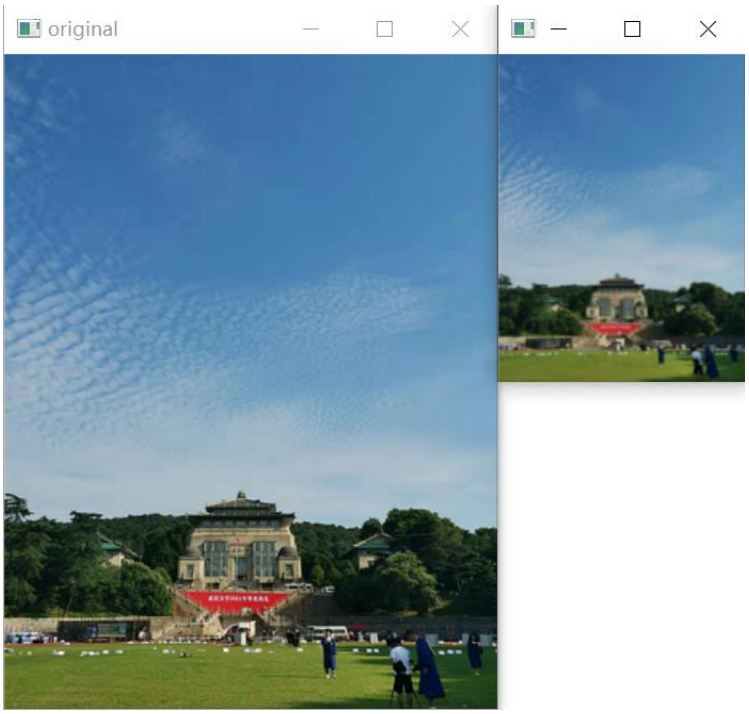


图 10-6 向下一次取样

多次向下取样的代码如下。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像
img = cv2.imread('nv.png')

#图像向下取样
r1 = cv2.pyrDown(img)
r2 = cv2.pyrDown(r1)
r3 = cv2.pyrDown(r2)

#显示图像
cv2.imshow('original', img)
cv2.imshow('PyrDown1', r1)
cv2.imshow('PyrDown2', r2)
cv2.imshow('PyrDown3', r3)
cv2.waitKey()
```

```
cv2.destroyAllWindows()
```

输出结果如图 10-7 所示，每次向下取样均为上次的四分之一，并且图像的清晰度会降低。

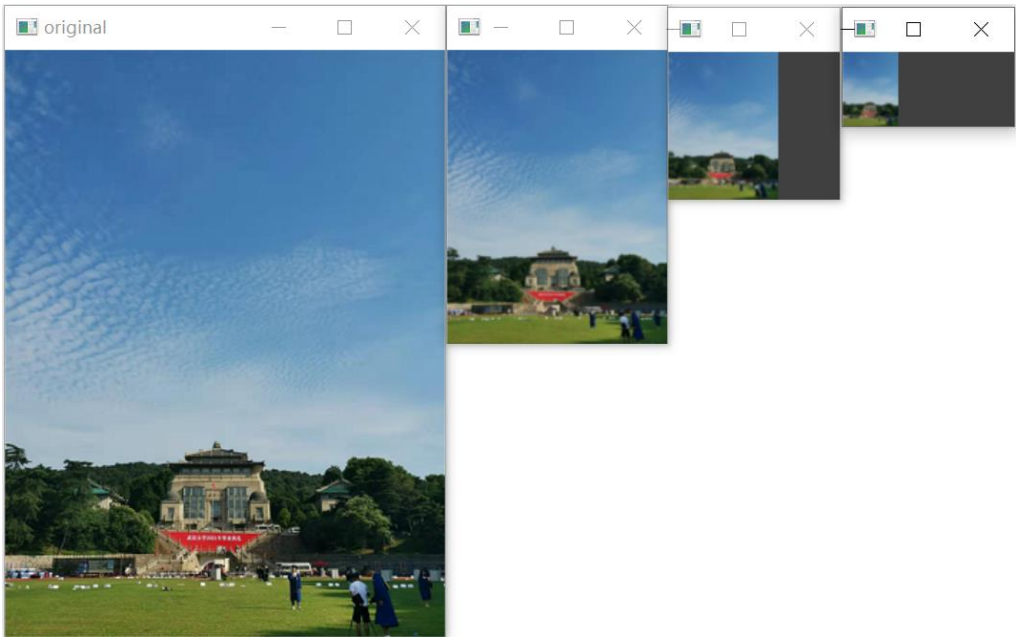


图 10-7 多次向下取样

4.总结

本文主要讲解图像金字塔处理，包括图像向上取样和向下取样。需要注意，向上取样放大后的图像比原始图像要模糊，而每次向下取样会删除偶数行和列，它会不停地丢失图像的信息。此外，向上采样和向下采样不是互逆的操作，经过两种操作后，是无法恢复原始图像的。

写到这里，第一部分图像处理基础知识介绍完毕。该部分包括 10 篇文章，分别是图像处理基础知识和 OpenCV 配置；OpenCV 入门详解（显示读取修改及保存图像）；OpenCV 绘制各类几何图形；图像算术与逻辑运算详解；图

像融合处理和 ROI 区域绘制；图像几何变换（平移、缩放、旋转、镜像、仿射、透视）；图像量化处理；图像采样处理和图像金字塔处理。这些基础知识是我们开启图像处理系列分享的基础，希望大家一定要主动完成文章中的所有代码。同时，如果您是一名初学者或学生，可以尝试学习该系列文章，既能提升您的编程兴趣，又能帮助您完成实战性的技能提升。接下来我们进入第二部分——图像运算和图像增强，进一步深入介绍图像处理相关知识点。

第二部分 图像运算和图像增强

第 11 篇 图像点运算之图像灰度化处理

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

第二部分将讲解图像运算和图像增强，第一篇文章是图像点运算的灰度化处理知识，包括各种灰度算法的实现，以及灰度线性变换和灰度非线性变换。

1.图像点运算概念

图像点运算（Point Operation）指对于一幅输入图像，将产生一幅输出图像，输出图像的每个像素点的灰度值由输入像素点决定。点运算实际上是灰度到灰度的映射过程，通过映射变换来达到增强或者减弱图像的灰度。还可以对图像进行求灰度直方图、线性变换、非线性变换以及图像骨架的提取。它与相邻的像素之间没有运算关系，是一种简单和有效的图像处理方法[1]。

图像的灰度变换可以通过有选择的突出图像感兴趣的特征或者抑制图像中不需要的特征，从而改善图像的质量，凸显图像的细节，提高图像的对比度。它也能有效地改变图像的直方图分布，使图像的像素值分布更为均匀[2-3]。它在实际中有很多的应用：

- 光度学标定

- 对比度增强
- 对比度扩展
- 显示标定
- 轮廓线确定

设输入图像为 A(x,y)，输出图像为 B(x,y)，则点运算可以表示为：

$$B(x,y) = f[A(x,y)] \tag{11-1}$$

图像点运算与几何运算存在差别，不会改变图像内像素点之间的空间位置关系。同时与局部（领域）运算也存在差别，输入像素和输出像素一一对应。

2.图像灰度化处理

图像灰度化是将一幅彩色图像转换为灰度化图像的过程。彩色图像通常包括 R、G、B 三个分量，分别显示出红绿蓝等各种颜色，灰度化就是使彩色图像的 R、G、B 三个分量相等的过程。灰度图像中每个像素仅具有一种样本颜色，其灰度是位于黑色与白色之间的多级色彩深度，灰度值大的像素点比较亮，反之比较暗，像素值最大为 255（表示白色），像素值最小为 0（表示黑色）。

假设某点的颜色由RGB(R,G,B)组成，常见灰度处理算法如表11-1所示：

表 11-1 灰度处理算法

算法名称	算法公式
最大值灰度处理	Gray=max(R, G, B)
浮点灰度处理	Gray=R×0.3+G×0.59+B×0.11
整数灰度处理	Gray=(R×30+G×59+B×11)/100

移位灰度处理	Gray=(R×28+G×151+B×77)>>8
平均灰度处理	Gray=(R+G+B)/3
加权平均灰度处理	Gray=R×0.299+G×0.587+B×0.144

表 11-1 中 Gray 表示灰度处理之后的颜色，然后将原始 RGB(R,G,B)颜色均匀地替换成新颜色 RGB(Gray,Gray,Gray)，从而将彩色图片转化为灰度图像。一种常见的方法是将 RGB 三个分量求和再取平均值，但更为准确的方法是设置不同的权重，将 RGB 分量按不同的比例进行灰度划分。比如人类的眼睛感官蓝色的敏感度最低，敏感最高的是绿色，因此将 RGB 按照 0.299、0.587、0.144 比例加权平均能得到较合理的灰度图像，如公式 11-2 所示[4-6]。

$$Gray = R\times 0.299 + G\times 0.587 + B\times 0.114 \quad (11-2)$$

在日常生活中，我们看到的大多数彩色图像都是 RGB 类型，但是在图像处理过程中，常常需要用到灰度图像、二值图像、HSV、HSI 等颜色，OpenCV 提供了 cvtColor()函数实现这些功能。其函数原型如下所示：

dst = cv2.cvtColor(src, code[, dst[, dstCn]])

- src 表示输入图像，需要进行颜色空间变换的原图像
- dst 表示输出图像，其大小和深度与 src 一致
- code 表示转换的代码或标识
- dstCn 表示目标图像通道数，其值为 0 时，则有 src 和 code 决定

该函数的作用是将一个图像从一个颜色空间转换到另一个颜色空间，其中，RGB 是指 Red、Green 和 Blue，一副图像由这三个通道（channel）构成；

Gray 表示只有灰度值一个通道；HSV 包含 Hue（色调）、Saturation（饱和度）和 Value（亮度）三个通道。

在 OpenCV 中，常见的颜色空间转换标识包括 CV_BGR2BGRA、CV_RGB2GRAY、CV_GRAY2RGB、CV_BGR2HSV、CV_BGR2XYZ、CV_BGR2HLS 等。下面是调用 cvtColor()函数将图像进行灰度化处理的代码。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取原始图片
src = cv2.imread('luo.png')

#图像灰度化处理
grayImage = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", grayImage)
```

```
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 11-1 所示，左边是彩色的“小珞珞”原图，右边是将彩色图像进行灰度化处理之后的灰度图。其中，灰度图将一个像素点的三个颜色变量设置为相等（R=G=B），此时该值称为灰度值。

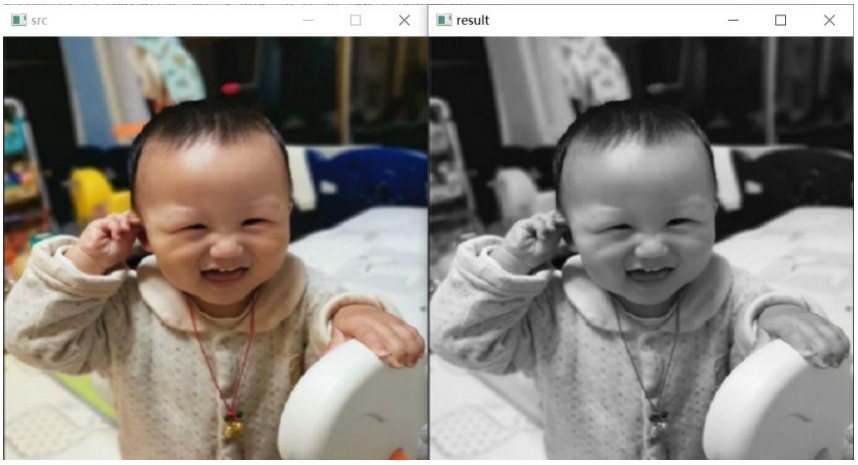


图 11-1 图像灰度化处理

同样，可以调用如下核心代码将彩色图像转换为 HSV 颜色空间，其输出结果如图 11-2 所示。

- grayImage = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)

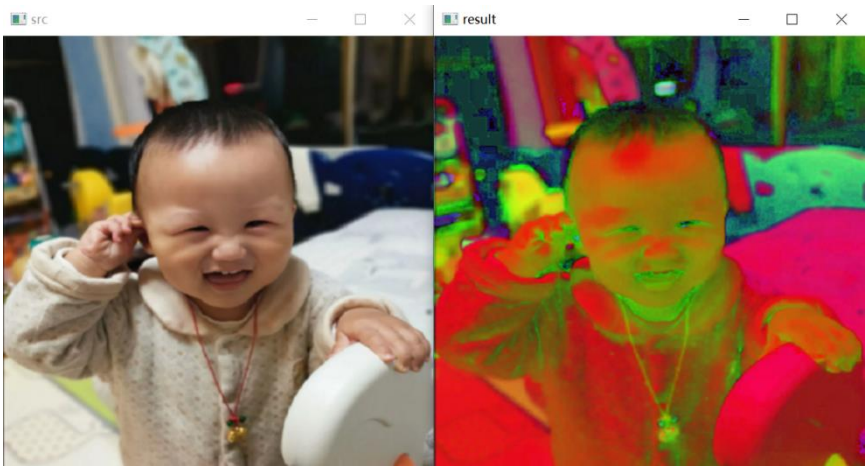


图 11-2 图像 HSV 转换

下面代码对比了九种常见的颜色空间，包括 BGR、RGB、GRAY、HSV、YCrCb、HLS、XYZ、LAB 和 YUV，并循环显示处理后的图像。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像
img_BGR = cv2.imread('luo.png')

img_RGB = cv2.cvtColor(img_BGR, cv2.COLOR_BGR2RGB)  #BGR 转换为 RGB
img_GRAY = cv2.cvtColor(img_BGR,
```

```
cv2.COLOR_BGR2GRAY)    #灰度化处理
img_HSV          =          cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2HSV)      #BGR 转 HSV
img_YCrCb          =          cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2YCrCb) #BGR 转 YCrCb
img_HLS          =          cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2HLS)      #BGR 转 HLS
img_XYZ          =          cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2XYZ)      #BGR 转 XYZ
img_LAB          =          cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2LAB)      #BGR 转 LAB
img_YUV          =          cv2.cvtColor(img_BGR,
cv2.COLOR_BGR2YUV)      #BGR 转 YUV

#调用 matplotlib 显示处理结果
titles = ['BGR', 'RGB', 'GRAY', 'HSV', 'YCrCb', 'HLS', 'XYZ',
'LAB', 'YUV']
images = [img_BGR, img_RGB, img_GRAY, img_HSV,
img_YCrCb,
          img_HLS, img_XYZ, img_LAB, img_YUV]
for i in range(9):
```

```
    plt.subplot(3, 3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 11-3 所示：

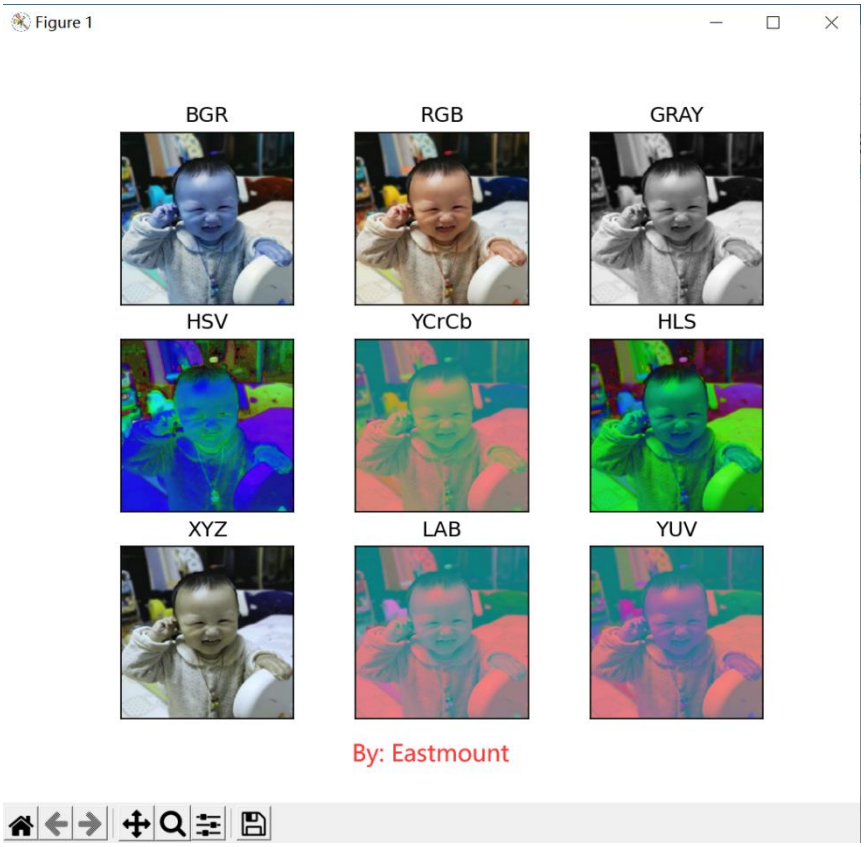


图 11-3 图像九种颜色空间相互转换

3.基于像素操作的图像灰度化处理

前面讲述了调用 OpenCV 中 cvtColor()函数实现图像灰度化的处理，接下来讲解基于像素操作的图像灰度化处理方法，主要是最大值灰度处理、平均灰

度处理和加权平均灰度处理方法。

（1）最大值灰度处理方法

该方法的灰度值等于彩色图像 R、G、B 三个分量中的最大值，公式如下：

$$gray(i,j) = \max(R(i,j), G(i,j), B(i,j)) \quad (11\text{-}3)$$

其方法灰度化处理后的灰度图亮度很高，实现代码如下。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像
img = cv2.imread('luo.png')

#获取图像高度和宽度
height = img.shape[0]
width = img.shape[1]

#创建一幅图像
grayimg = np.zeros((height, width, 3), np.uint8)
```

```
#图像最大值灰度处理
for i in range(height):
    for j in range(width):
        #获取图像 R G B 最大值
        gray = max(img[i,j][0], img[i,j][1], img[i,j][2])
        #灰度图像素赋值  gray=max(R,G,B)
        grayimg[i,j] = np.uint8(gray)

#显示图像
cv2.imshow("src", img)
cv2.imshow("gray", grayimg)
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 11-4 所示，其处理效果的灰度偏亮。

图 11-4 图像最大值灰度处理方法效果图

（2）平均灰度处理方法

该方法的灰度值等于彩色图像 R、G、B 三个分量灰度值的求和平均值，其计算公式如公式（11-4）所示：

$$gray(i,j)=\frac{R(i,j)+G(i,j)+B(i,j)}{3} \tag{11-4}$$

平均灰度处理方法实现代码如下。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('luo.png')
```

```
#获取图像高度和宽度
height = img.shape[0]
width = img.shape[1]

#创建一幅图像
grayimg = np.zeros((height, width, 3), np.uint8)

#图像平均灰度处理方法
for i in range(height):
    for j in range(width):
        #灰度值为 RGB 三个分量的平均值
        gray = (int(img[i,j][0]) + int(img[i,j][1]) + int(img[i,j][2])) / 3
        grayimg[i,j] = np.uint8(gray)

#显示图像
cv2.imshow("src", img)
cv2.imshow("gray", grayimg)

#等待显示
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 11-5 所示：

图 11-5 图像平均灰度处理方法效果图

（3）加权平均灰度处理方法

该方法根据色彩重要性，将三个分量以不同的权值进行加权平均。由于人眼对绿色的敏感最高，对蓝色敏感最低，因此，按下式对 RGB 三分量进行加权平均能得到较合理的灰度图像。

$$gray(i,j)=0.30\times R(i,j)+0.59\times G(i,j)+0.11\times B(i,j) \quad (11-5)$$

加权平均灰度处理方法实现代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('luo.png')


#获取图像高度和宽度
height = img.shape[0]
width = img.shape[1]


#创建一幅图像
grayimg = np.zeros((height, width, 3), np.uint8)


#图像平均灰度处理方法
for i in range(height):
    for j in range(width):
        #灰度加权平均法
        gray = 0.30 * img[i,j][0] + 0.59 * img[i,j][1] + 0.11 *
img[i,j][2]
        grayimg[i,j] = np.uint8(gray)
```

```
#显示图像
cv2.imshow("src", img)
cv2.imshow("gray", grayimg)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 11-6 所示：

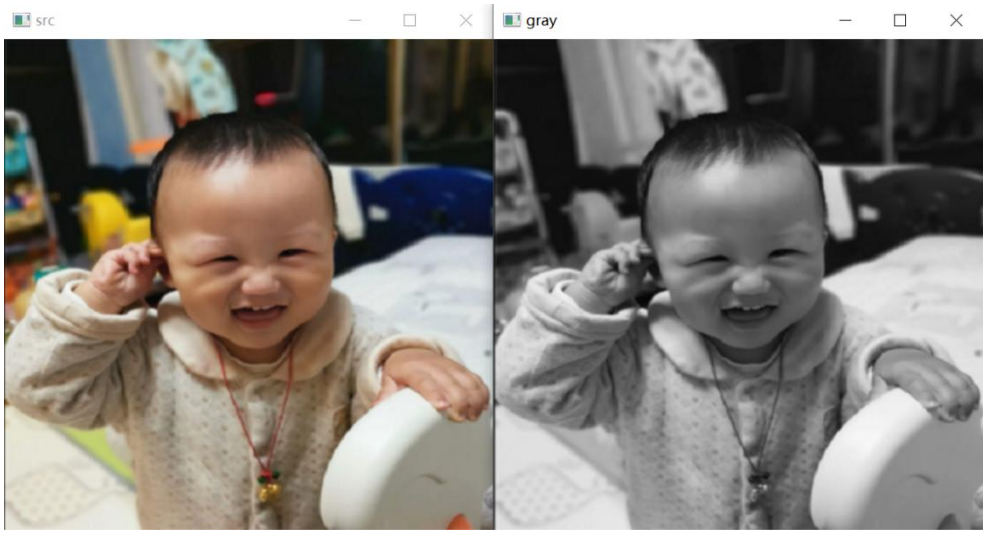


图 11-6 图像加权平均灰度处理方法效果图

4.总结

本文主要讲解图像点运算的灰度化处理，详细介绍常用的灰度化处理方法，并分享了图像颜色空间相互转换，以及三种灰度转换算法的实现。通过灰度处理，我们能有效将彩色图像转换为灰度图，为后续的边缘提取等处理提供支撑，也可

能实现图像处理软件最简单的彩色图转黑白的效果，希望对您有所帮助。

参考文献：

[1] 冈萨雷斯著，阮秋琦译. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 张铮，王艳平，薛桂香等. 数字图像处理与机器视觉——Visual C++与 Matlab 实现 [M]. 北京：人民邮电出版社，2014.

[3] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[4] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[5] Eastmount. [数字图像处理] 三.MFC 实现图像灰度、采样和量化功能详解[EB/OL]. (2015-05-28). https://blog.csdn.net/eastmount/article/details/46010637.

[6] Eastmount. [Python 图像处理] 十四.基于 OpenCV 和像素处理的图像灰度化处理 [EB/OL]. (2019-03-25). https://blog.csdn.net/Eastmount/article/details/88785768.

第 12 篇 图像灰度线性变换

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章引入图像灰度化处理。这篇文章将详细讲解图像灰度线性变换，包括灰度上移、对比度增强、对比度减弱和灰度反色变换。

1.灰度线性变换

图像的灰度线性变换是通过建立灰度映射来调整原始图像的灰度，从而改善图像的质量，凸显图像的细节，提高图像的对比度。灰度线性变换的计算公式如（12-1）所示：

$$D_B = f(D_A) = \alpha D_A + b \tag{12-1}$$

该公式中 D_B 表示灰度线性变换后的灰度值，D_A 表示变换前输入图像的灰度值，α 和 b 为线性变换方程 $f(D)$ 的参数，分别表示斜率和截距[1-4]。

- 当 α=1，b=0 时，保持原始图像
- 当 α=1，b!=0 时，图像所有的灰度值上移或下移
- 当 α=-1，b=255 时，原始图像的灰度值反转
- 当 α>1 时，输出图像的对比度增强

- 当 0< α<1 时，输出图像的对比度减小
- 当 α<0 时，原始图像暗区域变亮，亮区域变暗，图像求补

如图 12-1 所示，显示了图像的灰度线性变换对应的效果图。

图 12-1 图像的灰度线性变换效果图

2.图像灰度上移变换

该算法将实现图像灰度值的上移，从而提升图像的亮度。

- $D_B=D_A+50$

具体实现代码如下所示。由于图像的灰度值位于 0 至 255 区间之内，所以需要对灰度值进行溢出判断。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
import matplotlib.pyplot as plt


#读取原始图像

img = cv2.imread('luo.png')


#图像灰度转换

grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#获取图像高度和宽度

height = grayImage.shape[0]

width = grayImage.shape[1]


#创建一幅图像

result = np.zeros((height, width), np.uint8)


#图像灰度上移变换  DB=DA+50

for i in range(height):

    for j in range(width):


        if (int(grayImage[i,j]+50) > 255):

            gray = 255
```

```
            else:
                gray = int(grayImage[i,j]+50)

            result[i,j] = np.uint8(gray)

#显示图像
cv2.imshow("Gray Image", grayImage)
cv2.imshow("Result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 12-2 所示，图像的所有灰度值上移 50，图像变得更白了。注意，纯黑色对应的灰度值为 0，纯白色对应的灰度值为 255。

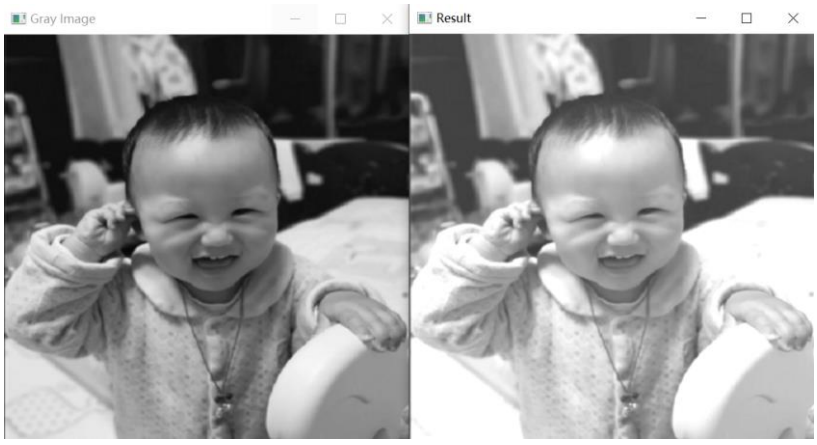


图 12-2 图像灰度值上移变换

3.图像对比度增强变换

该算法将增强图像的对比度，Python 实现代码如下所示。

- $D_B=D_A\times1.5$

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像
img = cv2.imread('luo.png')

#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]

#创建一幅图像
```

```
result = np.zeros((height, width), np.uint8)


#图像对比度增强变换 DB=DA×1.5
for i in range(height):
    for j in range(width):


        if (int(grayImage[i,j]*1.5) > 255):
            gray = 255
        else:
            gray = int(grayImage[i,j]*1.5)


        result[i,j] = np.uint8(gray)


#显示图像
cv2.imshow("Gray Image", grayImage)
cv2.imshow("Result", result)
```

其输出结果如图 12-3 所示，图像的所有灰度值增强 1.5 倍。

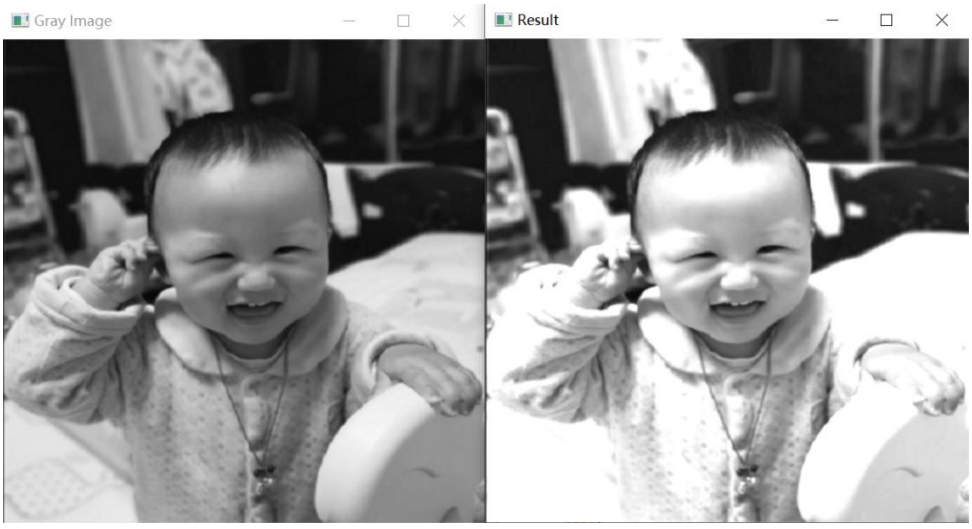


图 12-3 图像线性对比度增强

4.图像对比度减弱变换

该算法将减弱图像的对比度，Python 实现代码如下所示。

- DB=DA×0.8

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('luo.png')
```

```
#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]


#创建一幅图像
result = np.zeros((height, width), np.uint8)


#图像对比度减弱变换 DB=DA×0.8
for i in range(height):
    for j in range(width):
        gray = int(grayImage[i,j]*0.8)
        result[i,j] = np.uint8(gray)


#显示图像
cv2.imshow("Gray Image", grayImage)
cv2.imshow("Result", result)


#等待显示
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 12-4 所示，图像的所有灰度值减弱，图像变得更暗。

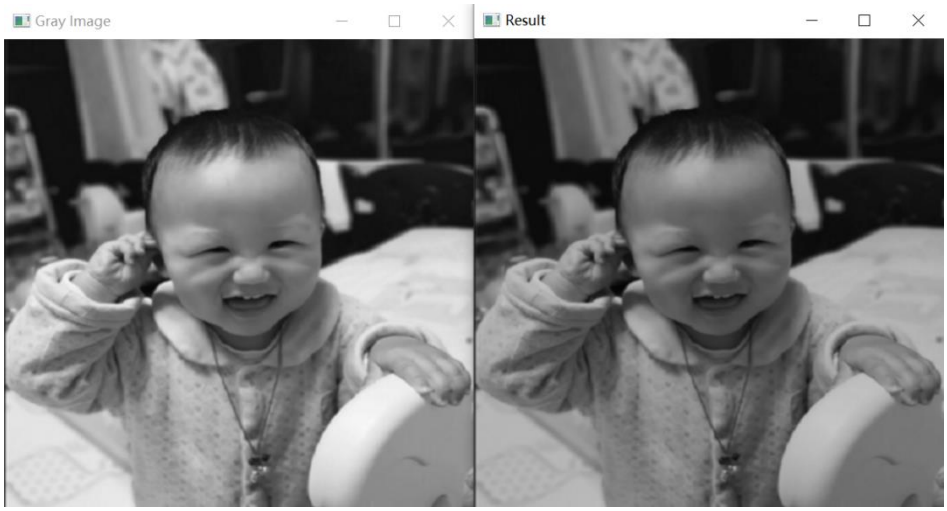


图 12-4 图像线性对比度减弱

5.图像灰度反色变换

反色变换又称为线性灰度求补变换，它是对原图像的像素值进行反转，即黑色变为白色，白色变为黑色的过程。

- $D_B=255-D_A$

其 Python 实现代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
import matplotlib.pyplot as plt


#读取原始图像

img = cv2.imread('luo.png')


#图像灰度转换

grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#获取图像高度和宽度

height = grayImage.shape[0]

width = grayImage.shape[1]


#创建一幅图像

result = np.zeros((height, width), np.uint8)


#图像灰度反色变换  DB=255-DA

for i in range(height):

    for j in range(width):

        gray = 255 - grayImage[i,j]

        result[i,j] = np.uint8(gray)
```

```
#显示图像
cv2.imshow("Gray Image", grayImage)
cv2.imshow("Result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其输出结果如图 12-5 所示，图像处理前后的灰度值是互补的。

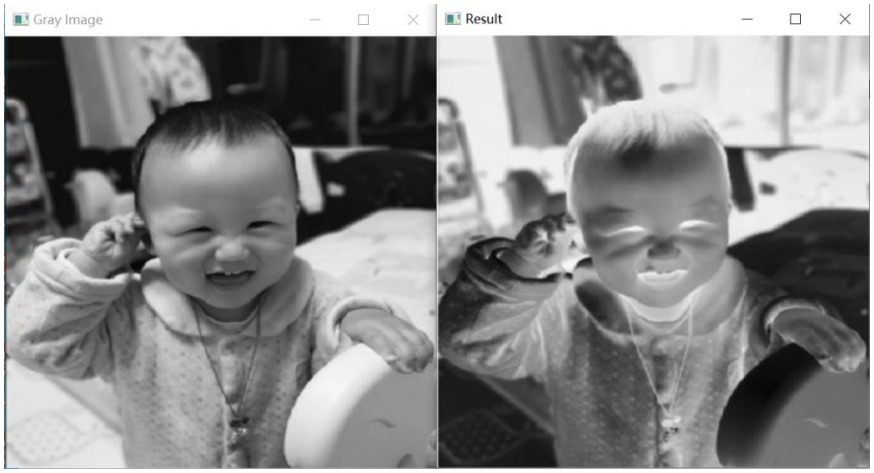


图 12-5 图像灰度反色变换

图像灰度反色变换在医学图像处理中有一定的应用，如图 12-6 所示：

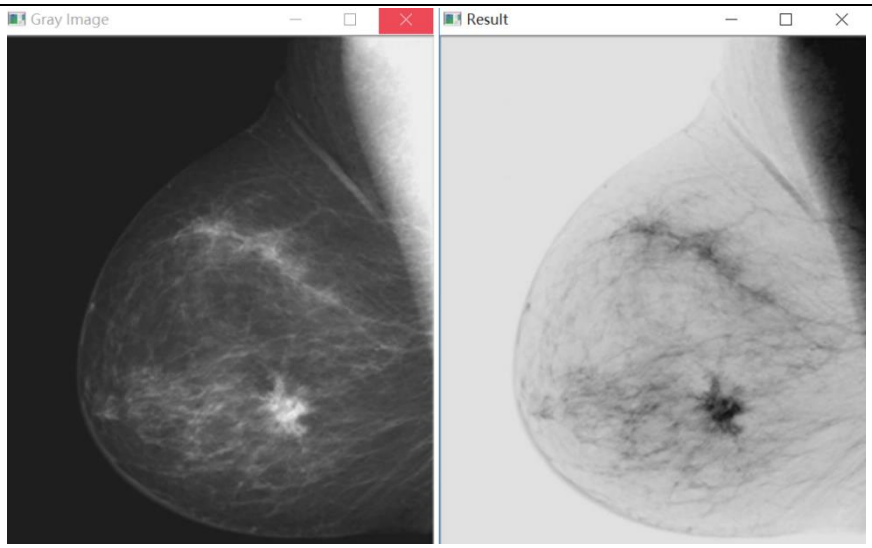


图 12-6 图像灰度反色变换处理医学图像

6.总结

本文主要讲解图像灰度线性变换，包括图像灰度上移、图像对比度增强变换、图像对比度减弱变换和图像灰度反色变换。希望大家一定要自己实现文章中的代码，更好地提升编程能力。

参考文献：

[1] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[2] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[3] Eastmount. [数字图像处理] 三.MFC 实现图像灰度、采样和量化功能详解[EB/OL]. (2015-05-28). https://blog.csdn.net/eastmount/article/details/46010637.

[4] Eastmount. [Python 图像处理] 十五.图像的灰度线性变换[EB/OL]. (2019-03-28). https://blog.csdn.net/Eastmount/article/details/88858696.

第 13 篇 图像灰度非线性变换

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍了图像灰度线性变换。这篇文章将详细讲解图像灰度非线性变换。图像灰度非线性变换主要包括对数变换、幂次变换、指数变换、分段函数变换，通过非线性关系对图像进行灰度处理，本文主要讲解三种常见类型的灰度非线性变换[1-4]。

1.图像灰度非线性变换

原始图像的灰度值按照 $D_B=D_A \times D_A/255$ 的公式进行非线性变换，其代码如下：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
#读取原始图像
img = cv2.imread('luo.png')

#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]

#创建一幅图像
result = np.zeros((height, width), np.uint8)

#图像灰度非线性变换：DB=DA×DA/255
for i in range(height):
    for j in range(width):
        gray = int(grayImage[i,j])*int(grayImage[i,j]) / 255
        result[i,j] = np.uint8(gray)

#显示图像
```

```
cv2.imshow("Gray Image", grayImage)
cv2.imshow("Result", result)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图像灰度非线性变换的输出结果如图 13-1 所示：

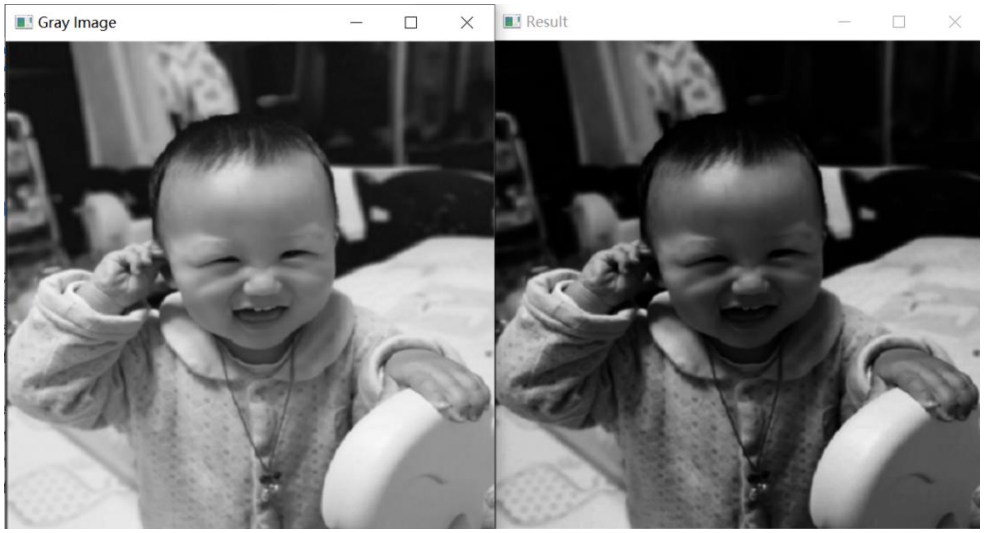


图 13-1 图像灰度非线性变换

2.图像灰度对数变换

图像灰度的对数变换一般表示如公式（13-1）所示：

$$D_B = c \times \log(1 + D_A) \qquad (13\text{-}1)$$

其中 c 为尺度比较常数，D_A 为原始图像灰度值，D_B 为变换后的目标灰度值。如图 13-2 所示，它表示对数曲线下的灰度值变化情况，其中 x 表示原始图像

的灰度值，y 表示对数变换之后的目标灰度值。

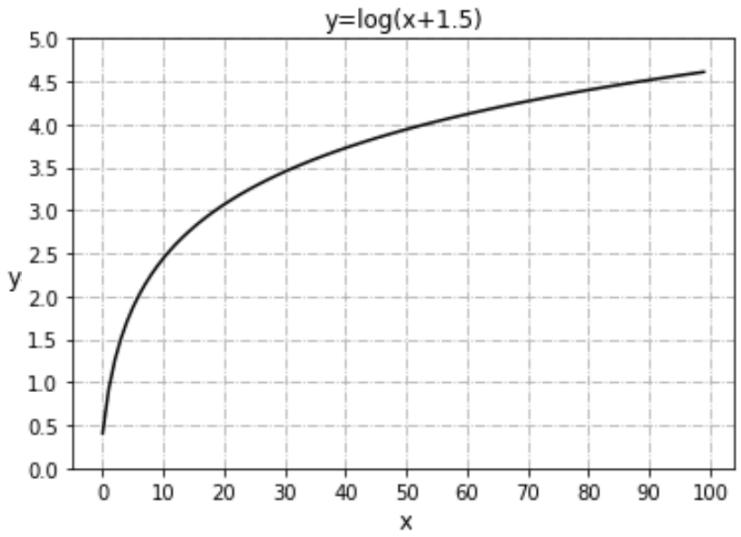


图 13-2 图像灰度对数变换示意图

由于对数曲线在像素值较低的区域斜率大，在像素值较高的区域斜率较小，所以图像经过对数变换后，较暗区域的对比度将有所提升。这种变换可用于增强图像的暗部细节，从而用来扩展被压缩的高值图像中的较暗像素。

对数变换实现了扩展低灰度值而压缩高灰度值的效果，被广泛地应用于频谱图像的显示中。一个典型的应用是傅立叶频谱，其动态范围可能宽达 0～106 直接显示频谱时，图像显示设备的动态范围往往不能满足要求，从而丢失大量的暗部细节；而在使用对数变换之后，图像的动态范围被合理地非线性压缩，从而可以清晰地显示。

在图 13-3 中，未经变换的频谱经过对数变换后，增加了低灰度区域的对比度，从而增强暗部的细节。

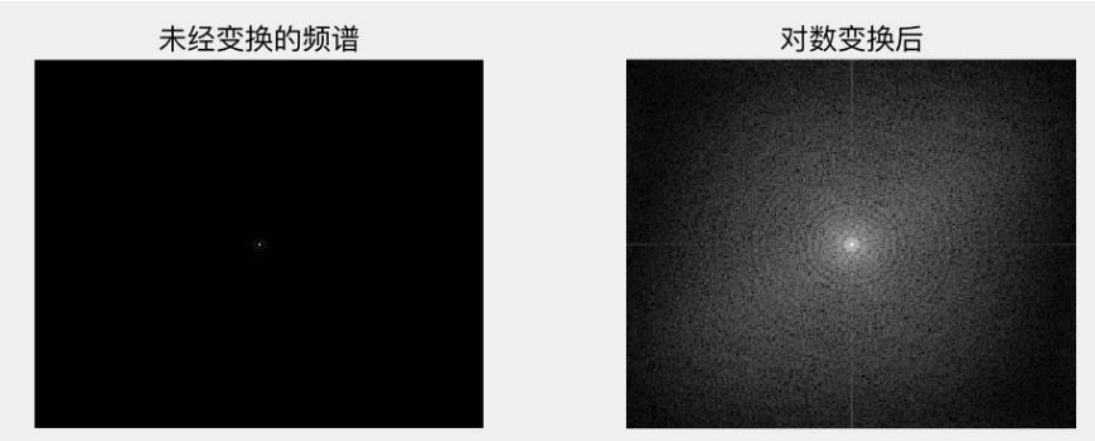


图 13-3 图像对数变换对比图

下面的代码实现了图像灰度的对数变换。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import numpy as np
import matplotlib.pyplot as plt
import cv2


#绘制曲线
def log_plot(c):
    x = np.arange(0, 256, 0.01)
    y = c * np.log(1 + x)
    plt.plot(x, y, 'r', linewidth=1)
    plt.rcParams['font.sans-serif']=['SimHei'] #正常显示中文
标签
```

```
    plt.title('对数变换函数')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.xlim(0, 255), plt.ylim(0, 255)
    plt.show()


#对数变换
def log(c, img):
    output = c * np.log(1.0 + img)
    output = np.uint8(output + 0.5)
    return output


#读取原始图像
img = cv2.imread('dark.png')


#绘制对数变换曲线
log_plot(42)


#图像灰度对数变换
output = log(42, img)
```

```
#显示图像
cv2.imshow('Input', img)
cv2.imshow('Output', output)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图 13-4 表示经过对数函数处理后的效果图，对数变换对于整体对比度偏低并且灰度值偏低的图像增强效果较好。

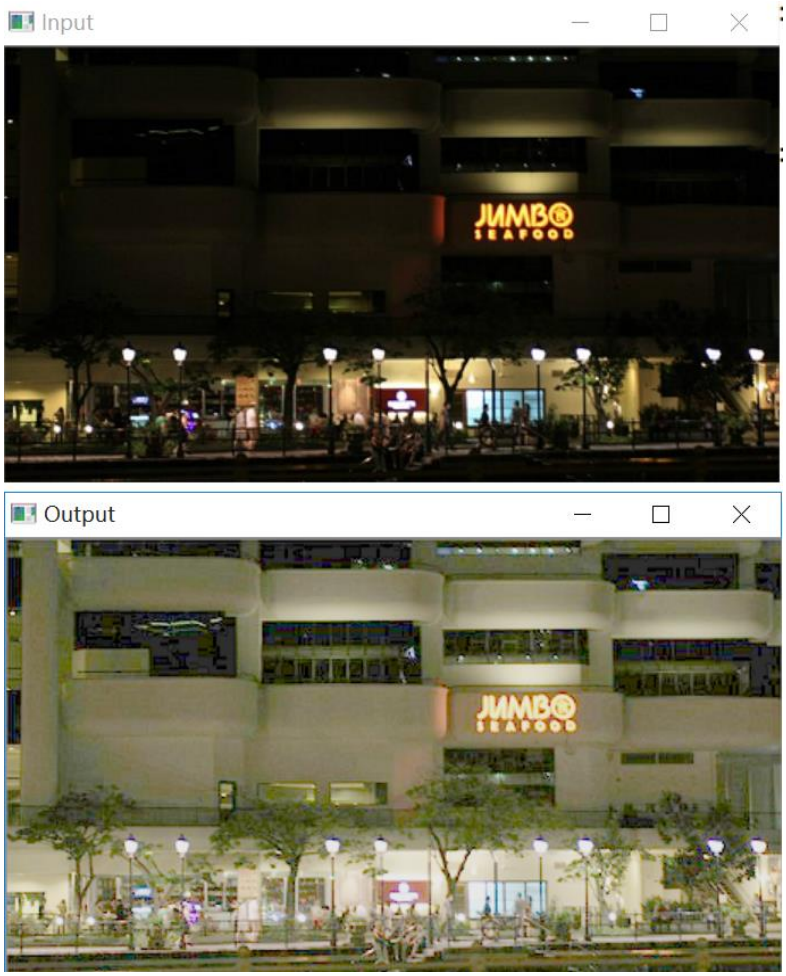


图 13-4 图像灰度对数变换

对应的对数函数曲线如图 13-5 所示，其中 x 表示原始图像的灰度值，y

表示对数变换之后的目标灰度值。

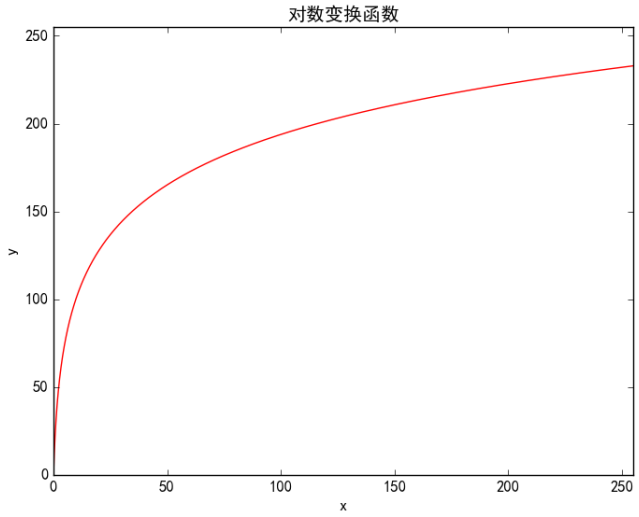


图 13-5 图像灰度对数变换曲线

3.图像灰度伽玛变换

伽玛变换又称为指数变换或幂次变换，是另一种常用的灰度非线性变换。图像灰度的伽玛变换一般表示如公式（13-2）所示：

$$D_B = c \times D_A^{\gamma} \qquad (13\text{-}2)$$

- 当 γ>1 时，会拉伸图像中灰度级较高的区域，压缩灰度级较低的部分。
- 当 γ<1 时，会拉伸图像中灰度级较低的区域，压缩灰度级较高的部分。
- 当 γ=1 时，该灰度变换是线性的，此时通过线性方式改变原图像。

Python 实现图像灰度的伽玛变换代码如下，主要调用幂函数实现。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import numpy as np
```

```
import matplotlib.pyplot as plt
import cv2


#绘制曲线
def gamma_plot(c, v):
    x = np.arange(0, 256, 0.01)
    y = c*x**v
    plt.plot(x, y, 'r', linewidth=1)
    plt.rcParams['font.sans-serif']=['SimHei'] #正常显示中文
标签
    plt.title('伽马变换函数')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.xlim([0, 255]), plt.ylim([0, 255])
    plt.show()


#伽玛变换
def gamma(img, c, v):
    lut = np.zeros(256, dtype=np.float32)
    for i in range(256):
        lut[i] = c * i ** v
```

```
    output_img = cv2.LUT(img, lut) #像素灰度值的映射
    output_img = np.uint8(output_img+0.5)
    return output_img

#读取原始图像
img = cv2.imread('white.png')

#绘制伽玛变换曲线
gamma_plot(0.00000005, 4.0)

#图像灰度伽玛变换
output = gamma(img, 0.00000005, 4.0)

#显示图像
cv2.imshow('Imput', img)
cv2.imshow('Output', output)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图 13-6 表示经过伽玛变换处理后的效果图，伽马变换对于图像对比度偏低，并且整体亮度值偏高（或由于相机过曝）情况下的图像增强效果明显。

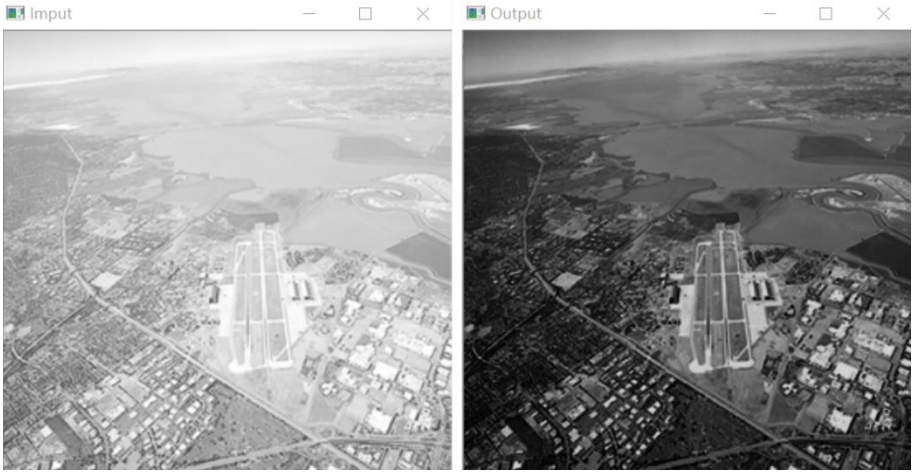


图 13-6 图像灰度伽玛变换

对应的伽马变换曲线如图 13-7 所示，其中 x 表示原始图像的灰度值，y 表示伽马变换之后的目标灰度值。

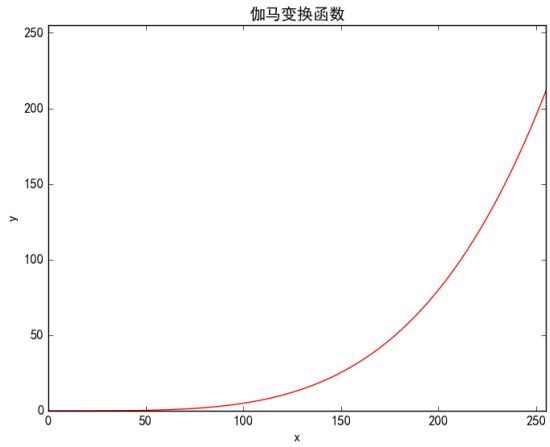


图 13-7 图像灰度伽玛变换曲线

4.总结

本文主要讲解图像灰度非线性变换，包括图像对数变换和伽马变换。其中，图像经过对数变换后，较暗区域的对比度将有所提升；而案例中经过伽玛变换处理的图像，整体亮度值偏高（或由于相机过曝）情况下的图像增强效果明显。这

些图像处理方法能有效提升图像的质量，为我们提供更好地感官效果。

参考文献：

1] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[2] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[3] Eastmount. [数字图像处理] 三.MFC 实现图像灰度、采样和量化功能详解[EB/OL]. (2015-05-28). https://blog.csdn.net/eastmount/article/details/46010637.

[4] Eastmount. [Python 图像处理] 十六.图像的灰度非线性变换之对数变换、伽马变换[EB/OL]. (2019-03-31). https://blog.csdn.net/Eastmount/article/details/88929290.

第 14 篇 图像点运算之图像阈值化处理

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面三篇文章介绍图像灰度化处理（含灰度线性变换和灰度非线性变换）。这篇文章将详细讲解图像阈值化处理，涉及阈值化处理、固定阈值化处理和自适应阈值化处理，这是图像边缘检测或图像增强等处理的基础。

图像阈值化可以理解为一个简单的图像分割操作，阈值又称为临界值，它的目的是确定出一个范围，然后这个范围内的像素点使用同一种方法处理，而阈值之外的部分则使用另一种处理方法或保持原样[1-2]。

1.图像阈值化

图像阈值化(Binarization)旨在剔除掉图像中一些低于或高于一定值的像素，从而提取图像中的物体，将图像的背景和噪声区分开来。

灰度化处理后的图像中，每个像素都只有一个灰度值，其大小表示明暗程度。阈值化处理可以将图像中的像素划分为两类颜色，常见的阈值化算法如公式（14-1）所示：

$$Gray(i,j)=\begin{cases}255, & Gray(i,j)\geq T\\ 0, & Gray(i,j)<T\end{cases} \quad (14-1)$$

当某个像素点的灰度 Gray(i,j)小于阈值 T 时，其像素设置为 0，表示黑色；当灰度 Gray(i,j)大于或等于阈值 T 时，其像素值为 255，表示白色。

在 Python 的 OpenCV 库中，提供了固定阈值化函数 threshold()和自适应阈值化函数 adaptiveThreshold()，将一幅图像进行阈值化处理[3-4]。

2.固定阈值化处理

OpenCV 中提供了函数 threshold()实现固定阈值化处理，其函数原型如下：

dst = cv2.threshold(src, thresh, maxval, type[, dst])

- src 表示输入图像的数组，8 位或 32 位浮点类型的多通道数
- dst 表示输出的阈值化处理后的图像，其类型和通道数与 src 一致
- thresh 表示阈值
- maxval 表示最大值，当参数阈值类型 type 选择 CV_THRESH_BINARY 或 CV_THRESH_BINARY_INV 时，该参数为阈值类型的最大值
- type 表示阈值类型

其中，threshold()函数不同类型的处理算法如表 14-1 所示。

表 7-1 图像阈值化处理不同类型对比

算法原型	算法含义
threshold(Gray,127,255,cv2.THRESH_BINARY)	像素点的灰度值大于阈值设其灰度值为最大值，小于阈值的像素点灰度值设定为0。
threshold(Gray,127,255,cv2.THRESH_BINARY_INV)	大于阈值的像素点的灰度值设定为 0，而小于该阈值的设定为 255。
threshold(Gray,127,255,cv2.THRESH_TRUNC)	像素点的灰度值小于阈值不改变，反之将像素点的灰度值设定为该阈值。
threshold(Gray,127,255,cv2.THRESH_TOZERO)	像素点的灰度值小于该阈值的不进行任何改变，而大于该阈值的部分，其灰度值全部变为 0。
threshold(Gray,127,255,cv2.THRESH_TOZERO_INV)	像素点的灰度值大于该阈值的不进行任何改变，小于该阈值其灰度值全部设定为0。

其对应的阈值化描述如图 14-1 所示：

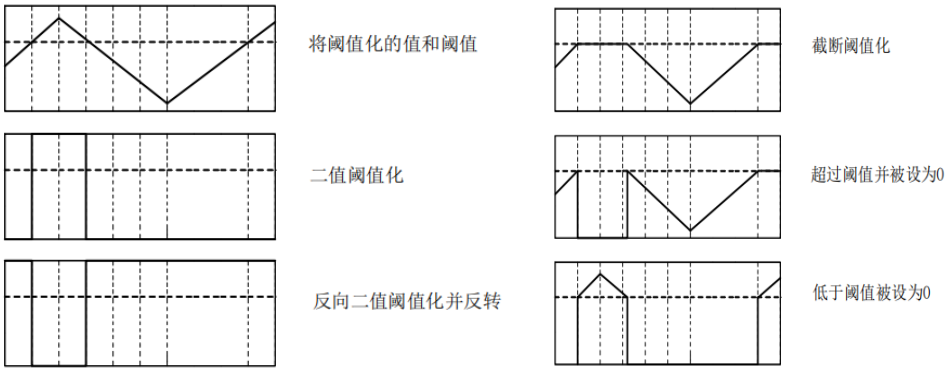


图 14-1 不同阈值类型的处理过程

阈值化处理广泛应用于各行各业，比如生物学中的细胞图分割、交通领域的车牌识别等。通过阈值化处理将所图像转换为黑白两色图，从而为后续的图像识别和图像分割提供更好的支撑作用。下面详细讲解五种阈值化处理算法。

（1）二进制阈值化

该函数的原型为 threshold(Gray,127,255,cv2.THRESH_BINARY)。其方法首先要选定一个特定的阈值量，比如 127，再按照如下所示的规则进行阈值化处理。

$$\mathrm{dst}(x,y)=\begin{cases}\max val & ,src(x,y)>thresh\\ 0 & ,src(x,y)\le thresh\end{cases} \qquad (14\text{-}2)$$

当前像素点的灰度值大于 thresh 阈值时（如 127），其像素点的灰度值设定为最大值（如 8 位灰度值最大为 255）；否则，像素点的灰度值设置为 0。如阈值为 127 时，像素点的灰度值为 163，则阈值化设置为 255；像素点的灰度值为 82，则阈值化设置为 0。

二进制阈值化处理的 Python 代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np


#读取图片
```

```
src = cv2.imread('luo.png')


#灰度图像处理
grayImage = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)


#二进制阈值化处理
r, b = cv2.threshold(grayImage, 127, 255,
cv2.THRESH_BINARY)


#显示图像
cv2.imshow("src", src)
cv2.imshow("result", b)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 14-2 所示，左边是小珞珞的原图，右边是将原图进行二进制阈值化处理的效果图。像素值大于 127 的设置为 255，小于等于 127 设置为 0。

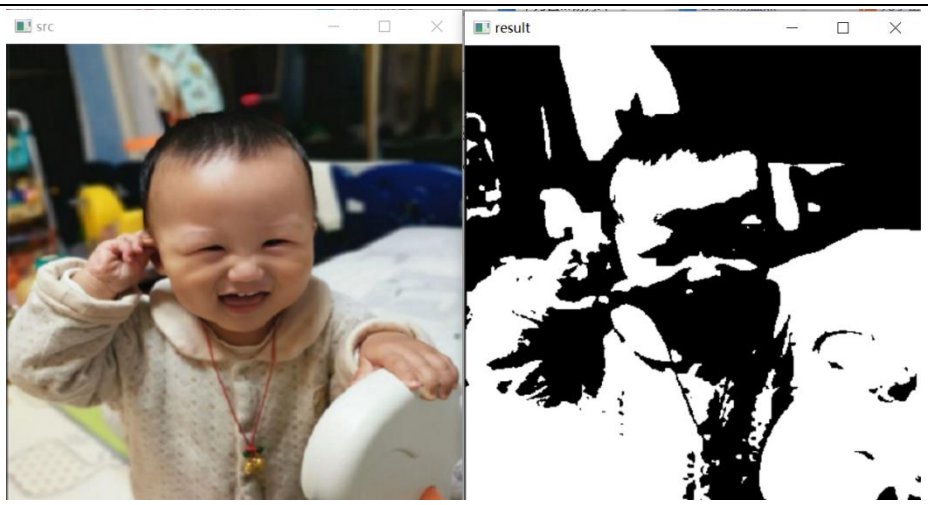


图 14-2 图像二进制阈值化处理

（2）反二进制阈值化

该函数的原型为 threshold(Gray,127,255,cv2.THRESH_BINARY_INV)。其方法首先要选定一个特定的阈值量，比如 127，再按照如下所示的规则进行阈值化处理。

$$\mathrm{dst}(x,y)=\begin{cases}0 & ,src(x,y)>thresh\\ \max val & ,src(x,y)\le thresh\end{cases} \qquad (14\text{-}3)$$

当前像素点的灰度值大于 thresh 阈值时（如 127），其像素点的灰度值设定为 0；否则，像素点的灰度值设置为最大值。如阈值为 127 时，像素点的灰度值为 211，则阈值化设置为 0；像素点的灰度值为 101，则阈值化设置为 255。

反二进制阈值化处理的 Python 代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
#读取图片
src = cv2.imread('luo.png')

#灰度图像处理
grayImage = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)

#反二进制阈值化处理
r, b = cv2.threshold(grayImage, 127, 255,
cv2.THRESH_BINARY_INV)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", b)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 14-3 所示：

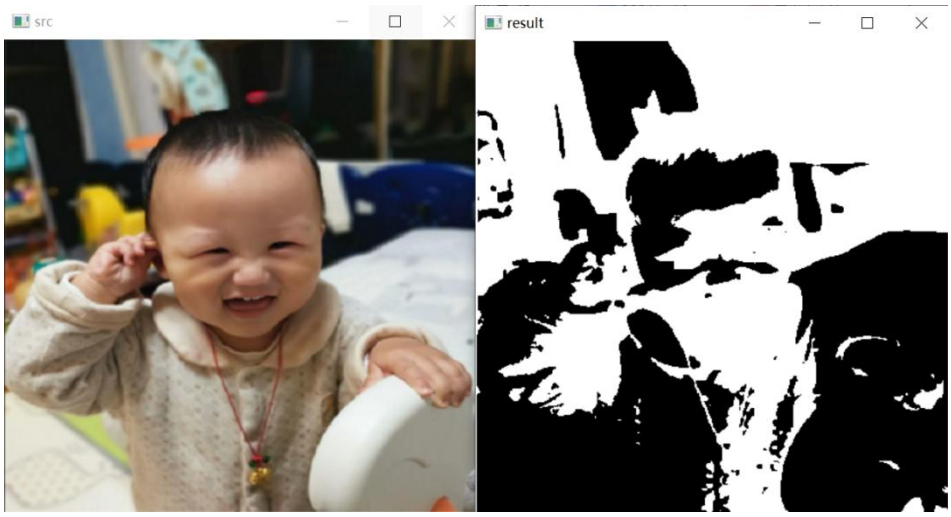


图 14-3 图像反二进制阈值化处理

（3）截断阈值化

该函数的原型为 threshold(Gray,127,255,cv2.THRESH_TRUNC)。图像中大于该阈值的像素点被设定为该阈值，小于或等于该阈值的保持不变，比如 127。新的阈值产生规则如下：

$$\mathrm{dst}(x,y)=\begin{cases}\mathrm{threshold} & ,src(x,y)>thresh\\ src(x,y) & ,src(x,y)\le thresh\end{cases} \qquad (14\text{-}4)$$

比如阈值为 127 时，像素点的灰度值为 167，则阈值化设置为 127；像素点的灰度值为 82，则阈值化设置为 82。截断阈值化处理的 Python 代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
#读取图片
src = cv2.imread('luo.png')

#灰度图像处理
grayImage = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)

#截断阈值化处理
r, b = cv2.threshold(grayImage, 127, 255,
cv2.THRESH_TRUNC)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", b)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 14-4 所示，图像经过截断阈值化处理将灰度值处理于 0 至 127 之间。

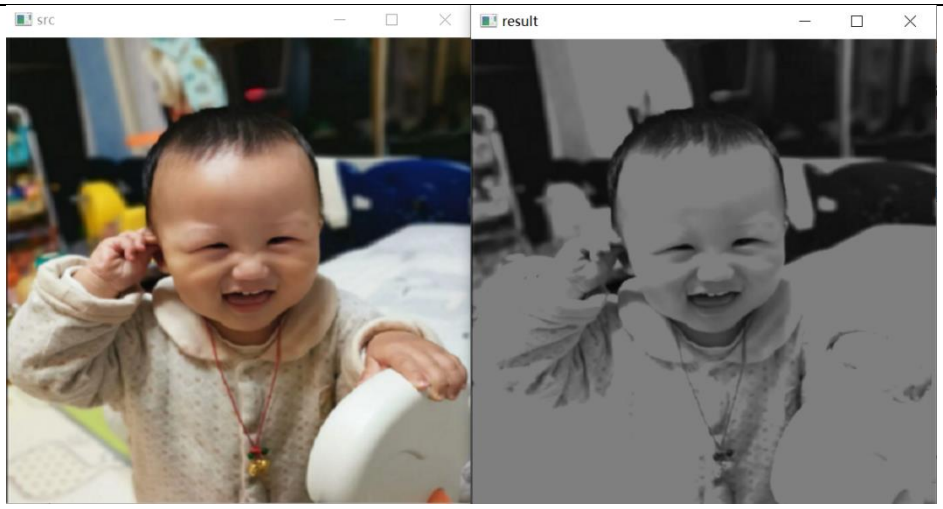


图 14-4 图像截断阈值化处理

（4）阈值化为 0

该函数的原型为 threshold(Gray,127,255,cv2.THRESH_TOZERO)。按照如下公式对图像的灰度值进行处理。

$$\mathrm{dst}(x,y)=\begin{cases} src(x,y) & ,src(x,y)>thresh \\ 0 & ,src(x,y)\le thresh \end{cases} \qquad (14\text{-}5)$$

当前像素点的灰度值大于 thresh 阈值时（如 127），其像素点的灰度值保持不变；否则，像素点的灰度值设置为 0。如阈值为 127 时，像素点的灰度值为 211，则阈值化设置为 211；像素点的灰度值为 101，则阈值化设置为 0。

图像阈值化为 0 处理的 Python 代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
#读取图片
src = cv2.imread('luo.png')

#灰度图像处理
grayImage = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)

#阈值化为 0 处理
r, b = cv2.threshold(grayImage, 127, 255,
cv2.THRESH_TOZERO)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", b)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 14-5 所示，该算法把比较亮的部分不变，比较暗的部分处理为 0。

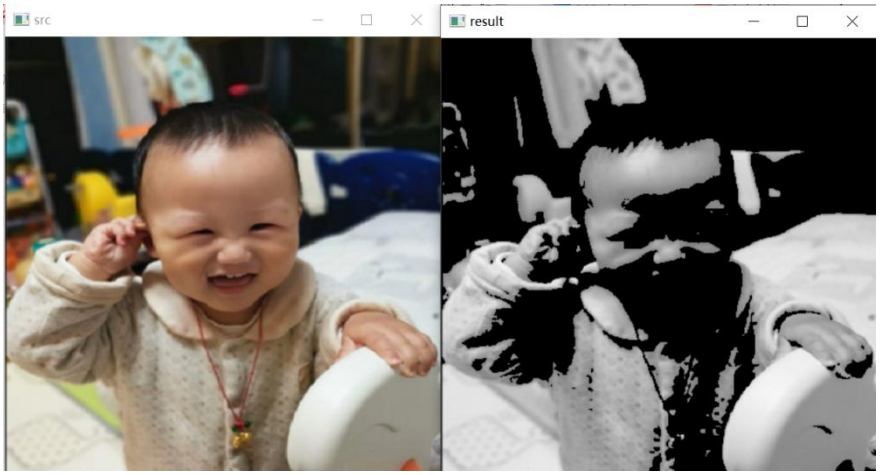


图 14-5 图像阈值化为 0 处理

（5）反阈值化为 0

该函数的原型为 threshold(Gray,127,255, cv2.THRESH_TOZERO_INV)。按照如下公式对图像的灰度值进行处理。

$$\mathrm{dst}(x,y)=\begin{cases}0 & ,src(x,y)>thresh\\ src(x,y) & ,src(x,y)\le thresh\end{cases} \qquad (14\text{-}6)$$

当前像素点的灰度值大于 thresh 阈值时（如 127），其像素点的灰度值设置为 0；否则，像素点的灰度值保持不变。如阈值为 127 时，像素点的灰度值为 211，则阈值化设置为 0；像素点的灰度值为 101，则阈值化设置为 101。

图像反阈值化为 0 处理的 Python 代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
#读取图片
src = cv2.imread('luo.png')


#灰度图像处理
GrayImage = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)


#二进制阈值化处理
r, b = cv2.threshold(GrayImage, 127, 255,
cv2.THRESH_TOZERO_INV)


#显示图像
cv2.imshow("src", src)
cv2.imshow("result", b)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 14-6 所示：

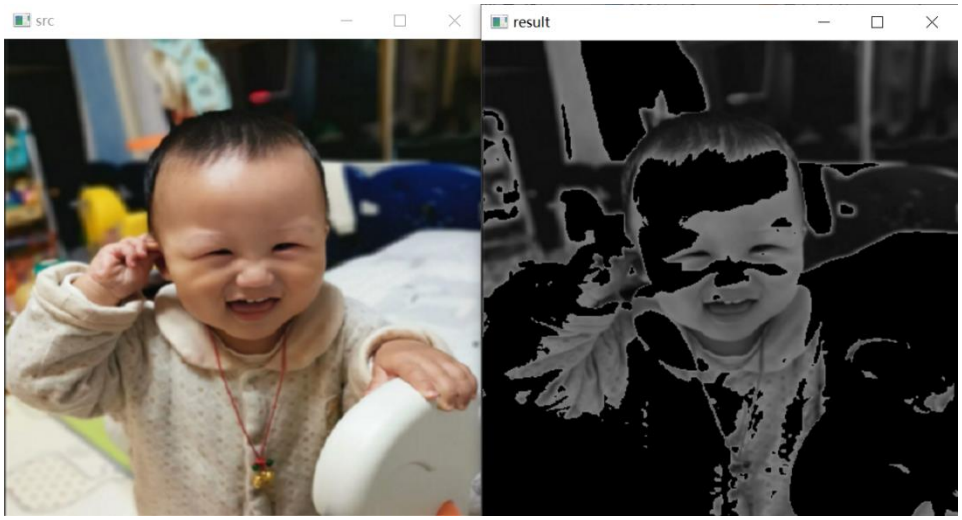


图 14-6 图像反阈值化为 0 处理

同样，我们在对民族图腾及图像进行识别和保护时，也需要进行图像阈值化处理。下面代码是对比苗族服饰图像五种固定阈值化处理的对比结果。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图像
img=cv2.imread('miao.png')
grayImage=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)


#阈值化处理
```

```
ret,thresh1=cv2.threshold(grayImage,127,255,cv2.THRES
H_BINARY)
ret,thresh2=cv2.threshold(grayImage,127,255,cv2.THRES
H_BINARY_INV)
ret,thresh3=cv2.threshold(grayImage,127,255,cv2.THRES
H_TRUNC)
ret,thresh4=cv2.threshold(grayImage,127,255,cv2.THRES
H_TOZERO)
ret,thresh5=cv2.threshold(grayImage,127,255,cv2.THRES
H_TOZERO_INV)

#显示结果
titles = ['Gray Image','BINARY','BINARY_INV','TRUNC',
'TOZERO','TOZERO_INV']
images = [grayImage, thresh1, thresh2, thresh3, thresh4,
thresh5]
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 14-7 所示：

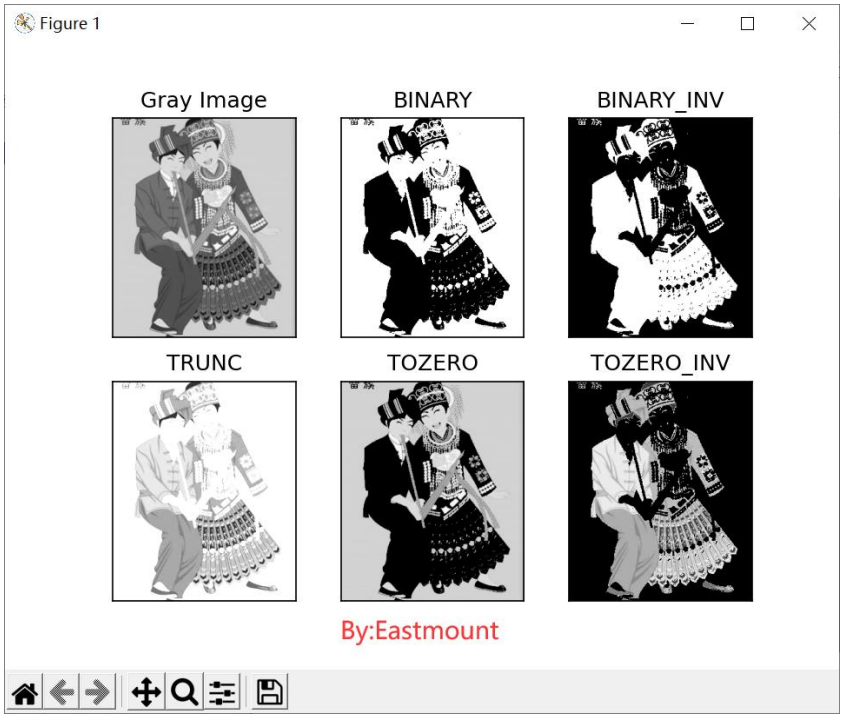


图 14-7 图像固定阈值化处理对比结果

3.自适应阈值化处理

前面讲解的是固定值阈值化处理方法，而当同一幅图像上的不同部分具有不同亮度时，上述方法就不在适用。此时需要采用自适应阈值化处理方法，根据图像上的每一个小区域，计算与其对应的阈值，从而使得同一幅图像上的不同区域采用不同的阈值，在亮度不同的情况下得到更好的结果。自适应阈值化处理在 OpenCV 中调用 cv2.adaptiveThreshold()函数实现，其原型如下所示：

```
dst = adaptiveThreshold(src, maxValue, adaptiveMethod,
thresholdType, blockSize, C[, dst])
```

- src 表示输入图像

- dst 表示输出的阈值化处理后的图像，其类型和尺寸需与 src 一致
- maxValue 表示给像素赋的满足条件的最大值
- adaptiveMethod 表示要适用的自适应阈值算法，常见取值包括 ADAPTIVE_THRESH_MEAN_C（阈值取邻域的平均值）或 ADAPTIVE_THRESH_GAUSSIAN_C（阈值取自邻域的加权和平均值，权重分布为一个高斯函数分布）
- thresholdType 表示阈值类型，取值必须为 THRESH_BINARY 或 THRESH_BINARY_INV
- blockSize 表示计算阈值的像素邻域大小，取值为 3、5、7 等
- C 表示一个常数，阈值等于平均值或者加权平均值减去这个常数

当阈值类型 thresholdType 为 THRESH_BINARY 时，其灰度图像转换为阈值化图像的计算公式如下所示：

$$dst(x,y)=\begin{cases} maxValue & ,src(x,y)>T(x,y) \\ 0 & ,src(x,y)\le T(x,y) \end{cases} \quad (14-7)$$

当阈值类型 thresholdType 为 THRESH_BINARY_INV 时，其灰度图像转换为阈值化图像的计算公式如下所示：

$$\mathrm{dst}(x,y)=\begin{cases} 0 & ,\mathrm{src}(x,y)>\mathrm{T}(x,y) \\ \mathrm{maxValue} & ,\mathrm{src}(x,y)\le \mathrm{T}(x,y) \end{cases} \quad (14-8)$$

其中，dst(x,y)表示阈值化处理后的灰度值，T(x,y)表示计算每个单独像素的阈值，其取值如下：

- 当 adaptiveMethod 参数采用 ADAPTIVE_THRESH_MEAN_C 时，阈值 T(x,y)为 blockSize×blockSize 邻域内（x，y）减去参数

C 的平均值。

- 当 adaptiveMethod 参数采用 ADAPTIVE_THRESH_GAUSSIAN_C 时，阈值 T(x,y) 为 blockSize×blockSize 邻域内（x，y）减去参数 C 与高斯窗交叉相关的加权总和。

下面的代码是对比固定值阈值化与自适应阈值化处理的方法。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

#读取图像
img = cv2.imread('miao.png')

#图像灰度化处理
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#固定值阈值化处理
r, thresh1 = cv2.threshold(grayImage, 127, 255,
```

```
cv2.THRESH_BINARY)

#自适应阈值化处理 方法一
thresh2 = cv2.adaptiveThreshold(grayImage, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,
11, 2)

#自适应阈值化处理 方法二
thresh3 = cv2.adaptiveThreshold(grayImage, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 11, 2)

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#显示图像
titles = ['灰度图像', '全局阈值', '自适应平均阈值', '自适应高斯阈值']
images = [grayImage, thresh1, thresh2, thresh3]
for i in range(4):
    plt.subplot(2, 2, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
```

```
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 14-8 所示，左上角为灰度化处理图像；右上角为固定值全局阈值化处理图像（cv2.threshold）；左下角为自适应邻域平均值分割，噪声较多；右下角为自适应邻域加权平均值分割，采用高斯函数分布，其效果相对较好。

图 14-8 图像自适应阈值化处理对比结果

4.总结

本章主要讲解了图像阈值化处理知识，调用 OpenCV 的 threshold()实现固定阈值化处理，调用 adaptiveThreshold()函数实现自适应阈值化处理。本文知识点将为后续的图像处理提供良好的基础。

参考文献：

[1] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[2] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[3] Eastmount. [Python 图像处理] 七.图像阈值化处理及算法对比[EB/OL]. (2018-10-30). https://blog.csdn.net/Eastmount/article/details/83548652.

[4] Eastmount. [Python 图像处理] 三十一.图像点运算处理两万字详细总结（灰度化处理、阈值化处理）[EB/OL]. (2020-11-13). https://blog.csdn.net/Eastmount/article/details/109649659.

第 15 篇 图像形态学处理之腐蚀和膨胀

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像阈值化处理。这篇文章将开始图像形态学知识，主要介绍图像腐蚀处理和膨胀处理。数学形态学（Mathematical Morphology）是一种应用于图像处理和模式识别领域的新方法。数学形态学（也称图像代数）表示以形态为基础对图像进行分析的数学工具，其基本思想是用具有一定形态的结构元素去量度和提取图像中对应形状以达到对图像分析和识别的目的。

1.形态学理论知识

数学形态学的应用可以简化图像数据，保持它们基本的形状特征，并出去不相干的结构。数学形态学的算法有天然的并行实现的结构，主要针对的是二值图像（0 或 1）。在图像处理方面，二值形态学经常应用到对图像进行分割、细化、抽取骨架、边缘提取、形状分析、角点检测，分水岭算法等。由于其算法简单，算法能够并行运算所以经常应用到硬件中[1-2]。

常见的图像形态学运算包括：

- 腐蚀

- 膨胀
- 开运算
- 闭运算
- 梯度运算
- 顶帽运算
- 底帽运算

这些运算在 OpenCV 中主要通过 MorphologyEx()函数实现，它能利用基本的膨胀和腐蚀技术，来执行更加高级形态学变换，如开闭运算、形态学梯度、顶帽、黑帽等，也可以实现最基本的图像膨胀和腐蚀。其函数原型如下：

dst = cv2.morphologyEx(src, model, kernel)

- src 表示原始图像
- model 表示图像进行形态学处理，包括：

  (1) cv2.MORPH_OPEN：开运算（Opening Operation）

  (2)cv2.MORPH_CLOSE：闭运算（Closing Operation）

  (3)cv2.MORPH_GRADIENT：形态学梯度（Morphological Gradient）

  (4)cv2.MORPH_TOPHAT：顶帽运算（Top Hat）

  (5)cv2.MORPH_BLACKHAT：黑帽运算（Black Hat）
- kernel 表示卷积核，可以用 numpy.ones()函数构建

2.图像腐蚀

图像的腐蚀（Erosion）和膨胀（Dilation）是两种基本的形态学运算，主要用来寻找图像中的极小区域和极大区域。图像腐蚀类似于“领域被蚕食”，它将图像中的高亮区域或白色部分进行缩减细化，其运行结果比原图的高亮区域更小。

设 A，B 为集合，A 被 B 的腐蚀，记为 A－B，其定义为：

$$A - B = \{x \mid B_x \subseteq A\} \tag{15-1}$$

该公式表示图像 A 用卷积模板 B 来进行腐蚀处理，通过模板 B 与图像 A 进行卷积计算，得出 B 覆盖区域的像素点最小值，并用这个最小值来替代参考点的像素值。如图 8-1 所示，将左边的原始图像 A 腐蚀处理为右边的效果图 A－B。

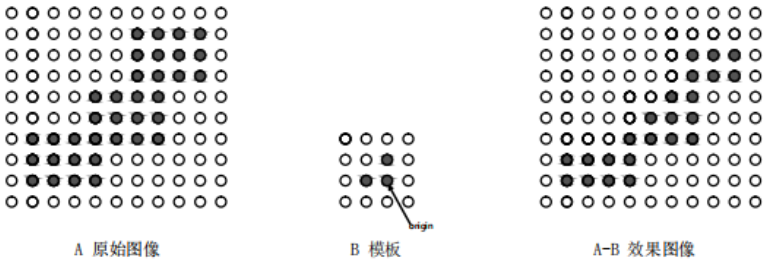


图 15-1 图像腐蚀处理原理

图像腐蚀主要包括二值图像和卷积核两个输入对象，卷积核是腐蚀中的关键数组，采用 Numpy 库可以生成。卷积核的中心点逐个像素扫描原始图像，被扫描到的原始图像中的像素点，只有当卷积核对应的元素值均为 1 时，其值才为 1，否则将其像素值修改为 0。在 Python 中，主要调用 OpenCV 的 erode()

函数实现图像腐蚀。其函数原型如下：

dst = cv2.erode(src, kernel, iterations)

- src 表示原始图像
- kernel 表示卷积核
- iterations 表示迭代次数，默认值为 1，表示进行一次腐蚀操作

可以采用函数 numpy.ones((5,5), numpy.uint8)创建 5×5 的卷积核，如下：

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (15\text{-}2)$$

图像腐蚀操作的代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取图片
src = cv2.imread('test01.jpg', cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((5,5), np.uint8)
```

```
#图像腐蚀处理
erosion = cv2.erode(src, kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", erosion)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 15-2 所示，左边表示原图，右边是腐蚀处理后的图像，可以发现图像中的干扰细线（噪声）被清洗干净。

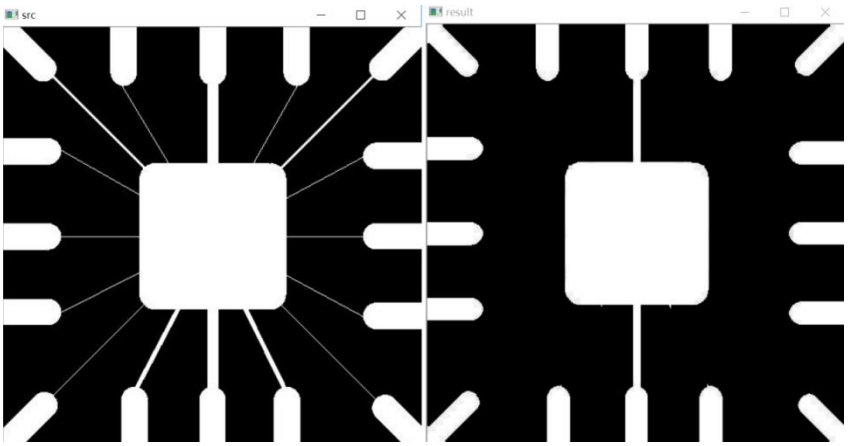


图 15-2 图像腐蚀处理

如果腐蚀之后的图像仍然存在噪声，可以设置迭代次数进行多次腐蚀操作。

比如进行 9 次腐蚀操作的核心代码如下：

- erosion = cv2.erode(src, kernel,iterations=9)

最终经过 9 次腐蚀处理的输出图像如图 15-3 所示。

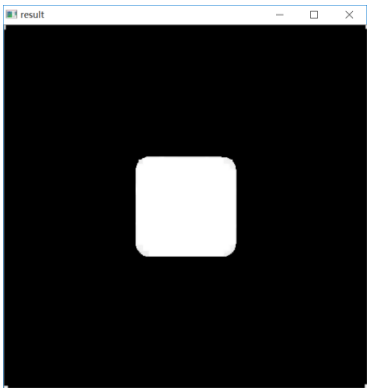


图 15-3 多次腐蚀处理后的图像

3.图像膨胀

图像膨胀是腐蚀操作的逆操作，类似于“领域扩张”，它将图像中的高亮区域或白色部分进行扩张，其运行结果比原图的高亮区域更大。

设 A，B 为集合，∅为空集，A 被 B 的膨胀，记为 $A \oplus B$，其中 $\oplus$ 为膨胀算子，膨胀定义为：

$$A \oplus B = \{x \mid B_x \cap A \neq \phi\} \quad (15\text{-}3)$$

该公式表示用 B 来对图像 A 进行膨胀处理，其中 B 是一个卷积模板，其形状可以为正方形或圆形，通过模板 B 与图像 A 进行卷积计算，扫描图像中的每一个像素点，用模板元素与二值图像元素做“与”运算，如果都为 0，那么目标像素点为 0，否则为 1。从而计算 B 覆盖区域的像素点最大值，并用该值替换参考点的像素值实现图像膨胀。图 15-4 是将左边的原始图像 A 膨胀处理为右边

的效果图 A⊕B。

图 15-4 图像膨胀处理原理图

图像被腐蚀处理后，它将去除噪声，但同时会压缩图像，而图像膨胀操作可以去除噪声并保持原有形状，如图 15-5 所示。

图 15-5 图像腐蚀和膨胀操作对比图

在 Python 中，主要调用 OpenCV 的 dilate()函数实现图像腐蚀。函数原型如下：

```
dst = cv2.dilate(src, kernel, iterations)
```

- src 表示原始图像
- kernel 表示卷积核，可以用 numpy.ones()函数构建
- iterations 表示迭代次数，默认值为 1，表示进行一次膨胀操作

图像膨胀操作的代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
```

```
import cv2
import numpy as np

#读取图片
src = cv2.imread('zhiwen.png',
cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((5,5), np.uint8)

#图像膨胀处理
erosion = cv2.dilate(src, kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", erosion)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 15-6 所示：

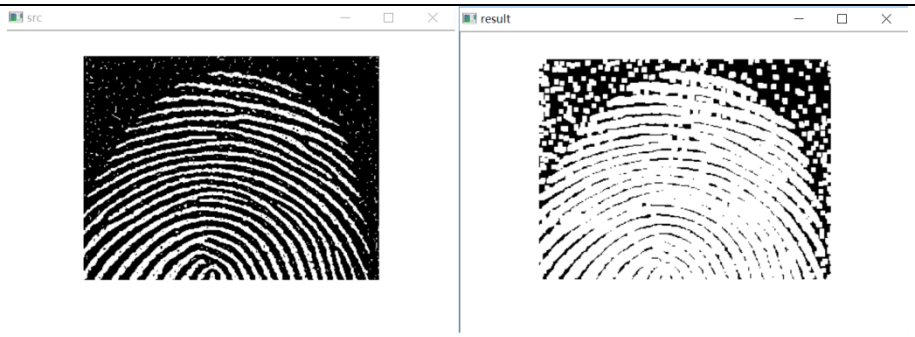


图 15-6 图像膨胀处理

4.总结

本章主要介绍图像形态学处理，详细讲解了图像腐蚀处理和膨胀处理。数学形态学是一种应用于图像处理和模式识别领域的新方法，其基本思想是用具有一定形态的结构元素去量度和提取图像中对应形状以达到对图像分析和识别目的。

参考文献：

[1] 冈萨雷斯著，阮秋琦译. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[4] Eastmount. [Python 图像处理] 八.图像腐蚀与图像膨胀[EB/OL]. (2018-10-31). https://blog.csdn.net/Eastmount/article/details/83581277.

第 16 篇 图像形态学处理之开运算、闭运算和梯度运算

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像阈值化的腐蚀处理和膨胀处理。这篇文章将继续介绍开运算、闭运算和梯度运算。数学形态学表示以形态为基础对图像进行分析的数学工具，其基本思想是用具有一定形态的结构元素去量度和提取图像中对应形状以达到对图像分析和识别的目的。

1.图像开运算

开运算一般能平滑图像的轮廓，削弱狭窄部分，去掉较细的突出。闭运算也是平滑图像的轮廓，与开运算相反，它一般熔合窄的缺口和细长的弯口，去掉小洞，填补轮廓上的缝隙。图像开运算是图像依次经过腐蚀、膨胀处理的过程，图像被腐蚀后将去除噪声，但同时也压缩了图像，接着对腐蚀过的图像进行膨胀处理，可以在保留原有图像的基础上去除噪声。其原理如图 16-1 所示。

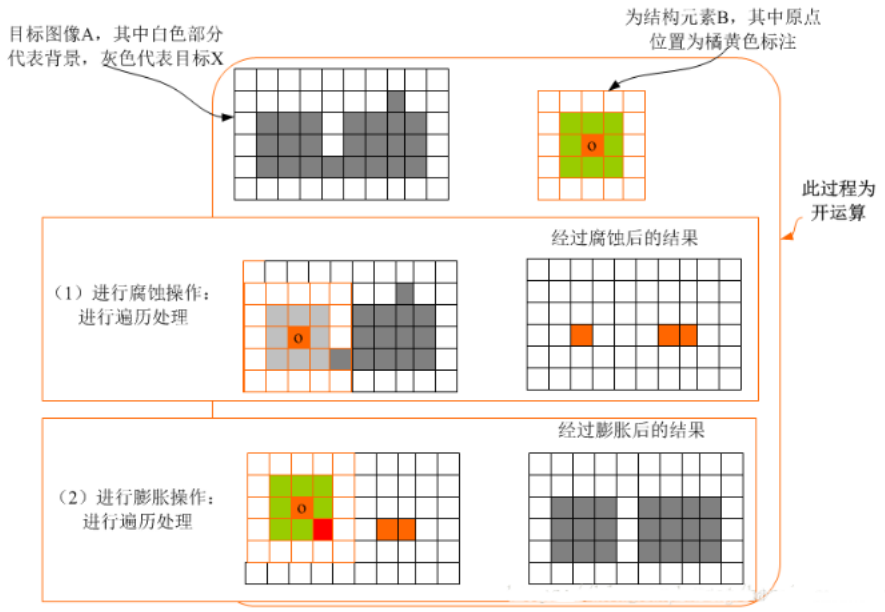


图 16-1 图像开运算原理图

设 A 是原始图像，B 是结构元素图像，则集合 A 被结构元素 B 做开运算，记为 A◦B，其定义为：

$$A \circ B = (A - B) \oplus B \tag{16-1}$$

换句话说，A 被 B 开运算就是 A 被 B 腐蚀后的结果再被 B 膨胀。图像开运算在 OpenCV 中主要使用函数 morphologyEx()，它是形态学扩展的一组函数，其函数原型如下：

```
dst = cv2.morphologyEx(src, cv2.MORPH_OPEN, kernel)
```

- src 表示原始图像
- cv2.MORPH_OPEN 表示图像进行开运算处理
- kernel 表示卷积核，可以用 numpy.ones()函数构建

图像开运算的代码如下所示：

```
# -*- coding: utf-8 -*-
```

```
# By：Eastmount
import cv2
import numpy as np

#读取图片
src = cv2.imread('test01.png',
cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((5,5), np.uint8)

#图像开运算
result = cv2.morphologyEx(src, cv2.MORPH_OPEN,
kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

输出结果如图 16-2 所示，左边为原始图像，右边为处理后的图像，可以看到原始图形中的噪声点被去除了部分。

图 16-2 图像开运算处理

但处理后的图像中仍然有部分噪声，如果想更彻底地去除，可以将卷积设置为 10×10 的模板，代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取图片
src                    =                    cv2.imread('test01.png',
```

```
cv2.IMREAD_UNCHANGED)


#设置卷积核
kernel = np.ones((10,10), np.uint8)


#图像开运算
result = cv2.morphologyEx(src, cv2.MORPH_OPEN,
kernel)


#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)


#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行结果如图 16-3 所示：

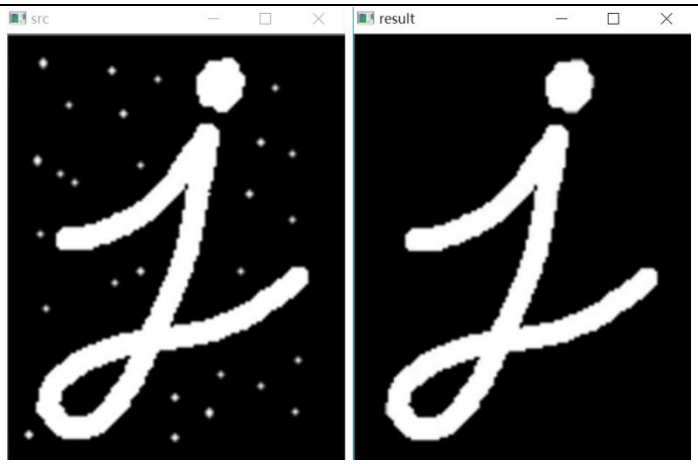


图 16-3 卷积 10×10 模板的开运算效果图

2.图像闭运算

图像闭运算是图像依次经过膨胀、腐蚀处理的过程，先膨胀后腐蚀有助于过滤前景物体内部的小孔或物体上的小黑点。其原理如图 16-4 所示：

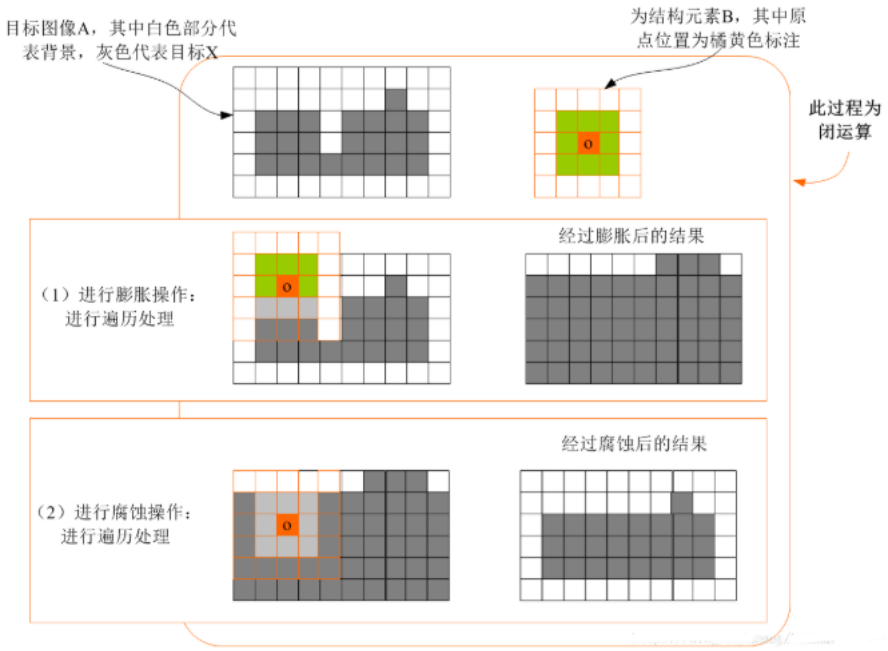


图 16-4 图像闭运算原理图

设 A 是原始图像，B 是结构元素图像，则集合 A 被结构元素 B 做开运算，

记为 A · B，其定义为：

$$A \bullet B = (A \oplus B) - B \qquad (16-2)$$

换句话说，A 被 B 闭运算就是 A 被 B 膨胀后的结果再被 B 腐蚀。图像开运算在 OpenCV 中主要使用函数 morphologyEx()，其函数原型如下：

dst = cv2.morphologyEx(src, cv2.MORPH_CLOSE, kernel)

- src 表示原始图像
- cv2.MORPH_CLOSE 表示图像进行闭运算处理
- kernel 表示卷积核，可以用 numpy.ones()函数构建

图像开运算的代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取图片
src = cv2.imread('test02.png',
cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((10,10), np.uint8)
```

```
#图像闭运算
result = cv2.morphologyEx(src, cv2.MORPH_CLOSE, kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 16-5 所示，它有效地去除了图像中间的小黑点（噪声）。

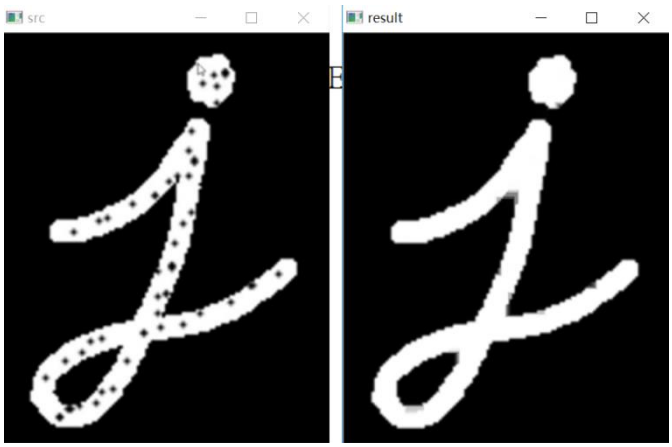
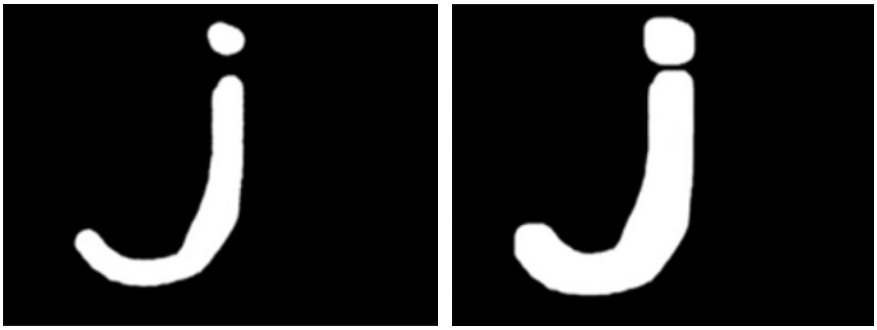


图 16-5 图像开运算处理

3.图像梯度运算

图像梯度运算是图像膨胀处理减去图像腐蚀处理后的结果，从而得到图像的轮廓，其原理如图 16-6 所示，（a）表示原始图像，（b）表示膨胀处理后的图像，（c）表示腐蚀处理后的图像，（d）表示图像梯度运算的效果图。



（a）原始图像　　（b）膨胀处理

（c）腐蚀处理　　（d）梯度运算

图 16-6 图像梯度运算原理

在 Python 中，图像梯度运算主要调用 morphologyEx()实现，其中参数 cv2.MORPH_GRADIENT 表示梯度处理，函数原型如下：

```
dst = cv2.morphologyEx(src, cv2.MORPH_GRADIENT, kernel)
```

- src 表示原始图像

- cv2.MORPH_GRADIENT 表示图像进行梯度运算处理
- kernel 表示卷积核，可以用 numpy.ones()函数构建

图像梯度运算的实现代码如下所示。

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np

#读取图片
src = cv2.imread('test03.png', cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((10,10), np.uint8)

#图像梯度运算
result = cv2.morphologyEx(src, cv2.MORPH_GRADIENT, kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)
```

```
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图像梯度运算处理的结果如图 16-7 所示，左边为原始图像，右边为处理后的效果图。

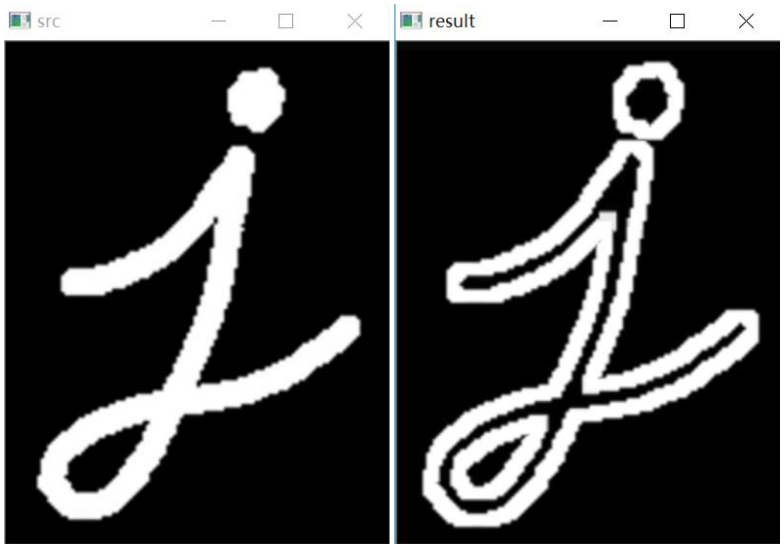


图 16-7 图像开运算处理

4.总结

本章主要介绍图像形态学处理，详细讲解了图像开运算、闭运算和梯度运算。数学形态学是一种应用于图像处理和模式识别领域的新方法，其基本思想是用具有一定形态的结构元素去量度和提取图像中对应形状以达到对图像分析和识别目的。

参考文献：

[1] 冈萨雷斯著，阮秋琦译. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[4] [7]Eastmount. [Python 图像处理] 九.形态学之图像开运算、闭运算、梯度运算 [EB/OL]. (2018-11-02). https://blog.csdn.net/Eastmount/article/details/83651172.

第 17 篇 图像形态学处理之顶帽运算和底帽运算

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像阈值化的开运算、闭运算和梯度运算。这篇文章将继续介绍顶帽运算和底帽运算。

1.图像顶帽运算

图像顶帽运算（top-hat transformation）又称为图像礼帽运算，它是用原始图像减去图像开运算后的结果，常用于解决由于光照不均匀图像分割出错的问题。其公式定义如下：

$$T_{hat}(A) = A - (A \circ B) \tag{17-1}$$

图像顶帽运算是用一个结构元通过开运算从一幅图像中删除物体，顶帽运算用于暗背景上的亮物体，它的一个重要用途是校正不均匀光照的影响。其效果图如图 17-1 所示。

（a）原始图像　　（b）开运算　　（c）顶帽运算

图 17-1 图像顶帽运算原理图

在 Python 中，图像顶帽运算主要调用 morphologyEx()实现，其中参数 cv2.MORPH_TOPHAT 表示顶帽处理，函数原型如下：

```
dst = cv2.morphologyEx(src, cv2.MORPH_TOPHAT, kernel)
```

- src 表示原始图像
- cv2.MORPH_TOPHAT 表示图像顶帽运算
- kernel 表示卷积核，可以用 numpy.ones()函数构建

假设存在一张光照不均匀的米粒图像，如图 17-2 所示，我们需要调用图像顶帽运算解决光照不均匀的问题。

图 17-2 光照不均匀的米粒图像

图像顶帽运算的 Python 代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np

#读取图片
src = cv2.imread('test01.png',
cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((10,10), np.uint8)

#图像顶帽运算
result = cv2.morphologyEx(src, cv2.MORPH_TOPHAT,
kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)
```

```
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其运行结果如图 17-3 所示。

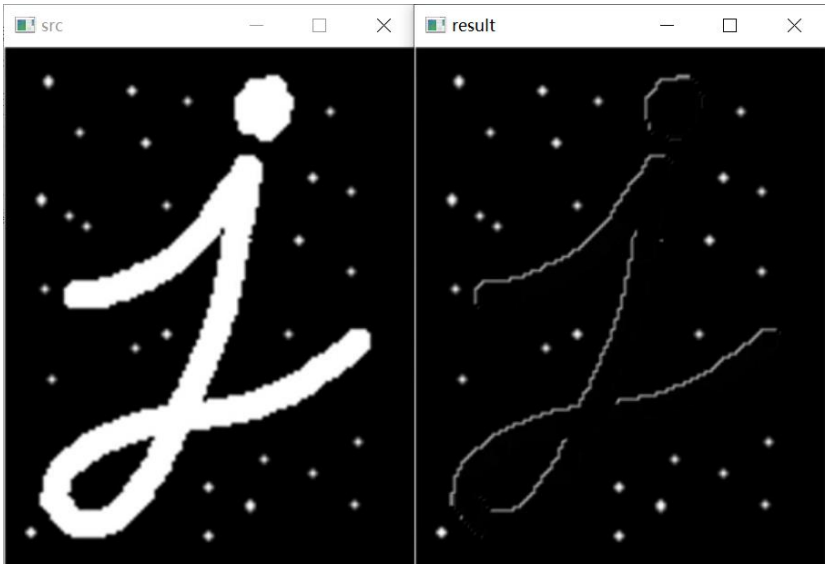


图 17-3 图像顶帽运算效果图

下图展示了“米粒”顶帽运算的效果图，可以看到顶帽运算后的图像删除了大部分非均匀背景，并将米粒与背景分离开来。

图 17-4 图像顶帽处理

为什么图像顶帽运算会消除光照不均匀的效果呢？通常可以利用灰度三维图来进行解释该算法。灰度三维图主要调用 Axes3D 包实现，对原图绘制灰度三维图的代码如下：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator,
FormatStrFormatter

#读取图像
img = cv.imread("test02.png")
img = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
imgd = np.array(img)        #image 类转 numpy

#准备数据
sp = img.shape
```

```
h = int(sp[0])            #图像高度(rows)
w = int(sp[1])            #图像宽度(colums) of image

#绘图初始处理
fig = plt.figure(figsize=(16,12))
ax = fig.gca(projection="3d")

x = np.arange(0, w, 1)
y = np.arange(0, h, 1)
x, y = np.meshgrid(x,y)
z = imgd
surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm)

#自定义 z 轴
ax.set_zlim(-10, 255)
ax.zaxis.set_major_locator(LinearLocator(10))    #设置 z 轴
网格线的疏密

#将 z 的 value 字符串转为 float 并保留 2 位小数
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'
))
```

```
# 设置坐标轴的 label 和标题
ax.set_xlabel('x', size=15)
ax.set_ylabel('y', size=15)
ax.set_zlabel('z', size=15)
ax.set_title("surface plot", weight='bold', size=20)

#添加右侧的色卡条
fig.colorbar(surf, shrink=0.6, aspect=8)
plt.show()
```

运行结果如图 17-5 所示，其中 x 表示原图像中的宽度坐标，y 表示原图像中的高度坐标，z 表示像素点(x, y)的灰度值。

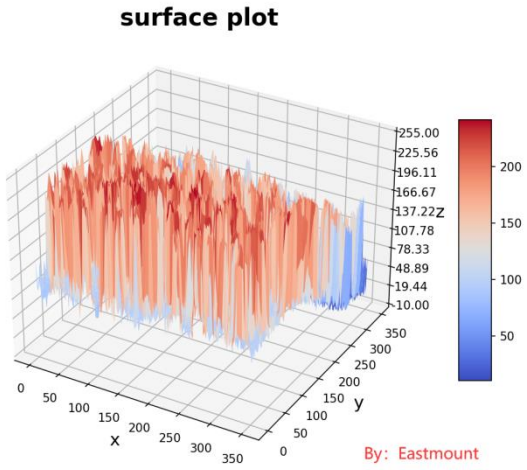


图 17-5 原始图像对应的灰度三维图

从图像中的像素走势显示了该图受各部分光照不均匀的影响，从而造成背景灰度不均现象，其中凹陷对应图像中灰度值比较小的区域。

通过图像白帽运算后的图像灰度三维图如图 17-6 所示，对应的灰度更集中于 10 至 100 区间，由此证明了不均匀的背景被大致消除了，有利于后续的阈值分割或图像分割。

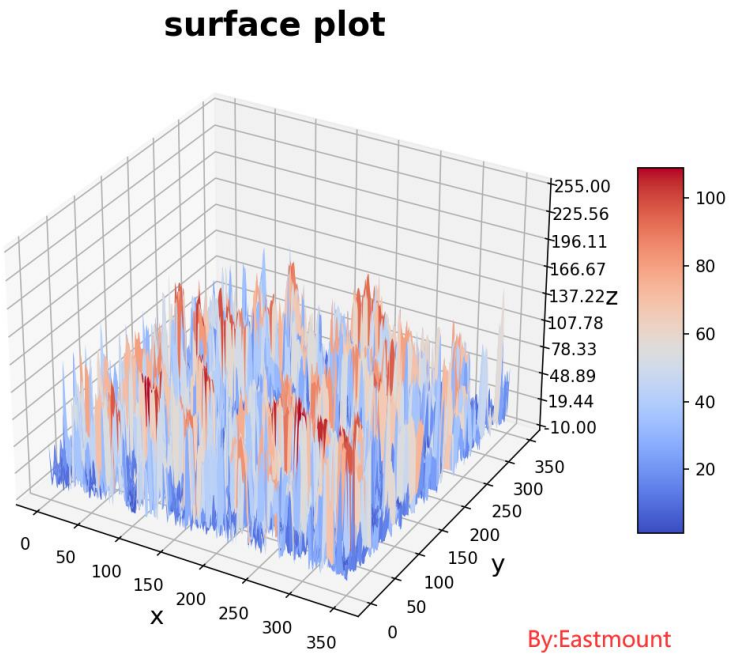


图 17-6 顶帽处理后的图像所对应的灰度三维图

绘制三维图增加的顶帽运算核心代码如下：

```
#读取图像
img = cv.imread("test02.png")
img = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
kernel = np.ones((10,10), np.uint8)
result = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)
imgd = np.array(result)       #image类转numpy
```

图 17-7 顶帽运算核心代码

2.图像底帽运算

图像底帽运算（bottom-hat transformation）又称为图像黑帽运算，它是用图像闭运算操作减去原始图像后的结果，从而获取图像内部的小孔或前景色

中黑点，也常用于解决由于光照不均匀图像分割出错的问题。其公式定义如下：

$$B_{hat}(A) = (A \bullet B) - A \qquad (17-2)$$

图像底帽运算是用一个结构元通过闭运算从一幅图像中删除物体，常用于校正不均匀光照的影响。其效果图如图 17-8 所示。

（b）原始图像　　（b）闭运算　　（c）底帽运算

图 17-8 图像底帽运算原理图

在 Python 中，图像底帽运算主要调用 morphologyEx()实现，其中参数 cv2.MORPH_BLACKHAT 表示底帽或黑帽处理，函数原型如下：

```
dst = cv2.morphologyEx(src, cv2.MORPH_BLACKHAT, kernel)
```

- src 表示原始图像
- cv2.MORPH_BLACKHAT 表示图像底帽或黑帽运算
- kernel 表示卷积核，可以用 numpy.ones()函数构建

Python 实现图像底帽运算的代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
```

```
#读取图片
src = cv2.imread('test02.png',
cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((10, 10), np.uint8)

#图像黑帽运算
result = cv2.morphologyEx(src, cv2.MORPH_BLACKHAT,
kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其运行结果如图 17-9 所示：

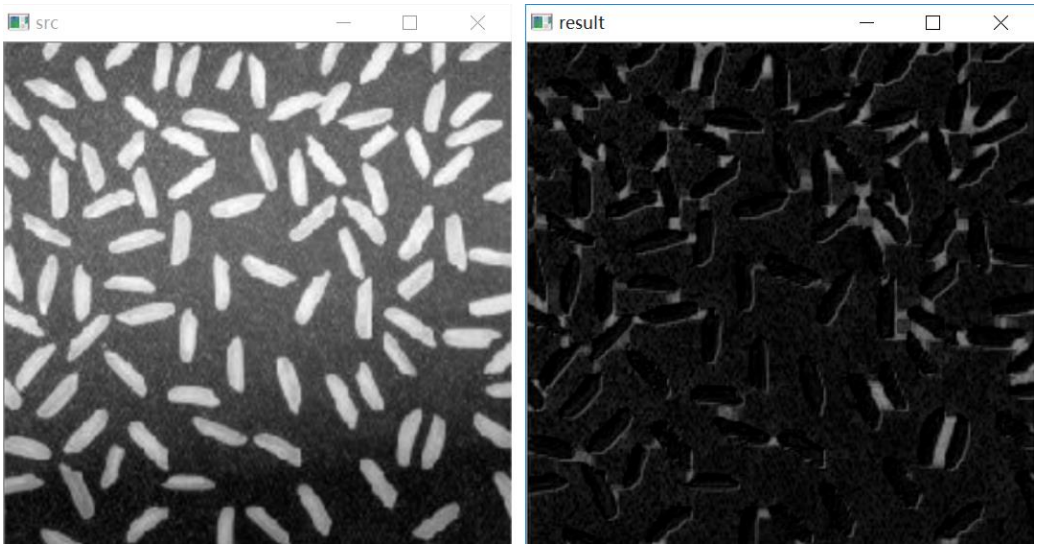


图 17-9 图像底帽处理

3.总结

该系列主要讲解了图像数学形态学知识，结合原理和代码详细介绍了图像腐蚀、图像膨胀、图像开运算和闭运算、图像顶帽运算和图像底帽运算等操作。这篇文章详细介绍了顶帽运算和底帽运算，它们将为后续的图像分割和图像识别提供有效支撑。

第 18 篇 图像直方图理论知识和绘制实现

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像形态学运算。这篇文章将进入新的系列，讲解图像直方图理论知识和绘制方法，本文将从 OpenCV 和 Matplotlib 两个方面介绍如何绘制直方图，这将为图像处理像素对比提供有效支撑。

1.图像直方图理论知识

灰度直方图是灰度级的函数，描述的是图像中每种灰度级像素的个数，反映图像中每种灰度出现的频率。假设存在一幅 6×6 像素的图像，接着统计其 1 至 6 灰度级的出现频率，并绘制如图 18-1 所示的柱状图，其中横坐标表示灰度级，纵坐标表示灰度级出现的频率[1-2]。

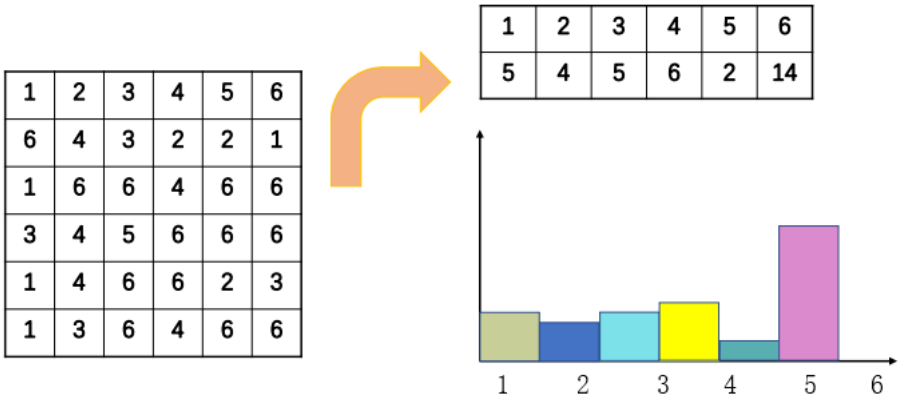


图 18-1 图像直方图统计原理

如果灰度级为 0-255（最小值 0 为黑色，最大值 255 为白色），同样可以绘制对应的直方图，如图 18-2 所示，左边是一幅灰度图像（Lena 灰度图），右边是对应各像素点的灰度级频率。

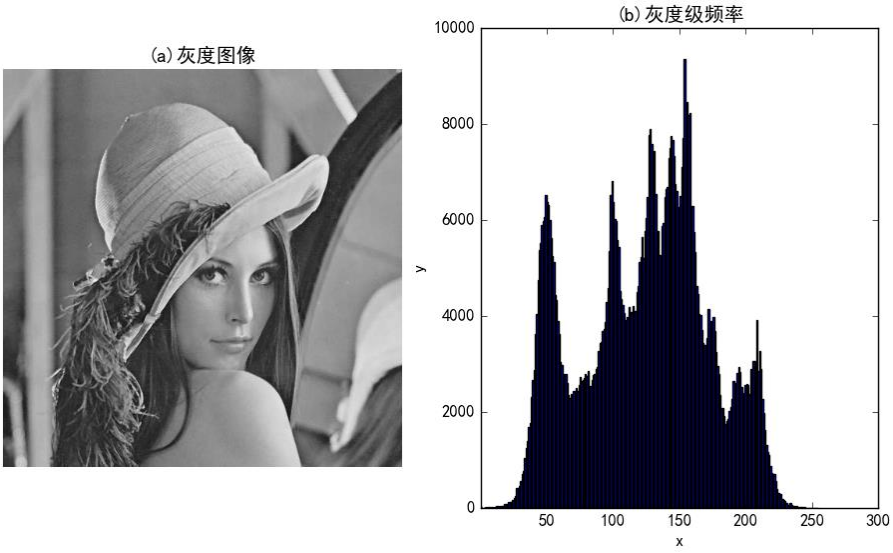


图 18-2 灰度直方图

为了让图像各灰度级的出现频数形成固定标准的形式，可以通过归一化方法对图像直方图进行处理，将待处理的原始图像转换成相应的标准形式[3]。假设变量 r 表示图像中像素灰度级，归一化处理后会将 r 限定在下述范围：

$$0 \le r \le 1 \tag{18-1}$$

在灰度级中，r为 0 时表示黑色，r为 1 时表示白色。对于一幅给定图像，每个像素值位于[0,1]区间之内，接着计算原始图像的灰度分布，用概率密度函数$P(r)$实现。为了更好地进行数字图像处理，必须引入离散形式。在离散形式下，用r_k表示离散灰度级，$P(r_k)$代替$P(r)$，并满足公式（18-2）。

$$\begin{aligned} &P_r(r_k) = \frac{n_k}{n} (0 \le r_k \le 1) \\ &k = 0,1,2,\Lambda\ ,l-1 \end{aligned} \tag{18-2}$$

公式中，n_k为图像中出现r_k这种灰度的像素数，n是图像中像素总数，$\frac{n_k}{n}$是概率论中的频数，l是灰度级总数（通常l为 256 级灰度）。接着在直角坐标系中做出r_k和$P(r_k)$的关系图，则成为灰度级的直方图[4]。

假设存在一幅 3×3 像素的图像，其像素值如公式（18-3）所示，则归一化直方图的步骤如下：

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 3 & 5 \\ 5 & 1 & 4 \end{bmatrix} \tag{18-3}$$

首先统计各灰度级对应的像素个数。用x数组统计像素点的灰度级，y数组统计具有该灰度级的像素个数。其中，灰度为 1 的像素共 3 个，灰度为 2 的像素共 1 个，灰度为 3 的像素共 2 个，灰度为 4 的像素共 1 个，灰度为 5 的像素共 2 个。

$$\begin{aligned} x &= [1,2,3,4,5] \\ y &= [3,1,2,1,2] \end{aligned} \tag{18-4}$$

接着统计总像素个数，如公式（18-5）所示。

$$n=(3+1+2+1+2)=9 \tag{18-5}$$

最后统计各灰度级的出现概率，通过公式（18-6）进行计算，其结果如下：

$$p=\frac{y}{n}=[\frac{3}{9},\frac{1}{9},\frac{2}{9},\frac{1}{9},\frac{2}{9}] \tag{18-6}$$

绘制的归一化图行如图18-3所示，横坐标表示图像中各个像素点的灰度级，纵坐标表示出现这个灰度级的概率。

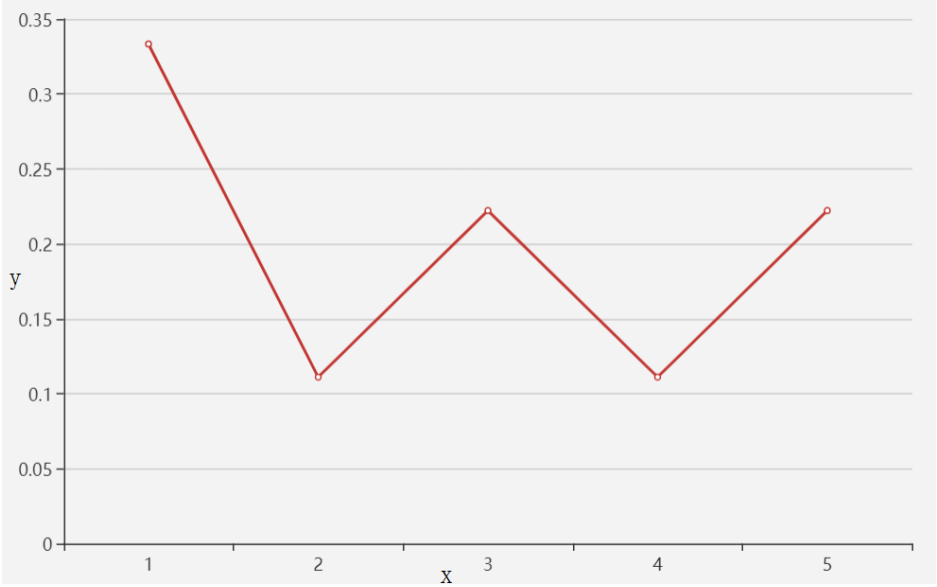


图 18-3 图像直方图归一化操作结果

直方图被广泛应用于计算机视觉领域，在使用边缘和颜色确定物体边界时，通过直方图能更好地选择边界阈值，进行阈值化处理。同时，直方图对物体与背景有较强对比的景物的分割特别有用，可以应用于检测视频中场景的变换及图像中的兴趣点。

2.OpenCV 绘制直方图

首先讲解使用 OpenCV 库绘制直方图的方法。在 OpenCV 中可以使用

calcHist()函数计算直方图，计算完成之后采用 OpenCV 中的绘图函数，如绘制矩形的 rectangle()函数，绘制线段的 line()函数来完成。其中，cv2.calcHist()的函数原型及常见六个参数如下：

hist = cv2.calcHist(images, channels, mask, histSize, ranges, accumulate)

- hist 表示直方图，返回一个二维数组
- images 表示输入的原始图像
- channels 表示指定通道，通道编号需要使用中括号，输入图像是灰度图像时，它的值为[0]，彩色图像则为[0]、[1]、[2]，分别表示蓝色（B）、绿色（G）、红色（R）
- mask 表示可选的操作掩码。如果要统计整幅图像的直方图，则该值为 None；如果要统计图像的某一部分直方图时，需要掩码来计算
- histSize 表示灰度级的个数，需要使用中括号，比如[256]
- ranges 表示像素值范围，比如[0, 255]
- accumulate 表示累计叠加标识，默认为 false，如果被设置为 true，则直方图在开始分配时不会被清零，该参数允许从多个对象中计算单个直方图，或者用于实时更新直方图；多个直方图的累积结果用于对一组图像的直方图计算

接下来的代码是计算图像各灰度级的大小、形状及频数，接着调用 plot()函数绘制直方图曲线。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

#读取图像
src = cv2.imread('lena-hd.png')

#计算 256 灰度级的图像直方图
hist = cv2.calcHist([src], [0], None, [256], [0,255])

#输出直方图大小、形状、数量
print(hist.size)
print(hist.shape)
print(hist)

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']
```

```
#显示原始图像和绘制的直方图
plt.subplot(121)
plt.imshow(src, 'gray')
plt.axis('off')
plt.title("(a)Lena 灰度图像")

plt.subplot(122)
plt.plot(hist, color='r')
plt.xlabel("x")
plt.ylabel("y")
plt.title("(b)直方图曲线")
plt.show()
```

上述代码绘制的“Lena”灰度图像所对应的直方图曲线如图 18-4 所示，图 18-4(a)表示原图像，图 18-4(b)表示对应的灰度直方图曲线。

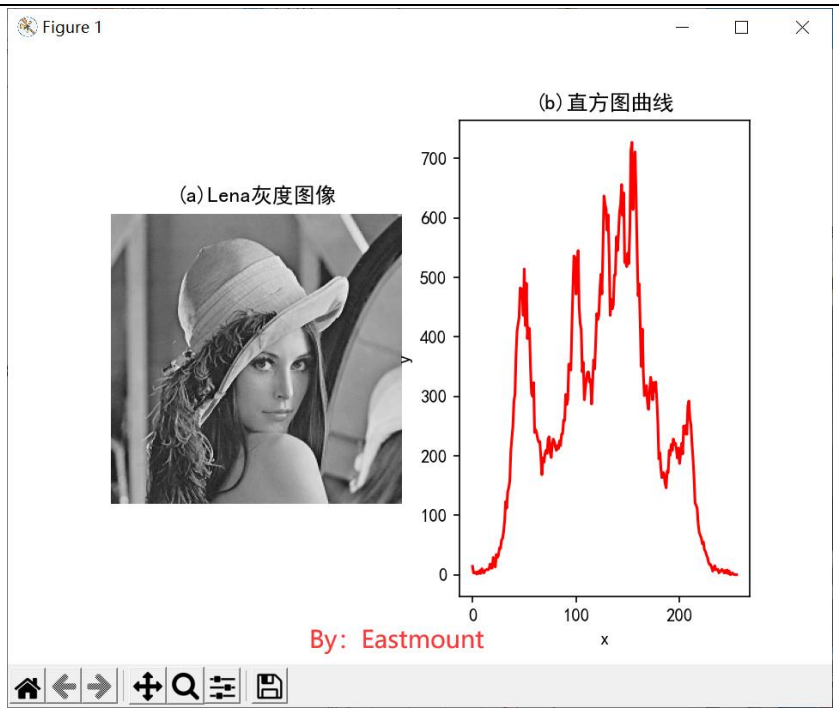


图 18-4 OpenCV 绘制 Lena 灰度直方图曲线

同时输出直方图的大小、形状及数量，如下所示：

```
256
(256L, 1L)
[[7.000e+00]
 [1.000e+00]
 [0.000e+00]
 [6.000e+00]
 [2.000e+00]
 ....
 [1.000e+00]
 [3.000e+00]
```

```
 [2.000e+00]
 [1.000e+00]
 [0.000e+00]]
```

彩色图像调用 OpenCV 绘制直方图的算法与灰度图像一样，只是从 B、G、R 三个放量分别进行计算及绘制，具体代码如下所示。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

#读取图像
src = cv2.imread('lena.png')

#转换为 RGB 图像
img_rgb = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)

#计算直方图
histb = cv2.calcHist([src], [0], None, [256], [0,255])
histg = cv2.calcHist([src], [1], None, [256], [0,255])
```

```
histr = cv2.calcHist([src], [2], None, [256], [0,255])


#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']


#显示原始图像和绘制的直方图
plt.subplot(121)
plt.imshow(img_rgb, 'gray')
plt.axis('off')
plt.title("(a)Lena 原始图像")
plt.subplot(122)
plt.plot(histb, color='b')
plt.plot(histg, color='g')
plt.plot(histr, color='r')
plt.xlabel("x")
plt.ylabel("y")
plt.title("(b)直方图曲线")
plt.show()
```

最终绘制的“Lena”彩色图像及其对应的彩色直方图曲线如图 18-5 所示，其中图 18-5(a)表示 Lena 原始图像，图 18-5(b)表示对应的彩色直方图曲线。

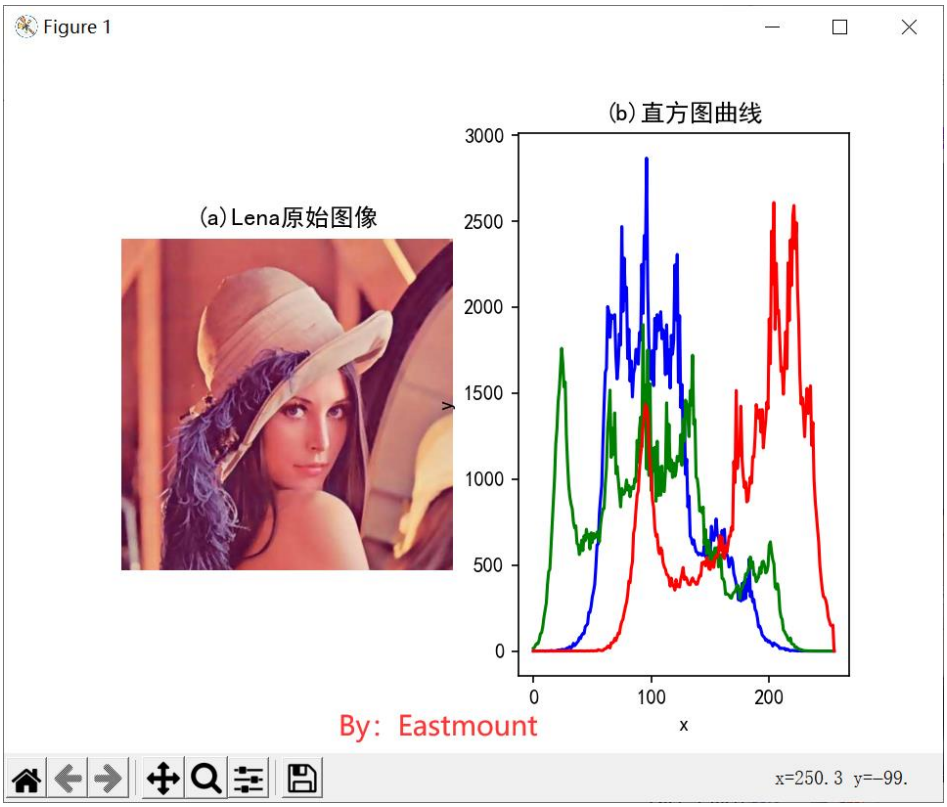


图 18-5 OpenCV 绘制彩色 Lena 图及其直方图曲线

3.Matplotlib 绘制直方图

Matplotlib 是 Python 强大的数据可视化工具，主要用于绘制各种 2D 图形。本小节 Python 绘制直方图主要调用 matplotlib.pyplot 库中 hist()函数实现，它会根据数据源和像素级绘制直方图。其函数主要包括五个常用的参数，如下所示：

```
n, bins, patches = plt.hist(arr, bins=50, normed=1, facecolor='green', alpha=0.75)
```

- arr 表示需要计算直方图的一维数组
- bins 表示直方图显示的柱数，可选项，默认值为 10

- normed 表示是否将得到的直方图进行向量归一化处理，默认值为 0
- facecolor 表示直方图颜色
- alpha 表示透明度
- n 为返回值，表示直方图向量
- bins 为返回值，表示各个 bin 的区间范围
- patches 为返回值，表示返回每个 bin 里面包含的数据，是一个列表

图像直方图的 Python 实现代码如下所示，该示例主要是通过 matplotlib.pyplot 库中的 hist()函数绘制的。注意，读取的“lena-hd.png”图像的像素为二维数组，而 hist()函数的数据源必须是一维数组，通常需要通过函数 ravel()拉直图像。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图像
src = cv2.imread('lena-hd.png')
```

```
#绘制直方图
plt.hist(src.ravel(), 256)
plt.xlabel("x")
plt.ylabel("y")
plt.show()

#显示原始图像
cv2.imshow("src", src)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

读取显示的“lena”灰度图像如图 18-6 所示。

图 18-6 Lena 原始灰度图像

最终的灰度直方图如图 18-7 所示，它将 Lena 图 256 级灰度和各个灰度级的频数绘制出来，其中 x 轴表示图像的 256 级灰度，y 轴表示各个灰度级的频数。

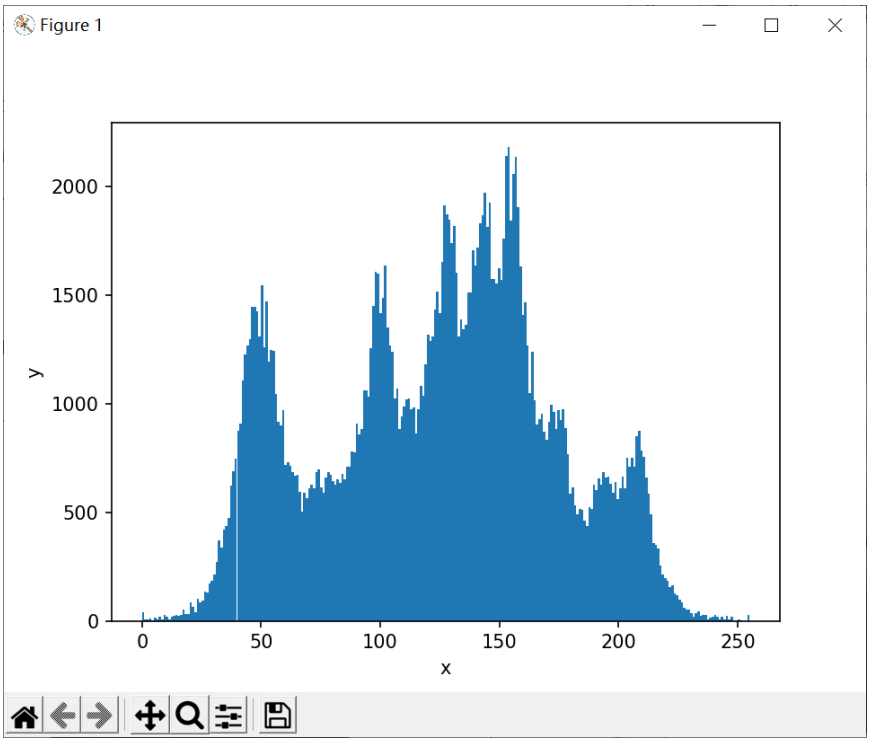


图 18-7 Lena 灰度图像对应的直方图

如果调用下列函数，则绘制的直方图是经过标准化处理，并且颜色为绿色、透明度为 0.75 的直方图，如图 18-8 所示。

❖ plt.hist(src.ravel(), bins=256, density=1, facecolor='green', alpha=0.75)

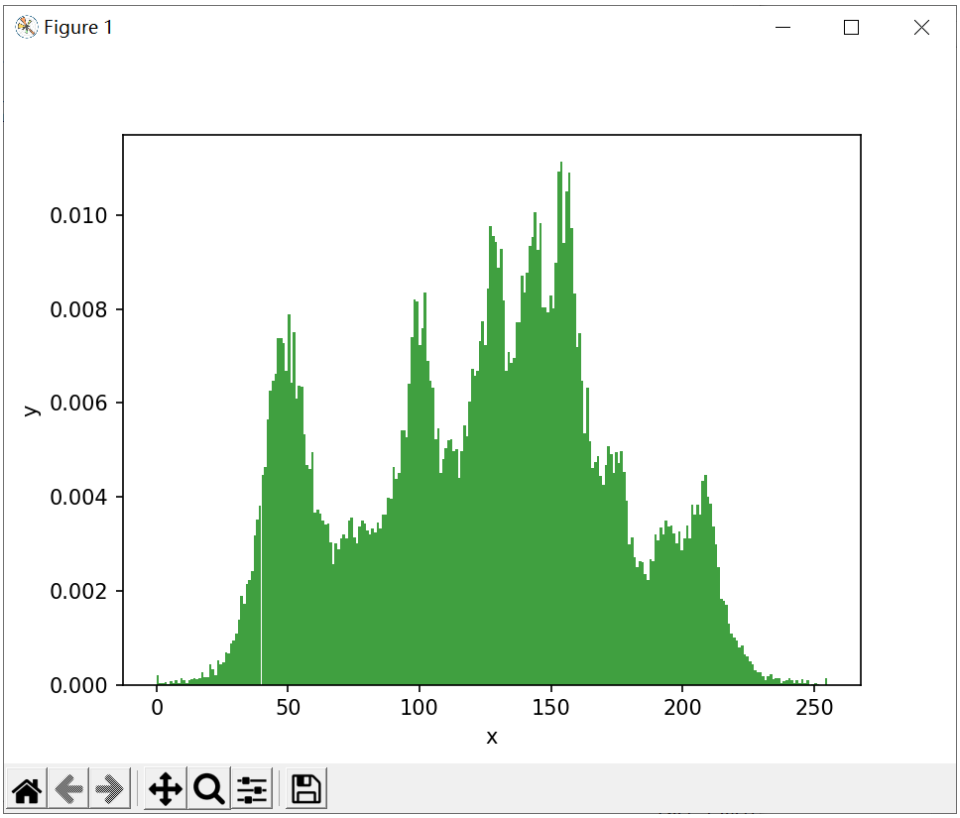


图 18-8 Lena 灰度图像归一化处理后的直方图

彩色直方图是高维直方图的特例，它统计彩色图片 RGB 各分量出现的频率，即彩色概率分布信息。彩色图片的直方图和灰度直方图一样，只是分别画出三个通道的直方图，然后再进行叠加，其代码如下所示。Lena 彩色原始图像如图 18-9 所示。

图 18-9 Lena 彩色原始图像

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
src = cv2.imread('Lena.png')

#获取 BGR 三个通道的像素值
b, g, r = cv2.split(src)
```

```
#绘制直方图
plt.figure("Lena")
#蓝色分量
plt.hist(b.ravel(), bins=256, density=1, facecolor='b',
edgecolor='b', alpha=0.75)
#绿色分量
plt.hist(g.ravel(), bins=256, density=1, facecolor='g',
edgecolor='g', alpha=0.75)
#红色分量
plt.hist(r.ravel(), bins=256, density=1, facecolor='r',
edgecolor='r', alpha=0.75)
plt.xlabel("x")
plt.ylabel("y")
plt.show()

#显示原始图像
cv2.imshow("src", src)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

绘制的彩色直方图如图 18-10 所示，包括红色、绿色、蓝色三种对比。

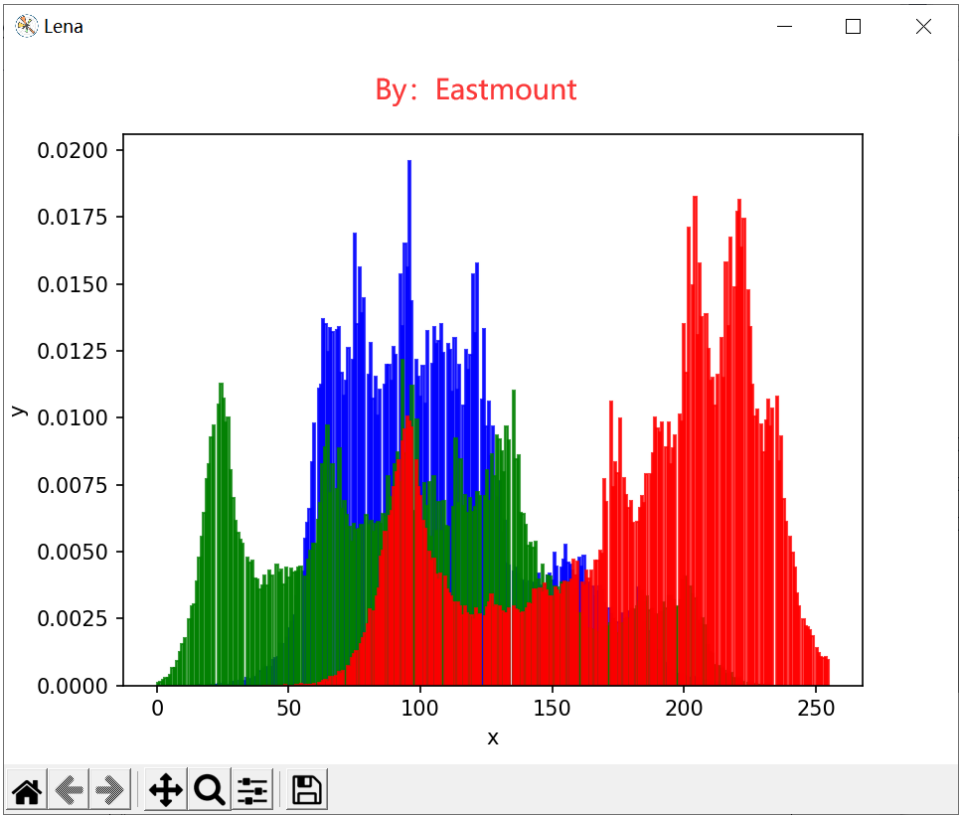


图 18-10 Lena 彩色原始图像的直方图

如果希望将三个颜色分量的柱状图分开绘制并进行对比，则使用下面的代码实现，调用 plt.figure(figsize=(8, 6))函数绘制窗口，以及 plt.subplot()函数分别绘制 4 个子图。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
```

```
#读取图像
src = cv2.imread('lena.png')


#转换为 RGB 图像
img_rgb = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)


#获取 BGR 三个通道的像素值
b, g, r = cv2.split(src)
print(r,g,b)


plt.figure(figsize=(8, 6))


#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']


#原始图像
plt.subplot(221)
plt.imshow(img_rgb)
plt.axis('off')
plt.title("(a)原图像")
```

```
#绘制蓝色分量直方图
plt.subplot(222)
plt.hist(b.ravel(), bins=256, density=1, facecolor='b',
edgecolor='b', alpha=0.75)
plt.xlabel("x")
plt.ylabel("y")
plt.title("(b)蓝色分量直方图")

#绘制绿色分量直方图
plt.subplot(223)
plt.hist(g.ravel(), bins=256, density=1, facecolor='g',
edgecolor='g', alpha=0.75)
plt.xlabel("x")
plt.ylabel("y")
plt.title("(c)绿色分量直方图")

#绘制红色分量直方图
plt.subplot(224)
plt.hist(r.ravel(), bins=256, density=1, facecolor='r',
edgecolor='r', alpha=0.75)
plt.xlabel("x")
```

```
plt.ylabel("y")
plt.title("(d)红色分量直方图")
plt.show()
```

最终输出的图形如图 18-11 所示,，图 18-11(a)表示原图像，图 18-11(b)表示蓝色分量直方图，图 18-11(c)表示绿色分量直方图，图 18-11(d)表示红色分类直方图。

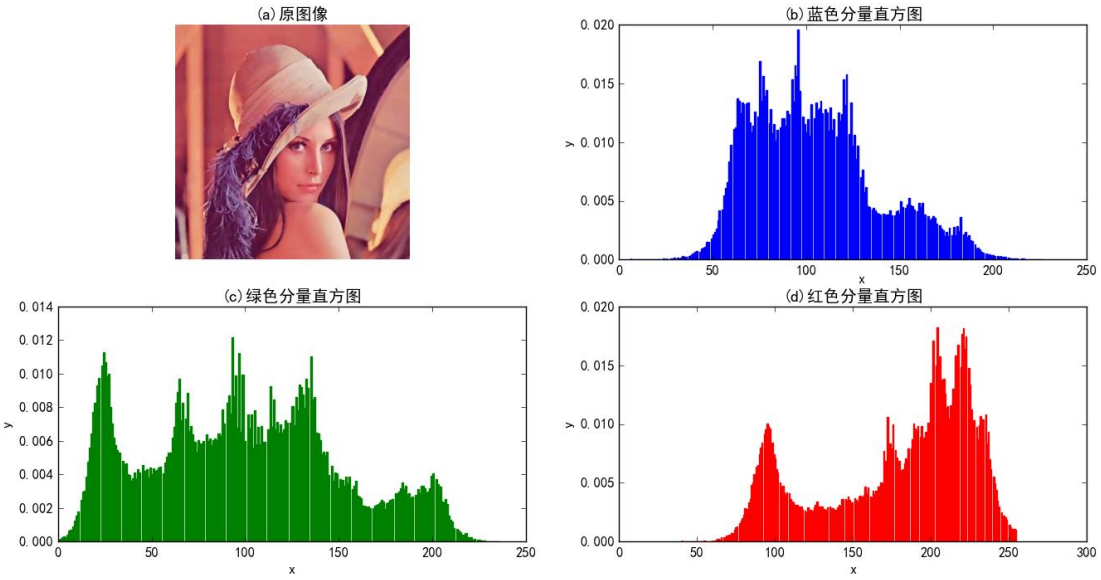


图 18-11 Lena 直方图对比

4.总结

本文主要讲解图像直方图理论知识以及直方图绘制方法，并且包括 Matplotlib 和 OpenCV 两种统计及绘制方法。灰度直方图是灰度级的函数，描述的是图像中每种灰度级像素的个数，反映图像中每种灰度出现的频率。这篇文章的知识点将为后续图像处理和图像运算对比提供支撑。

参考文献：

[1] 冈萨雷斯. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社, 2013.

[2] 张恒博，欧宗瑛. 一种基于色彩和灰度直方图的图像检索方法[J]. 计算机工程, 2004.

[3] Eastmount. [数字图像处理] 四.MFC 对话框绘制灰度直方图[EB/OL]. (2015-05-31). https://blog.csdn.net/eastmount/article/details/46237463.

[4] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社, 2008.

[5] Eastmount. [Python 图像处理] 十一.灰度直方图概念及 OpenCV 绘制直方图[EB/OL]. (2018-11-06). https://blog.csdn.net/Eastmount/article/details/83758402.

第 19 篇 图像灰度直方图对比分析

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面的文章详细介绍了图像灰度变换和阈值变换，本篇文章将结合直方图分别对比图像灰度变换前后的变化，方便读者更清晰地理解灰度变换和阈值变换。

1.灰度增强直方图对比

图像灰度上移变换使用的表达式为：

- $D_B=D_A+50$

该算法将实现图像灰度值的上移，从而提升图像的亮度，结合直方图对比的实现代码如下所示。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
#读取图像
img = cv2.imread('lena-hd.png')

#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]
result = np.zeros((height, width), np.uint8)

#图像灰度上移变换 DB=DA+50
for i in range(height):
    for j in range(width):
        if (int(grayImage[i,j]+50) > 255):
            gray = 255
        else:
            gray = int(grayImage[i,j]+50)
        result[i,j] = np.uint8(gray)
```

```
#计算原图的直方图
hist = cv2.calcHist([img], [0], None, [256], [0,255])

#计算灰度变换的直方图
hist_res = cv2.calcHist([result], [0], None, [256], [0,255])

#原始图像
plt.figure(figsize=(8, 6))
plt.subplot(221), plt.imshow(img, 'gray'), plt.title("(a)"),
plt.axis('off')

#绘制掩膜
plt.subplot(222), plt.plot(hist), plt.title("(b)"), plt.xlabel("x"),
plt.ylabel("y")

#绘制掩膜设置后的图像
plt.subplot(223), plt.imshow(result, 'gray'), plt.title("(c)"),
plt.axis('off')

#绘制直方图
plt.subplot(224), plt.plot(hist_res), plt.title("(d)"),
```

```
plt.xlabel("x"), plt.ylabel("y")
plt.show()
```

其运行结果如图 19-1 所示，其中（a）表示原始图像，（b）表示对应的灰度直方图，（c）表示灰度上移后的图像，（d）是对应的直方图。对比发现，图 19-1（d）比图 19-1（b）的灰度级整体高了 50，曲线整体向右平移了 50 个单位。

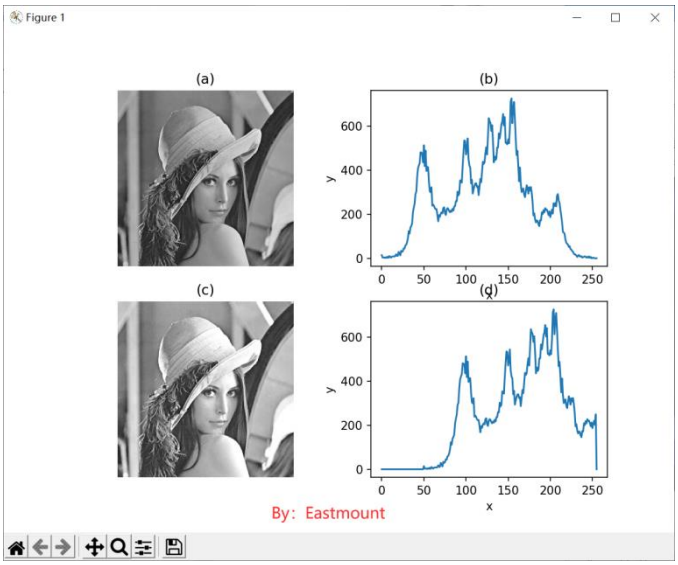


图 19-1 图像灰度上移直方图对比

2.灰度减弱直方图对比

该算法将减弱图像的对比度，使用的表达式为：

- $D_B=D_A\times 0.8$

Python 结合直方图实现灰度对比度减弱的代码如下所示。

```
# -*- coding: utf-8 -*-
# By：Eastmount
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图像
img = cv2.imread('lena-hd.png')


#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]
result = np.zeros((height, width), np.uint8)


#图像对比度减弱变换  DB=DA×0.8
for i in range(height):
    for j in range(width):
        gray = int(grayImage[i,j]*0.8)
        result[i,j] = np.uint8(gray)
```

```
#计算原图的直方图
hist = cv2.calcHist([img], [0], None, [256], [0,255])

#计算灰度变换的直方图
hist_res = cv2.calcHist([result], [0], None, [256], [0,255])

#原始图像
plt.figure(figsize=(8, 6))
plt.subplot(221), plt.imshow(img, 'gray'), plt.title("(a)"),
plt.axis('off')

#绘制掩膜
plt.subplot(222), plt.plot(hist), plt.title("(b)"), plt.xlabel("x"),
plt.ylabel("y")

#绘制掩膜设置后的图像
plt.subplot(223), plt.imshow(result, 'gray'), plt.title("(c)"),
plt.axis('off')

#绘制直方图
plt.subplot(224), plt.plot(hist_res), plt.title("(d)"),
```

```
plt.xlabel("x"), plt.ylabel("y")
plt.show()
```

其运行结果如图 19-2 所示，其中（a）和（b）表示原始图像和对应的灰度直方图，（c）和（d）表示灰度减弱或对比度缩小的图像及对应的直方图。图 19-2（d）比图 19-2（b）的灰度级整体缩小了 0.8 倍，绘制的曲线更加密集。

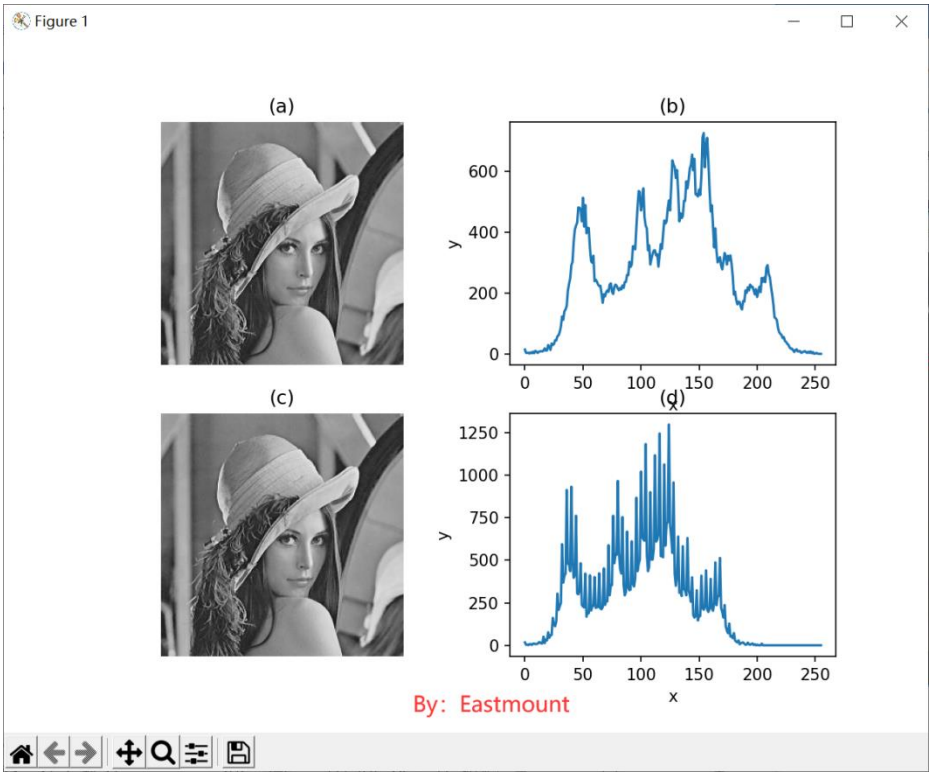


图 19-2 图像灰度减弱直方图对比

3.图像反色直方图对比

该算法将图像的颜色反色，对原图像的像素值进行反转，即黑色变为白色，白色变为黑色，使用的表达式为：

- $D_B=255-D_A$

实现代码如下所示。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('lena-hd.png')

#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]
result = np.zeros((height, width), np.uint8)

#图像灰度反色变换 DB=255-DA
for i in range(height):
    for j in range(width):
```

```
            gray = 255 - grayImage[i,j]
            result[i,j] = np.uint8(gray)

#计算原图的直方图
hist = cv2.calcHist([img], [0], None, [256], [0,255])

#计算灰度变换的直方图
hist_res = cv2.calcHist([result], [0], None, [256], [0,255])

#原始图像
plt.figure(figsize=(8, 6))
plt.subplot(221), plt.imshow(img, 'gray'), plt.title("(a)"),
plt.axis('off')

#绘制掩膜
plt.subplot(222), plt.plot(hist), plt.title("(b)"), plt.xlabel("x"),
plt.ylabel("y")

#绘制掩膜设置后的图像
plt.subplot(223), plt.imshow(result, 'gray'), plt.title("(c)"),
plt.axis('off')
```

```
#绘制直方图
plt.subplot(224), plt.plot(hist_res), plt.title("(d)"),
plt.xlabel("x"), plt.ylabel("y")
plt.show()
```

其运行结果如图 19-3 所示，其中（a）和（b）表示原始图像和对应的灰度直方图，（c）和（d）表示灰度反色变换图像及对应的直方图。图 19-3（d）与图 19-3（b）是反相对称的，整个灰度值满足 $D_B=255-D_A$ 表达式。

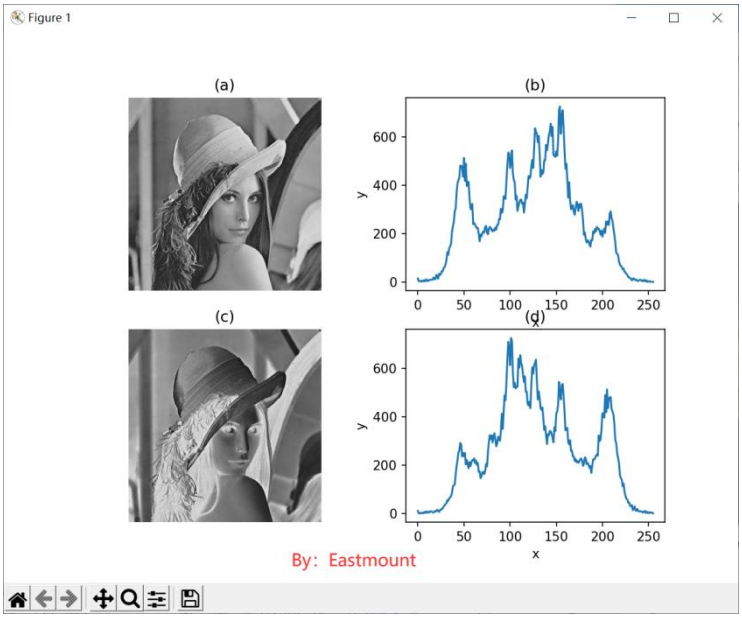


图 19-3 图像灰度反色变换直方图对比

4.图像对数变换直方图对比

该算法将增加低灰度区域的对比度，从而增强暗部的细节，使用的表达式为：

- $D_B = c \times \log(1+D_A)$

下面代码实现了图像灰度的对数变换及直方图对比。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('lena-hd.png')

#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]
result = np.zeros((height, width), np.uint8)

#图像灰度对数变换
for i in range(height):
    for j in range(width):
```

```
            gray = 42 * np.log(1.0 + grayImage[i,j])
            result[i,j] = np.uint8(gray)

#计算原图的直方图
hist = cv2.calcHist([img], [0], None, [256], [0,255])

#计算灰度变换的直方图
hist_res = cv2.calcHist([result], [0], None, [256], [0,255])

#原始图像
plt.figure(figsize=(8, 6))
plt.subplot(221), plt.imshow(img, 'gray'), plt.title("(a)"),
plt.axis('off')

#绘制原始图像直方图
plt.subplot(222), plt.plot(hist), plt.title("(b)"), plt.xlabel("x"),
plt.ylabel("y")

#灰度变换后的图像
plt.subplot(223), plt.imshow(result, 'gray'), plt.title("(c)"),
plt.axis('off')
```

```
#灰度变换图像的直方图
plt.subplot(224), plt.plot(hist_res), plt.title("(d)"),
plt.xlabel("x"), plt.ylabel("y")
plt.show()
```

其运行结果如图 19-4 所示，其中（a）和（b）表示原始图像和对应的灰度直方图，（c）和（d）表示灰度对数变换图像及对应的直方图。

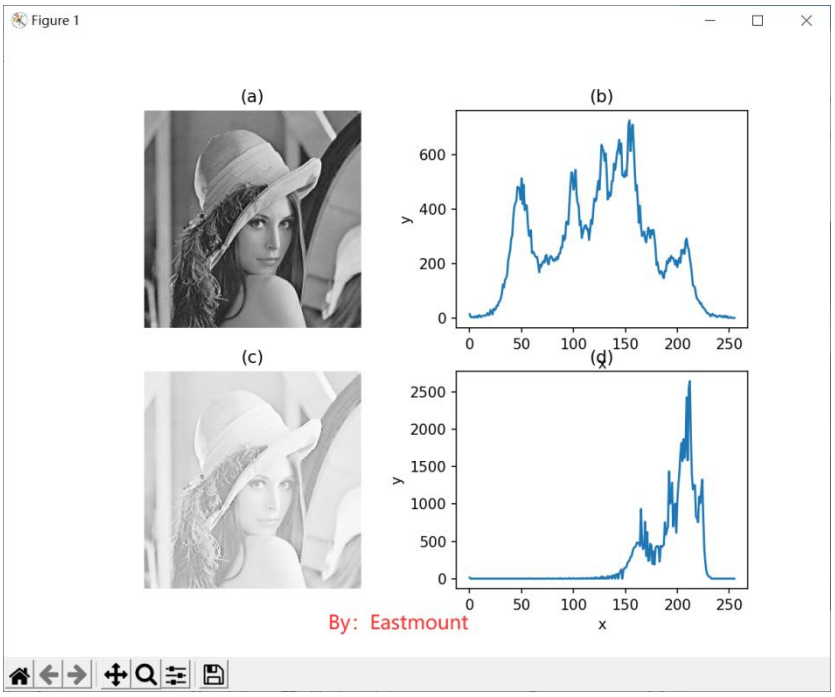


图 19-4 图像灰度对数变换直方图对比

5.图像阈值化处理直方图对比

该算法原型为 threshold(Gray,127,255,cv2.THRESH_BINARY)，当前像素点的灰度值大于 thresh 阈值时（如 127），其像素点的灰度值设定为

最大值（如 9 位灰度值最大为 255）；否则，像素点的灰度值设置为 0。二进制阈值化处理及直方图对比的 Python 代码如下所示。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('lena-hd.png')

#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#二进制阈值化处理
r, result = cv2.threshold(grayImage, 127, 255,
cv2.THRESH_BINARY)

#计算原图的直方图
hist = cv2.calcHist([img], [0], None, [256], [0,256])
```

```
#计算阈值化处理的直方图
hist_res = cv2.calcHist([result], [0], None, [256], [0,256])

#原始图像
plt.figure(figsize=(8, 6))
plt.subplot(221), plt.imshow(img, 'gray'), plt.title("(a)"),
plt.axis('off')

#绘制原始图像直方图
plt.subplot(222), plt.plot(hist), plt.title("(b)"), plt.xlabel("x"),
plt.ylabel("y")

#阈值化处理后的图像
plt.subplot(223), plt.imshow(result, 'gray'), plt.title("(c)"),
plt.axis('off')

#阈值化处理图像的直方图
plt.subplot(224), plt.plot(hist_res), plt.title("(d)"),
plt.xlabel("x"), plt.ylabel("y")
plt.show()
```

其运行结果如图 19-5 所示，其中（a）和（b）表示原始图像和对应的灰度

直方图，（c）和（d）表示图像阈值化处理及对应的直方图，图 19-5（d）中可以看到，灰度值仅仅分布于 0（黑色）和 255（白色）两种灰度级。

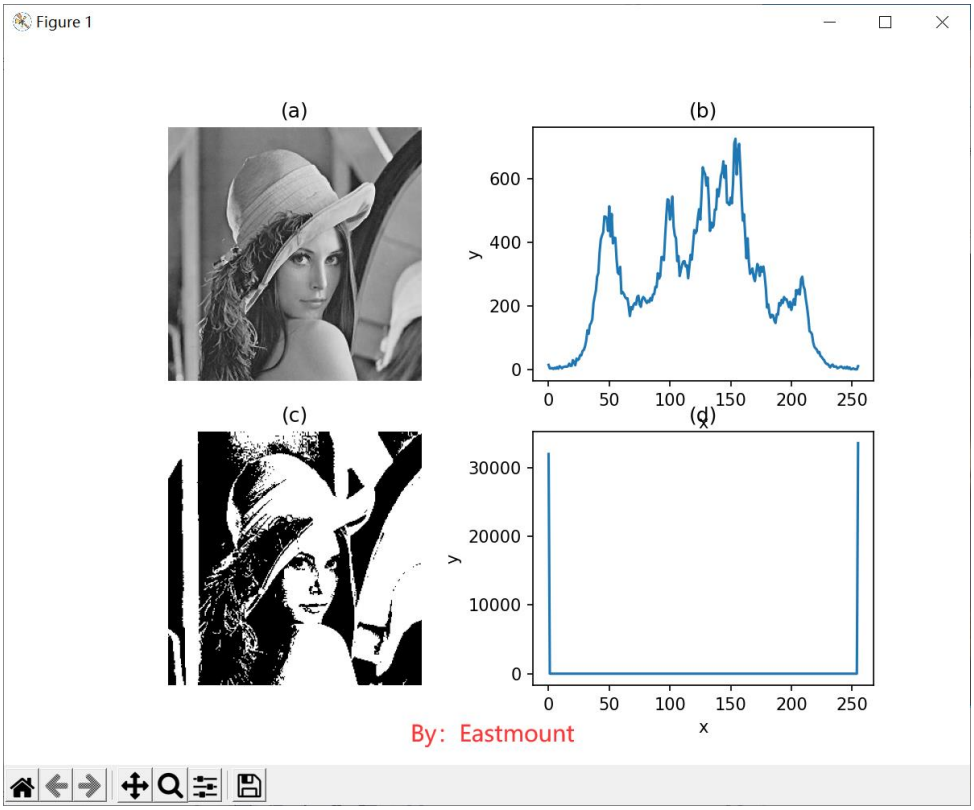


图 19-5 图像阈值化处理直方图对比

6.总结

本文主要讲解图像直方图理论知识以及直方图绘制方法，包括灰度增强直方图对比、灰度减弱直方图对比、图像反色直方图对比、图像对数变换直方图对比、图像阈值化处理直方图对比。灰度直方图是灰度级的函数，描述的是图像中每种灰度级像素的个数，反映图像中每种灰度出现的频率。这篇文章的知识点将为后续图像处理和图像运算对比提供支撑。

第 20 篇 图像掩膜直方图和 HS 直方图

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像灰度直方图对比分析。这篇文章将继续讲解图像掩膜直方图和 HS 直方图，并分享一个通过直方图判断白天与黑夜的案例。

1.图像掩膜直方图

如果要统计图像的某一部分直方图，就需要使用掩码（蒙板）来进行计算。假设将要统计的部分设置为白色，其余部分设置为黑色，然后使用该掩膜进行直方图绘制，其完整代码如下所示。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
```

```
#读取图像
img = cv2.imread('luo.png')

#转换为 RGB 图像
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#设置掩膜
mask = np.zeros(img.shape[:2], np.uint8)
mask[100:300, 100:300] = 255
masked_img = cv2.bitwise_and(img, img, mask=mask)

#图像直方图计算
hist_full = cv2.calcHist([img], [0], None, [256], [0,256]) #通
道[0]-灰度图

#图像直方图计算(含掩膜)
hist_mask = cv2.calcHist([img], [0], mask, [256], [0,256])

plt.figure(figsize=(8, 6))
```

```
#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#原始图像
plt.subplot(221)
plt.imshow(img_rgb, 'gray')
plt.axis('off')
plt.title("(a)原始图像")

#绘制掩膜
plt.subplot(222)
plt.imshow(mask, 'gray')
plt.axis('off')
plt.title("(b)掩膜")

#绘制掩膜设置后的图像
plt.subplot(223)
plt.imshow(masked_img, 'gray')
plt.axis('off')
plt.title("(c)图像掩膜处理")
```

```
#绘制直方图
plt.subplot(224)
plt.plot(hist_full)
plt.plot(hist_mask)
plt.title("(d)直方图曲线")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

其运行结果如图 20-1 所示，它使用了一个 200×200 像素的掩膜进行实验。其中图 20-1(a)表示原始图像，图 20-1(b)表示 200×200 像素的掩膜，图 20-1(c)表示原始图像进行掩膜处理，图 20-1(d)表示直方图曲线，蓝色曲线为原始图像的灰度值直方图分布情况，绿色波动更小的曲线为掩膜直方图曲线。

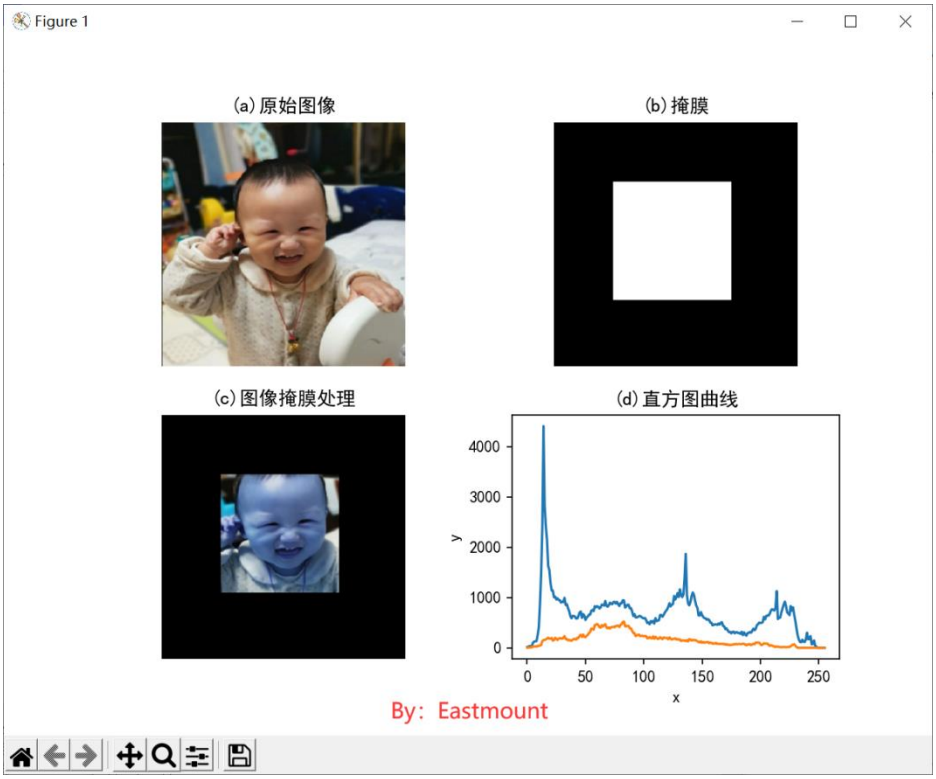


图 20-1 掩膜直方图曲线

2.图像 HS 直方图

为了刻画图像中颜色的直观特性，常常需要分析图像的 HSV 空间下的直方图特性。HSV 空间是由色调(Hue)、饱和度(Saturation)、以及亮度(Value) 构成，因此在进行直方图计算时，需要先将源 RGB 图像转化为 HSV 颜色空间图像，然后将对应的 H 和 S 通道进行单元划分，再其二维空间上计算相对应直方图，再计算直方图空间上的最大值并归一化绘制相应的直方图信息，从而形成色调-饱和度直方图（或 H-S 直方图）。该直方图通常应用在目标检测、特征分析以及目标特征跟踪等场景[1-2]。

由于 H 和 S 分量与人感受颜色的方式是紧密相连，V 分量与图像的彩色信

息无关，这些特点使得 HSV 模型非常适合于借助人的视觉系统来感知彩色特性的图像处理算法。

下面的代码是具体的实现代码，使用matplotlib.pyplot库中的imshow()函数来绘制具有不同颜色映射的 2D 直方图。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('luo.png')

#转换为 RGB 图像
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#图像 HSV 转换
hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

#计算 H-S 直方图
hist = cv2.calcHist(hsv, [0,1], None, [180,256],
```

```
[0,180,0,256])


#原始图像
plt.figure(figsize=(8, 6))
plt.subplot(121), plt.imshow(img_rgb, 'gray'), plt.title("(a)"),
plt.axis('off')


#绘制 H-S 直方图
plt.subplot(122), plt.imshow(hist, interpolation='nearest'),
plt.title("(b)")
plt.xlabel("x"), plt.ylabel("y")
plt.show()
```

图 20-2(a)表示原始输入图像，图 20-2(b)是原图像对应的彩色直方图，其中 X 轴表示饱和度（S），Y 轴表示色调（H）。在直方图中，可以看到 H=140 和 S=130 附近的一些高值，它对应于艳丽的色调。

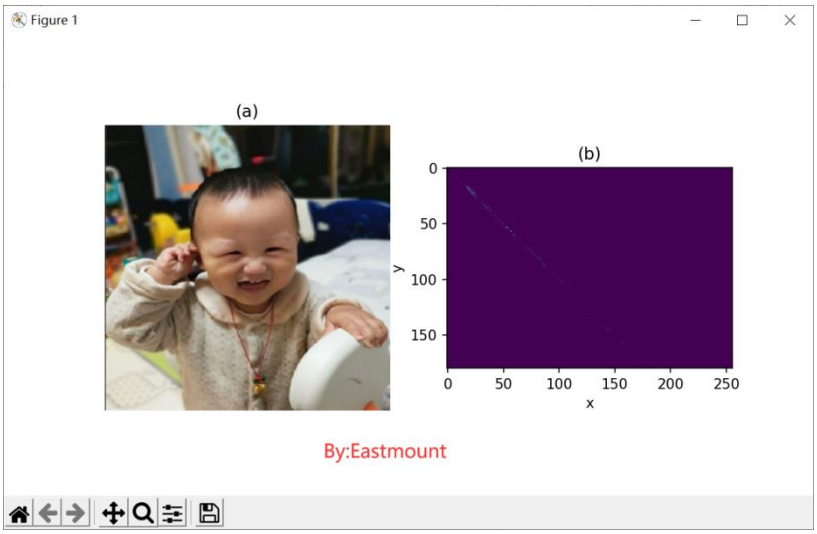


图 20-2 图像 H-S 直方图

3.直方图判断白天黑夜

接着讲述一个应用直方图的案例，通过直方图来判断一幅图像是黑夜或白天。常见的方法是通过计算图像的灰度平均值、灰度中值或灰度标准差，再与自定义的阈值进行对比，从而判断是黑夜还是白天[3-4]。

- 灰度平均值：该值等于图像中所有像素灰度值之和除以图像的像素个数。
- 灰度中值：对图像中所有像素灰度值进行排序，然后获取所有像素最中间的值，即为灰度中值。
- 灰度标准差：又常称均方差，是离均差平方的算术平均数的平方根。标准差能反映一个数据集的离散程度，是总体各单位标准值与其平均数离差平方的算术平均数的平方根。如果一幅图看起来灰蒙蒙的， 那灰度标准差就小；如果一幅图看起来很鲜艳，那对比度就很大，标准差也大。

下面的代码是计算灰度“Lena”图的灰度平均值、灰度中值和灰度标准差。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#函数：获取图像的灰度平均值
def fun_mean(img, height, width):
    sum_img = 0
    for i in range(height):
        for j in range(width):
            sum_img = sum_img + int(img[i,j])
    mean = sum_img / (height * width)
    return mean


#函数：获取中位数
def fun_median(data):
    length = len(data)
    data.sort()
    if (length % 2)== 1:
        z = length // 2
```

```
        y = data[z]
    else:
        y = (int(data[length//2]) + int(data[length//2-1])) / 2
    return y


#读取图像
img = cv2.imread('lena-hd.png')


#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#获取图像高度和宽度
height = grayImage.shape[0]
width = grayImage.shape[1]


#计算图像的灰度平均值
mean = fun_mean(grayImage, height, width)
print("灰度平均值：", mean)


#计算图像的灰度中位数
value = grayImage.ravel() #获取所有像素值
```

```
median = fun_median(value)
print("灰度中值：", median)


#计算图像的灰度标准差
std = np.std(value, ddof = 1)
print("灰度标准差", std)
```

其运行结果如图 20-3 所示，图 20-3(a)为原始图像，图 20-3(b)为处理结果。其灰度平均值为 123，灰度中值为 129，灰度标准差为 48.39。

图 20-3 图像平均值、灰度中值和灰度标准差计算

下面讲解另一种用来判断图像是白天还是黑夜的方法，其基本步骤如下：

（1）读取原始图像，转换为灰度图，并获取图像的所有像素值；

（2）设置灰度阈值并计算该阈值以下的像素个数。比如像素的阈值设置为 50，统计低于 50 的像素值个数；

（3）设置比例参数，对比该参数与低于该阈值的像素占比，如果低于参数则预测为白天，高于参数则预测为黑夜。比如该参数设置为 0.8，像素的灰度值低于阈值 50 的个数占整幅图像所有像素个数的 90%，则认为该图像偏暗，故

预测为黑夜；否则预测为白天。

具体实现的代码如下所示。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#函数：判断黑夜或白天
def func_judge(img):
    #获取图像高度和宽度
    height = grayImage.shape[0]
    width = grayImage.shape[1]
    piexs_sum = height * width
    dark_sum = 0   #偏暗像素个数
    dark_prop = 0 #偏暗像素所占比例

    for i in range(height):
        for j in range(width):
            if img[i, j] < 50: #阈值为 50
                dark_sum += 1
```

```
    #计算比例
    print(dark_sum)
    print(piexs_sum)
    dark_prop = dark_sum * 1.0 / piexs_sum
    if dark_prop >=0.8:
        print("This picture is dark!", dark_prop)
    else:
        print("This picture is bright!", dark_prop)

#读取图像
img = cv2.imread('day.png')

#转换为 RGB 图像
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#图像灰度转换
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#计算 256 灰度级的图像直方图
hist = cv2.calcHist([grayImage], [0], None, [256], [0,255])
```

```
#判断黑夜或白天
func_judge(grayImage)

#显示原始图像和绘制的直方图
plt.subplot(121), plt.imshow(img_rgb, 'gray'), plt.axis('off'),
plt.title("(a)")
plt.subplot(122), plt.plot(hist, color='r'), plt.xlabel("x"), plt.ylabel("y"),
plt.title("(b)")


plt.show()
```

第一张测试图输出的结果如图 20-4 所示，其中图 20-4(a)为原始图像，图 20-4(b)为对应直方图曲线。

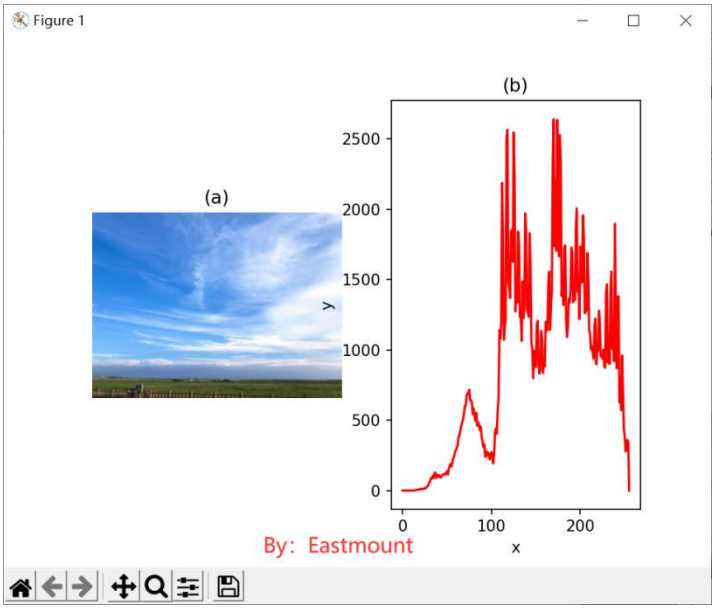


图 20-4 图像灰度计算

最终输出结果为“('This picture is bright!', 0.010082704388303882)”，该预测为白天。

```
2231
221270
This picture is bright! 0.010082704388303882
>>>
```

图 20-5 图像预测为白天

第二张测试图输出的结果如图 20-6 所示，其中图 20-6(a)为原始图像，图 20-6(b)为对应直方图曲线。

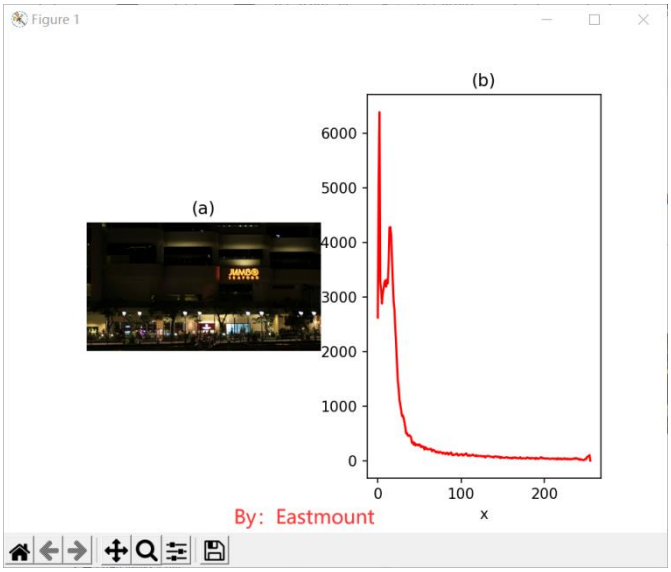


图 20-6 图像灰度计算

最终输出结果为“('This picture is dark!', 0.8511824175824175)”，该预测为黑夜。

```
96822
113750
This picture is dark! 0.8511824175824175
>>>
```

图 20-7 图像预测为黑夜

4.总结

本章主要讲解图像直方图相关知识点，包括掩膜直方图和 HS 直方图，并通过直方图判断黑夜与白天，通过案例分享直方图的实际应用。希望对您有所帮助，后续将进入图像增强相关知识点。

参考文献：

[1] 冈萨雷斯. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社, 2013.

[2]张恒博，欧宗瑛．一种基于色彩和灰度直方图的图像检索方法[J]．计算机工程，2004.

[3]Eastmount.[数字图像处理] 四.MFC 对话框绘制灰度直方图[EB/OL].(2015-05-31). https://blog.csdn.net/eastmount/article/details/46237463.

[4]ZJE_ANDY．python3+opencv 利用灰度直方图来判断图片的亮暗情况[EB/OL].(2018-06-20). https://blog.csdn.net/u014453898/article/details/80745987.

[5]阮秋琦．数字图像处理学（第 3 版）[M]．北京：电子工业出版社，2008.

[6] Eastmount.[Python 图像处理] 十一.灰度直方图概念及 OpenCV 绘制直方图[EB/OL]. (2018-11-06). https://blog.csdn.net/Eastmount/article/details/83758402.

第 21 篇 图像增强和直方图均衡化处理

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一个系列介绍了图像直方图相关知识点。这篇文章将进入图像增强系列，首先我们介绍图像增强基础知识和直方图均衡化处理方法。

1.图像增强

图像增强（Image Enhancement）是指按照某种特定的需求，突出图像中有用的信息，去除或者削弱无用的信息。图像增强的目的是使处理后的图像更适合人眼的视觉特性或易于机器识别。在医学成像、遥感成像、人物摄影等领域，图像增强技术都有着广泛的应用。图像增强同时可以作为目标识别、目标跟踪、特征点匹配、图像融合、超分辨率重构等图像处理算法的预处理算法[1]。

随着消费型和专业型数码相机的日益普及，海量的图像数据正在被产生但由于场景条件的影响，很多在高动态范围场景、昏暗环境或特殊光线条件下拍摄的图像视觉效果不佳，需要进行后期增强处理压缩或拉伸动态范围或提取一致色感才能满足显示和印刷的要求。图 21-1 所示是需要进行增强处理的图像的举例。

(a) (b)

图 21-1 需进行增强处理的图像举例

图 21-1(a)为一幅在高动态范围的场景中拍摄的照片，图中左上角天空区域亮度很高，但右侧大半边的树所在区域却很暗，细节几乎不可见，需采用色调映射方法进行增强，以压缩动态范围增强暗处细节；图 21-1(b)为一幅在水下拍摄的图像,整体动态范围很低,细节辨识不清,全图都需要进行对比度增强[2]。

图像增强通常划分为如图 21-2 所示的分类，其中最重要的是图像平滑和图像锐化处理，接下来也会分别进行详细介绍。

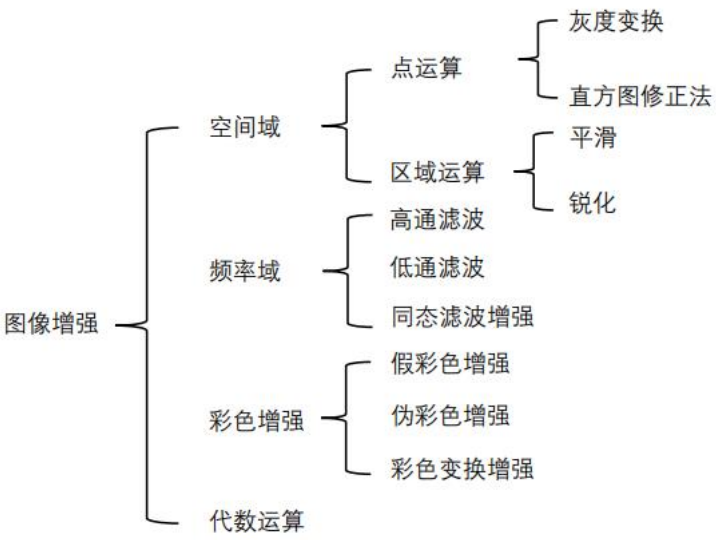


图 21-2 图像增强分类

图像增强按其变换处理所在的作用域不同而被分为空间域方法和频率域方法两大类。而由于具体的应用目的不同，其图像实际增强处理所用到的方法和增强的内容有一定的差异，但图像增强处理的各目标和方法并不互相排斥，某些应用中需要同时联合几种方法来实现最好的增强效果[3]。

（1）空间域

空间域增强通常包含图像灰度级变换、图像直方变换、直方均衡以及使用模糊逻辑和基于优化的增强算法，如使用遗传算法和细菌觅食等算法进行优化处理以达到图像增强的目的。空间域图像增强方法的一般定义如公式（21-1）。

$$g(x,y)=T[f(x,y)] \tag{21-1}$$

其中，$f(x,y)$为输入的待增强的图像，$g(x,y)$为处理后的增强图像，T 为空间域变换函数，表示对原图像 $f(x,y)$ 在像素空间所进行的各种变换操作。当 T 操作定义在单个像素点 (x,y) 上时，称该操作为点操作；而空间滤波指 T 操作作用于像素点 (x,y) 的邻域上时的相应处理。

直方图均衡是图像增强处理中对比度变换调整中最典型的方法。该方法是空域增强中最常用、最简单有效的方法之一，其采用灰度统计特征，将原始图像中的灰度直方图从较为集中的某个灰度区间转变为均匀分布于整个灰度区域范围的变换方法。空间域增强方法按处理策略的差异，又可分为：

❖ 全局一致性处理

全局一致性方法较为简单，仅对图像空间像素值进行统一的调整，则未考虑像素点在空间中的分布特性。全局直方图均衡方法的主要优点是算法简单、速度

快，可自动增强图像。全局直方图均衡方法的缺点是对噪声敏感、细节信息易失，在某些结果区域产生过增强问题，且对对比度增强的力度相对较低。

❖ 局部自适应处理

局部自适应方法较复杂，主要针对图像局部对比度、边缘等特殊区域信息进行增强。局部直方图均衡的主要优点是局部自适应，可最大限度的增强图像细节；其缺点是增强图像质量操控困难，并会随之引入噪声[4]。

（2）频率域

基于频域的图像增强算法基础为卷积理论，该方法把图像视为波，然后利用信号处理手段来处理图像。其通用的数学表示如下公式(21-2)所示。

$$G(u,v)=H(u,v)*F(u,v) \tag{21-2}$$

由上式逆变换后，产生增强后的图像 $g(x,y)$，其表达如下公式(21-3)所示：

$$g(x,y)=F^{-1}[G(u,v)]=F^{-1}[H(u,v)*F(u,v)] \tag{21-3}$$

其中，$g(x,y)$ 为增强后的图像，$F(x,y)$ 为原图像的傅立叶变换，$H(x,y)$ 为滤波变换函数，通过大量的实验研究，发现增强处理后的图像具有比原图像更加清晰的细节。常用的滤波方法有低通、高通、带阻及同态滤波等。

频域图像增强方法从本质上是一种间接对图像进行变换处理的方法，其最早的变换理论，由傅立叶的《热分析理论》指出的周期函数表达可由不同频率和不同倍乘系数表达的正/余弦和形式表征。

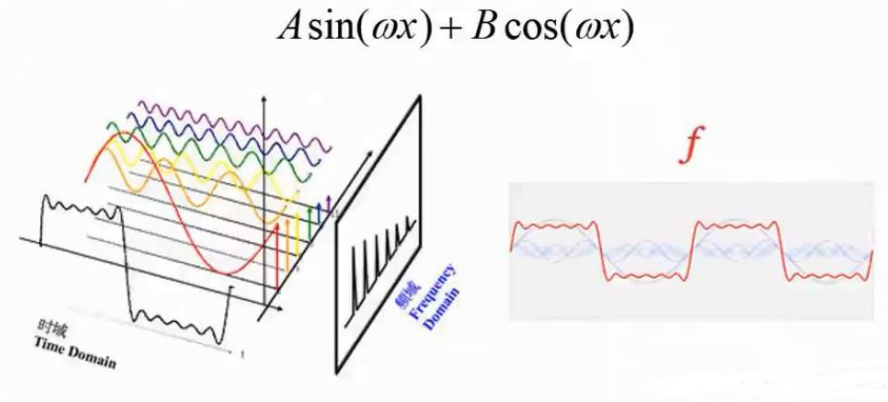


图 21-3 图像频域分析

随着图像处理应用不断发展，频率域变换方法近年来在小波变换基础上发展了具有更高精度和更好稀疏表达特性的方法，更加适合于表达图像的边缘轮廓信息，如 Curvelet 和 Contourlet 变换。这些超小波变换都是基于变换域的新型的多尺度分析方法，在图像对比度增强、降噪、图像融合与分割等方面得到了广泛地应用[5]。

2.直方图均衡化原理

直方图均衡化是图像灰度变化的一个重要处理，被广泛应用于图像增强领域。它是指通过某种灰度映射将原始图像的像素点均匀地分布在每一个灰度级上，其结果将产生一幅灰度级分布概率均衡的图像。直方图均衡化的中心思想是把原始图像的灰度直方图从比较集中的某个灰度区间转变为全范围均匀分布的灰度区间，通过该处理，增加了像素灰度值的动态范围，从而达到增强图像整体对比度的效果。直方图均衡化包括三个核心步骤：

- 统计直方图中每个灰度级出现的次数；

- 计算累计归一化直方图；
- 重新计算像素点的像素值。

直方图均衡化示意图如图 21-4 所示，左边的像素值集中于中心部分，通过均衡化处理后，图像峰值不再这么高，同时 0-50、200-255 灰度值部分存在像素值。简单理解就是将图像的灰度值进行拉伸。

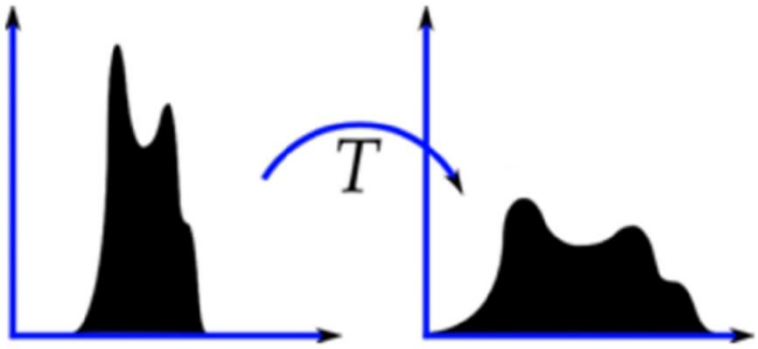


图 21-4 直方图均衡化

图 21-5 将左边像素偏暗的原始图像进行直方图均衡化处理，得到右边的图像色彩更均衡。

图 21-5 直方图均衡化处理

直方图均衡化算法的详细处理过程如下：

■ **第一步，计算原始图像直方图的概率密度。**

如公式(21-3)所示。其中，r_k表示第 k 个灰度级（k=0,1,2,..,L-1），L 最大值为 256；n_k表示图像中灰度级为 r_k的像素个数；N 表示图像中像素的总个数；$P(r_k)$为图像中第 k 个灰度级占总像素数的比例。

$$P(r_k)=\frac{n_k}{N} \tag{21-3}$$

■ **第二步，通过灰度变换函数 T 计算新图像灰度级的概率密度。**

新图像灰度级的概率密度是原始图像灰度级概率密度的累积，如公式(21-4)所示。其中，k 是新图像的灰度级，k=0,1,2,..,L-1；表示原始图像的第 k 个灰度级；为直方图均衡化处理后的第 k 个灰度级。

$$s_k=T(r_k)=\sum_{j=0}^{k}P(r_j) \tag{21-4}$$

■ **第三步，计算新图像的灰度值。**

由于公式(21-4)计算所得的 s_k位于 0 至 1 之间，需要乘以图像的最大灰度级 L，转换为最终的灰度值 res，如公式(21-5)所示。

$$res = s_k \times L \tag{21-5}$$

图 21-6(a)表示原始图像，图 21-6(b)表示其对应的直方图，x 轴表示 256 个灰度级，y 轴表示各个灰度级出现的频数。

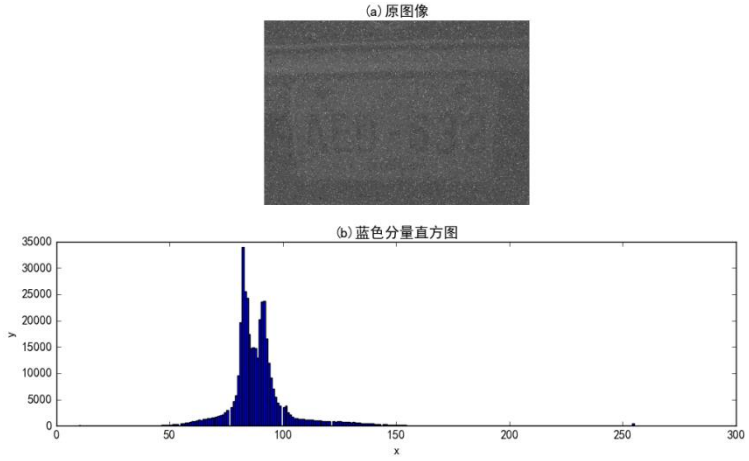


图 21-6 原始图像及其直方图

图 21-7(a)表示直方图均衡化处理后的图像，图 21-7(b)表示其对应的直方图。从效果图可以看出，经过直方图均衡化处理，图像变得更加清晰，图像的灰度级分布也更加均匀[6]。

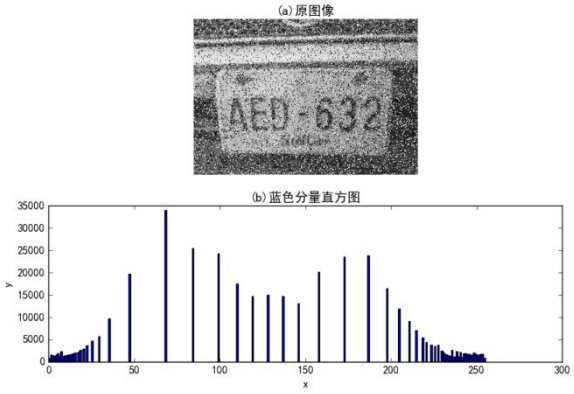


图 21-7 直方图均衡化后的图像及其直方图

下面举一个简单的例子说明直方图均衡化的算法流程。假设存在如公式（21-6）所示的 7×7 个像素点的图像。

$$\begin{bmatrix} 0 & 1 & 4 & 1 & 7 & 3 & 3 \\ 0 & 0 & 4 & 0 & 0 & 1 & 3 \\ 1 & 2 & 7 & 5 & 7 & 4 & 6 \\ 0 & 4 & 0 & 1 & 1 & 6 & 6 \\ 7 & 1 & 2 & 2 & 7 & 3 & 3 \\ 4 & 5 & 7 & 4 & 2 & 7 & 2 \\ 0 & 7 & 1 & 5 & 2 & 0 & 1 \end{bmatrix} \tag{21-6}$$

- **首先通过各像素值出现的次数。**

其统计结果如表 21-1 所示，其中像素值 0 出现了 9 次，像素值 1 出现了 9 次，依次类推，像素值 7 出现了 8 次。

表 21-1 各像素值统计直方图

像素级	个数
0	9
1	9
2	6
3	5
4	6
5	3
6	3
7	8

- **接着将表 21-1 的结果进行归一化处理。**

即当前像素值出现的次数除以总的像素个数（总个数为 49），比如像素值 0 的归一化结果为 0.18，像素值 1 的归一化结果为 0.18，依次类推，像素值 7 的

归一化结果为 0.16。

表 21-2 各像素值归一化处理后的直方图

像素级	百分比
0	0.18
1	0.18
2	0.12
3	0.10
4	0.12
5	0.06
6	0.06
7	0.16

- **然后统计各像素值的累计直方图。**

比如像素值 0 的累计直方图百分比为 0.18，像素值 1 的累计直方图百分比为像素值 0（百分比为 0.184）加上像素值 1（百分比为 0.184），最终结果四舍五入为 0.37，依次类推，像素值 7 的累计直方图百分比为 1.00。

表 21-3 各像素值的累计直方图

像素级	百分比
0	0.18
1	0.37
2	0.49
3	0.59

4	0.71
5	0.78
6	0.84
7	1.00

- **最后将各像素值的累计直方图百分比乘以像素范围的最大值。**

此处的范围为 0 到 7，故乘以最大值 7，得到一个新的结果，即为直方图均衡化处理的结果，如表 21-4 所示。

表 21-4 各像素值的累计直方图

像素级	累计直方图百分比	计算过程	新的像素级
0	0.18	0.18×7	1
1	0.37	0.37×7	3
2	0.49	0.49×7	3
3	0.59	0.59×7	4
4	0.71	0.71×7	5
5	0.78	0.78×7	5
6	0.84	0.84×7	6

7	1.00	1.00×7	7

表 21-4 将原始像素级的 8 个结果（0、1、2、3、4、5、6、7）直方图均衡化为 6 个新的结果（1、3、4、5、6、7），生成如图 21-8 所示的直方图。其中图 21-8(a)是原始图像各像素级的直方图，图 21-8(b)是直方图均衡化处理后的各像素级的直方图，最终结果更加均衡。直方图均衡化可以让色彩细节更丰富，获取更多的信息，常用于人脸识别、车牌识别、医疗图像处理等领域。

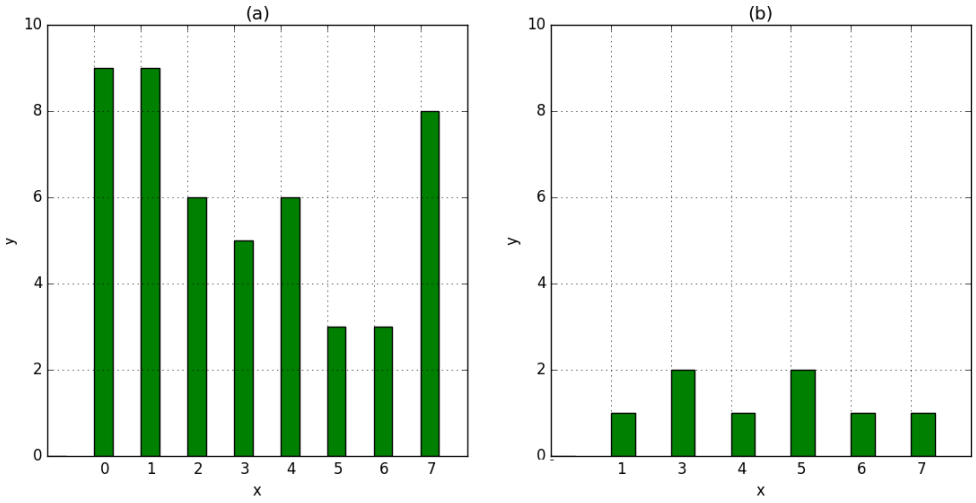


图 21-8 直方图均衡化处理前后的直方图对比

3.直方图均衡化处理

Python 调用 OpenCV 中的 cv2.equalizeHist()函数实现直方图均衡化处理，并且为全局直方图均衡化。其函数原型如下所示，输出的 dst 图像与输入图像 src 具有相同的大小和类型。

- dst = cv2.equalizeHist(src)
  - src 表示输入图像，即原图像

- dst 表示目标图像，直方图均值化处理的结果

下面的代码是调用 cv2.equalizeHist()函数实现图像直方图均衡化处理。

```
# -*- coding: utf-8 -*-
# By：Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('luo.png')

#灰度转换
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#直方图均衡化处理
result = cv2.equalizeHist(gray)

#显示图像
cv2.imshow("Input", gray)
cv2.imshow("Result", result)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

输出结果如图 21-9 所示。

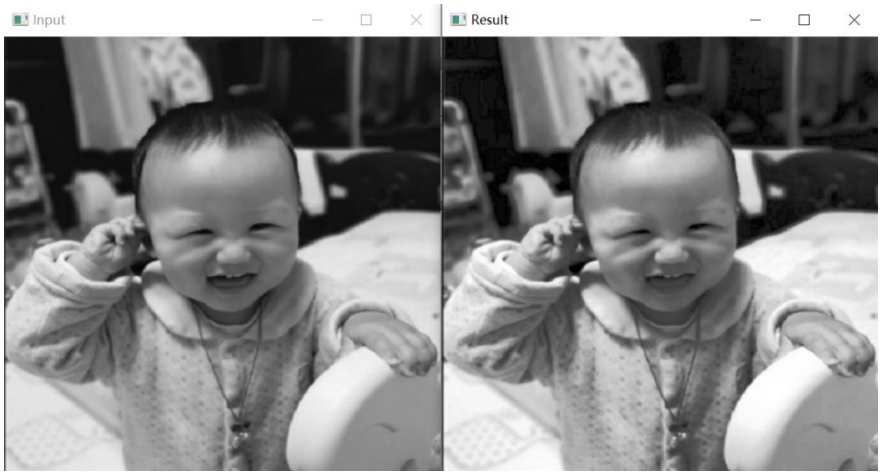


图 21-9 图像直方图均衡化处理

下图是风景图的直方图均衡化处理，左边为原始图像，右边为直方图均衡化处理图像，它有效提升了图像的亮度及细节。

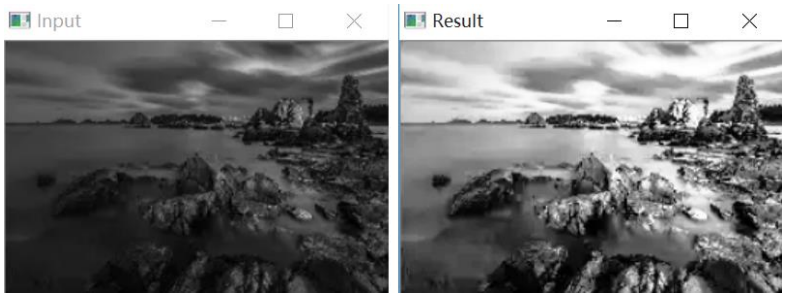


图 21-10 图像直方图均衡化处理

下面代码调用函数 ravel()绘制了对应的直方图，并进行直方图均衡化处理前后的图像对比。

```
# -*- coding: utf-8 -*-

# By：Eastmount
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('lena.bmp')

#灰度转换
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#直方图均衡化处理
result = cv2.equalizeHist(gray)

#显示图像
plt.subplot(221)
plt.imshow(gray, cmap=plt.cm.gray), plt.axis("off"),
plt.title('(a)')
plt.subplot(222)
plt.imshow(result, cmap=plt.cm.gray), plt.axis("off"),
plt.title('(b)')
plt.subplot(223)
```

```
plt.hist(img.ravel(), 256), plt.title('(c)')
plt.subplot(224)
plt.hist(result.ravel(), 256), plt.title('(d)')
plt.show()
```

输出结果如图 21-11 所示，图 21-11(a)为原始图像，对应的直方图为图 21-11(c)，图 21-11(b)和图 21-11(d)为直方图处理后的图像及对应直方图，它让图像的灰度值分布更加均衡。

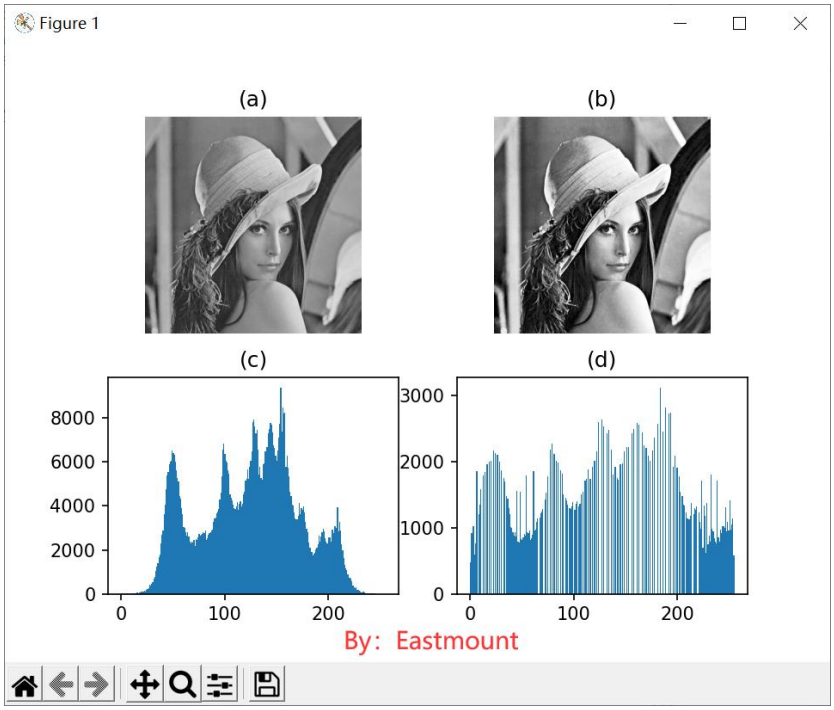


图 21-11 图像直方图均衡化处理

如果需要对彩色图片进行全局直方图均衡化处理，则需要分解 RGB 三色通道，分别进行处理后再进行通道合并。完整代码如下所示：

```
# -*- coding: utf-8 -*-
# By：Eastmount
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('yxz.jpg')

# 彩色图像均衡化 需要分解通道 对每一个通道均衡化
(b, g, r) = cv2.split(img)
bH = cv2.equalizeHist(b)
gH = cv2.equalizeHist(g)
rH = cv2.equalizeHist(r)

# 合并每一个通道
result = cv2.merge((bH, gH, rH))
cv2.imshow("Input", img)
cv2.imshow("Result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
#绘制直方图
plt.figure("Hist")
#蓝色分量
plt.hist(bH.ravel(), bins=256, normed=1, facecolor='b',
edgecolor='b', hold=1)
#绿色分量
plt.hist(gH.ravel(), bins=256, normed=1, facecolor='g',
edgecolor='g', hold=1)
#红色分量
plt.hist(rH.ravel(), bins=256, normed=1, facecolor='r',
edgecolor='r', hold=1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

输出结果如图 21-12 所示，左边为彩色原始图像，右边为直方图均衡化处理图像，它有效提升了图像的亮度及细节。

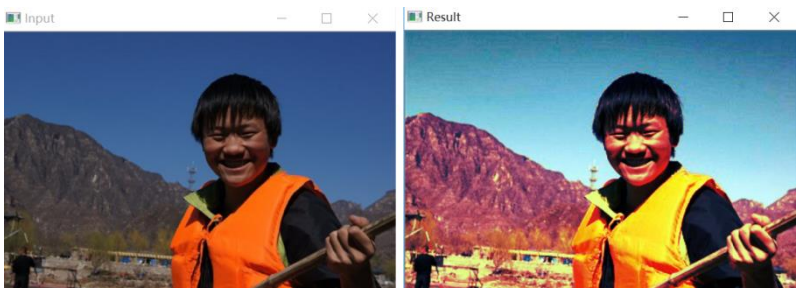


图 21-12 彩色图像直方图均衡化处理

对应的直方图如图 21-13 所示，其中 x 轴表示图像的灰度级，y 轴表示各灰度级所对应的直方图。

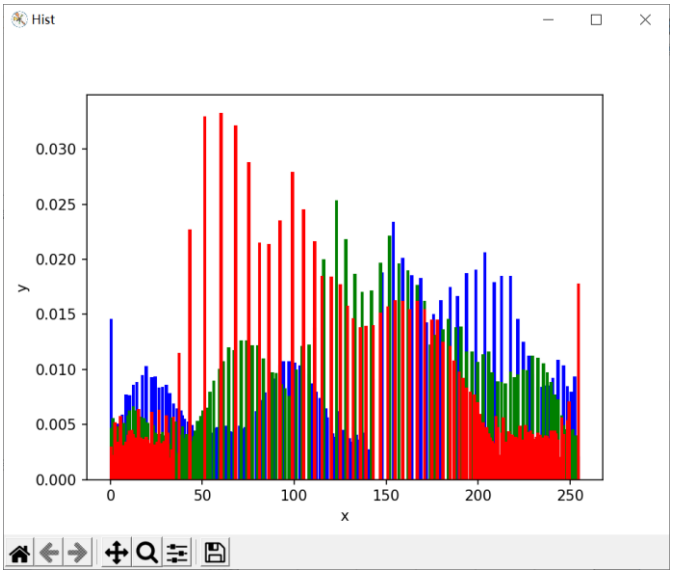


图 21-13 图像直方图

同样，小珞珞图像直方图均衡化的效果如下图所示，总体而言原图图像比较偏黄，更偏近于人肤色；处理后的效果图增强了暗处和面部表情饱和度。

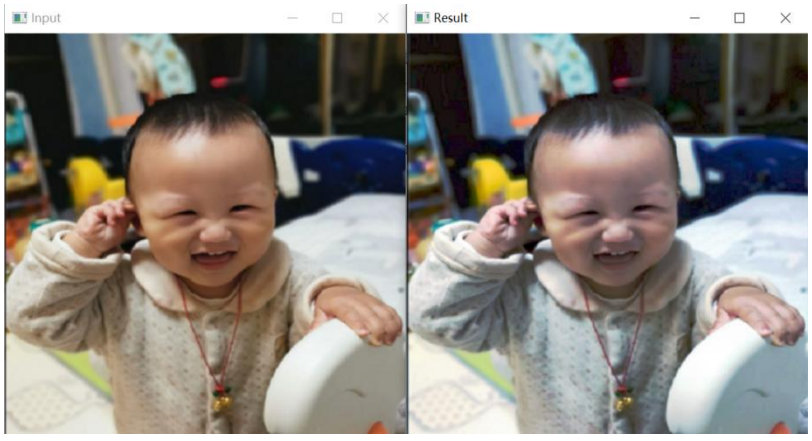


图 21-14 小珞珞彩色图像直方图均衡化处理

4.总结

本章主要讲解图像增强和直方图均衡化原理知识，接着通过 OpenCV 实现图像均衡化处理，并显示对应的直方图显示处理前后的效果。直方图均衡化被广泛应用于图像增强领域，通过某种灰度映射将原始图像的像素点均匀地分布在每一个灰度级上，其结果将产生一幅灰度级分布概率均衡的图像。直方图均衡化的中心思想是把原始图像的灰度直方图从比较集中的某个灰度区间转变为全范围均匀分布的灰度区间，通过该处理，增加了像素灰度值的动态范围，从而达到增强图像整体对比度的效果。

参考文献：

[1] 王浩,张叶,沈宏海,张景忠.图像增强算法综述[J].中国光学,2017,10(04):438-448.

[2] 许欣. 图像增强若干理论方法与应用研究[D].南京理工大学,2010.

[3] 李艳梅. 图像增强的相关技术及应用研究[D].电子科技大学,2013.

[4] 冈萨雷斯. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社, 2013.

[5] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社, 2008.

[6] eastmount.[数字图像处理] 五.MFC 图像点运算之灰度线性变化、灰度非线性变化、阈值化和均衡化处理详解[EB/OL]. (2015-06-02). https://blog.csdn.net/eastmount/article/details/46312145.

[7] 毛星云, 冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社, 2015.

[8] eastmount. [Python 图像处理] 三十八.OpenCV 图像增强和图像去雾万字详解（直方图均衡化、局部直方图均衡化、自动色彩均衡化）[EB/OL]. (2021-03-12). https://blog.csdn.net/ Eastmount/article/details/114706950.

第 22 篇 局部直方图均衡化和自动色彩均衡化处理

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像增强概念和直方图均衡化。这篇文章将继续讲解图像增强，包括图像局部直方图均衡化和自动色彩均衡化处理。

1.局部直方图均衡化

前文通过调用 OpenCV 中 equalizeHist()函数实现直方图均衡化处理，该方法简单高效，但其实它是一种全局意义上的均衡化处理，很多时候这种操作不是很好，会把某些不该调整的部分给均衡处理了。同时，图像中不同的区域灰度分布相差甚远，对它们使用同一种变换常常产生不理想的效果，实际应用中，常常需要增强图像的某些局部区域的细节。

为了解决这类问题，Pizer 等提出了局部直方图均衡化的方法（AHE），但 AHE 方法仅仅考虑了局部区域的像素，忽略了图像其他区域的像素，且对于图像中相似区域具有过度放大噪声的缺点。为此 K. Zuiderveld 等人提出了对比度受限 CLAHE 的图像增强方法，通过限制局部直方图的高度来限制局部对比度的增强幅度，从而限制噪声的放大及局部对比度的过增强，该方法常用于图像

增强，也可以被用来进行图像去雾操作[1-2]。

在 OpenCV 中，调用函数 createCLAHE()实现对比度受限的局部直方图均衡化。它将整个图像分成许多小块（比如按 10×10 作为一个小块），那么对每个小块进行均衡化。这种方法主要对于图像直方图不是那么单一的（比如存在多峰情况）图像比较实用。其函数原型如下所示：

- retval = createCLAHE([, clipLimit[, tileGridSize]])
  - clipLimit 参数表示对比度的大小
  - tileGridSize 参数表示每次处理块的大小

调用 createCLAHE()实现对比度受限的局部直方图均衡化的代码如下：

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图片
img = cv2.imread('lena.bmp')


#灰度转换
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
#局部直方图均衡化处理
clahe = cv2.createCLAHE(clipLimit=2,
tileGridSize=(10,10))

#将灰度图像和局部直方图相关联，把直方图均衡化应用到灰度图
result = clahe.apply(gray)

#显示图像
plt.subplot(221)
plt.imshow(gray, cmap=plt.cm.gray), plt.axis("off"),
plt.title('(a)')
plt.subplot(222)
plt.imshow(result, cmap=plt.cm.gray), plt.axis("off"),
plt.title('(b)')
plt.subplot(223)
plt.hist(img.ravel(), 256), plt.title('(c)')
plt.subplot(224)
plt.hist(result.ravel(), 256), plt.title('(d)')
plt.show()
```

输出结果如图 22-1 所示，图 22-1(a)为原始图像，对应的直方图为图 22-1(c)，图 22-1(b)和图 22-1(d)为对比度受限的局部直方图均衡化处理后的图

像及对应直方图，它让图像的灰度值分布更加均衡。可以看到，相对于全局的直方图均衡化，这个局部的均衡化似乎得到的效果更自然一点。

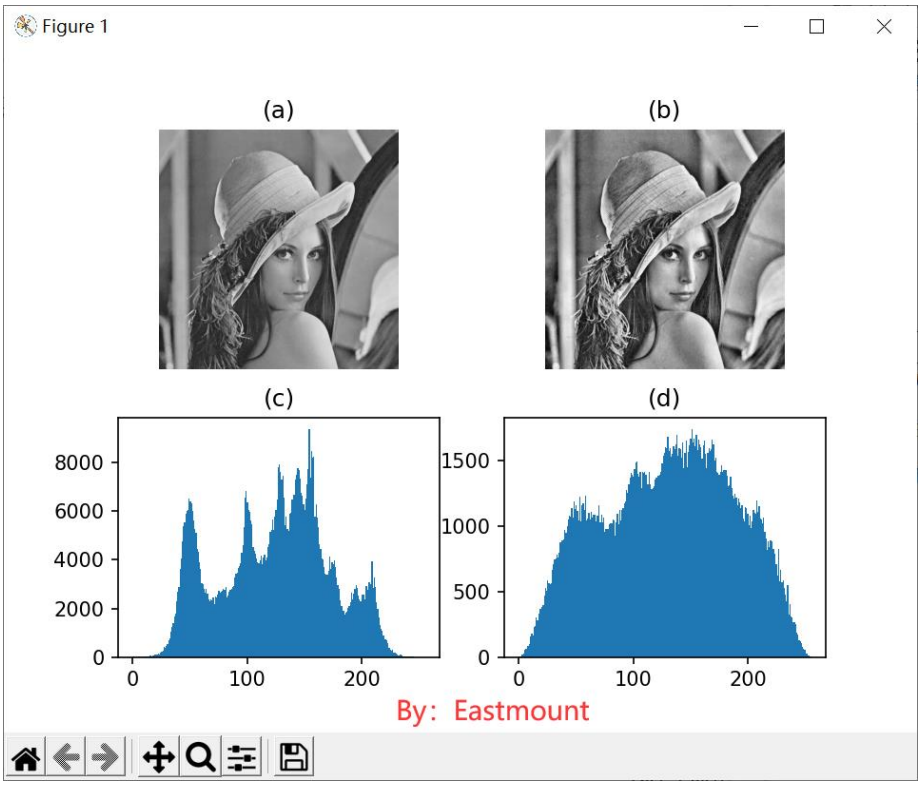


图 22-1 图像对比度受限的局部直方图均衡化处理

2.自动色彩均衡化

Retinex 算法是代表性的图像增强算法，它根据人的视网膜和大脑皮层模拟对物体颜色的波长光线反射能力而形成，对复杂环境下的一维条码具有一定范围内的动态压缩，对图像边缘有着一定自适应的增强。自动色彩均衡（Automatic Color Enhancement，ACE）算法是在 Retinex 算法的理论上提出的，它通过计算图像目标像素点和周围像素点的明暗程度及其关系来对最终的像素值进行校正，实现图像的对比度调整，产生类似人体视网膜的色彩恒常性和亮度恒

常性的均衡，具有很好的图像增强效果[3-4]。

ACE 算法包括两个步骤，一是对图像进行色彩和空域调整，完成图像的色差校正，得到空域重构图像；二是对校正后的图像进行动态扩展。ACE 算法计算公式如下：

$$\mathrm{Y}=\frac{\sum(g(I(\mathrm{x}_0)-I(\mathrm{x}))\cdot w(\mathrm{x}_0,x))}{\sum(w(x_0,x))} \tag{22-1}$$

其中，W 是权重参数，离中心点像素越远的 W 值越小；g 是相对对比度调节参数，其计算方法如公式(22-2)所示，a 表示控制参数，值越大细节增强越明显。

$$\mathrm{g}(x)=\max(\min(ax,1.0),-1.0) \tag{22-2}$$

图 22-2 是条形码图像进行 ACE 图像增强后的效果图，通过图像增强后的图 7(b)对比度更强，改善了原图像的明暗程度，增强的同时保持了图像的真实性。

(a)原图　　(b)增强后效果图

图 22-2 ACE 图像增强效果图

由于 OpenCV 中暂时没有 ACE 算法包，下面的代码是借鉴

“zmshy2128”老师的文章，修改实现的彩色直方图均衡化处理[5]。

```
# -*- coding: utf-8 -*-
# By:Eastmount
#  参考 zmshy2128 老师文章
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt


#线性拉伸处理
#去掉最大最小 0.5%的像素值  线性拉伸至[0,1]
def stretchImage(data, s=0.005, bins = 2000):
    ht = np.histogram(data, bins);
    d = np.cumsum(ht[0])/float(data.size)
    lmin = 0; lmax=bins-1
    while lmin<bins:
        if d[lmin]>=s:
            break
        lmin+=1
    while lmax>=0:
        if d[lmax]<=1-s:
```

```
            break
        lmax-=1
    return np.clip((data-ht[1][lmin])/(ht[1][lmax]-
ht[1][lmin]), 0,1)

#根据半径计算权重参数矩阵
g_para = {}
def getPara(radius = 5):
    global g_para
    m = g_para.get(radius, None)
    if m is not None:
        return m
    size = radius*2+1
    m = np.zeros((size, size))
    for h in range(-radius, radius+1):
        for w in range(-radius, radius+1):
            if h==0 and w==0:
                continue
            m[radius+h, radius+w] =
1.0/math.sqrt(h**2+w**2)
    m /= m.sum()
```

```
    g_para[radius] = m
    return m


#常规的 ACE 实现
def zmIce(I, ratio=4, radius=300):
    para = getPara(radius)
    height,width = I.shape


    #Python3 报错如下 使用列表 append 修改
    zh = []
    zw = []
    n = 0
    while n < radius:
        zh.append(0)
        zw.append(0)
        n += 1
    for n in range(height):
        zh.append(n)
    for n in range(width):
        zw.append(n)
    n = 0
```

```
    while n < radius:
        zh.append(height-1)
        zw.append(width-1)
        n += 1
    #print(zh)
    #print(zw)

    Z = I[np.ix_(zh, zw)]
    res = np.zeros(I.shape)
    for h in range(radius*2+1):
        for w in range(radius*2+1):
            if para[h][w] == 0:
                continue
            res += (para[h][w] * np.clip((I-Z[h:h+height,
w:w+width])*ratio, -1, 1))
    return res

#单通道 ACE 快速增强实现
def zmIceFast(I, ratio, radius):
    print(I)
    height, width = I.shape[:2]
```

```
    if min(height, width) <=2:
        return np.zeros(I.shape)+0.5
    Rs = cv2.resize(I, (int((width+1)/2), int((height+1)/2)))
    Rf = zmIceFast(Rs, ratio, radius)                    #递归调用
    Rf = cv2.resize(Rf, (width, height))
    Rs = cv2.resize(Rs, (width, height))

    return Rf+zmIce(I,ratio, radius)-zmIce(Rs,ratio,radius)

#rgb 三通道分别增强 ratio 是对比度增强因子 radius 是卷积模板半径
def zmIceColor(I, ratio=4, radius=3):
    res = np.zeros(I.shape)
    for k in range(3):
        res[:,:,k] = stretchImage(zmIceFast(I[:,:,k], ratio, radius))
    return res

#主函数
if __name__ == '__main__':
    img = cv2.imread('test01.png')
```

```
res = zmIceColor(img/255.0)*255
cv2.imwrite('Ice.jpg', res)
```

运行结果如图22-3和图22-4所示，ACE算法能有效进行图像去雾处理，实现图像的细节增强。

图 22-3 ACE 图像增强效果图

图 22-4 ACE 图像去雾处理

3.总结

本文主要讲解图像局部直方图均衡化和自动色彩均衡化处理。这些算法可以广泛应用于图像增强、图像去噪、图像去雾等领域。

参考文献：

[1] 王浩,张叶,沈宏海,张景忠.图像增强算法综述[J].中国光学,2017,10(04):438-448.

[2] 李艳梅. 图像增强的相关技术及应用研究[D].电子科技大学,2013.

[3] S. Bidon, Olivier Besson, J. Y. Tourneret. The Adaptive Coherence Estimator is the Generalized Likelihood Ratio Test for a Class of Heterogeneous Environments[J]. IEEE Signal Processing Letters, 2008, 15: 281-284.

[4] eastmount. [Python 图像处理] 三十八.OpenCV 图像增强和图像去雾万字详解(直方图均衡化、局部直方图均衡化、自动色彩均衡化 ）[EB/OL]. (2021-03-12). https://blog.csdn.net/ Eastmount/article/details/114706950.

[5] zmshy2128. 自动色彩均衡（ ACE ）快速算法[EB/OL]. (2016-12-05). https://www.cnblogs.com/zmshy2128/p/6135551.html.

第 23 篇 图像平滑之均值滤波、方框滤波、高斯滤波

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像增强知识，包括直方图均衡化、局部直方图均衡化、自动色彩均衡化。这篇文章将详细讲解图像平滑知识，包括均值滤波、方框滤波和高斯滤波。常用于消除噪声的图像平滑方法包括三种线性滤波（均值滤波、方框滤波、高斯滤波）和两种非线性滤波（中值滤波、双边滤波），本文将详细讲解三种线性滤波方法。

1.图像平滑

图像平滑是一项简单且使用频率很高的图像处理方法，可以用来压制、弱化或消除图像中的细节、突变、边缘和噪声，最常见的是用来减少图像上的噪声[1]。何为图像噪声？噪声是妨碍人的感觉器官所接受信源信息理解的因素，是不可预测只能用概率统计方法认识的随机误差。从图 23-1 中，可以观察到噪声的特点：位置随机、大小不规则，将这种噪声称为随机噪声，这是一种常见的噪声类型。

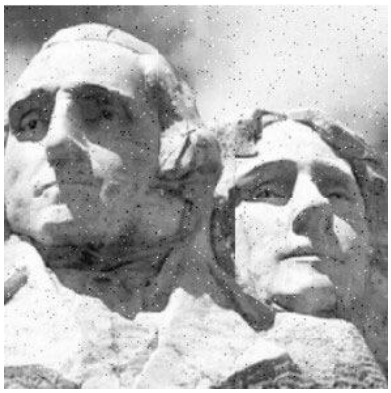


图 23-1 噪声示例

图 23-2 是一个图像平滑的示例，图中左半部分是包含噪声的原始输入图像，右半部分是进行图像平滑后的图像。通过对比容易观察到，在平滑后的图像中，物体中的噪声得到了有效地抑制和消除，但花的边缘部分被进行了模糊，这种将图像中的冗余信息进行抑制，即花的噪声进行消除的过程被称为图像平滑[2]。

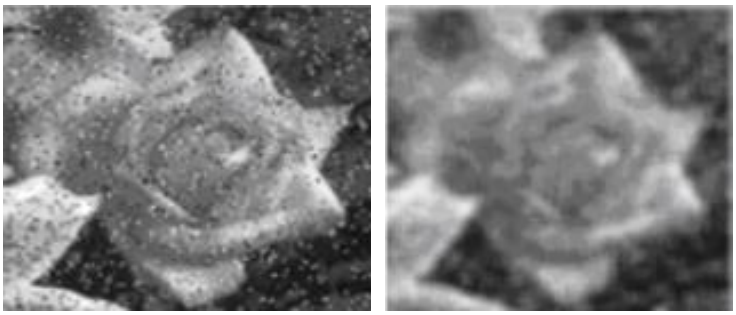


图 23-2 图像平滑实例

一幅图像不可避免地要受到各种噪声源的干扰，所以噪声滤除往往是图像处理中的第一步，滤波效果好坏将直接影响后续处理结果，噪声滤除在图像处理中占有相当重要的地位。噪声滤除算法多种多样，可以从设计方法上分为线性滤波算法和非线性滤波算法两大类。

（1）线性滤波

在图像处理中，对邻域中的像素的计算为线性运算时，如利用窗口函数进行平滑加权求和的运算，或者某种卷积运算，都可以称为线性滤波。在数字信号处理和数字图像处理的早期研究中，线性滤波器是噪声抑制处理的主要手段，如均值滤波、方框滤波、高斯滤波等。

线性滤波算法对高斯型噪声有较好的滤波效果，而当信号频谱与噪声频谱混叠时或者当信号中含有非叠加性噪声时（例如由系统非线性引起的噪声或存在非高斯噪声等），线性滤波器的处理结果就很难令人满意。

（2）非线性滤波

非线性滤波利用原始图像跟模版之间的一种逻辑关系得到结果，如中值滤波、双边滤波等。非线性滤波技术从某种程度上弥补了线性滤波方法的不足，由于它能够在滤除噪声的同时较好地保持图像信号的高频细节，从而得到广泛的应用。著名学者 Tukey [3]于 1971 年首次提出了一种非线性滤波器——中值滤波器，从此揭开了非线性滤波方法研究的序幕。非线性滤波技术发展到现在，基于中值滤波的改进算法层出不穷，在非线性滤波算法中占有重要的地位。另外很多新的非线性滤波算法也相继涌现，如基于数学形态学的滤波方法、基于模糊理论的滤波方法、基于神经网络的滤波方法等，它们为图像滤波技术提供新的思路[4-5]。

后文将详细介绍以下常用的一些滤波器，包括均值滤波、方框滤波、高斯吕波、中值滤波等，如表 23-1 所示。

表 23-1 常见的图像平滑算法

算法名称	OpenCV 函数	滤波方式

均值滤波	blur()	线性滤波
方框滤波	boxblur()	线性滤波
高斯滤波	GaussianBlur()	线性滤波
中值滤波	medianBlur()	非线性滤波
双边滤波	bilateralFilter()	非线性滤波

图 23-3 为这五种滤波的效果对比，从滤波的结果可以看出各种滤波算法对图像的作用非常不同，有些变化非常大，有些甚至跟原图一样。在实际应用时，应根据噪声的特点、期望的图像和边缘特征等来选择合适的滤波器，这样才能发挥图像滤波的最大优点。

(a)原图　　(b)均值滤波　　(c)方框滤波

(d)高斯滤波　　(e)中值滤波　　(f)双边滤波

图 23-3 滤波效果对比

在图像产生、传输和复制过程中，常常会因为多方面原因而被噪声干扰或出现数据丢失，降低了图像的质量。这就需要对图像进行一定的增强处理以减小这些缺陷带来的影响[6]。

2.均值滤波

均值滤波是最简单的一种线性滤波算法，它是指在原始图像上对目标像素给一个模板，该模板包括了其周围的临近像素（以目标像素为中心的周围 8 个像素，构成一个滤波模板，即去掉目标像素本身），再用模板中的全体像素的平均值来代替原来的像素值。换句话说，均值滤波输出图像的每一个像素值是其周围 M×M 个像素值的加权平均值。

图 23-4 表示均值滤波处理的过程，中心红色点的像素值为蓝色背景区域像素值求和的均值。5×5 的矩阵称之为模糊内核，针对原始图像内的像素点，均值滤波采用核对其像素逐个进行均值处理，并得到最终的效果图。

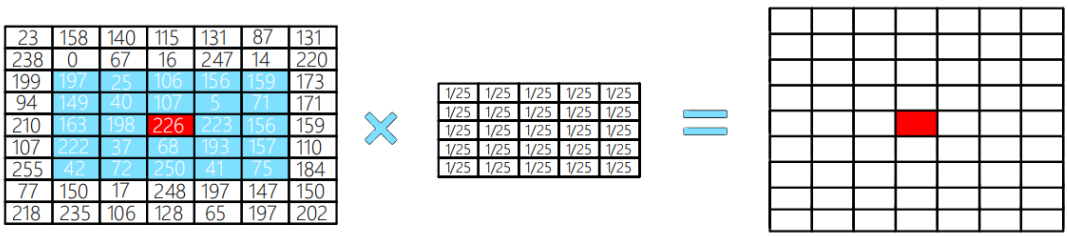


图 23-4 均值滤波处理过程

其中红色区域的像素值均值滤波处理过程为：

$$K = ((197+25+106+156+159)+ (149+40+107+5+71)+ (163+198+226+223+156) +(222+37+68+193+157)+(42+72+250+41+75)) / 25 \quad (23-1)$$

均值滤波算法比较简单，计算速度较快，对周期性的干扰噪声有很好的抑制作用，但是它不能很好地保护图像的细节，在图像去噪的同时，也破坏了图像的细节部分，从而使图像变得模糊。

Python 调用 OpenCV 中的 cv2.blur()函数实现均值滤波处理，其函数原型如下所示，输出的 dst 图像与输入图像 src 具有相同的大小和类型。

dst = blur(src, ksize[, dst[, anchor[, borderType]]])

- src 表示输入图像，它可以有任意数量的通道，但深度应为 CV_8U、CV_16U、CV_16S、CV_32F 或 CV_64F
- ksize 表示模糊内核大小，以（宽度，高度）的形式呈现
- anchor 表示锚点，即被平滑的那个点，其默认值 Point（-1，-1）表示位于内核的中央，可省略
- borderType 表示边框模式，用于推断图像外部像素的某种边界模式，默认值为 BORDER_DEFAULT，可省略

常见的模糊内核包括（3，3）和（5，5），如公式（23-2）和（23-3）所示：

$$K(3,3)=\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \tag{23-2}$$

$$K(5,5)=\frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix} \tag{23-3}$$

图像均值滤波的 Python 实现代码如下所示，需要注意的是，代码中使用的

是 3×3 的模板，plt.rcParams 是用于设置中文汉字正常显示。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('lena-zs.png')
source = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#均值滤波
result = cv2.blur(source, (3,3))

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', '均值滤波']
images = [source, result]
for i in range(2):
```

```
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

“lena”图输出结果如图 23-5 所示，左边表示含有噪声的待处理原图，右边是均值滤波处理后的图像，图像中的椒盐噪声被去除了。

图 23-5 图像 3×3 核的均值滤波处理

如果图像中的噪声仍然存在，可以增加模糊内核的大小，比如使用 5×5、10×10，甚至 20×20 的模板。图 23-6 就是使用 10×10 的内核，但是处理后的图像会逐渐变得更模糊。

图 23-6 图像 10×10 核的均值滤波处理

图像均值滤波是通过模糊内核对图像进行平滑处理，由于模糊内核中的每个权重值都相同，故称为均值。该方法在一定程度上消除了原始图像中的噪声，降低了原始图像的对比度，但也存在一定缺陷，它在降低噪声的同时使图像变得模糊，尤其是边缘和细节处，而且模糊内核越大，模糊程度越严重。

3.方框滤波

图像平滑利用卷积模板逐一处理图像中每个像素，这一过程可以形象地比作对原始图像的像素进行过滤整理，把邻域像素逐一处理的算法过程称为滤波器。常见的线性滤波器包括均值滤波和方框滤波。

方框滤波又称为盒式滤波，它利用卷积运算对图像邻域的像素值进行平均处理，从而实现消除图像中的噪声。方框滤波和和均值滤波的模糊内核基本一样，区别为是否需要进行均一化处理。

Python 调用 OpenCV 中的 cv2.boxFilter()函数实现方框滤波处理，其

函数原型如下所示：

dst = boxFilter(src, depth, ksize[, dst[, anchor[, normalize[, borderType]]]])

- src 表示输入图像
- dst 表示输出图像，其大小和类型与输入图像相同
- depth 表示输出图像深度，通常设置为“-1”，表示与原图深度一致
- ksize 表示模糊内核大小，以（宽度，高度）的形式呈现
- normalize 表示是否对目标图像进行归一化处理，默认值为 true
- anchor 表示锚点，即被平滑的那个点，其默认值 Point（-1，-1）表示位于内核的中央，可省略
- borderType 表示边框模式，用于推断图像外部像素的某种边界模式，默认值为 BORDER_DEFAULT，可省略

常见的模糊内核 ksize 包括（3，3）和（5，5），如下所示：

$$K(3,3)=\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \tag{23-4}$$

$$K(5,5)=\frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix} \tag{23-5}$$

参数 normalize 表示是否对目标图像进行归一化处理。

（1）当 normalize 为 true 时，需要执行归一化处理，方框滤波就变成了均值滤波。其中，归一化就是把要处理的像素值都缩放到一个范围内，以便统一

处理和直观量化。

（2）当 normalize 为 false 时，表示非归一化的方框滤波，不进行均值化处理，实际上就是求周围各像素的和。但此时很容易发生溢出，多个像素值相加后的像素值大于 255，溢出后的像素值均设置为 255，即白色。

参数 normalize 的定义如公式（23-6）所示。

$$H=\frac{1}{\alpha}\begin{bmatrix}1 & \Lambda & 1\\ M & O & M\\ 1 & \Lambda & 1\end{bmatrix},\quad \alpha=\begin{cases}\dfrac{1}{\text{width}\times height} & normalize=true\\ 1 & normalize=false\end{cases} \tag{23-6}$$

图像方框滤波的 Python 实现代码如下所示，代码中使用 3×3 的核，normalize=0 表示不进行图像归一化处理。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图片
img = cv2.imread('lena-zs.png')
source = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)


#方框滤波
result = cv2.boxFilter(source, -1, (3,3), normalize=0)
```

```
#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', '方框滤波']
images = [source, result]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

方框滤波非归一化处理的输出结果如图 23-7 所示，处理后的效果图中包含很多白色的像素点，这是因为图像像素求和结果发生溢出（超过 255）。由此可见，进行非归一化的处理时，得到的图像包含白色过多，对源图像的毁坏太大。

图 23-7 图像 3×3 核的非归一化方框滤波处理

如果设置 2×2 的模糊内核，其非归一化的方框滤波处理效果更好一些，如图 23-8 所示。核心代码为：

- cv2.boxFilter(source, -1, (2,2), normalize=0)

图 23-8 图像 2×2 核的非归一化方框滤波处理

下面代码是使用 3×3 内核，进行归一化方框滤波处理的代码，其输出结果与 3×3 内核均值滤波完全相同。

```
# -*- coding: utf-8 -*-
```

```
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('lena-zs.png')
source = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#方框滤波
result = cv2.boxFilter(source, -1, (3,3), normalize=1)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', '方框滤波']
images = [source, result]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
```

```
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 23-9 所示：

图 23-9 图像 3×3 核的归一化方框滤波处理

4.高斯滤波

为了克服局部平均法造成图像模糊的弊端，又提出了一些保持边缘细节的局部平滑算法，图像高斯滤波（高斯平滑）就是这样一种算法。它是应用邻域平均思想对图像进行平滑的一种线性平滑滤波，对于抑制服从正态分布的噪声非常有效，适用于消除高斯噪声，被广泛应用于图像处理的减噪过程。

图像高斯滤波为图像不同位置的像素值赋予了不同的权重，距离越近的点权重越大，距离越远的点权重越小。它与方框滤波和均值滤波不同，它对邻域内的像素进行平均时，为不同位置的像素赋予不同的权值。通俗地讲，高斯滤波就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其

他像素值（权重不同）经过加权平均后得到。

下面是常用的 3×3 和 5×5 内核的高斯滤波模板。

$$K(3,3)=\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \tag{23-7}$$

$$K(5,5)=\frac{1}{273}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix} \tag{23-8}$$

高斯滤波引入了数学中的高斯函数（正态分布函数），一个二维高斯函数如下公式（23-9）所示，其中 σ 为标准差。高斯加权平均中，最重要是 σ 的选取，标准差代表数据离散程度，如果 σ 较小，则高斯分布中心区域将更加聚集，平滑效果更差；反之，如果 σ 较大，高斯分布中心区域将更离散，平滑效果更明显[10]。

$$h(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{23-9}$$

高斯滤波的核心思想是对高斯函数进行离散化，以离散点上的高斯函数值为权值，对图像中的每个像素点做一定范围邻域内的加权平均，从而有效地消除高斯噪声。高斯滤波让临近中心的像素点具有更高的重要度，对周围像素计算加权平均值，如图 23-10 所示，其中心位置权重最高为 0.4。

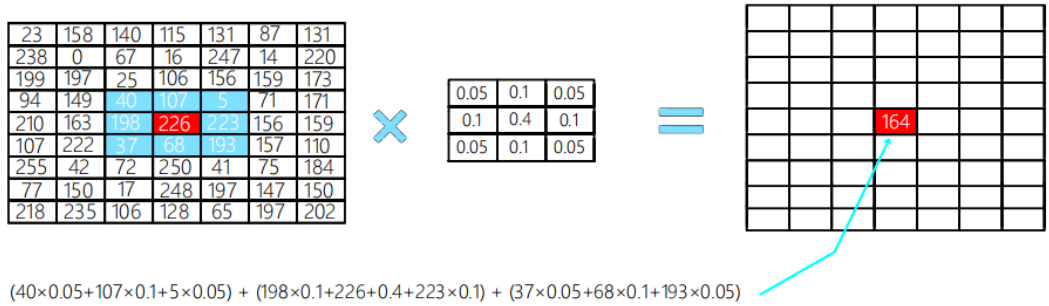


图 23-10 高斯滤波处理过程

Python 中 OpenCV 主要调用 GaussianBlur()函数实现高斯平滑处理，函数原型如下所示：

dst = GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])

- src 表示待处理的输入图像
- dst 表示输出图像，其大小和类型与输入图像相同
- ksize 表示高斯滤波器模板大小，ksize.width 和 ksize.height 可以不同，但它们都必须是正数和奇数，它们也可以是零，即（0, 0）
- sigmaX 表示高斯核函数在 X 方向的高斯内核标准差
- sigmaY 表示高斯核函数在 Y 方向的高斯内核标准差。如果 sigmaY 为零，则设置为等于 sigmaX，如果两个 sigma 均为零，则分别从 ksize.width 和 ksize.height 计算得到
- borderType 表示边框模式，用于推断图像外部像素的某种边界模式，默认值为 BORDER_DEFAULT，可省略

下面代码是使用 7×7 核模板进行高斯滤波处理。

```
# -*- coding: utf-8 -*-
```

```
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('lena-zs.png')
source = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#高斯滤波
result = cv2.GaussianBlur(source, (7,7), 0)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', '高斯滤波']
images = [source, result]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
```

```
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 23-11 所示，左边为待处理图像，右边为高斯滤波处理后图像。

图 23-11 高斯滤波 7×7 核的处理

图 23-12 是使用 15×15 高斯核模板进行高斯滤波处理的效果图，由图可知，图像在去除噪声的同时也变得更加模糊。

图 23-12 高斯滤波 15×15 核的处理

总之，高斯滤波作为最有效的滤波器之一，它对于抑制服从正态分布的噪声

非常有效。

5.总结

本文主要讲解了常用于消除噪声的图像平滑方法，常见方法包括三种线性滤波（均值滤波、方框滤波、高斯滤波）和两种非线性滤波（中值滤波、双边滤波）。这篇文章介绍了均值滤波、方框滤波和高斯滤波，通过原理和代码进行对比，分别讲述了各种滤波方法的优缺点，有效地消除了图像的噪声，并保留图像的边缘轮廓。

参考文献：

[1] 冈萨雷斯著，阮秋琦译. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] zhu_hongji. [OpenCV 学习笔记] 之图像平滑（线性/非线性滤波器）[EB/OL]. (2018-08-11). https://blog.csdn.net/zhu_hongji/article/details/81479571.

[3] 陆瑶. 图像处理与 matlab 实例之图像平滑（一）[EB/OL]. (2017-07-23). https://www.cnblogs.com/luyaoblog/p/7160948.html.

[4] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[5] 石振刚. 基于模糊逻辑的图像处理算法研究[D]. 东北大学, 2009.

[6] 马光豪. 基于稀疏高频梯度和联合双边滤波的图像平滑算法研究[D].山东大学, 2018.

[7] 陈初侠. 图像滤波及边缘检测与增强技术研究[D].合肥工业大学, 2009.

[8] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[9] Eastmount. [Python 图像处理] 四.图像平滑之均值滤波、方框滤波、高斯滤波及中值滤波 [EB/OL]. (2018-09-02). https://blog.csdn.net/Eastmount/article/details/82216380.

[10] Eastmount. [数字图像处理] 七.MFC 图像增强之图像普通平滑、高斯平滑、Laplacian 、 Sobel 、 Prewitt 锐化详解 [EB/OL]. (2015-06-08). https://blog.csdn.net/eastmount/article/ details/46378783.

第 24 篇 图像平滑之中值滤波、双边滤波

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

上一篇文章介绍图像平滑知识，包括均值滤波、方框滤波和高斯滤波。这篇文章将继续讲解图像平滑知识，包括中值滤波和双边滤波。常用于消除噪声的图像平滑方法包括三种线性滤波（均值滤波、方框滤波、高斯滤波）和两种非线性滤波（中值滤波、双边滤波），本文将详细讲解两种非线性滤波方法。

1.中值滤波

前面讲述的都是线性平滑滤波，它们的中间像素值都是由邻域像素值线性加权得到的，接下来将讲解一种非线性平滑滤波——中值滤波。中值滤波通过计算每一个像素点某邻域范围内所有像素点灰度值的中值，来替换该像素点的灰度值，从而让周围的像素值更接近真实情况，消除孤立的噪声。

中值滤波对脉冲噪声有良好的滤除作用，特别是在滤除噪声的同时，能够保护图像的边缘和细节，使之不被模糊处理，这些优良特性是线性滤波方法所不具有的，从而使其常常被应用于消除图像中的椒盐噪声[1-2]。

中值滤波算法的计算过程如图 24-1 所示。选择含有五个点的窗口，依次扫

描该窗口中的像素，每个像素点所对应的灰度值按照升序或降序排列，然后获取最中间的值来替换该点的灰度值。

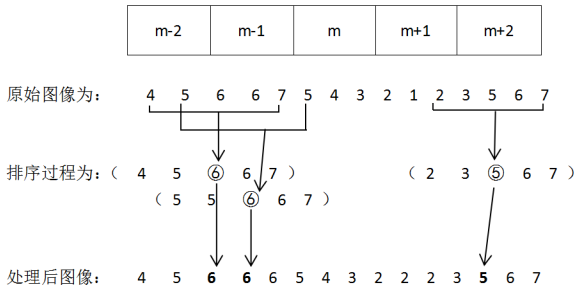


图 24-1 中值滤波算法计算过程

上图展示的是矩形窗口，常用的窗口还包括正方形、十字形、环形和圆形等，不同形状的窗口会带来不同的过滤效果，其中正方形和圆形窗口适合于外轮廓边缘较长的图像，十字形窗口适合于带尖角形状的图像。

OpenCV 将中值滤波封装在 medianBlur()函数中，其函数原型如下所示：

```
dst = medianBlur(src, ksize[, dst])
```

- src 表示待处理的输入图像
- dst 表示输出图像，其大小和类型与输入图像相同
- ksize 表示内核大小，其值必须是大于 1 的奇数，如 3、5、7 等

下面是调用 medianBlur()函数实现中值滤波的代码。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
#读取图片
img = cv2.imread('lena-zs.png')
source = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#中值滤波
result = cv2.medianBlur(source, 3)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', '中值滤波']
images = [source, result]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 24-1 所示，它有效地过滤掉了“lena”图中的噪声，并且很好地保护了图像的边缘信息，使之不被模糊处理。

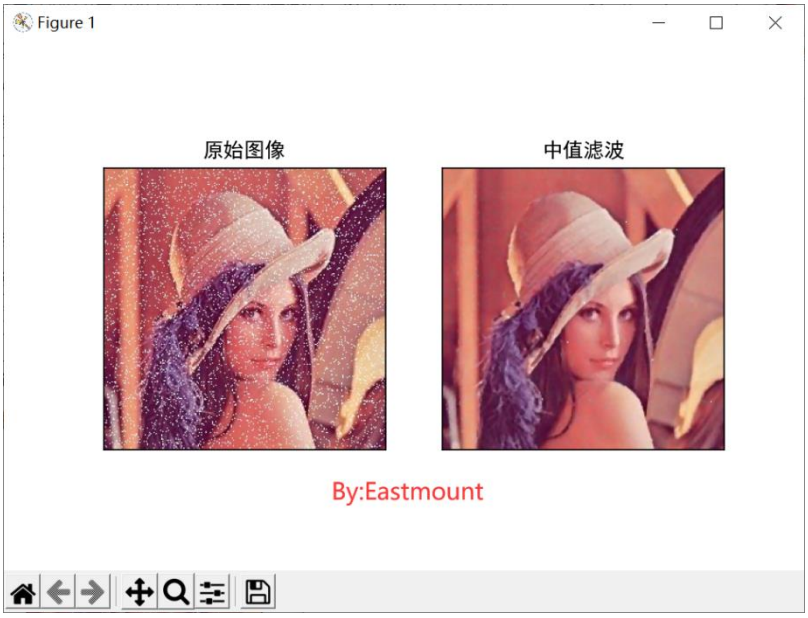


图 24-1 中值滤波 3×3 核的处理

2.双边滤波

双边滤波（Bilateral filter）是由 Tomasi 和 Manduchi 在 1998 年发明的一种各向异性滤波，它一种非线性的图像平滑方法，结合了图像的空间邻近度和像素值相似度（即空间域和值域）的一种折中处理，从而达到保边去噪的目的。双边滤波的优势是能够做到边缘的保护，其他的均值滤波、方框滤波和高斯滤波在去除噪声的同时，都会有较明显的边缘模糊，对于图像高频细节的保护效果并不好[3]。

双边滤波比高斯滤波多了一个高斯方差 sigma-d，它是基于空间分布的高斯滤波函数。所以在图像边缘附近，离的较远的像素点不会过于影响到图像边缘上的像素点，从而保证了图像边缘附近的像素值得以保存。但是双边滤波也存在一定的缺陷，由于它保存了过多的高频信息，双边滤波不能有效地过滤掉彩色图

像中的高频噪声，只能够对低频信息进行较好地去噪[4]。

在双边滤波器中，输出的像素值依赖于邻域像素值的加权值组合，对输入图像进行局部加权平均得到输出图像 $\hat{f}$ 的像素值，其公式如下所示：

$$\hat{f}(x,y)=\frac{\sum_{(i,j)\in D_{x,y}} g(x,y)w(x,y,i,j)}{\sum_{(i,j)\in D_{x,y}} w(x,y,i,j)} \qquad (24-1)$$

式中$D_{x,y}$表示中心点(x,y)的(2N+1)×(2N+1)的领域像素，$\hat{f}(x,y)$值依赖于领域像素值$g(x,y)$的加权平均。权重系数$w(x,y,i,j)$取决于空间域核(domain)和值域核（ range ）的乘积。

空间域核的定义如公式（ 24-2 ）所示。

$$w_{\mathrm{d}}(x,y,i,j)=\exp(-\frac{(x-i)^2+(y-j)^2}{2\sigma_d^2}) \qquad (24-2)$$

值域核的定义如公式（ 24-3 ）所示。

$$w_{\mathrm{s}}(x,y,i,j)=\exp(-\frac{|g(x,y)-g(i,j)|^2}{2\sigma_s^2}) \qquad (24-3)$$

两者相乘之后，就会产生依赖于数据的双边滤波权重函数，如下所示：

$$w_{\mathrm{s}}(x,y,i,j)=w_{\mathrm{d}}(x,y,i,j)\times w_{\mathrm{s}}(x,y,i,j) \qquad (24-4)$$

从式子（ 24-4 ）可以看出，双边滤波器的加权系数是空间邻近度因子w_d和像素亮度相似因子w_s的非线性组合。前者随着像素点与中心点之间欧几里德距离的增加而减小，后者随着像素亮度之差的增大而减小[5-6]。

在图像变化平缓的区域，邻域内亮度值相差不大，双边滤波器转化为高斯低通滤波器；在图像变化剧烈的区域，邻域内像素亮度值相差较大，滤波器利用边缘点附近亮度值相近的像素点的亮度平均值替代原亮度值。因此，双边滤波器既

平滑了图像，又保持了图像边缘，其原理图如图 24-2 所示。

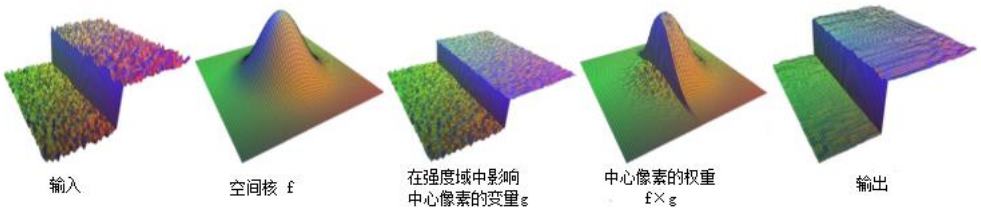


图 24-2 双边滤波原理

OpenCV 将中值滤波封装在 bilateralFilter()函数中，其函数原型如下所示：

dst = bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]])

- src 表示待处理的输入图像
- dst 表示输出图像，其大小和类型与输入图像相同
- d 表示在过滤期间使用的每个像素邻域的直径。如果这个值我们设其为非正数，则它会由 sigmaSpace 计算得出
- sigmaColor 表示颜色空间的标准方差。该值越大，表明像素邻域内较远的颜色会混合在一起，从而产生更大面积的半相等颜色区域
- sigmaSpace 表示坐标空间的标准方差。该值越大，表明像素的颜色足够接近，从而使得越远的像素会相互影响，更大的区域中相似的颜色获取相同的颜色，当 d>0，d 指定了邻域大小且与 sigmaSpace 无关。否则，d 正比于 sigmaSpace
- borderType 表示边框模式，用于推断图像外部像素的某种边界模式，默认值为 BORDER_DEFAULT，可省略

下面是调用 bilateralFilter()函数实现双边滤波的代码，其中 d 为 15，sigmaColor 设置为 150，sigmaSpace 设置为 150。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('lena-zs.png')
source = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#双边滤波
result = cv2.bilateralFilter(source, 15, 150, 150)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', '双边滤波']
images = [source, result]
```

```
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 24-3 所示：

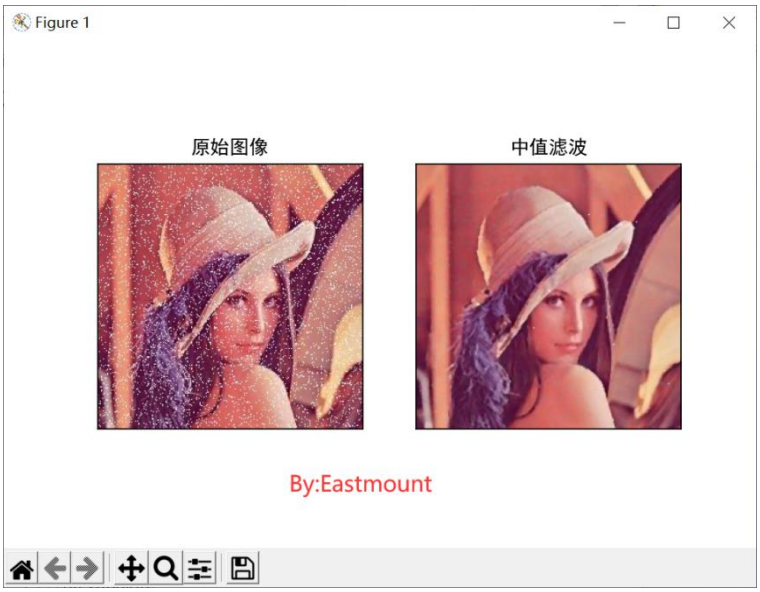


图 24-3 双边滤波处理

3.总结

本文主要讲解了常用于消除噪声的图像平滑方法，常见方法包括三种线性滤波（均值滤波、方框滤波、高斯滤波）和两种非线性滤波（中值滤波、双边滤波）。这篇文章介绍了中值滤波和双边滤波，通过原理和代码进行对比，分别讲述了各种滤波方法的优缺点，有效地消除了图像的噪声，并保留图像的边缘轮廓。

参考文献：

[1] 冈萨雷斯著，阮秋琦译. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 陈初侠. 图像滤波及边缘检测与增强技术研究[D].合肥工业大学, 2009.

[4] Eastmount. [Python 图像处理] 四.图像平滑之均值滤波、方框滤波、高斯滤波及中值滤波 [EB/OL]. (2018-09-02). https://blog.csdn.net/Eastmount/article/details/82216380.

[5] Eastmount. [数字图像处理] 七.MFC 图像增强之图像普通平滑、高斯平滑、Laplacian 、 Sobel 、 Prewitt 锐化详解 [EB/OL]. (2015-06-08). https://blog.csdn.net/eastmount/article/ details/46378783.

[6] 毛星云. [OpenCV 入门教程之九] 非线性滤波专场：中值滤波、双边滤波[EB/OL]. (2014-04-08). https://blog.csdn.net/poem_qianmo/article/details/23184547.

[7] C. Tomasi, R Manduchi. Bilateral Filtering for Gray and Color images[C]. Proceedings of the IEEE International Conference on Computer Vision, Bombay, India. 1998:839-846.

第 25 篇 图像锐化之 Roberts、Prewitt 算子实现边缘检测

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

在图像收集和传输过程中，可能会受一些外界因素造成图像模糊和有噪声的情况，从而影响到后续的图像处理和识别。此时可以通过图像锐化和边缘检测，加强原图像的高频部分，锐化突出图像的边缘细节，改善图像的对比度，使模糊的图像变得更清晰。图像锐化和边缘检测主要包括一阶微分锐化和二阶微分锐化，本文主要讲解常见的图像锐化和边缘检测方法，即 Roberts 算子和 Prewitt 算子，希望对您有所帮助。

1.图像锐化

由于收集图像数据的器件或传输图像的通道存在一些质量缺陷，或者受其他外界因素的影响，使得图像存在模糊和有噪声的情况，从而影响到图像识别工作的开展。一般来说，图像的能量主要集中在其低频部分，噪声所在的频段主要在高频段，同时图像边缘信息主要集中在其高频部分。这将导致原始图像在平滑处理之后，图像边缘和图像轮廓模糊的情况出现。为了减少这类不利效果的影响，

就需要利用图像锐化技术，使图像的边缘变得清晰[1]。

图像锐化处理的目的是为了使图像的边缘、轮廓线以及图像的细节变得清晰，经过平滑的图像变得模糊的根本原因是图像受到了平均或积分运算，因此可以对其进行逆运算，从而使图像变得清晰。微分运算是求信号的变化率，具有较强高频分量作用。从频率域来考虑，图像模糊的实质是因为其高频分量被衰减，因此可以用高通滤波器来使图像清晰。但要注意能够进行锐化处理的图像必须有较高的性噪比，否则锐化后图像性噪比反而更低，从而使得噪声增加比信号还要多，因此一般是先去除或减轻噪声后再进行锐化处理。这时需要开展图像锐化和边缘检测处理，加强原图像的高频部分，锐化突出图像的边缘细节，改善图像的对比度，使模糊的图像变得更清晰。

图像锐化和边缘提取技术可以消除图像中的噪声，提取图像信息中用来表征图像的一些变量，为图像识别提供基础。通常使用灰度差分法对图像的边缘、轮廓进行处理，将其凸显。图像锐化的方法分为高通滤波和空域微分法，本章主要介绍 Robert 算子、Prewitt 算子、Sobel 算子、Laplacian 算子、Scharr 算子等[2-3]。

（1）一阶微分算子

一阶微分算子一般借助空域微分算子通过卷积完成，但实际上数字图像处理中求导是利用差分近似微分来进行的。梯度对应一阶导数，梯度算子是一阶导数算子。对一个连续函数 f(x,y)，它在位置(x,y)梯度可表示为一个矢量：

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \tag{25-1}$$

梯度的模值为公式（25-2）所示。

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{25-2}$$

梯度的方向在$f(x,y)$最大变化率方向上，梯度方向如公式（25-3）所示。

$$\angle\nabla f(x,y) = arctan\left(\frac{\partial f}{\partial x}\right)/\left(\frac{\partial f}{\partial x}\right) \tag{25-3}$$

对于数字图像，导数可以用差分来近似，则梯度可以表示为：

$$\nabla f \approx \big(f(i+1,j) - f(i,j), f(i,j+1) - f(i,j)\big) \tag{25-4}$$

在实际中常用区域模板卷积来近似计算，对水平方向和垂直方向各用一个模板，再通过两个模板组合起来构成一个梯度算子。根据模板的大小，其中元素值的不同，可以提出多种模板，构成不同的检测算子，后文中将对各种算子进行详细介绍。

由梯度的计算可知，在图像灰度变化较大的边沿区域其梯度值大，在灰度变化平缓的区域梯度值较小，而在灰度均匀的区域其梯度值为零。根据得到的梯度值来返回像素值，如将梯度值大的像素设置成白色，梯度值小的设置为黑色，这样就可以将边缘提取出来了，或者是加强梯度值大的像素灰度值就可以突出细节了达到了锐化的目的。

（2）二阶微分算子

二阶微分算子是求图像灰度变化导数的导数，对图像中灰度变化强烈的地方很敏感，从而可以突出图像的纹理结构。当图像灰度变化剧烈时，进行一阶微分则会形成一个局部的极值，对图像进行二阶微分则会形成一个过零点，并且在零点两边产生一个波峰和波谷，设定一个阈值检测到这个过零点，如图 25-1 所示。

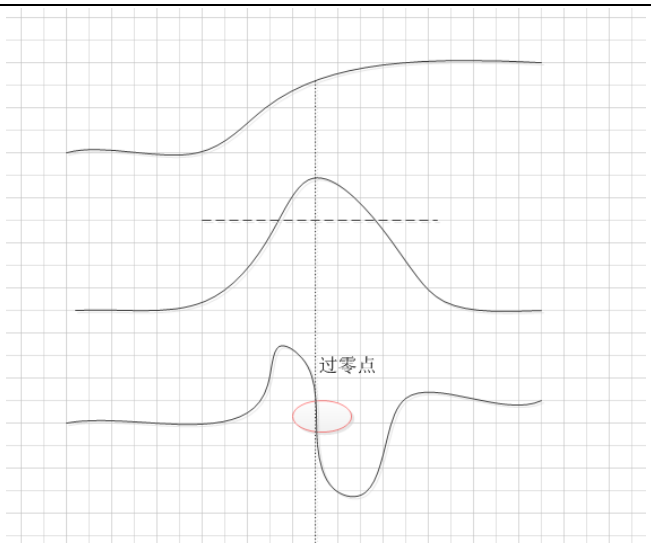


图 25-1 二阶微分算子原理

这样做的好处有两个，一是二阶微分关心的是图像灰度的突变而不强调灰度缓慢变化的区域，对边缘的定位能力更强；二是 Laplacian 算子是各向同性的，即具有旋转不变性，在一阶微分里，是用|dx|+|dy|来近似一个点的梯度，当图像旋转一个角度时，这个值就会变化，但对于 Laplacian 算子来说，不管图像怎么旋转，得到的相应值是一样的。

想要确定过零点要以 p 为中心的一个 3×3 领域，p 点为过零点意味着至少有两个相对的领域像素的符号不同。有四种要检测的情况：左/右、上/下、两个对角。如果 g(x,y)的值与一个阈值比较，那么不仅要求相对领域的符号不同，数值差的绝对值也要超过这个阈值，这时 p 称为一个过零点像素。二阶微分的定义为：

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x) \tag{25-5}$$

二阶微分在恒定灰度区域的微分值为零，在灰度台阶或斜坡起点处微分值非零，沿着斜坡的微分值为零。与一阶微分算子相比较，一阶微分算子获得的边界

是比较粗略的边界，反映的边界信息较少，但是所反映的边界比较清晰；二阶微分算子获得的边界是比较细致的边界，反映的边界信息包括了许多的细节信息，但是所反映的边界不是太清晰。

2.Roberts 算子

Roberts 算子又称为交叉微分算法，它是基于交叉差分的梯度算法，通过局部差分计算检测边缘线条。常用来处理具有陡峭的低噪声图像，当图像边缘接近于正 45 度或负 45 度时，该算法处理效果更理想，其缺点是对边缘的定位不太准确，提取的边缘线条较粗。

Roberts 算子的模板分为水平方向和垂直方向，如公式（25-6）所示，从其模板可以看出，Roberts 算子能较好的增强正负 45 度的图像边缘[4]。

$$d_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad d_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{25-6}$$

如公式（25-7）所示，分别表示图像的水平方向和垂直方向的计算公式。

$$\begin{aligned} d_x(i,j) &= f(i+1,j+1) - f(i,j) \\ d_y(i,j) &= f(i,j+1) - f(i+1,j) \end{aligned} \tag{25-7}$$

Roberts 算子像素的最终计算公式如下：

$$S = \sqrt{\left(d_x(i,j)^2 + d_y(i,j)^2\right)} \tag{25-8}$$

在 Python 中，Roberts 算子主要通过 Numpy 定义模板，再调用 OpenCV 的 filter2D()函数实现边缘提取[3]。该函数主要是利用内核实现对图

像的卷积运算，其函数原型如下所示：

```
dst = filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])
```

- src 表示输入图像
- dst 表示输出的边缘图，其大小和通道数与输入图像相同
- ddepth 表示目标图像所需的深度
- kernel 表示卷积核，一个单通道浮点型矩阵
- anchor 表示内核的基准点，其默认值为（-1，-1），位于中心位置
- delta 表示在储存目标图像前可选的添加到像素的值，默认值为 0
- borderType 表示边框模式

在进行 Roberts 算子处理之后，还需要调用 convertScaleAbs()函数计算绝对值，并将图像转换为 8 位图进行显示。其算法原型如下：

```
dst = convertScaleAbs(src[, dst[, alpha[, beta]]])
```

- src 表示原数组
- dst 表示输出数组，深度为 8 位
- alpha 表示比例因子
- beta 表示原数组元素按比例缩放后添加的值

最后调用 addWeighted()函数计算水平方向和垂直方向的 Roberts 算子。其运行代码如下：

```
# -*- coding: utf-8 -*-
```

```
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#Roberts 算子
kernelx = np.array([[-1,0],[0,1]], dtype=int)
kernely = np.array([[0,-1],[1,0]], dtype=int)
x = cv2.filter2D(grayImage, cv2.CV_16S, kernelx)
y = cv2.filter2D(grayImage, cv2.CV_16S, kernely)
#转 uint8
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Roberts = cv2.addWeighted(absX,0.5,absY,0.5,0)
```

```
#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', 'Roberts 算子']
images = [lenna_img, Roberts]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 25-2 所示，左边为原始图像，右边为 Roberts 算子图像锐化提取的边缘轮廓。

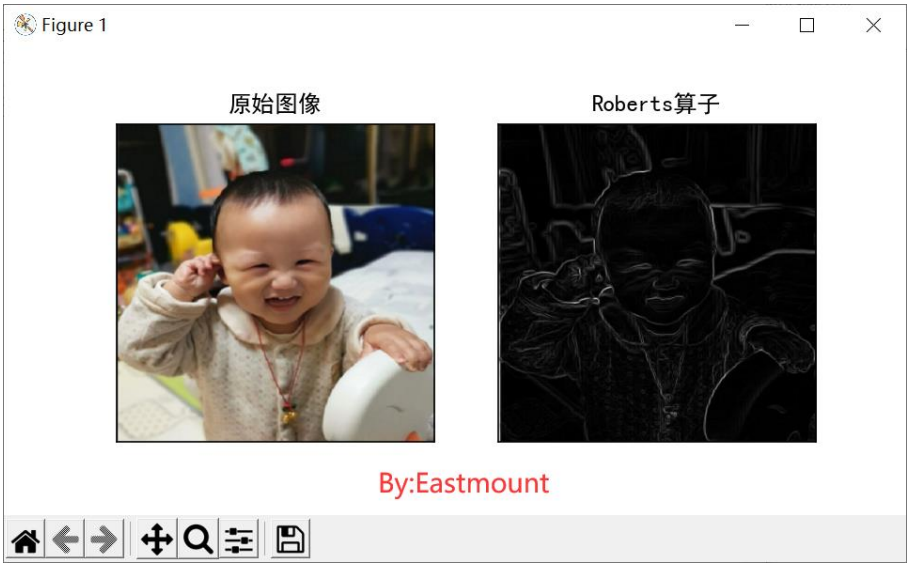


图 25-2 Roberts 算子边缘提取

3.Prewitt 算子

Prewitt 是一种图像边缘检测的微分算子，其原理是利用特定区域内像素灰度值产生的差分实现边缘检测。由于 Prewitt 算子采用 3×3 模板对区域内的像素值进行计算，而 Robert 算子的模板为 2×2，故 Prewitt 算子的边缘检测结果在水平方向和垂直方向均比 Robert 算子更加明显。Prewitt 算子适合用来识别噪声较多、灰度渐变的图像，其计算公式如下所示。

$$d_x=\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix}\quad d_y=\begin{bmatrix}-1 & -1 & -1\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix} \tag{25-9}$$

具体的水平和垂直方向计算公式如下所示：

$$d_x(i,j)=[f(i-1,j-1)+f(i-1,j)+f(i-1,j+1)]-[f(i+1,j-1)+f(i+1,j)+f(i+1,j+1)]$$

$$d_y(i,j)=[f(i-1,j+1)+f(i,j+1)+f(i+1,j+1)]-[f(i-1,j-1)+f(i,j-1)+f(i+1,j-1)] \tag{25-10}$$

Prewitt 算子像素的最终计算如公式（25-11）所示。

$$S=\sqrt{(d_x(i,j)^2+d_y(i,j)^2)} \tag{25-11}$$

在 Python 中，Prewitt 算子的实现过程与 Roberts 算子比较相似。通过 Numpy 定义模板，再调用 OpenCV 的 filter2D()函数实现对图像的卷积运算，最终通过 convertScaleAbs()和 addWeighted()函数实现边缘提取，代码如下所示：

```
# -*- coding: utf-8 -*-
```

```
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#Prewitt 算子
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]],dtype=int)
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]],dtype=int)
x = cv2.filter2D(grayImage, cv2.CV_16S, kernelx)
y = cv2.filter2D(grayImage, cv2.CV_16S, kernely)
#转 uint8
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Prewitt = cv2.addWeighted(absX,0.5,absY,0.5,0)
```

```
#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', 'Prewitt 算子']
images = [lenna_img, Prewitt]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

最终运行结果如图 25-3 所示，左边为原始图像，右边为 Prewitt 算子图像锐化提取的边缘轮廓，其效果图的边缘检测结果在水平方向和垂直方向均比 Robert 算子更加明显。

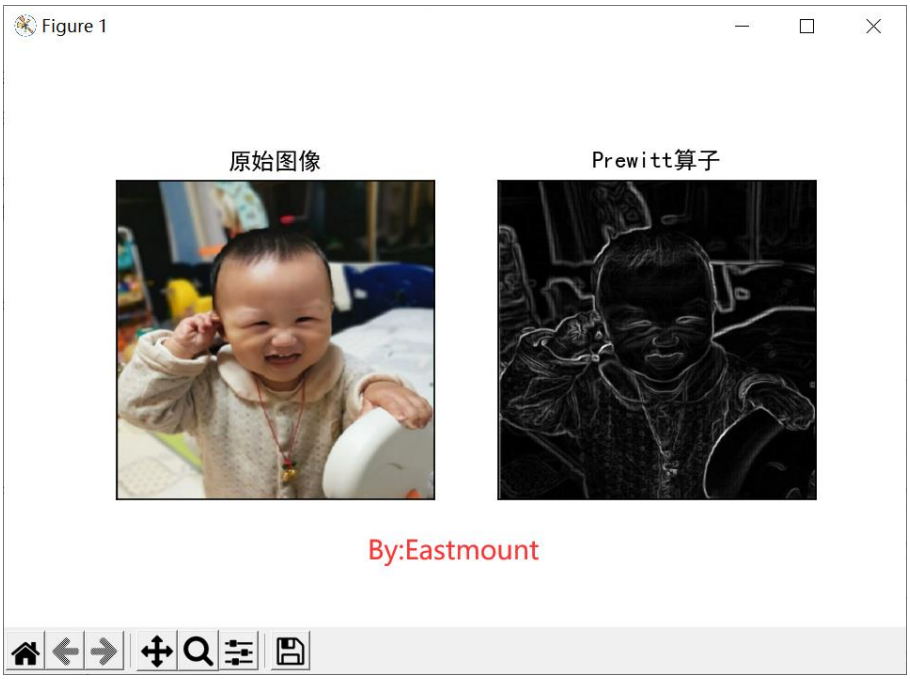


图 25-3 Prewitt 算子边缘提取

4.总结

本文主要介绍图像锐化和边缘检测知识，详细讲解了 Roberts 算子和 Prewitt 算子，并通过小珞珞图像进行边缘轮廓提取。图像锐化和边缘提取技术可以消除图像中的噪声，提取图像信息中用来表征图像的一些变量，为图像识别提供基础。

参考文献：

[1] 冈萨雷斯著，阮秋琦译. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] DSQiu. 图像锐化（增强）和边缘检测 [EB/OL]. (2012-08-20). https://dsqiu.iteye.com/blog/1638589.

[3]阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[4]杨秀璋，于小民，范郁锋，李娜. 基于苗族服饰的图像锐化和边缘提取技术研究[J]. 现代计算机，2018-10.

[5]ChuanjieZhu. 图像边缘检测——一阶微分算子 Roberts、Sobel、Prewitt、Kirsch 、 Robinson （ Matlab 实 现 ） [EB/OL]. (2017-10-25). https://blog.csdn.net/u014485485/article/details/78339420.

[6]Eastmount. [数字图像处理] 七.MFC 图像增强之图像普通平滑、高斯平滑、Laplacian 、 Sobel 、 Prewitt 锐 化 详 解 [EB/OL]. (2015-06-08). https://blog.csdn.net/eastmount/article/details/ 46378783.

第 26 篇 图像锐化之 Sobel、Laplacian 算子实现边缘检测

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

在图像收集和传输过程中，可能会受一些外界因素造成图像模糊和有噪声的情况，从而影响到后续的图像处理和识别。此时可以通过图像锐化和边缘检测，加强原图像的高频部分，锐化突出图像的边缘细节，改善图像的对比度，使模糊的图像变得更清晰。图像锐化和边缘检测主要包括一阶微分锐化和二阶微分锐化，本文主要讲解常见的图像锐化和边缘检测方法，即 Sobel 算子和 Laplacian 算子，希望对您有所帮助。

1.Sobel 算子

Sobel 算子是一种用于边缘检测的离散微分算子，它结合了高斯平滑和微分求导。该算子用于计算图像明暗程度近似值，根据图像边缘旁边明暗程度把该区域内超过某个数的特定点记为边缘。Sobel 算子在 Prewitt 算子的基础上增加了权重的概念，认为相邻点的距离远近对当前像素点的影响是不同的，距离越近的像素点对应当前像素的影响越大，从而实现图像锐化并突出边缘轮廓[1-4]。

Sobel 算子的边缘定位更准确，常用于噪声较多、灰度渐变的图像。其算法模板如公式（26-1）所示，其中 dx 表示水平方向，dy 表示垂直方向[3]。

$$d_x=\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}\quad d_y=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \tag{26-1}$$

其像素计算公式如下：

$$\begin{aligned} d_x(i,j)=&[f(i-1,j-1)+2f(i-1,j)+f(i-1,j+1)]\\ &-[f(i+1,j-1)+2f(i+1,j)+f(i+1,j+1)]\\ d_y(i,j)=&[f(i-1,j+1)+2f(i,j+1)+f(i+1,j+1)]\\ &-[f(i-1,j-1)+2f(i,j-1)+f(i+1,j-1)] \end{aligned} \tag{26-2}$$

Sobel 算子像素的最终计算公式如下：

$$S=\sqrt{\left(d_x(i,j)^2+d_y(i,j)^2\right)} \tag{26-3}$$

Sobel 算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。对噪声具有平滑作用，提供较为精确的边缘方向信息。因为 Sobel 算子结合了高斯平滑和微分求导（分化），因此结果会具有更多的抗噪性，当对精度要求不是很高时，Sobel 算子是一种较为常用的边缘检测方法。

Python 和 OpenCV 将 Sobel 算子封装在 Sobel()函数中，其函数原型如下所示：

```
dst = Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])
```

- src 表示输入图像
- dst 表示输出的边缘图，其大小和通道数与输入图像相同

- ddepth 表示目标图像所需的深度，针对不同的输入图像，输出目标图像有不同的深度
- dx 表示 x 方向上的差分阶数，取值 1 或 0
- dy 表示 y 方向上的差分阶数，取值 1 或 0
- ksize 表示 Sobel 算子的大小，其值必须是正数和奇数
- scale 表示缩放导数的比例常数，默认情况下没有伸缩系数
- delta 表示将结果存入目标图像之前，添加到结果中的可选增量值
- borderType 表示边框模式，更多详细信息查阅 BorderTypes

注意，在进行 Sobel 算子处理之后，还需要调用 convertScaleAbs()函数计算绝对值，并将图像转换为 8 位图进行显示。其算法原型如下：

```
dst = convertScaleAbs(src[, dst[, alpha[, beta]]])
```

- src 表示原数组
- dst 表示输出数组，深度为 8 位
- alpha 表示比例因子
- beta 表示原数组元素按比例缩放后添加的值

Sobel 算子的实现代码如下所示。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
```

```
import matplotlib.pyplot as plt


#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)


#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#Sobel 算子
x = cv2.Sobel(grayImage, cv2.CV_16S, 1, 0) #对 x 求一阶导
y = cv2.Sobel(grayImage, cv2.CV_16S, 0, 1) #对 y 求一阶导
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)


#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']


#显示图形
titles = ['原始图像', 'Sobel 算子']
```

```
images = [lenna_img, Sobel]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 26-1 所示：

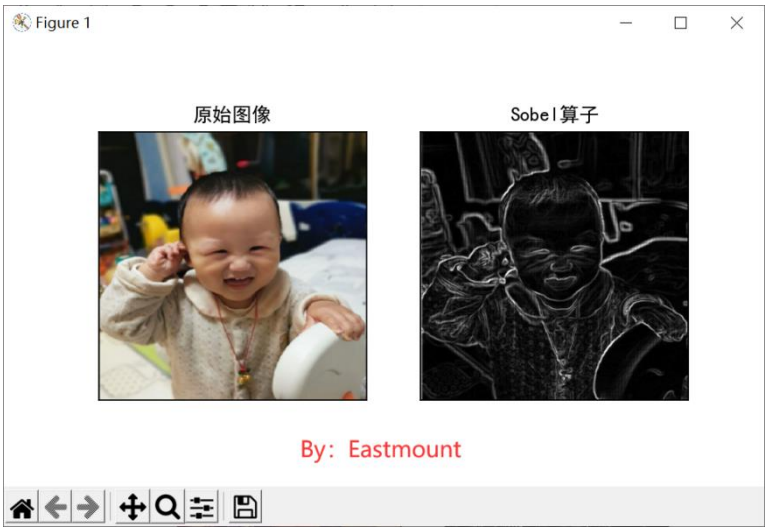


图 26-1 Sobel 算子边缘提取

2.Laplacian 算子

拉普拉斯(Laplacian)算子是 n 维欧几里德空间中的一个二阶微分算子，常用于图像增强领域和边缘提取。它通过灰度差分计算邻域内的像素，基本流程是：

- 判断图像中心像素灰度值与它周围其他像素的灰度值；

- 如果中心像素的灰度更高，则提升中心像素的灰度；
- 反之降低中心像素的灰度，从而实现图像锐化操作。

在算法实现过程中，Laplacian 算子通过对邻域中心像素的四方向或八方向求梯度，再将梯度相加起来判断中心像素灰度与邻域内其他像素灰度的关系，最后通过梯度运算的结果对像素灰度进行调整[2]。

一个连续的二元函数 f(x,y)，其拉普拉斯运算定义为：

$$\nabla^2 \mathrm{f} = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{26-4}$$

Laplacian 算子分为四邻域和八邻域，四邻域是对邻域中心像素的四方向求梯度，八邻域是对八方向求梯度。其中，四邻域模板如公式（26-5）所示：

$$\mathrm{H} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{26-5}$$

其像素的计算公式可以简化为：

$$\mathrm{g}(i,j) = 4f(i,j) - f(i+1,j) - f(i-1,j) - f(i,j-1) - f(i,j+1) \tag{26-6}$$

通过模板可以发现，当邻域内像素灰度相同时，模板的卷积运算结果为 0；当中心像素灰度高于邻域内其他像素的平均灰度时，模板的卷积运算结果为正数；当中心像素的灰度低于邻域内其他像素的平均灰度时，模板的卷积为负数。对卷积运算的结果用适当的衰弱因子处理并加在原中心像素上，就可以实现图像的锐化处理。

Laplacian 算子的八邻域模板如下：

$$H=\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix} \tag{26-7}$$

其像素的计算公式可以简化为：

$$g(i,j)=8f(i,j)-f(i+1,j-1)-f(i+1,j)-f(i+1,j+1) \tag{26-8}$$

$$-f(i,j-1)-f(i,j+1)-f(i-1,j-1)-f(i-1,j)-f(i-1,j+1)$$

Python 和 OpenCV 将 Laplacian 算子封装在 Laplacian()函数中，其函数原型如下所示：

dst = Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]])

- src 表示输入图像
- dst 表示输出的边缘图，其大小和通道数与输入图像相同
- ddepth 表示目标图像所需的深度
- ksize 表示用于计算二阶导数的滤波器的孔径大小，其值必须是正数和奇数，且默认值为 1，更多详细信息查阅 getDerivKernels
- scale 表示计算拉普拉斯算子值的可选比例因子。默认值为 1，更多详细信息查阅 getDerivKernels
- delta 表示将结果存入目标图像之前，添加到结果中的可选增量值，默认值为 0
- borderType 表示边框模式，更多详细信息查阅 BorderTypes

注意，Laplacian 算子其实主要是利用 Sobel 算子的运算，通过加上

Sobel 算子运算出的图像 x 方向和 y 方向上的导数，得到输入图像的图像锐化结果。

同时，在进行 Laplacian 算子处理之后，还需要调用 convertScaleAbs() 函数计算绝对值，并将图像转换为 8 位图进行显示。其算法原型如下：

```
dst = convertScaleAbs(src[, dst[, alpha[, beta]]])
```

- src 表示原数组
- dst 表示输出数组，深度为 8 位
- alpha 表示比例因子
- beta 表示原数组元素按比例缩放后添加的值

当 ksize=1 时，Laplacian()函数采用 3×3 的孔径（四邻域模板）进行变换处理。下面的代码是采用 ksize=3 的 Laplacian 算子进行图像锐化处理，其代码如下：

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

```
#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#拉普拉斯算法
dst = cv2.Laplacian(grayImage, cv2.CV_16S, ksize = 3)
Laplacian = cv2.convertScaleAbs(dst)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', 'Laplacian 算子']
images = [lenna_img, Laplacian]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 26-2 所示：

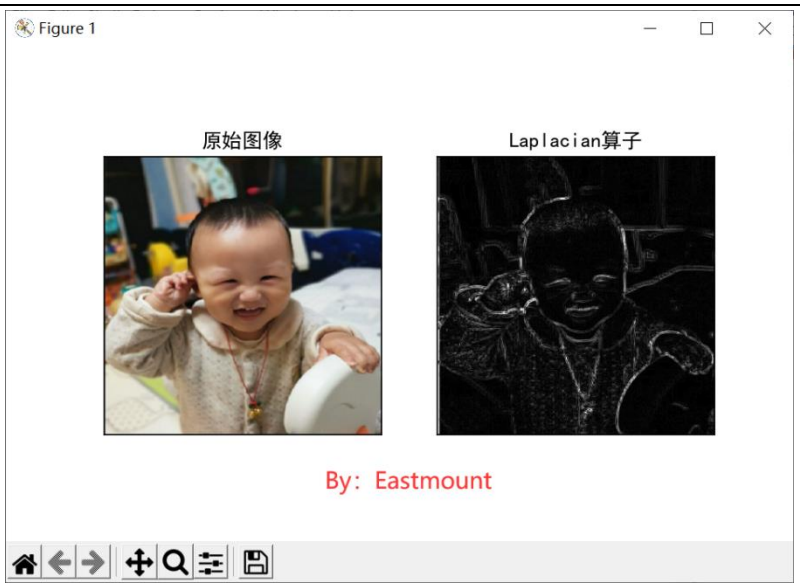


图 26-2 Laplacian 算子边缘提取

边缘检测算法主要是基于图像强度的一阶和二阶导数，但导数通常对噪声很敏感，因此需要采用滤波器来过滤噪声，并调用图像增强或阈值化算法进行处理，最后再进行边缘检测。下面是采用高斯滤波去噪和阈值化处理之后，再进行边缘检测的过程，并对比了四种常见的边缘提取算法。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#高斯滤波
gaussianBlur = cv2.GaussianBlur(grayImage, (3,3), 0)

#阈值处理
ret, binary = cv2.threshold(gaussianBlur, 127, 255,
cv2.THRESH_BINARY)

#Roberts 算子
kernelx = np.array([[-1,0],[0,1]], dtype=int)
kernely = np.array([[0,-1],[1,0]], dtype=int)
x = cv2.filter2D(binary, cv2.CV_16S, kernelx)
y = cv2.filter2D(binary, cv2.CV_16S, kernely)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Roberts = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#Prewitt 算子
```

```
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]], dtype=int)
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]], dtype=int)
x = cv2.filter2D(binary, cv2.CV_16S, kernelx)
y = cv2.filter2D(binary, cv2.CV_16S, kernely)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Prewitt = cv2.addWeighted(absX,0.5,absY,0.5,0)

#Sobel 算子
x = cv2.Sobel(binary, cv2.CV_16S, 1, 0)
y = cv2.Sobel(binary, cv2.CV_16S, 0, 1)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#拉普拉斯算法
dst = cv2.Laplacian(binary, cv2.CV_16S, ksize = 3)
Laplacian = cv2.convertScaleAbs(dst)

#效果图
titles = ['Source Image', 'Binary Image', 'Roberts Image',
```

```
            'Prewitt Image','Sobel Image', 'Laplacian Image']
images = [lenna_img, binary, Roberts, Prewitt, Sobel, Laplacian]
for i in np.arange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 26-3 所示。其中，Laplacian 算子对噪声比较敏感，由于其算法可能会出现双像素边界，常用来判断边缘像素位于图像的明区或暗区，很少用于边缘检测；Robert 算子对陡峭的低噪声图像效果较好，尤其是边缘正负 45 度较多的图像，但定位准确率较差；Prewitt 算子对灰度渐变的图像边缘提取效果较好，而没有考虑相邻点的距离远近对当前像素点的影响；Sobel 算子考虑了综合因素，对噪声较多的图像处理效果更好。

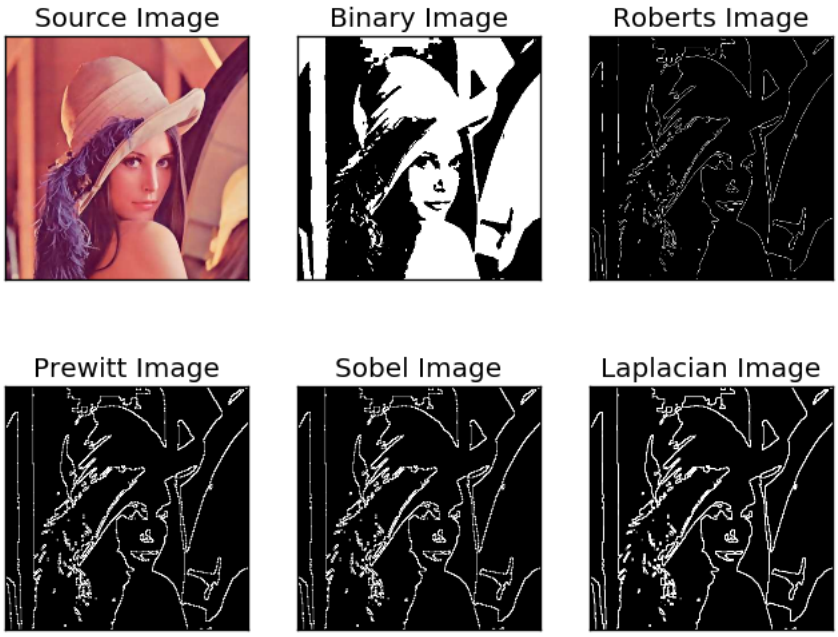


图 26-3 四种算子的边缘提取对比

3.总结

本文主要介绍图像锐化和边缘检测知识，详细讲解了 Sobel 算子和 Laplacian 算子，并通过小珞珞图像进行边缘轮廓提取。图像锐化和边缘提取技术可以消除图像中的噪声，提取图像信息中用来表征图像的一些变量，为图像识别提供基础。

参考文献：

[1] 冈萨雷斯著，阮秋琦译. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] DSQiu. 图像锐化（增强）和边缘检测 [EB/OL]. (2012-08-20).

https://dsqiu.iteye.com/blog/1638589.

[3]阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[4]杨秀璋，于小民，范郁锋，李娜. 基于苗族服饰的图像锐化和边缘提取技术研究[J]. 现代计算机，2018-10.

[5]Eastmount. [数字图像处理] 七.MFC 图像增强之图像普通平滑、高斯平滑、Laplacian 、 Sobel 、 Prewitt 锐 化 详 解 [EB/OL]. (2015-06-08). https://blog.csdn.net/eastmount/article/details/ 46378783.

第 27 篇 图像锐化之 Scharr、Canny、LOG 实现边缘检测

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

在图像收集和传输过程中，可能会受一些外界因素造成图像模糊和有噪声的情况，从而影响到后续的图像处理和识别。此时可以通过图像锐化和边缘检测，加强原图像的高频部分，锐化突出图像的边缘细节，改善图像的对比度，使模糊的图像变得更清晰。图像锐化和边缘检测主要包括一阶微分锐化和二阶微分锐化，本文主要讲解常见的图像锐化和边缘检测方法，即 Scharr 算子、Canny 算子和 LOG 算子，希望对您有所帮助。

1.Scharr 算子

由于 Sobel 算子在计算相对较小的核的时候，其近似计算导数的精度比较低，比如一个 3×3 的 Sobel 算子，当梯度角度接近水平或垂直方向时，其不精确性就越发明显。Scharr 算子同 Sobel 算子的速度一样快，但是准确率更高，尤其是计算较小核的情景，所以利用 3×3 滤波器实现图像边缘提取更推荐使用 Scharr 算子。

Scharr 算子又称为 Scharr 滤波器，也是计算 x 或 y 方向上的图像差分，在 OpenCV 中主要是配合 Sobel 算子的运算而存在的，其滤波器的滤波系数如下：

$$d_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \quad d_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \qquad (27\text{-}1)$$

Scharr 算子的函数原型如下所示，和 Sobel 算子几乎一致，只是没有 ksize 参数。

dst = Scharr(src, ddepth, dx, dy[, dst[, scale[, delta[, borderType]]]])

- src 表示输入图像
- dst 表示输出的边缘图，其大小和通道数与输入图像相同
- ddepth 表示目标图像所需的深度，针对不同的输入图像，输出目标图像有不同的深度
- dx 表示 x 方向上的差分阶数，取值 1 或 0
- dy 表示 y 方向上的差分阶数，取值 1 或 0
- scale 表示缩放导数的比例常数，默认情况下没有伸缩系数
- delta 表示将结果存入目标图像之前，添加到结果中的可选增量值
- borderType 表示边框模式，更多详细信息查阅 BorderTypes

Scharr 算子的实现代码如下所示。

```
# -*- coding: utf-8 -*-
# By:Eastmount
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Scharr 算子
x = cv2.Scharr(grayImage, cv2.CV_32F, 1, 0) #X 方向
y = cv2.Scharr(grayImage, cv2.CV_32F, 0, 1) #Y 方向
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Scharr = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']
```

```
#显示图形
titles = ['原始图像', 'Scharr 算子']
images = [lenna_img, Scharr]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 27-1 所示：

图 27-1 Scharr 算子边缘提取

2.Cann 算子

John F.Canny 于 1986 年发明了一个多级边缘检测算法——Canny 边缘检测算子，并创立了边缘检测计算理论（Computational theory of edge detection），该理论有效地解释了这项技术的工作理论。

边缘检测通常是在保留原有图像属性的情况下，对图像数据规模进行缩减，提取图像边缘轮廓的处理方式。Canny 算法是一种被广泛应用于边缘检测的标准算法，其目标是找到一个最优的边缘检测解或找寻一幅图像中灰度强度变化最强的位置。最优边缘检测主要通过低错误率、高定位性和最小响应三个标准进行评价。Canny 算子的实现步骤如下：

第一步，使用高斯平滑（如公式 27-2 所示）去除噪声。

$$K(5,5)=\frac{1}{273}\begin{bmatrix}1&4&7&4&1\\4&16&26&16&4\\7&26&41&26&7\\4&16&26&16&4\\1&4&7&4&1\end{bmatrix} \tag{27-2}$$

第二步，按照 Sobel 滤波器步骤计算梯度幅值和方向，寻找图像的强度梯度。先将卷积模板分别作用 x 和 y 方向，再计算梯度幅值和方向，其公式如下所示。梯度方向一般取 0 度、45 度、90 度和 135 度四个方向。

$$d_x=\begin{bmatrix}-1&0&+1\\-2&0&+2\\-1&0&+1\end{bmatrix}\quad d_y=\begin{bmatrix}-1&-2&-1\\0&0&0\\+1&+2&+1\end{bmatrix} \tag{27-3}$$

$$S=\sqrt{\left(d_x(i,j)^2+d_y(i,j)^2\right)} \tag{27-4}$$

$$\theta=\arctan(\frac{d_y}{d_x}) \tag{27-5}$$

第三步，通过非极大值抑制（Non-maximum Suppression）过滤掉非边缘像素，将模糊的边界变得清晰。该过程保留了每个像素点上梯度强度的极大值，过滤掉其他的值。对于每个像素点，它进行如下操作：

- 将其梯度方向近似为以下值中的一个，包括 0、45、90、135、180、225、270 和 315，即表示上下左右和 45 度方向；
- 比较该像素点和其梯度正负方向的像素点的梯度强度，如果该像素点梯度强度最大则保留，否则抑制（删除，即置为 0）。其处理后效果如图 27-2 所示，左边表示梯度值，右边表示非极大值抑制处理后的边缘。

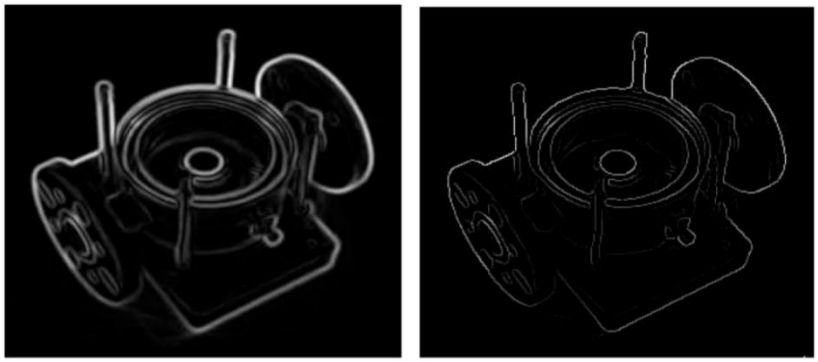


图 27-2 非极大值抑制处理

第四步，利用双阈值方法来确定潜在的边界。经过非极大抑制后图像中仍然有很多噪声点，此时需要通过双阈值技术处理，即设定一个阈值上界和阈值下界。图像中的像素点如果大于阈值上界则认为必然是边界（称为强边界，strong edge），小于阈值下界则认为必然不是边界，两者之间的则认为是候选项（称为弱边界，weak edge）。经过双阈值处理的图像如图 27-3 所示，左边为非极大值抑制处理后的边缘，右边为双阈值技术处理的效果图。

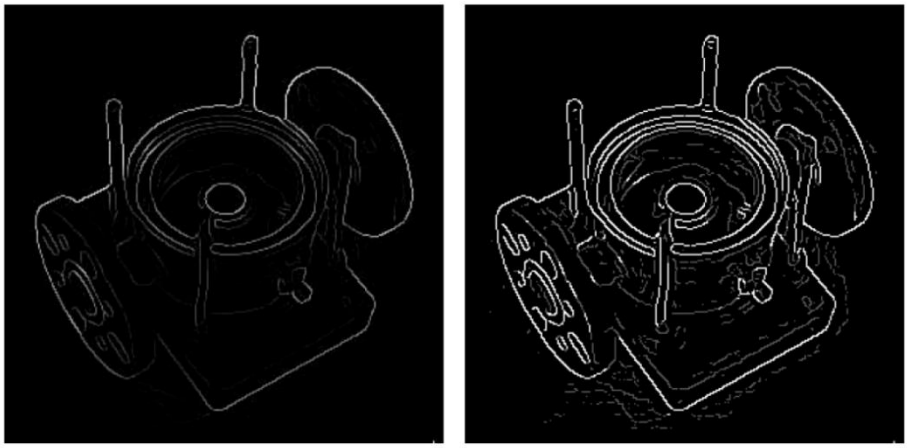


图 27-3 双阈值处理

第五步，利用滞后技术来跟踪边界。若某一像素位置和强边界相连的弱边界认为是边界，其他的弱边界则被删除。

在 OpenCV 中，Canny()函数原型如下所示：

```
edges = Canny(image, threshold1, threshold2[, edges[, apertureSize[,
 L2gradient]]])
```

- image 表示输入图像
- edges 表示输出的边缘图，其大小和类型与输入图像相同
- threshold1 表示第一个滞后性阈值
- threshold2 表示第二个滞后性阈值
- apertureSize 表示应用 Sobel 算子的孔径大小，其默认值为 3
- L2gradient 表示一个计算图像梯度幅值的标识，默认值为 false

Canny 算子的边缘提取实现代码如下所示：

```
# -*- coding: utf-8 -*-
```

```
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#高斯滤波降噪
gaussian = cv2.GaussianBlur(grayImage, (3,3), 0)

#Canny 算子
Canny = cv2.Canny(gaussian, 50, 150)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']
```

```
#显示图形
titles = ['原始图像', 'Canny 算子']
images = [lenna_img, Canny]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 27-4 所示：

图 27-4 Canny 处理

3.LOG 算子

LOG（Laplacian of Gaussian）边缘检测算子是 David Courtnay

Marr 和 Ellen Hildreth 在 1980 年共同提出的，也称为 Marr & Hildreth 算子，它根据图像的信噪比来求检测边缘的最优滤波器。该算法首先对图像做高斯滤波，然后再求其拉普拉斯（Laplacian）二阶导数，根据二阶导数的过零点来检测图像的边界，即通过检测滤波结果的零交叉（Zero crossings）来获得图像或物体的边缘。

LOG 算子综合考虑了对噪声的抑制和对边缘的检测两个方面，并且把 Gauss 平滑滤波器和 Laplacian 锐化滤波器结合了起来，先平滑掉噪声，再进行边缘检测，所以效果会更好。 该算子与视觉生理中的数学模型相似，因此在图像处理领域中得到了广泛的应用。它具有抗干扰能力强，边界定位精度高，边缘连续性好，能有效提取对比度弱的边界等特点。

常见的 LOG 算子是 5×5 模板，如下所示：

$$\begin{bmatrix} -2 & -4 & -4 & -4 & -2 \\ -4 & 0 & 8 & 0 & -4 \\ -4 & 8 & 24 & 8 & -4 \\ -4 & 0 & 8 & 0 & -4 \\ -2 & -4 & -4 & -4 & -2 \end{bmatrix} \tag{27-6}$$

由于 LOG 算子到中心的距离与位置加权系数的关系曲线像墨西哥草帽的剖面，所以 LOG 算子也叫墨西哥草帽滤波器，如图 27-5 所示。

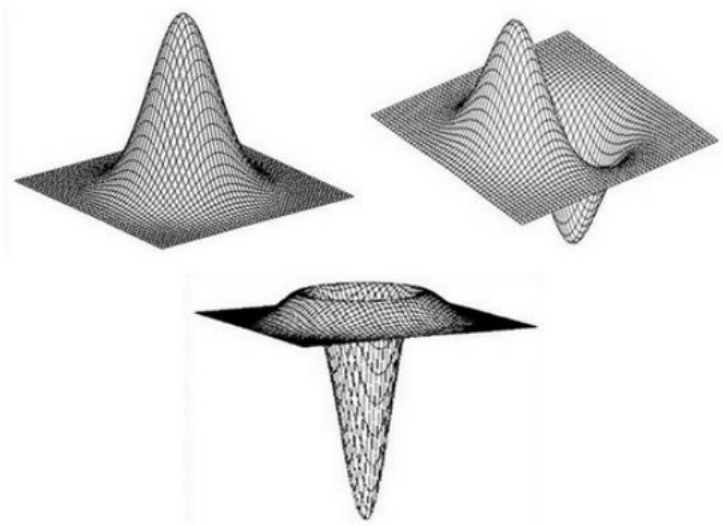


图 27-5 墨西哥草帽滤波器

LOG 算子的边缘提取实现代码如下所示：

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('luo.png')
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
#先通过高斯滤波降噪
gaussian = cv2.GaussianBlur(grayImage, (3,3), 0)

#再通过拉普拉斯算子做边缘检测
dst = cv2.Laplacian(gaussian, cv2.CV_16S, ksize = 3)
LOG = cv2.convertScaleAbs(dst)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = ['原始图像', 'LOG 算子']
images = [lenna_img, LOG]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

其运行结果如图 27-6 所示：

图 27-6 LOG 算子边缘提取

4.总结

该系列文章主要通过 Roberts 算子、Prewitt 算子、Sobel 算子、Laplacian 算子、Scharr 算子、Canny 算子和 LOG 算子实现图像锐化和边缘检测，有效地提取了图像的轮廓，并进行了详细地实验处理。

第三部分 图像识别及图像处理经典案例

- ✧ 第 47 篇 Pytorch 构建 Faster-RCNN 检测小麦图像
- ✧ 第 48 篇 GAN 入门知识详解及手写数字图像生成

第 28 篇 图像分割理论和基于阈值及边缘检测的图像分割

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

第一部分作者介绍了图像处理基础知识，第二部分介绍了图像运算和图像增强，接下来第三部分我们将详细讲解图像识别及图像处理经典案例，该部分属于高阶图像处理知识，能进一步加深我们的理解和实践能力。

图像分割是将图像分成若干具有独特性质的区域并提取感兴趣目标的技术和过程，它是图像处理和图像分析的关键步骤。主要分为基于阈值的分割方法、基于区域的分割方法、基于边缘的分割方法和基于特定理论的分割方法。本文将重点围绕图像处理实例，详细讲解各种图像分割的方法[1]。

1.图像分割

图像分割（Image Segmentation）技术是计算机视觉领域的重要研究方向，是图像语义理解和图像识别的重要一环。它是指将图像分割成若干具有相似性质的区域的过程，研究方法包括基于阈值的分割方法、基于区域的分割方法、基于边缘的分割方法和基于特定理论的分割方法（含图论、聚类、深度语义等）。该技术广泛应用于场景物体分割、人体背景分割、三维重建、车牌识别、人脸识

别、无人驾驶、增强现实等行业[2-4]。如图 28-1 所示，它将鲜花颜色划分为四个层级。

图 28-1 图像分割效果图

图像分割的目标是根据图像中的物体将图像的像素分类，并提取感兴趣的目标。图像分割是图像识别和计算机视觉至关重要的预处理，没有正确的分割就不可能有正确的识别。图像分割主要依据图像中像素的亮度及颜色，但计算机在自动处理分割时，会遇到各种困难，如光照不均匀、噪声影响、图像中存在不清晰的部分以及阴影等，常常发生图像分割错误。同时，随着深度学习和神经网络的发展，基于深度学习和神经网络的图像分割技术有效提高了分割的准确率，能够较好地解决图像中噪声和不均匀问题。

2.基于阈值的图像分割

最常用的图像分割方法是将图像灰度分为不同的等级，然后用设置灰度门限的方法确定有意义的区域或欲分割的物体边界。图像阈值化（Binarization）旨在剔除掉图像中一些低于或高于一定值的像素，从而提取图像中的物体，将图像的背景和噪声区分开来。图像阈值化可以理解为一个简单的图像分割操作，阈值

又称为临界值，它的目的是确定出一个范围，然后这个范围内的像素点使用同一种方法处理，而阈值之外的部分则使用另一种处理方法或保持原样。

阈值化处理可以将图像中的像素划分为两类颜色，常见的阈值化算法如公式（28-1）所示，当某个像素点的灰度 Gray(i,j)小于阈值 T 时，其像素设置为 0，表示黑色；当灰度 Gray(i,j)大于或等于阈值 T 时，其像素值为 255，表示白色。

$$Gray(i,j)=\begin{cases}255, & Gray(i,j)\geq T\\ 0, & Gray(i,j)<T\end{cases} \qquad (28\text{-}1)$$

在 Python 的 OpenCV 库中，提供了固定阈值化函数 threshold()和自适应阈值化函数 adaptiveThreshold()，将一幅图像进行阈值化处理，前面第 14 篇文章详细介绍了图像阈值化处理方法，下面代码对比了不同阈值化算法的图像分割结果。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取图像
img=cv2.imread('scenery.png')
grayImage=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
#阈值化处理
ret,thresh1=cv2.threshold(grayImage,127,255,cv2.THRES
H_BINARY)
ret,thresh2=cv2.threshold(grayImage,127,255,cv2.THRES
H_BINARY_INV)
ret,thresh3=cv2.threshold(grayImage,127,255,cv2.THRES
H_TRUNC)
ret,thresh4=cv2.threshold(grayImage,127,255,cv2.THRES
H_TOZERO)
ret,thresh5=cv2.threshold(grayImage,127,255,cv2.THRES
H_TOZERO_INV)

#显示结果
titles = ['Gray Image','BINARY','BINARY_INV',
          'TRUNC','TOZERO','TOZERO_INV']
images = [grayImage, thresh1, thresh2, thresh3, thresh4,
thresh5]
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
```

```
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 28-2 所示，它将彩色风景图像转换成五种对应的阈值处理效果，包括二进制阈值化（BINARY）、反二进制阈值化（BINARY_INV）、截断阈值化（THRESH_TRUNC）、阈值化为 0（THRESH_TOZERO）、反阈值化为 0（THRESH_TOZERO_INV）。

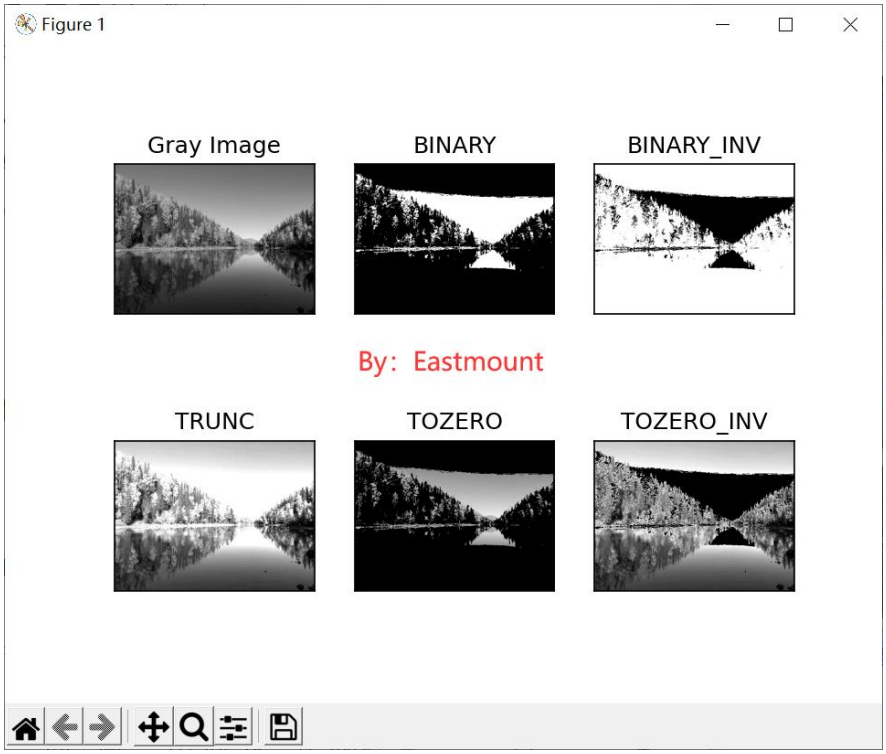


图 28-2 图像阈值化分割

3.基于边缘检测的图像分割

图像中相邻区域之间的像素集合共同构成了图像的边缘。基于边缘检测的图像分割方法是通过确定图像中的边缘轮廓像素，然后将这些像素连接起来构建区

域边界的过程。由于沿着图像边缘走向的像素值变化比较平缓，而沿着垂直于边缘走向的像素值变化比较大，根据该特点，通常会采用一阶导数和二阶导数来描述和检测边缘。在前文中，详细讲解了 Python 边缘检测的方法，下面的代码是对比常用的微分算子，如 Roberts、Prewitt、Sobel、Laplacian、Scharr、Canny、LOG 等算子。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('scenery.png')
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#高斯滤波
gaussianBlur = cv2.GaussianBlur(grayImage, (3,3), 0)
```

```
#阈值处理
ret, binary = cv2.threshold(gaussianBlur, 127, 255,
cv2.THRESH_BINARY)


#Roberts 算子
kernelx = np.array([[-1,0],[0,1]], dtype=int)
kernely = np.array([[0,-1],[1,0]], dtype=int)
x = cv2.filter2D(binary, cv2.CV_16S, kernelx)
y = cv2.filter2D(binary, cv2.CV_16S, kernely)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Roberts = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)


#Prewitt 算子
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]], dtype=int)
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]], dtype=int)
x = cv2.filter2D(binary, cv2.CV_16S, kernelx)
y = cv2.filter2D(binary, cv2.CV_16S, kernely)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Prewitt = cv2.addWeighted(absX,0.5,absY,0.5,0)
```

```
#Sobel 算子
x = cv2.Sobel(binary, cv2.CV_16S, 1, 0)
y = cv2.Sobel(binary, cv2.CV_16S, 0, 1)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#拉普拉斯算法
dst = cv2.Laplacian(binary, cv2.CV_16S, ksize = 3)
Laplacian = cv2.convertScaleAbs(dst)



# Scharr 算子
x = cv2.Scharr(gaussianBlur, cv2.CV_32F, 1, 0) #X 方向
y = cv2.Scharr(gaussianBlur, cv2.CV_32F, 0, 1) #Y 方向
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Scharr = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#Canny 算子
```

```
Canny = cv2.Canny(gaussianBlur, 50, 150)


#先通过高斯滤波降噪
gaussian = cv2.GaussianBlur(grayImage, (3,3), 0)


#再通过拉普拉斯算子做边缘检测
dst = cv2.Laplacian(gaussian, cv2.CV_16S, ksize = 3)
LOG = cv2.convertScaleAbs(dst)


#效果图
titles = ['Source Image', 'Binary Image', 'Roberts Image',
          'Prewitt Image','Sobel Image', 'Laplacian Image',
          'Scharr Image', 'Canny Image', 'LOG Image']
images = [lenna_img, binary, Roberts,
          Prewitt, Sobel, Laplacian,
          Scharr, Canny, LOG]
for i in np.arange(9):
    plt.subplot(3,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 28-3 所示，它依次为原始图像、二值化图像、Roberts 算子分割图、Prewitt 算子分割图、Sobel 算子分割图、Laplacian 算子分割图、Scharr 算子分割图、Canny 算子分割图和 LOG 算子分割图。

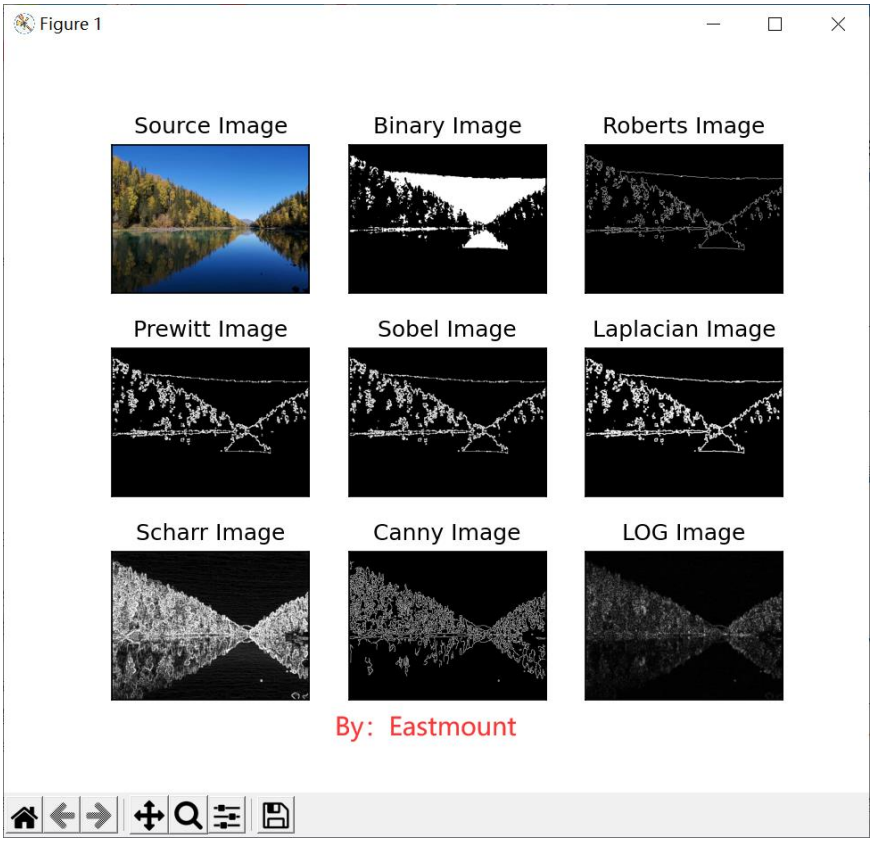


图 28-3 基于边缘检测的图像分割对比

下面讲解另一种边缘检测的方法。在 OpenCV 中，可以通过 cv2.findContours()函数从二值图像中寻找轮廓，其函数原型如下所示[3]：

```
image, contours, hierarchy = findContours(image, mode, method[, contours[, hierarchy[, offset]]])
```

- image 表示输入图像，即用于寻找轮廓的图像，为 8 位单通道
- contours 表示检测到的轮廓，其函数运行后的结果存在该变量中，

每个轮廓存储为一个点向量

- hierarchy 表示输出变量，包含图像的拓扑信息，作为轮廓数量的表示，它包含了许多元素，每个轮廓 contours[i]对应 4 个 hierarchy 元素 hierarchy[i][0]至 hierarchy[i][3]，分别表示后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号
- mode 表示轮廓检索模式。cv2.RETR_EXTERNAL 表示只检测外轮廓；cv2.RETR_LIST 表示提取所有轮廓，且检测的轮廓不建立等级关系；cv2.RETR_CCOMP 提取所有轮廓，并建立两个等级的轮廓，上面的一层为外边界，里面一层为内孔的边界信；cv2.RETR_TREE 表示提取所有轮廓，并且建立一个等级树或网状结构的轮廓
- method 表示轮廓的近似方法。cv2.CHAIN_APPROX_NONE 存储所有的轮廓点，相邻的两个点的像素位置差不超过 1，即 max（abs(x1-x2), abs(y1-y2)）=1；cv2.CHAIN_APPROX_SIMPLE 压缩水平方向、垂直方向、对角线方向的元素，只保留该方向的终点坐标，例如一个矩阵轮廓只需 4 个点来保存轮廓信息；cv2.CHAIN_APPROX_TC89_L1 和 cv2.CHAIN_APPROX_TC89_KCOS 使用 Teh-Chinl Chain 近似算法
- offset 表示每个轮廓点的可选偏移量

在使用 findContours()函数检测图像边缘轮廓后，通常需要和

drawContours()函数联合使用，接着绘制检测到的轮廓，drawContours()函数的原型如下：

```
image = drawContours(image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]])
```

- image 表示目标图像，即所要绘制轮廓的背景图片
- contours 表示所有的输入轮廓，每个轮廓存储为一个点向量
- contourIdx 表示轮廓绘制的指示变量，如果为负数表示绘制所有轮廓
- color 表示绘制轮廓的颜色
- thickness 表里绘制轮廓线条的粗细程度，默认值为 1
- lineType 表示线条类型，默认值为 8，可选线包括 8（8 连通线型）、4（4 连通线型）、CV_AA（抗锯齿线型）
- hierarchy 表示可选的层次结构信息
- maxLevel 表示用于绘制轮廓的最大等级，默认值为 INT_MAX
- offset 表示每个轮廓点的可选偏移量

下面代码是使用 cv2.findContours()检测图像轮廓，并调用cv2.drawContours()函数绘制出轮廓线条。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('scenery.png')
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#阈值化处理
ret, binary = cv2.threshold(grayImage, 0, 255,

cv2.THRESH_BINARY+cv2.THRESH_OTSU)

#边缘检测
contours, hierarchy = cv2.findContours(binary,
cv2.RETR_TREE,

cv2.CHAIN_APPROX_SIMPLE)
```

```
#轮廓绘制
cv2.drawContours(img, contours, -1, (0, 255, 0), 1)


#显示图像 5
cv2.imshow('gray', binary)
cv2.imshow('res', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图 28-4 为图像阈值化处理效果图，图 28-5 为最终提取的风景图的边缘线条。

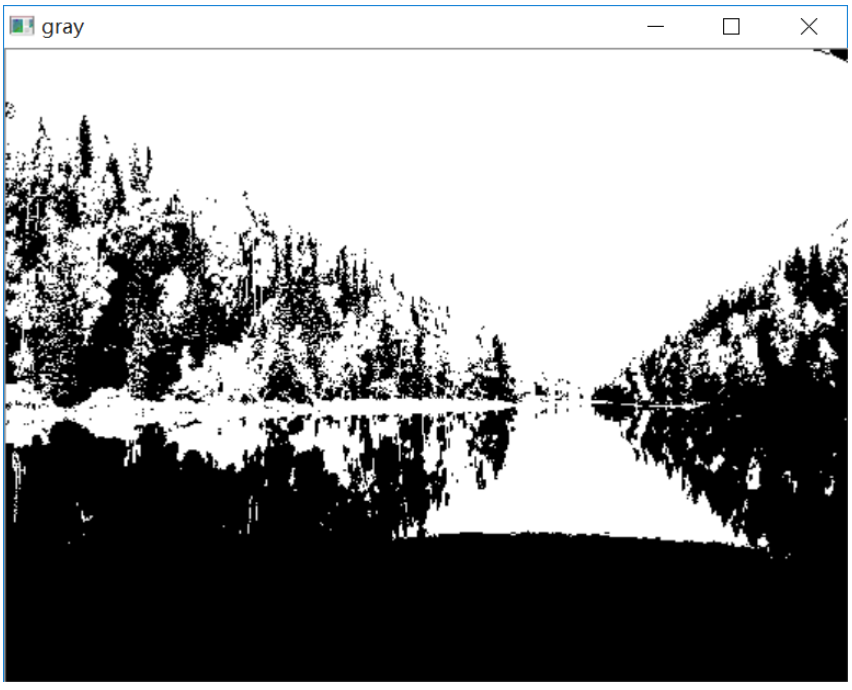


图 28-4 图像阈值化处理

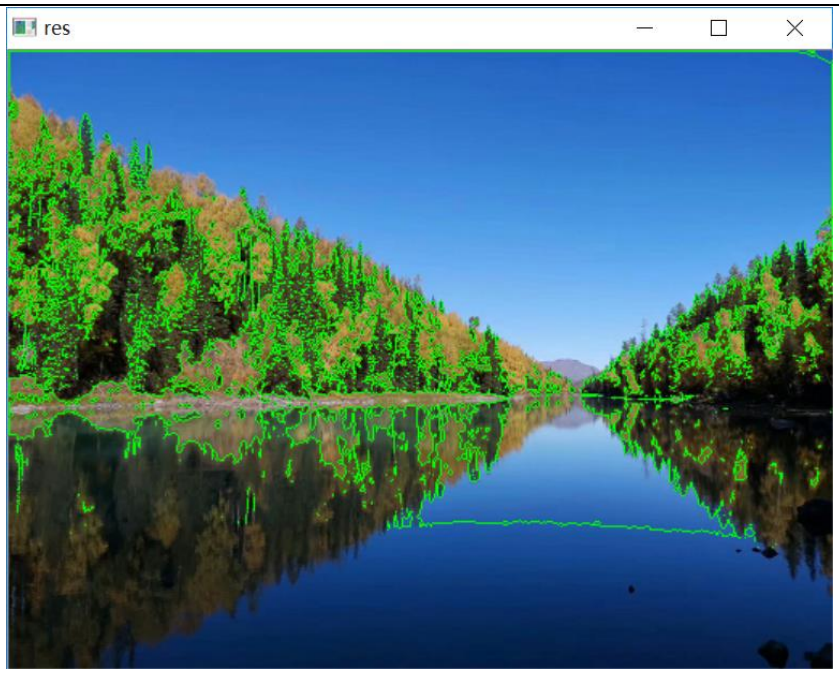


图 28-5 提取图像轮廓线条

4.总结

本文主要讲解了常用的图像分割方法，包括基于阈值的图像分割方法、基于边缘检测的图像分割方法。希望读者能结合本文知识点，围绕自己的研究领域或工程项目进行深入的学习，实现所需的图像处理。

参考文献：

[1] 冈萨雷斯著. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[4] 张铮，王艳平，薛桂香等. 数字图像处理与机器视觉——Visual C++与 Matlab 实现[M]. 北京：人民邮电出版社，2014.

[5] Jumping boy. python+OpenCV 图像处理(十一)图像轮廓检测[EB/OL]. (2018-08-30). https://blog.csdn.net/qq_40962368/article/details/82078732.

[6] Eastmount. [Python 图像处理] 四十.全网首发 Python 图像分割万字详解(阈值分割、边缘分割、纹理分割、分水岭算法、K-Means 分割、漫水填充分割、区域定位)[EB/OL]. (2021-05-18). https://blog.csdn.net/Eastmount/article/details/116952580.

[7] 杨秀璋, 颜娜. Python 网络数据爬取及分析从入门到精通 (分析篇) [M]. 北京：北京航天航空大学出版社, 2018.

[8] Fukunaga K. and Hostetler L.D. The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition[J]. IEEE Transactions on Information Theory, 1975, 21, 32-10.

第 29 篇 基于纹理背景和聚类算法的图像分割

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

图像分割是将图像分成若干具有独特性质的区域并提取感兴趣目标的技术和过程，它是图像处理和图像分析的关键步骤。主要分为基于阈值的分割方法、基于区域的分割方法、基于边缘的分割方法和基于特定理论的分割方法。上一篇文章引入了图像分割知识，这篇文章将详细讲解基于纹理背景的图像分割和基于聚类算法的图像分割。

1.基于纹理背景的图像分割

该部分主要讲解基于图像纹理信息（颜色）、边界信息（反差）和背景信息的图像分割算法。在 OpenCV 中，GrabCut 算法能够有效地利用纹理信息和边界信息分割背景，提取图像目标物体。该算法是微软研究院基于图像分割和抠图的课题，它能有效地将目标图像分割提取，如图 29-1 所示[1]。

图 29-1 GrabCut 提取图像目标物体

GrabCut 算法原型如下所示：

mask, bgdModel, fgdModel = grabCut(img, mask, rect, bgdModel, fgdModel, iterCount[, mode])

- image 表示输入图像，为 8 位三通道图像
- mask 表示蒙板图像，输入/输出的 8 位单通道掩码，确定前景区域、背景区域、不确定区域。当模式设置为 GC_INIT_WITH_RECT 时，该掩码由函数初始化
- rect 表示前景对象的矩形坐标，其基本格式为(x, y, w, h)，分别为左上角坐标和宽度、高度
- bdgModel 表示后台模型使用的数组，通常设置为大小为（1, 65）np.float64 的数组
- fgdModel 表示前台模型使用的数组，通常设置为大小为（1, 65）np.float64 的数组
- iterCount 表示算法运行的迭代次数
- mode 是 cv::GrabCutModes 操作模式之一，cv2.GC_INIT_WITH_RECT 或 cv2.GC_INIT_WITH_MASK 表示使用矩阵模式或蒙板模式

下面是 Python 的实现代码，通过调用 np.zeros()函数创建掩码、fgbModel 和 bgModel，接着定义 rect 矩形范围，调用函数 grabCut()实现图像分割。由于该方法会修改掩码，像素会被标记为不同的标志来指明它们是背景或前景。接着将所有的 0 像素和 2 像素点赋值为 0（背景），而所有的 1 像素和 3 像素点赋值为 1（前景），完整代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

#读取图像
img = cv2.imread('nv.png')

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#设置掩码、fgbModel、bgModel
mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1,65), np.float64)
```

```
fgdModel = np.zeros((1,65), np.float64)


#矩形坐标
rect = (100, 100, 500, 800)


#图像分割
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5,
            cv2.GC_INIT_WITH_RECT)


#设置新掩码：0 和 2 做背景
mask2 = np.where((mask==2)|(mask==0), 0,
1).astype('uint8')


#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']


#显示原图
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.subplot(1,2,1)
plt.imshow(img)
plt.title('(a)原始图像')
```

```
plt.xticks([]), plt.yticks([])


#使用蒙板来获取前景区域
img = img*mask2[:, :, np.newaxis]
plt.subplot(1,2,2)
plt.imshow(img)
plt.title('(b)目标图像')


plt.colorbar()
plt.xticks([]), plt.yticks([])
plt.show()
```

输出图像如图 29-2 所示，图 29-2(a)为原始图像，图 29-2(b)为图像分割后提取的目标人物，但人物右部分的背景仍然存在。如何移除这些背景呢？这里需要使用自定义的掩码进行提取，读取一张灰色背景轮廓图，从而分离背景与前景，希望读者下来实现该功能。

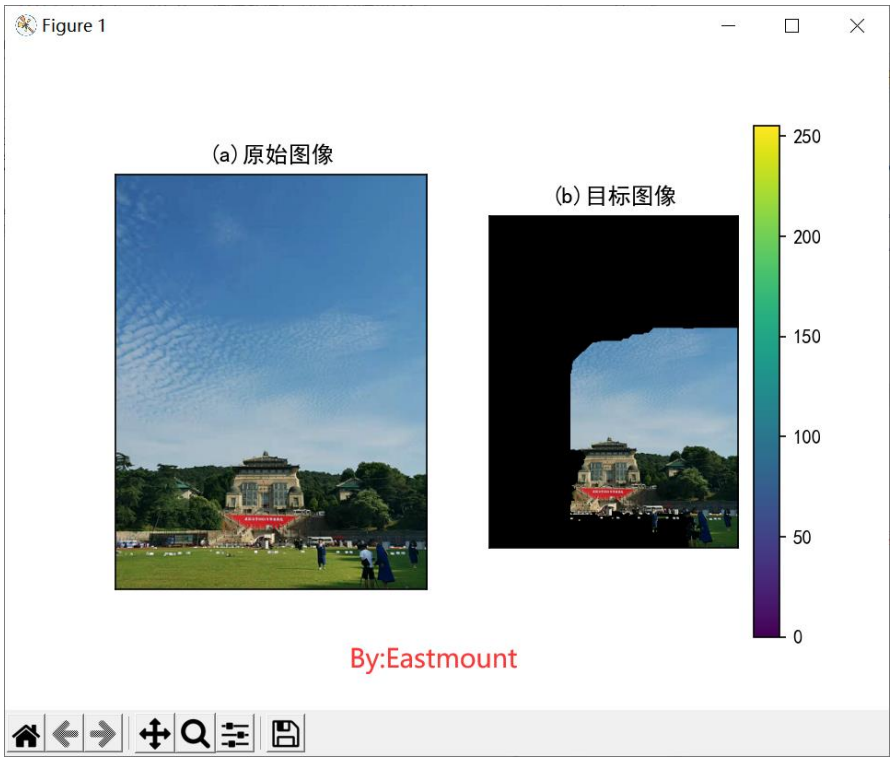


图 29-2 图像分割提取目标人物

2.基于 K-Means 聚类算法的区域分割

K-Means 聚类是最常用的聚类算法，最初起源于信号处理，其目标是将数据点划分为 K 个类簇，找到每个簇的中心并使其度量最小化。该算法的最大优点是简单、便于理解，运算速度较快，缺点是只能应用于连续型数据，并且要在聚类前指定聚集的类簇数[2]。

下面是 K-Means 聚类算法的分析流程，步骤如下：

- ✧ 第一步，确定 K 值，即将数据集聚集成 K 个类簇或小组；
- ✧ 第二步，从数据集中随机选择 K 个数据点作为质心（Centroid）或数据中心；

- ✧ 第三步，分别计算每个点到每个质心之间的距离，并将每个点划分到离最近质心的小组，跟定了那个质心；
- ✧ 第四步，当每个质心都聚集了一些点后，重新定义算法选出新的质心；
- ✧ 第五步，比较新的质心和老的质心，如果新质心和老质心之间的距离小于某一个阈值，则表示重新计算的质心位置变化不大，收敛稳定，则认为聚类已经达到了期望的结果，算法终止；
- ✧ 第六步，如果新的质心和老的质心变化很大，即距离大于阈值，则继续迭代执行第三步到第五步，直到算法终止。

图 29-3 是对身高和体重进行聚类的算法，将数据集的人群聚集成三类。

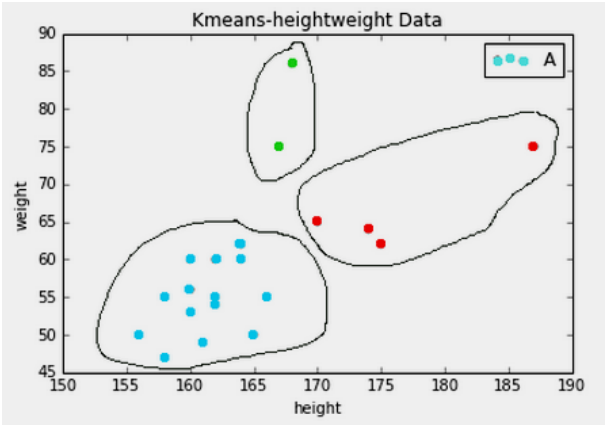


图 29-3 height-weight 聚类

在图像处理中，通过 K-Means 聚类算法可以实现图像分割、图像聚类、图像识别等操作，本小节主要用来进行图像颜色分割。假设存在一张 100×100 像素的灰度图像，它由 10000 个 RGB 灰度级组成，我们通过 K-Means 可以将这些像素点聚类成 K 个簇，然后使用每个簇内的质心点来替换簇内所有的像素点，这样就能实现在不改变分辨率的情况下量化压缩图像颜色，实现图像颜

色层级分割。

在 OpenCV 中，Kmeans()函数原型如下所示：

```
retval, bestLabels, centers = kmeans(data, K, bestLabels,
criteria, attempts, flags[, centers])
```

- data 表示聚类数据，最好是 np.flloat32 类型的 N 维点集
- K 表示聚类类簇数
- bestLabels 表示输出的整数数组，用于存储每个样本的聚类标签索引
- criteria 表示算法终止条件，即最大迭代次数或所需精度。在某些迭代中，一旦每个簇中心的移动小于 criteria.epsilon，算法就会停止
- attempts 表示重复试验 kmeans 算法的次数，算法返回产生最佳紧凑性的标签
- flags 表示初始中心的选择，两种方法是 cv2.KMEANS_PP_CENTERS ; 和 cv2.KMEANS_RANDOM_CENTERS
- centers 表示集群中心的输出矩阵，每个集群中心为一行数据

下面使用该方法对灰度图像颜色进行分割处理，需要注意，在进行 K-Means 聚类操作之前，需要将 RGB 像素点转换为一维的数组，再将各形式的颜色聚集在一起，形成最终的颜色分割。

```
# -*- coding: utf-8 -*-
```

```
# By: Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像灰度颜色
img = cv2.imread('scenery.png', 0)

#获取图像高度、宽度
rows, cols = img.shape[:]

#图像二维像素转换为一维
data = img.reshape((rows * cols, 1))
data = np.float32(data)

#定义中心（type,max_iter,epsilon)
criteria = (cv2.TERM_CRITERIA_EPS +
            cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

#设置标签
flags = cv2.KMEANS_RANDOM_CENTERS
```

```
#K-Means 聚类  聚集成 4 类
compactness, labels, centers = cv2.kmeans(data, 4, None,
criteria, 10, flags)

#生成最终图像
dst = labels.reshape((img.shape[0], img.shape[1]))

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图像
titles = ['原始图像', '聚类图像']
images = [img, dst]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray'),
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 29-4 所示，左边为灰度图像，右边为 K-Means 聚类后的图像，它将灰度级聚集成四个层级，相似的颜色或区域聚集在一起。

图 29-4 灰度图像 K-Means 聚类处理

下面代码是对彩色图像进行颜色分割处理，它将彩色图像聚集成 2 类、4 类和 64 类。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread('scenery.png')


#图像二维像素转换为一维
```

```
data = img.reshape((-1,3))
data = np.float32(data)

#定义中心 (type,max_iter,epsilon)
criteria = (cv2.TERM_CRITERIA_EPS +
            cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

#设置标签
flags = cv2.KMEANS_RANDOM_CENTERS

#K-Means 聚类 聚集成 2 类
compactness, labels2, centers2 = cv2.kmeans(data, 2,
None, criteria, 10, flags)

#K-Means 聚类 聚集成 4 类
compactness, labels4, centers4 = cv2.kmeans(data, 4,
None, criteria, 10, flags)

#K-Means 聚类 聚集成 8 类
compactness, labels8, centers8 = cv2.kmeans(data, 8,
None, criteria, 10, flags)
```

```
#K-Means 聚类 聚集成 16 类
compactness, labels16, centers16 = cv2.kmeans(data, 16,
None, criteria, 10, flags)

#K-Means 聚类 聚集成 64 类
compactness, labels64, centers64 = cv2.kmeans(data, 64,
None, criteria, 10, flags)

#图像转换回 uint8 二维类型
centers2 = np.uint8(centers2)
res = centers2[labels2.flatten()]
dst2 = res.reshape((img.shape))

centers4 = np.uint8(centers4)
res = centers4[labels4.flatten()]
dst4 = res.reshape((img.shape))

centers8 = np.uint8(centers8)
res = centers8[labels8.flatten()]
dst8 = res.reshape((img.shape))
```

```
centers16 = np.uint8(centers16)
res = centers16[labels16.flatten()]
dst16 = res.reshape((img.shape))

centers64 = np.uint8(centers64)
res = centers64[labels64.flatten()]
dst64 = res.reshape((img.shape))

#图像转换为 RGB 显示
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
dst2 = cv2.cvtColor(dst2, cv2.COLOR_BGR2RGB)
dst4 = cv2.cvtColor(dst4, cv2.COLOR_BGR2RGB)
dst8 = cv2.cvtColor(dst8, cv2.COLOR_BGR2RGB)
dst16 = cv2.cvtColor(dst16, cv2.COLOR_BGR2RGB)
dst64 = cv2.cvtColor(dst64, cv2.COLOR_BGR2RGB)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图像
```

```
titles = ['原始图像', '聚类图像  K=2', '聚类图像  K=4',
          '聚类图像  K=8', '聚类图像  K=16', '聚类图像  K=64']
images = [img, dst2, dst4, dst8, dst16, dst64]
for i in range(6):
    plt.subplot(2,3,i+1), plt.imshow(images[i], 'gray'),
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 29-5 所示，它对比了原始图像和各 K-Means 聚类处理后的图像。当 K=2 时，聚集成 2 种颜色；当 K=4 时，聚集成 4 种颜色；当 K=8 时，聚集成 8 种颜色；当 K=16 时，聚集成 16 种颜色；当 K=64 时，聚集成 64 种颜色。

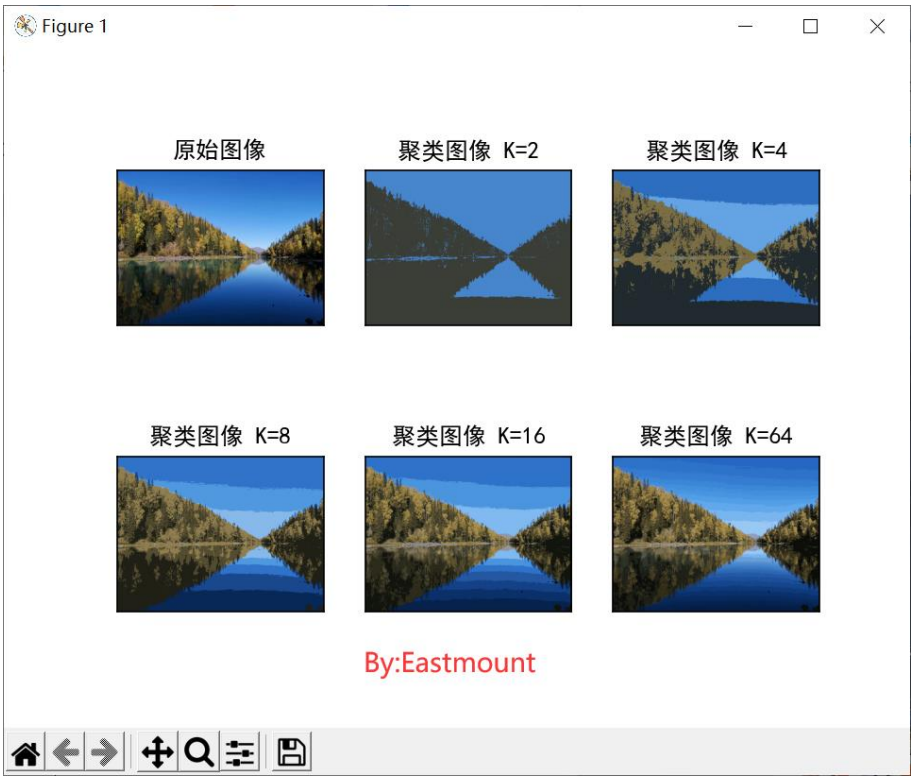


图 29-5 彩色图像 K-Means 聚类对比处理

同样，如果是人物图像显示如图 29-6 所示，比如小珞珞。

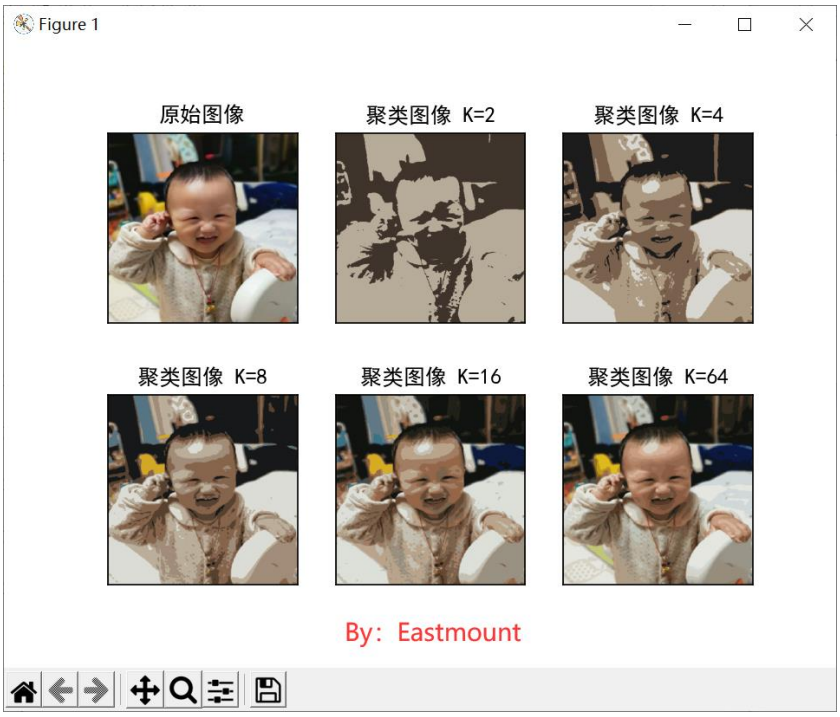


图 29-6 彩色图像 K-Means 聚类对比处理

第 30 篇 基于均值漂移算法和分水岭算法的图像分割

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

图像分割是将图像分成若干具有独特性质的区域并提取感兴趣目标的技术和过程，它是图像处理和图像分析的关键步骤。主要分为基于阈值的分割方法、基于区域的分割方法、基于边缘的分割方法和基于特定理论的分割方法。这篇文章将详细讲解基于均值漂移算法和分水岭算法的图像分割。

1.基于均值漂移算法的图像分割

均值漂移（Mean Shfit）算法是一种通用的聚类算法，最早是 1975 年 Fukunaga 等人在一篇关于概率密度梯度函数的估计论文中提出[1]。它是一种无参估计算法，沿着概率梯度的上升方向寻找分布的峰值。Mean Shift 算法先算出当前点的偏移均值，移动该点到其偏移均值，然后以此为新的起始点，继续移动，直到满足一定的条件结束。

图像分割中可以利用均值漂移算法的特性，实现彩色图像分割。在

OpenCV 中提供的函数为 pyrMeanShiftFiltering()，该函数严格来说并不是图像分割，而是图像在色彩层面的平滑滤波，它可以中和色彩分布相近的颜色，平滑色彩细节，侵蚀掉面积较小的颜色区域，所以在 OpenCV 中它的后缀是滤波“Filter”，而不是分割“segment”。该函数原型如下所示：

```
dst = pyrMeanShiftFiltering(src, sp, sr[, dst[, maxLevel[, termcrit]]])
```

- src 表示输入图像，8 位三通道的彩色图像
- dst 表示输出图像，需同输入图像具有相同的大小和类型
- sp 表示定义漂移物理空间半径的大小
- sr 表示定义漂移色彩空间半径的大小
- maxLevel 表示定义金字塔的最大层数
- termcrit 表示定义的漂移迭代终止条件，可以设置为迭代次数满足终止，迭代目标与中心点偏差满足终止，或者两者的结合

均值漂移 pyrMeanShiftFiltering()函数的执行过程是如下：

- 构建迭代空间。以输入图像上任一点 P_0 为圆心，建立以 sp 为物理空间半径，sr 为色彩空间半径的球形空间，物理空间上坐标为 x 和 y，色彩空间上坐标为 RGB 或 HSV，构成一个空间球体。其中 x 和 y 表示图像的长和宽，色彩空间 R、G、B 在 0 至 255 之间。
- 求迭代空间的向量并移动迭代空间球体重新计算向量，直至收敛。在上一步构建的球形空间中，求出所有点相对于中心点的色彩向量之和，移动迭代空间的中心点到该向量的终点，并再次计算该球形空间中所有点

的向量之和，如此迭代，直到在最后一个空间球体中所求得向量和的终点就是该空间球体的中心点 P_n，迭代结束。

- 更新输出图像 dst 上对应的初始原点 P_0 的色彩值为本轮迭代的终点 P_n 的色彩值，完成一个点的色彩均值漂移。
- 对输入图像 src 上其他点，依次执行上述三个步骤，直至遍历完所有点后，整个均值偏移色彩滤波完成。

下面的代码是图像均值漂移的实现过程：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像灰度颜色
img = cv2.imread('scenery.png')

spatialRad = 50 #空间窗口大小
colorRad = 50 #色彩窗口大小
maxPyrLevel = 2 #金字塔层数

#图像均值漂移分割
```

```
dst = cv2.pyrMeanShiftFiltering( img, spatialRad, colorRad,
maxPyrLevel)


#显示图像
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

当漂移物理空间半径设置为 50，漂移色彩空间半径设置为 50，金字塔层数为 2，输出的效果图如图 30-1 所示。

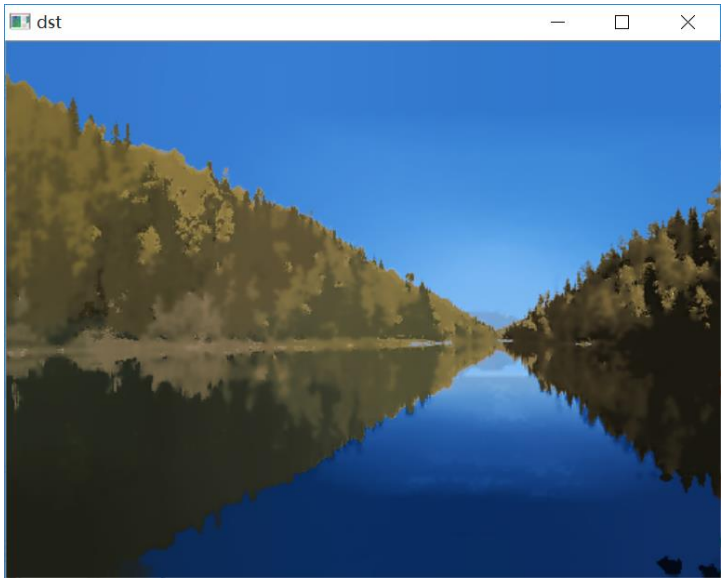


图 30-1 均值漂移分割图像

当漂移物理空间半径设置为 20，漂移色彩空间半径设置为 20，金字塔层数为 2，输出的效果图如图 30-2 所示。对比可以发现，半径为 20 时，图像色彩细节大部分存在，半径为 50 时，森林和水面的色彩细节基本都已经丢失。

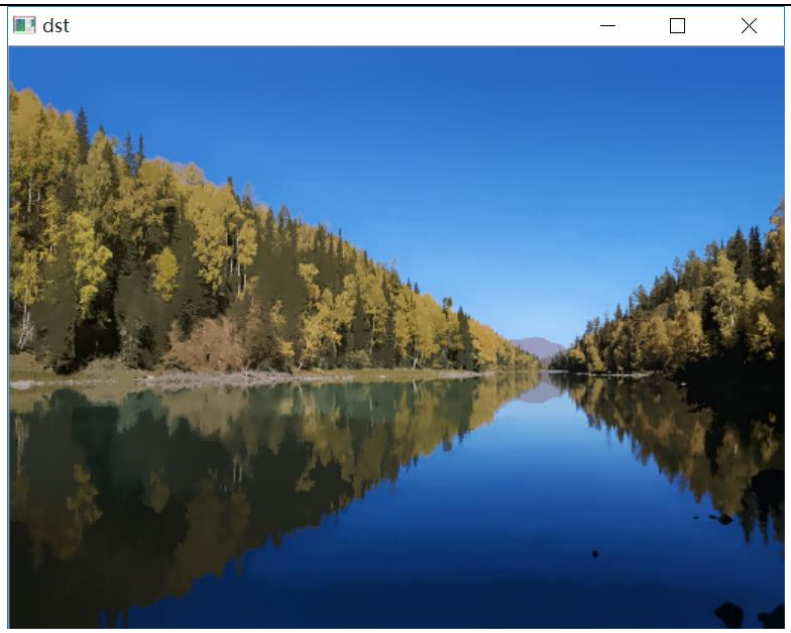


图 30-2 均值漂移分割清晰图像

写到这里，均值偏移算法对彩色图像的分割平滑操作就完成了，为了达到更好地分割目的，借助漫水填充函数进行下一步处理，在下一篇文章将详细介绍，这里只是引入该函数。完整代码如下所示：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像灰度颜色
img = cv2.imread('scenery.png')
```

```
#获取图像行和列
rows, cols = img.shape[:2]

#mask 必须行和列都加 2 且必须为 uint8 单通道阵列
mask = np.zeros([rows+2, cols+2], np.uint8)

spatialRad = 100 #空间窗口大小
colorRad = 100     #色彩窗口大小
maxPyrLevel = 2    #金字塔层数

#图像均值漂移分割
dst = cv2.pyrMeanShiftFiltering( img, spatialRad, colorRad,
maxPyrLevel)

#图像漫水填充处理
cv2.floodFill(dst, mask, (30, 30), (0, 255, 255),
                (100, 100, 100), (50, 50, 50),
                cv2.FLOODFILL_FIXED_RANGE)

#显示图像
cv2.imshow('src', img)
```

```
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

输出的效果图如图 30-3 所示，它将天空染成黄色。

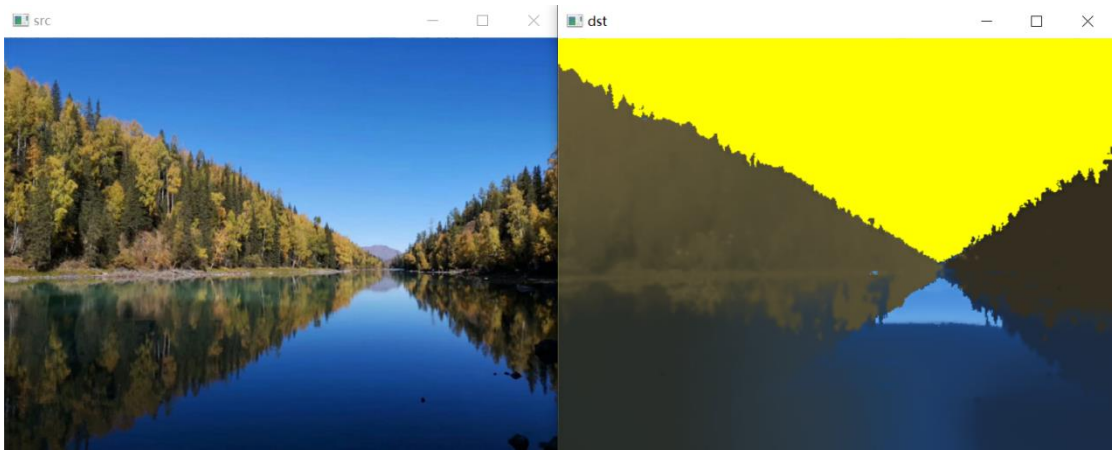


图 30-3 均值漂移结合漫水填充分割图像

2.基于分水岭算法的图像分割

图像分水岭算法（Watershed Algorithm）是将图像的边缘轮廓转换为“山脉”，将均匀区域转换为“山谷”，从而提升分割效果的算法[3]。分水岭算法是基于拓扑理论的数学形态学的分割方法，灰度图像根据灰度值把像素之间的关系看成山峰和山谷的关系，高亮度（灰度值高）的地方是山峰，低亮度（灰度值低）的地方是山谷。接着给每个孤立的山谷（局部最小值）不同颜色的水（Label），当水涨起来，根据周围的山峰（梯度），不同的山谷也就是不同颜色的像素点开始合并，为了避免这个现象，可以在水要合并的地方建立障碍，直到所有山峰都被淹没。所创建的障碍就是分割结果，这个就是分水岭的原理[3]。

分水岭算法的计算过程是一个迭代标注过程，主要包括排序和淹没两个步骤。由于图像会存在噪声或缺失等问题，该方法会造成分割过度。OpenCV 提供了 watershed()函数实现图像分水岭算法，并且能够指定需要合并的点，其函数原型如下所示：

```
markers = watershed(image, markers)
```

- image 表示输入图像，需为 8 位三通道的彩色图像
- markers 表示用于存储函数调用之后的运算结果，输入/输出 32 位单通道图像的标记结构，输出结果需和输入图像的尺寸和类型一致。

下面是分水岭算法实现图像分割的过程。假设存在一幅彩色硬币图像，如图 30-4 所示，硬币相互之间挨着。

图 30-4 原始硬币图像

第一步，通过图像灰度化和阈值化处理提取图像灰度轮廓，采用 OTSU 二值化处理获取硬币的轮廓。

```
# -*- coding: utf-8 -*-
```

```
# By: Eastmount
import numpy as np
import cv2
from matplotlib import pyplot as plt

#读取原始图像
img = cv2.imread('coin.jpg')

#图像灰度化处理
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#图像阈值化处理
ret, thresh = cv2.threshold(gray, 0, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

#显示图像
cv2.imshow('src', img)
cv2.imshow('res', thresh)
cv2.waitKey()
cv2.destroyAllWindows()
```

输出结果如图 30-5 所示。

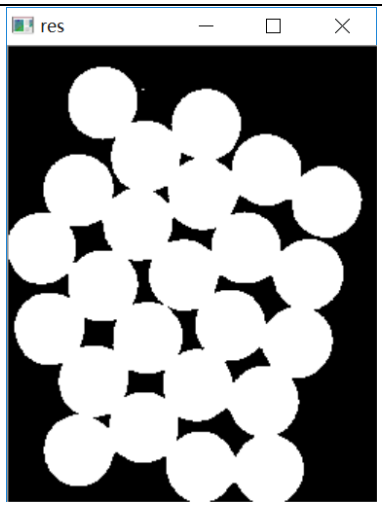


图 30-5 灰度化与阈值化处理后的图像

第二步，通过形态学开运算过滤掉小的白色噪声。同时，由于图像中的硬币是紧挨着的，所以不能采用图像腐蚀去掉边缘的像素，而是选择距离转换，配合一个适当的阈值进行物体提取。这里引入一个图像膨胀操作，将目标边缘扩展到背景，以确定结果的背景区域。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import numpy as np
import cv2
from matplotlib import pyplot as plt

#读取原始图像
img = cv2.imread('coin.jpg')

#图像灰度化处理
```

```
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#图像阈值化处理
ret, thresh = cv2.threshold(gray, 0, 255,

cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

#图像开运算消除噪声
kernel = np.ones((3,3),np.uint8)
opening =
cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel,
iterations = 2)

#图像膨胀操作确定背景区域
sure_bg = cv2.dilate(opening,kernel,iterations=3)

#距离运算确定前景区域
dist_transform =
cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,
0.7*dist_transform.max(), 255, 0)
```

```
#寻找未知区域
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图像
titles = ['原始图像', '阈值化', '开运算',
          '背景区域', '前景区域', '未知区域']
images = [img, thresh, opening, sure_bg, sure_fg,
unknown]
for i in range(6):
    plt.subplot(2,3,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图 30-6 所示，包括原始图像、阈值化处理、开运算、背景区域、前景区域、未知区域等。由图可知，在使用阈值过滤的图像里，确认了图像的硬币区域，而在有些情况，可能对前景分割更感兴趣，而不关心目标是否需要

分开或挨着，那时可以采用腐蚀操作来求解前景区域。

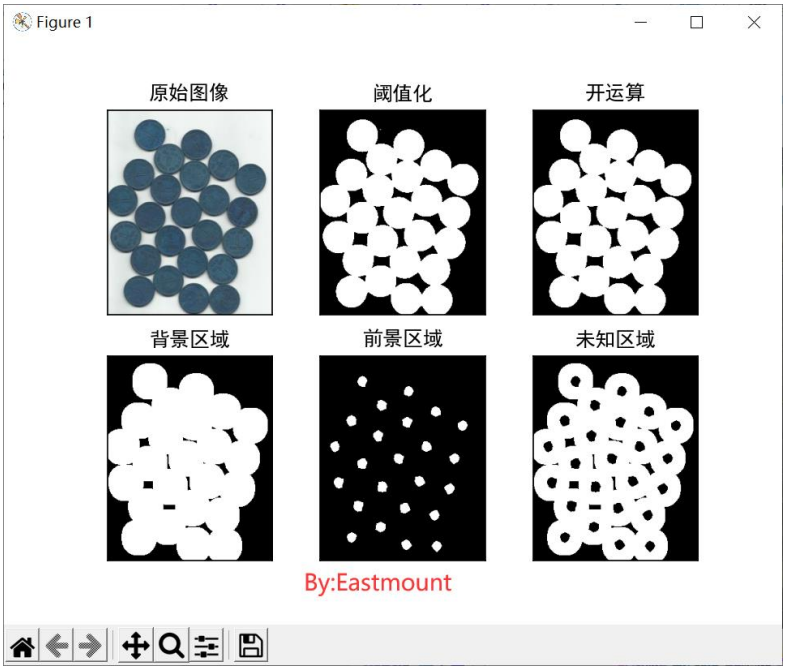


图 30-6 背景区域和前景区域确定

第三步，当前处理结果中，已经能够区分出前景硬币区域和背景区域。接着我们创建标记变量，在该变量中标记区域，已确认的区域（前景或背景）用不同的正整数标记出来，不确认的区域保持 0，使用 cv2.connectedComponents()函数来将图像背景标记成 0，其他目标用从 1 开始的整数标记。注意，如果背景被标记成 0，分水岭算法会认为它是未知区域，所以要用不同的整数来标记。

最后，调用 watershed()函数实现分水岭图像分割，标记图像会被修改，边界区域会被标记成 0，完整代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import numpy as np
```

```
import cv2
from matplotlib import pyplot as plt

#读取原始图像
img = cv2.imread('coin.jpg')

#图像灰度化处理
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#图像阈值化处理
ret, thresh = cv2.threshold(gray, 0, 255,

cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

#图像开运算消除噪声
kernel = np.ones((3,3),np.uint8)
opening =
cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel,
iterations = 2)

#图像膨胀操作确定背景区域
```

```
sure_bg = cv2.dilate(opening,kernel,iterations=3)


#距离运算确定前景区域
dist_transform =
cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,
0.7*dist_transform.max(), 255, 0)


#寻找未知区域
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)


#标记变量
ret, markers = cv2.connectedComponents(sure_fg)


#所有标签加一，以确保背景不是 0 而是 1
markers = markers+1


#用 0 标记未知区域
markers[unknown==255]=0
```

```
#分水岭算法实现图像分割
markers = cv2.watershed(img, markers)
img[markers == -1] = [255,0,0]


#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']


#显示图像
titles = ['标记区域', '图像分割']
images = [markers, img]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

最终分水岭算法的图像分割如图 30-7 所示，它将硬币的轮廓成功提取。

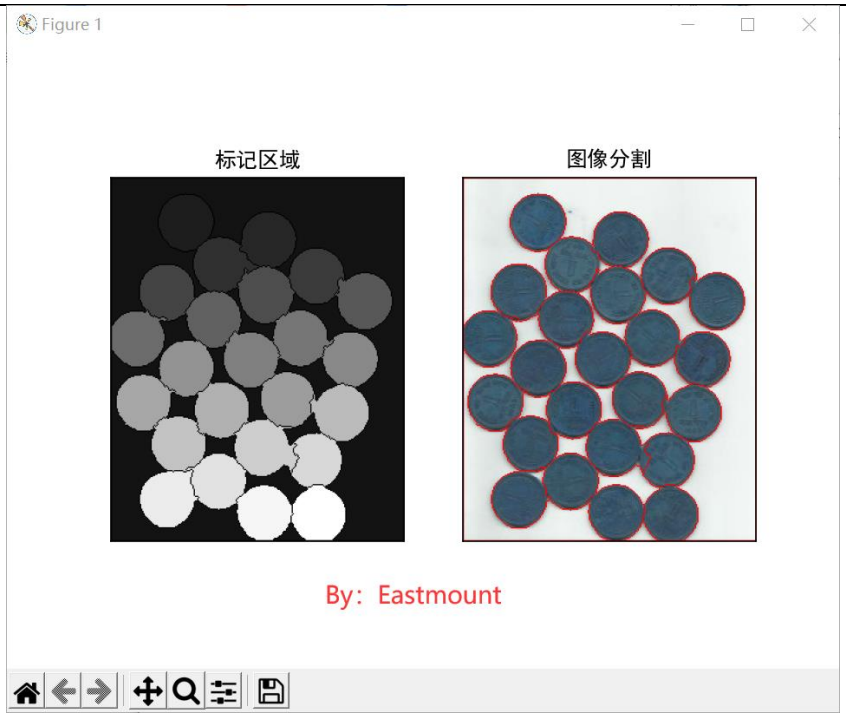


图 30-7 分水岭算法提取图像硬币轮廓

图 30-8 是采用分水岭算法提取图像 Windows 中心轮廓的效果图。

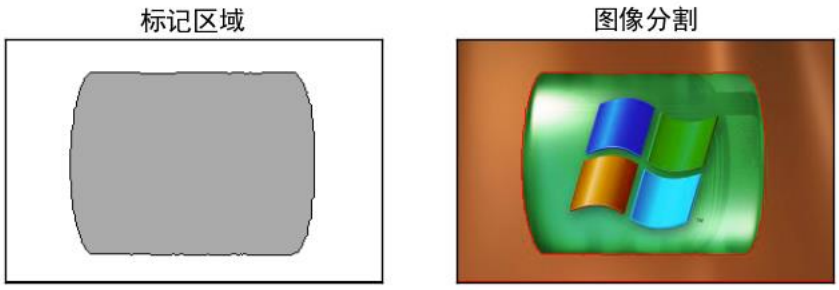


图 30-8 分水岭算法提取图像 Windows 轮廓

分水岭算法对微弱边缘具有良好的响应，图像中的噪声、物体表面细微的灰度变化，都会产生过度分割的现象。但同时应当看出，分水岭算法对微弱边缘具有良好的响应，是得到封闭连续边缘的保证。另外，分水岭算法所得到的封闭的集水盆，为分析图像的区域特征提供了可能。

3.总结

本文主要讲解了图像分割方法，包括基于均值漂移算法的图像分割方法、基于分水岭算法的图像分割方法，通过这些处理能有效分割图像的背景和前景，识别某些图像的区域。

参考文献：

[1] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[2] Eastmount. [Python 图像处理] 四十.全网首发 Python 图像分割万字详解（阈值分割、边缘分割、纹理分割、分水岭算法、K-Means 分割、漫水填充分割、区域定位）[EB/OL]. (2021-05-18). https://blog.csdn.net/Eastmount/article/details/116952580.

第 31 篇 图像漫水填充分割应用

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

图像分割是将图像分成若干具有独特性质的区域并提取感兴趣目标的技术和过程，它是图像处理和图像分析的关键步骤。主要分为基于阈值的分割方法、基于区域的分割方法、基于边缘的分割方法和基于特定理论的分割方法。这篇文章将详细讲解漫水填充分割应用，至此，几种图像分割方法介绍完毕。

1.图像漫水填充

图像漫水填充（FloodFill）是指用一种特定的颜色填充联通区域，通过设置可连通像素的上下限以及连通方式来达到不同的填充效果。漫水填充通常被用来标记或分离图像的一部分以便对其进行深入的处理或分析。

图像漫水填充主要是遴选出与种子点联通且颜色相近的像素点，接着对像素点的值进行处理。如果遇到掩码，则根据掩码进行处理。其原理类似 Photoshop 的魔术棒选择工具，漫水填充将查找和种子点联通的颜色相同的点，而魔术棒选择工具是查找和种子点联通的颜色相近的点，将和初始种子像素颜色相近的点压进栈作为新种子。基本工作步骤如下：

- 选定种子点（x，y）；
- 检查种子点的颜色，如果该点颜色与周围连接点的颜色不相同，则将周围点颜色设置为该点颜色；如果相同则不做处理。但是周围点不一定都会变成和种子点的颜色相同，如果周围连接点在给定的范围（从 loDiff 到 upDiff）内或在种子点的像素范围内才会改变颜色；
- 检测其他连接点，进行第 2 个步骤的处理，直到没有连接点，即到达检测区域边界停止。

2.图像漫水填充分割实现

在 OpenCV 中，主要通过 floodFill()函数实现漫水填充分割，它将用指定的颜色从种子点开始填充一个连接域。其函数原型如下所示：

floodFill(image, mask, seedPoint, newVal[, loDiff[, upDiff[, flags]]])

- image 表示输入/输出 1 通道或 3 通道，6 位或浮点图像
- mask 表示操作掩码，必须为 8 位单通道图像，其长宽都比输入图像大两个像素点。注意，漫水填充不会填充掩膜 mask 的非零像素区域，mask 中与输入图像(x,y)像素点相对应的点的坐标为(x+1,y+1)。
- seedPoint 为 Point 类型，表示漫水填充算法的起始点
- newVal 表示像素点被染色的值，即在重绘区域像素的新值

- loDiff 表示当前观察像素值与其部件邻域像素值或待加入该部件的种子像素之间的亮度或颜色之负差的最大值，默认值为 Scalar( )
- upDiff 表示当前观察像素值与其部件邻域像素值或待加入该部件的种子像素之间的亮度或颜色之正差的最大值，默认值为 Scalar( )
- flags 表示操作标识符，此参数包括三个部分：低八位 0-7bit 表示邻接性（4 邻接或 8 邻接）；中间八位 8-15bit 表示掩码的填充颜色，如果中间八位为 0 则掩码用 1 来填充；高八位 16-31bit 表示填充模式，可以为 0 或者以下两种标志符的组合，FLOODFILL_FIXED_RANGE 表示此标志会考虑当前像素与种子像素之间的差，否则就考虑当前像素与相邻像素的差。FLOODFILL_MASK_ONLY 表示函数不会去填充改变原始图像,而是去填充掩码图像 mask，mask 指定的位置为零时才填充，不为零不填充。

在 Python 和 OpenCV 实现代码中，它设置种子点位置为(10,200)；设置颜色为黄色(0,255,255)；连通区范围设定为 loDiff 和 upDiff；标记参数设置为 CV_FLOODFILL_FIXED_RANGE ，它表示待处理的像素点与种子点作比较，在范围之内，则填充此像素，即种子漫水填充满足：

- src(seed.x, seed.y) - loDiff <= src(x, y) <= src(seed.x, seed.y) +upDiff

完整代码如下：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np

#读取原始图像
img = cv2.imread('windows.png')

#获取图像行和列
rows, cols = img.shape[:2]

#目标图像
dst = img.copy()

#mask 必须行和列都加 2 且必须为 uint8 单通道阵列
#mask 多出来的 2 可以保证扫描的边界上的像素都会被处理
mask = np.zeros([rows+2, cols+2], np.uint8)

#图像漫水填充处理
#种子点位置(30,30) 设置颜色(0,255,255) 连通区范围设定 loDiff
upDiff
```

```
#src(seed.x, seed.y) - loDiff <= src(x, y) <= src(seed.x,
seed.y) +upDiff
cv2.floodFill(dst, mask, (30, 30), (0, 255, 255),
                (100, 100, 100), (50, 50, 50),
                cv2.FLOODFILL_FIXED_RANGE)


#显示图像
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

输出结果如图 31-1 所示，左边为原始图像，右边为将 Windows 图标周围填充为黄色的图像。

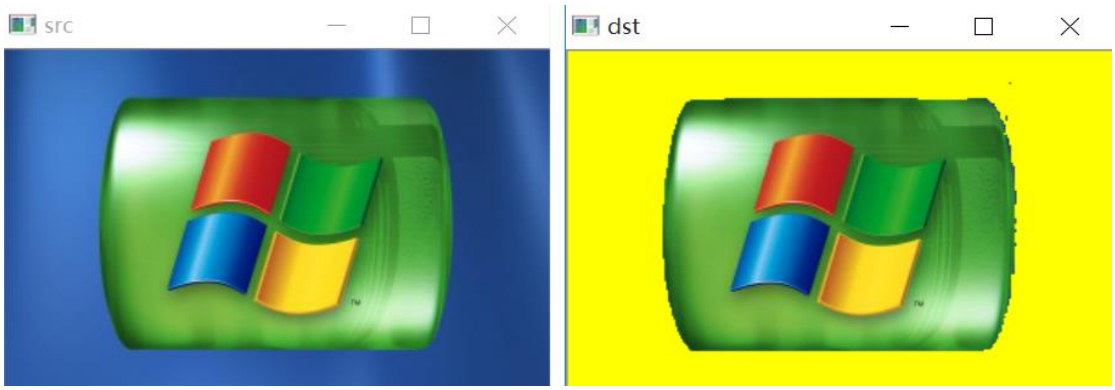


图 31-1 图像漫水填充分割

3.图像漫水填充分割自动软件

下面补充另一段代码，它将打开一幅图像，点击鼠标选择种子节点，移动滚动条设定连通区范围的 loDiff 和 upDiff 值，并产生动态的漫水填充分割。

注意，该部分代码中涉及鼠标、键盘、滚动条等操作，希望读者下来学习相关知识，该系列文章更多是讲解 Python 图像处理的算法原理及代码实现。

```
# coding:utf-8
import cv2
import random
import sys
import numpy as np


#使用说明  点击鼠标选择种子点
help_message = '''USAGE: floodfill.py [<image>]
Click on the image to set seed point
Keys:
  f - toggle floating range
  c - toggle 4/8 connectivity
  ESC - exit
'''
```

```
if __name__ == '__main__':

    #输出提示文本
    print(help_message)

    #读取原始图像
    img = cv2.imread('scenery.png')

    #获取图像高和宽
    h, w = img.shape[:2]

    #设置掩码 长和宽都比输入图像多两个像素点
    mask = np.zeros((h+2, w+2), np.uint8)

    #设置种子节点和 4 邻接
    seed_pt = None
    fixed_range = True
    connectivity = 4

    #图像漫水填充分割更新函数
    def update(dummy=None):
```

```
        if seed_pt is None:
            cv2.imshow('floodfill', img)
            return

        #建立图像副本并漫水填充
        flooded = img.copy()
        mask[:] = 0 #掩码初始为全 0
        lo = cv2.getTrackbarPos('lo', 'floodfill') #像素邻域负差
最大值
        hi = cv2.getTrackbarPos('hi', 'floodfill') #像素邻域正差
最大值
        print('lo=', lo, 'hi=', hi)

        #低位比特包含连通值 4 (缺省) 或 8
        flags = connectivity

        #考虑当前像素与种子像素之间的差（高比特也可以为 0）
        if fixed_range:
            flags |= cv2.FLOODFILL_FIXED_RANGE

        #以白色进行漫水填充
```

```
        cv2.floodFill(flooded, mask, seed_pt,
                      (random.randint(0,255),
random.randint(0,255),
                       random.randint(0,255)),        (lo,)*3,
(hi,)*3, flags)

        #选定基准点用红色圆点标出
        cv2.circle(flooded, seed_pt, 2, (0, 0, 255), -1)
        print("send_pt=", seed_pt)

        #显示图像
        cv2.imshow('floodfill', flooded)

    #鼠标响应函数
    def onmouse(event, x, y, flags, param):
        global seed_pt #基准点

        #鼠标左键响应选择漫水填充基准点
        if flags & cv2.EVENT_FLAG_LBUTTON:
            seed_pt = x, y
            update()
```

```
#执行图像漫水填充分割更新操作
update()

#鼠标更新操作
cv2.setMouseCallback('floodfill', onmouse)

#设置进度条
cv2.createTrackbar('lo', 'floodfill', 20, 255, update)
cv2.createTrackbar('hi', 'floodfill', 20, 255, update)

#按键响应操作
while True:
    ch = 0xFF & cv2.waitKey()
    #退出
    if ch == 27:
        break
    #选定时 flags 的高位比特位 0
    #邻域的选定为当前像素与相邻像素的差，联通区域会很大
    if ch == ord('f'):
        fixed_range = not fixed_range
```

```
            print('using %s range' % ('floating', 'fixed')[fixed_range])
            update()
        #选择 4 方向或则 8 方向种子扩散
        if ch == ord('c'):
            connectivity = 12-connectivity
            print('connectivity =', connectivity)
            update()
    cv2.destroyAllWindows()
```

当鼠标选定的种子点为（242,96），观察点像素邻域负差最大值“lo”为138，观察点像素邻域正差最大值“hi”为 147 时，图像漫水填充效果如图 31-2 所示，它将天空和中心水面填充成黄色。

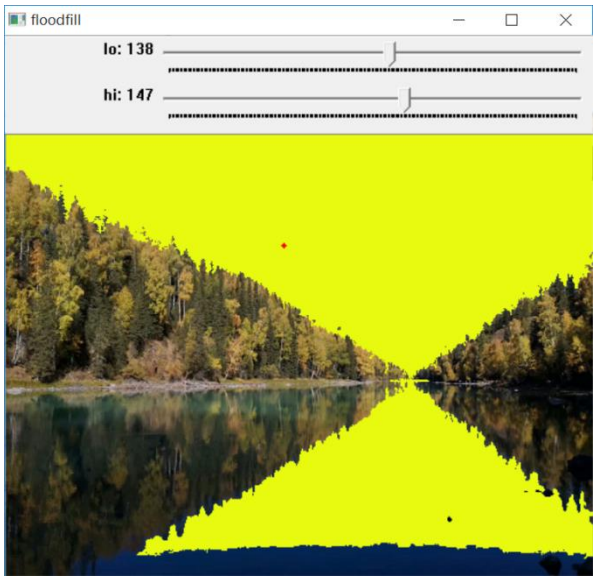


图 31-2 图像天空和中心水面的漫水填充分割效果

当鼠标选定的种子点为（328, 202），观察点像素邻域负差最大值“lo”

为 142，观察点像素邻域正差最大值“hi”为 45 时，图像漫水填充效果如图 31-3 所示，它将图像两旁的森林和水面填充成蓝紫色。

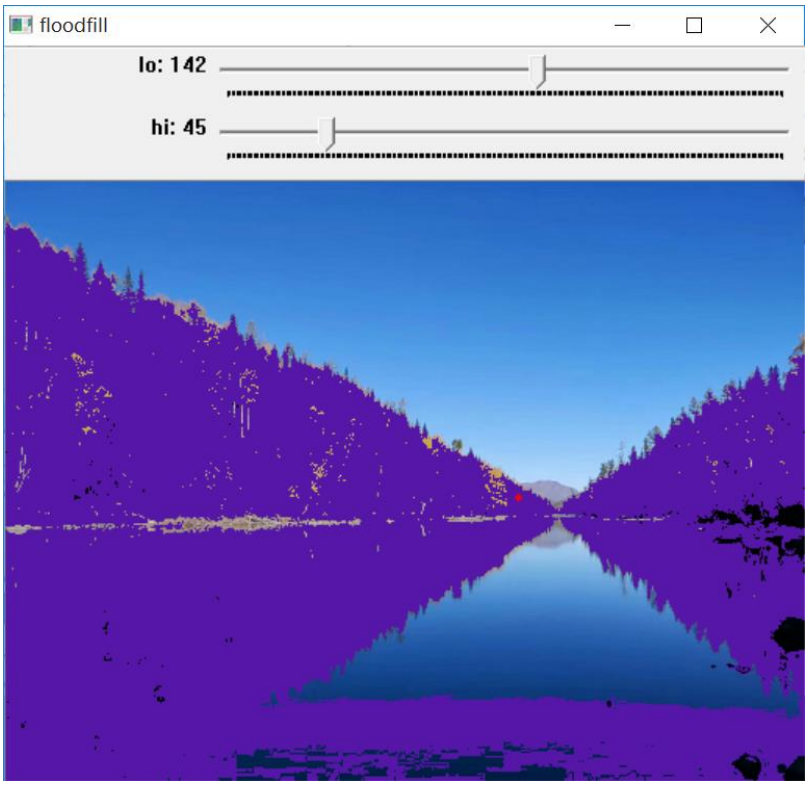


图 31-3 图像两旁森林和水面的漫水填充分割效果

4.总结

写到这里，图像分割知识点就介绍完毕，包括基于阈值的图像分割方法、基于边缘检测的图像分割方法、基于纹理背景的图像分割方法和基于特定理论的图像分割方法。其中，基于特定理论的分割方法又分别讲解了基于 K-Means 聚类、均值漂移、分水岭算法的图像分割方法。最后通过漫水填充分割案例加深了读者的印象。希望读者能结合本章知识点，围绕自己的研究领域或工程项目进行深入的学习，实现所需的图像处理。

参考文献：

[1] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[2] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[3] Robert Laganiere. OpenCV2 计算机视觉编程手册[M]. 北京：科学出版社，2013.

[4] xxxss. OpenCV-Python 教程:31.分水岭算法对图像进行分割[EB/OL]. (2). https://www.jianshu.com/p/de81d6029235.

[5] Python_Zhou. python+opencv 漫水填充(floodFill)实例详解[EB/OL]. (2018-11-13). https://blog.csdn.net/weixin_42508025/article/details/84029054.

[6] 浅墨_毛星云. [OpenCV 入门教程之十五] 水漫金山：OpenCV 漫水填充算法（Floodfill）[EB/OL]. (2014-06-03). https://blog.csdn.net/poem_qianmo/article/details/28261997.

[7] kalp_yp. 漫水填充实例详解[EB/OL]. (2018-06-21). https://blog.csdn.net/u013539952/article/details/80702849.

[8] 太子洗马. OpenCv 漫水填充 floodFill 详解[EB/OL]. (2018-07-09). https://blog.csdn.net/weixin_42296411/article/details/80966724.

[9] 菜鸟知识搬运工. OpenCV3 学习（7.1）——图像分割之一（漫水填充 FloodFill）[EB/Ol]. (2019-02-03). https://blog.csdn.net/qq_30815237/article/details/86759450.

[10] gjy095. Opencv Python 版学习笔记（二）漫水填充[EB/OL]. (2013-06-28).

https://blog.csdn.net/gjy095/article/details/9198845.

[11] it2153534. [Python]OpenCV 学习总结及文字检测[EB/OL]. (2018-01-28).

https://blog.csdn.net/it2153534/article/details/79185397.

第 32 篇 图像傅里叶变换和傅里叶逆变换详解

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

在数字图像处理中，有两个经典的变换被广泛应用——傅里叶变换和霍夫变换。其中，傅里叶变换主要是将时间域上的信号转变为频率域上的信号，用来进行图像除噪、图像增强等处理；霍夫变换主要用来辨别找出物件中的特征，用来进行特征检测、图像分析、数位影像处理等处理。本文主要讲解图像傅里叶变换和傅里叶逆变换。

1.图像傅里叶变换和逆变换

傅里叶变换（Fourier Transform，简称 FT）常用于数字信号处理，它的目的是将时间域上的信号转变为频率域上的信号。随着域的不同，对同一个事物的了解角度也随之改变，因此在时域中某些不好处理的地方，在频域就可以较为简单的处理。同时，可以从频域里发现一些原先不易察觉的特征。傅里叶定理指出“任何连续周期信号都可以表示成（或者无限逼近）一系列正弦信号的叠加。”[1]

傅里叶公式（32-1）如下，其中 w 表示频率，t 表示时间，e^{-iwt} 为复变函

数。它将时间域的函数表示为频率域的函数 f(t)的积分[2]。

$$\mathrm{F}(w)=F[f(t)]=\int_{-\infty}^{\infty}f(t)e^{-iwt}dt \qquad (32-1)$$

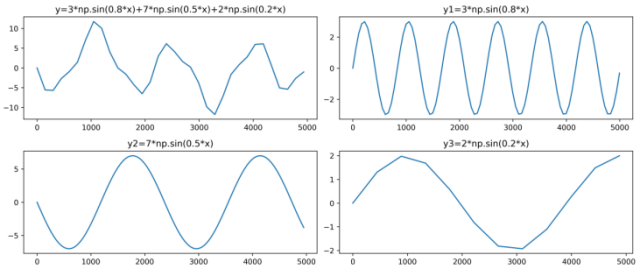


图 32-1 傅里叶变换示例

傅里叶变换认为一个周期函数（信号）包含多个频率分量，任意函数（信号）f(t)可通过多个周期函数（或基函数）相加合成。从物理角度理解，傅里叶变换是以一组特殊的函数（三角函数）为正交基，对原函数进行线性变换，物理意义便是原函数在各组基函数的投影。如 32-1 图所示，它是由三条正弦曲线组合成。其函数为（32-2）所示[3]。

$$y=3\times\sin(0.8\cdot x)+7\times\sin(\frac{1}{3}\cdot x+2)+2\times\sin(0.2\cdot x+3) \qquad (32-2)$$

傅里叶变换可以应用于图像处理中，经过对图像进行变换得到其频谱图。从谱频图里频率高低来表征图像中灰度变化剧烈程度。图像中的边缘信号和噪声信号往往是高频信号，而图像变化频繁的图像轮廓及背景等信号往往是低频信号。这时可以有针对性的对图像进行相关操作，例如图像除噪、图像增强和锐化等。

二维图像的傅里叶变换可以用以下数学公式（32-3）表达，其中 f 是空间域（Spatial Domain））值，F 是频域（Frequency Domain）值。

$$\mathrm{F}(k,l)=\sum_{\mathrm{i}=0}^{N-1}\sum_{\mathrm{j}=0}^{N-1}f(i,j)e^{-i2\pi(\frac{ki}{N}+\frac{li}{N})} \qquad (32-3)$$
$$e^{ix}=\cos x+i\sin x$$

对上面的傅里叶变换有了大致的了解之后，下面通过 Numpy 和 OpenCV 分别讲解图像傅里叶变换的算法及操作代码。

2.OpenCV 实现图像傅里叶变换和逆变换

（1）OpenCV 实现傅里叶变换

OpenCV 中相应的函数是 cv2.dft()和用 Numpy 输出的结果一样，但是是双通道的。第一个通道是结果的实数部分，第二个通道是结果的虚数部分，并且输入图像要首先转换成 np.float32 格式。其函数原型如下所示：

```
dst = cv2.dft(src, dst=None, flags=None, nonzeroRows=None)
```

- src 表示输入图像，需要通过 np.float32 转换格式
- dst 表示输出图像，包括输出大小和尺寸
- flags 表示转换标记，其中 DFT _INVERSE 执行反向一维或二维转换，而不是默认的正向转换；DFT _SCALE 表示缩放结果，由阵列元素的数量除以它；DFT _ROWS 执行正向或反向变换输入矩阵的每个单独的行，该标志可以同时转换多个矢量，并可用于减少开销以执行 3D 和更高维度的转换等；DFT _COMPLEX_OUTPUT 执行 1D 或 2D 实数组的正向转换，这是最快的选择，默认功能；DFT _REAL_OUTPUT 执行一维或二维复数阵列的逆变换，结果通常是相同大小的复数数组，但如果输入数组具有共轭复数对称性，则输出为真实数组

- nonzeroRows 表示当参数不为零时，函数假定只有 nonzeroRows 输入数组的第一行（未设置）或者只有输出数组的第一个（设置）包含非零，因此函数可以处理其余的行更有效率，并节省一些时间；这种技术对计算阵列互相关或使用 DFT 卷积非常有用

注意，由于输出的频谱结果是一个复数，需要调用 cv2.magnitude()函数将傅里叶变换的双通道结果转换为 0 到 255 的范围。其函数原型如下：

cv2.magnitude(x, y)

- x 表示浮点型 X 坐标值，即实部
- y 表示浮点型 Y 坐标值，即虚部

最终输出结果为幅值，即：$\mathrm{dst}(I) = \sqrt{x(I)^2 + y(I)^2}$

下面的代码是调用 cv2.dft()进行傅里叶变换的一个简单示例。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import numpy as np
import cv2
from matplotlib import pyplot as plt
import matplotlib

#读取图像
```

```
img = cv2.imread('lena-hd.png', 0)

#傅里叶变换
dft = cv2.dft(np.float32(img), flags = cv2.DFT_COMPLEX_OUTPUT)

#将频谱低频从左上角移动至中心位置
dft_shift = np.fft.fftshift(dft)

#频谱图像双通道复数转换为 0-255 区间
result = 20*np.log(cv2.magnitude(dft_shift[:,:,0], dft_shift[:,:,1]))

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#显示图像
plt.subplot(121), plt.imshow(img, cmap = 'gray')
plt.title(u'(a)原始图像'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(result, cmap = 'gray')
plt.title(u'(b)傅里叶变换处理'), plt.xticks([]), plt.yticks([])
```

```
plt.show()
```

输出结果如图 32-2 所示，图(a)为原始“Lena”图，图(b)为转换后的频谱图像，并且保证低频位于中心位置。

图 32-2 OpenCV 傅里叶变换对比图

（2）OpenCV 实现傅里叶逆变换

在 OpenCV 中，通过函数 cv2.idft()实现傅里叶逆变换，其返回结果取决于原始图像的类型和大小，原始图像可以为实数或复数。其函数原型如下所示：

```
dst = cv2.idft(src[, dst[, flags[, nonzeroRows]]])
```

- src 表示输入图像，包括实数或复数
- dst 表示输出图像
- flags 表示转换标记
- nonzeroRows 表示要处理的 dst 行数，其余行的内容未定义（请

参阅 dft 描述中的卷积示例）

注意，由于输出的频谱结果是一个复数，需要调用 cv2.magnitude()函数将傅里叶变换的双通道结果转换为 0 到 255 的范围。其函数原型如下：

cv2.magnitude(x, y)

- x 表示浮点型 X 坐标值，即实部
- y 表示浮点型 Y 坐标值，即虚部

最终输出结果为幅值，即：$\mathrm{dst}(I) = \sqrt{x(I)^2 + y(I)^2}$

下面的代码是调用 cv2.idft()进行傅里叶逆变换的一个简单示例。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import numpy as np
import cv2
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv2.imread('lena-hd.png', 0)

#傅里叶变换
dft = cv2.dft(np.float32(img), flags =
```

```
cv2.DFT_COMPLEX_OUTPUT)
dftshift = np.fft.fftshift(dft)
res1= 20*np.log(cv2.magnitude(dftshift[:,:,0], dftshift[:,:,1]))

#傅里叶逆变换
ishift = np.fft.ifftshift(dftshift)
iimg = cv2.idft(ishift)
res2 = cv2.magnitude(iimg[:,:,0], iimg[:,:,1])

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#显示图像
plt.subplot(131), plt.imshow(img, 'gray'), plt.title('(a)原始图像
')
plt.axis('off')
plt.subplot(132), plt.imshow(res1, 'gray'), plt.title('(b)傅里叶
变换处理')
plt.axis('off')
plt.subplot(133), plt.imshow(res2, 'gray'), plt.title('(b)傅里叶
变换逆处理')
```

```
plt.axis('off')
plt.show()
```

输出结果如图 32-3 所示，图(a)为原始“Lena”图，图(b)为傅里叶变换后的频谱图像，图(c)为傅里叶逆变换，频谱图像转换为原始图像的过程。

图 32-3 OpenCV 傅里叶逆变换对比图

3.NumPy 实现图像傅里叶变换和逆变换

（1）Numpy 实现傅里叶变换

Numpy 中的 FFT 包提供了函数 np.fft.fft2()可以对信号进行快速傅里叶变换，其函数原型如下所示，该输出结果是一个复数数组（Complex Ndarry）[2]。

```
fft2(a, s=None, axes=(-2, -1), norm=None)
```

- a 表示输入图像，阵列状的复杂数组
- s 表示整数序列，可以决定输出数组的大小。输出可选形状（每个

转换轴的长度），其中 s[0]表示轴 0，s[1]表示轴 1。对应 fit(x,n)函数中的 n，沿着每个轴，如果给定的形状小于输入形状，则将剪切输入。如果大于则输入将用零填充。如果未给定's'，则使用沿'axles'指定的轴的输入形状

- axes 表示整数序列，用于计算 FFT 的可选轴。如果未给出，则使用最后两个轴。“axes”中的重复索引表示对该轴执行多次转换，一个元素序列意味着执行一维 FFT
- norm 包括 None 和 ortho 两个选项，规范化模式（请参见 numpy.fft）。默认值为无

Numpy 中的 fft 模块有很多函数，相关函数如下：

```
#计算一维傅里叶变换
numpy.fft.fft(a, n=None, axis=-1, norm=None)
#计算二维的傅里叶变换
numpy.fft.fft2(a, n=None, axis=-1, norm=None)
#计算 n 维的傅里叶变换
numpy.fft.fftn()
#计算 n 维实数的傅里叶变换
numpy.fft.rfftn()
#返回傅里叶变换的采样频率
numpy.fft.fftfreq()
#将 FFT 输出中的直流分量移动到频谱中央
```

numpy.fft.shift()

下面的代码是通过 Numpy 库实现傅里叶变换，调用 np.fft.fft2()快速傅里叶变换得到频率分布，接着调用 np.fft.fftshift()函数将中心位置转移至中间，最终通过 Matplotlib 显示效果图。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv.imread('lena-hd.png', 0)

#快速傅里叶变换算法得到频率分布
f = np.fft.fft2(img)

#默认结果中心点位置是在左上角,
#调用 fftshift()函数转移到中间位置
fshift = np.fft.fftshift(f)
```

```
#fft 结果是复数，其绝对值结果是振幅
fimg = np.log(np.abs(fshift))

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#展示结果
plt.subplot(121), plt.imshow(img, 'gray'), plt.title('(a)原始图像
')
plt.axis('off')
plt.subplot(122), plt.imshow(fimg, 'gray'), plt.title('(b)傅里叶
变换处理')
plt.axis('off')
plt.show()
```

输出结果如图 32-4 所示，图(a)为原始图像，图(b)为频率分布图谱，其中越靠近中心位置频率越低，越亮（灰度值越高）的位置代表该频率的信号振幅越大。

图 32-4 傅里叶变换效果图

需要注意，傅里叶变换得到低频、高频信息，针对低频和高频处理能够实现不同的目的。同时，傅里叶过程是可逆的，图像经过傅里叶变换、逆傅里叶变换能够恢复原始图像。

下面代码呈现了原始图像在变化方面的一种表示：图像最明亮的像素放到中央，然后逐渐变暗，在边缘上的像素最暗。这样可以发现图像中亮、暗像素的百分比，即为频域中的振幅 AA 的强度。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import matplotlib
```

```
#读取图像
img = cv.imread('luo.png', 0)

#傅里叶变换
f = np.fft.fft2(img)

#转移像素做幅度普
fshift = np.fft.fftshift(f)

#取绝对值：将复数变化成实数取对数的目的为了将数据变化到 0-255
res = np.log(np.abs(fshift))

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#展示结果
plt.subplot(121), plt.imshow(img, 'gray'), plt.title('(a)原始图像
'), plt.axis('off')
plt.subplot(122), plt.imshow(res, 'gray'), plt.title('(b)傅里叶变
换处理'), plt.axis('off')
plt.show()
```

输出结果如图 32-5 所示，图(a)为原始图像，图(b)为频率分布图谱。

图 32-5 傅里叶变换频谱图

（2）Numpy 实现傅里叶逆变换

下面介绍 Numpy 实现傅里叶逆变换，它是傅里叶变换的逆操作，将频谱图像转换为原始图像的过程。通过傅里叶变换将转换为频谱图，并对高频（边界）和低频（细节）部分进行处理，接着需要通过傅里叶逆变换恢复为原始效果图。频域上对图像的处理会反映在逆变换图像上，从而更好地进行图像处理。

图像傅里叶变化主要使用的函数如下所示：

```
#实现图像逆傅里叶变换，返回一个复数数组
numpy.fft.ifft2(a, n=None, axis=-1, norm=None)
#fftshit()函数的逆函数，它将频谱图像的中心低频部分移动至左上角
numpy.fft.fftshift()
#将复数转换为 0 至 255 范围
```

```
iimg = numpy.abs(逆傅里叶变换结果)
```

下面的代码分别实现了傅里叶变换和傅里叶逆变换。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv.imread('luo.png', 0)

#傅里叶变换
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
res = np.log(np.abs(fshift))

#傅里叶逆变换
ishift = np.fft.ifftshift(fshift)
iimg = np.fft.ifft2(ishift)
iimg = np.abs(iimg)
```

```
#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#展示结果
plt.subplot(131), plt.imshow(img, 'gray'), plt.title('(a)原始图像
')
plt.axis('off')
plt.subplot(132), plt.imshow(res, 'gray'), plt.title('(b)傅里叶变
换处理')
plt.axis('off')
plt.subplot(133), plt.imshow(iimg, 'gray'), plt.title('(c)傅里叶
逆变换处理')
plt.axis('off')
plt.show()
```

输出结果如图 32-6 所示，从左至右分别为原始图像、频谱图像、逆傅里叶变换转换图像。

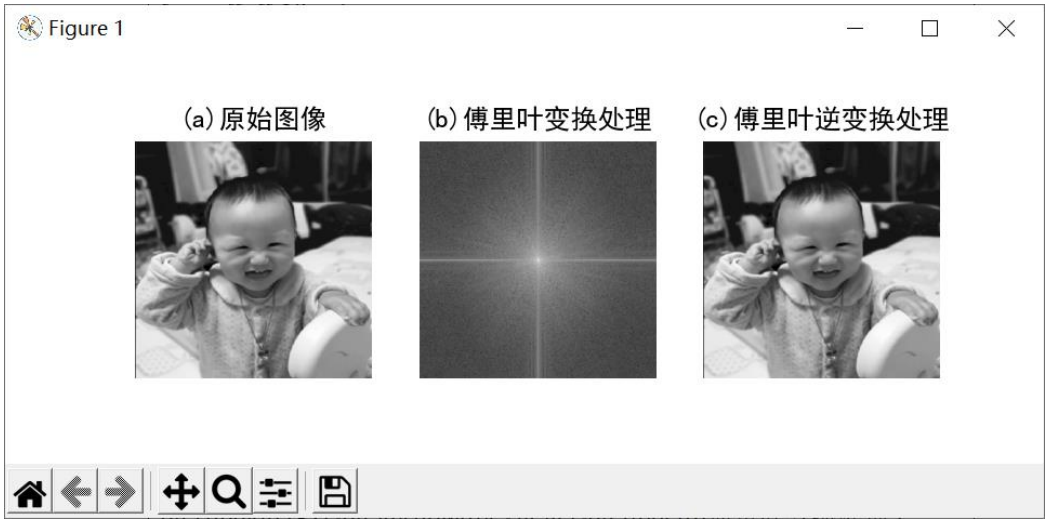


图 32-6 逆傅里叶变换对比图

4.高通滤波和低通滤波

傅里叶变换的目的并不是为了观察图像的频率分布（至少不是最终目的），更多情况下是为了对频率进行过滤，通过修改频率以达到图像增强、图像去噪、边缘检测、特征提取、压缩加密等目的。

过滤的方法一般有三种：

- 低通（Low-pass）
- 高通（High-pass）
- 带通（Band-pass）

所谓低通就是保留图像中的低频成分，过滤高频成分，可以把过滤器想象成一张渔网，想要低通过滤器，就是将高频区域的信号全部拉黑，而低频区域全部保留。例如，在一幅大草原的图像中，低频对应着广袤且颜色趋于一致的草原，表示图像变换缓慢的灰度分量；高频对应着草原图像中的老虎等边缘信息，表示

图像变换较快的灰度分量，由于灰度尖锐过度造成[4]。

（1）高通滤波器

高通滤波器是指通过高频的滤波器，衰减低频而通过高频，常用于增强尖锐的细节，但会导致图像的对比度会降低。该滤波器将检测图像的某个区域，根据像素与周围像素的差值来提升像素的亮度。图 32-7 展示了“Lena”图对应的频谱图像，其中心区域为低频部分。

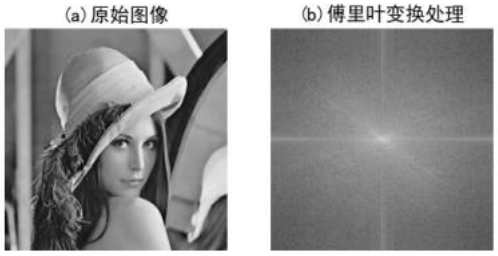


图 32-7 傅里叶变换

接着通过高通滤波器覆盖掉中心低频部分，将 255 两点变换为 0，同时保留高频部分，其处理过程如图 32-8 所示。

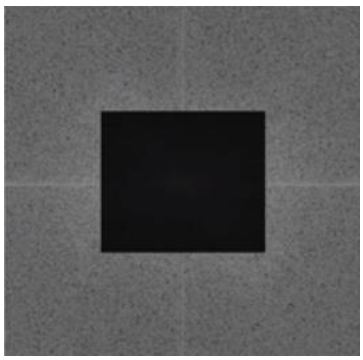


图 32-8 高通滤波器处理

其中心黑色模板生成的核心代码如下：

```
rows, cols = img.shape

crow,ccol = int(rows/2), int(cols/2)
```

```
fshift[crow-30:crow+30, ccol-30:ccol+30] = 0
```

通过高通滤波器将提取图像的边缘轮廓，生成如图 32-9 所示图像。

图 32-9 高通滤波器提取边缘轮廓

完整代码如下所示：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv.imread('lena-hd.png', 0)

#傅里叶变换
```

```
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)

#设置高通滤波器
rows, cols = img.shape
crow,ccol = int(rows/2), int(cols/2)
fshift[crow-30:crow+30, ccol-30:ccol+30] = 0

#傅里叶逆变换
ishift = np.fft.ifftshift(fshift)
iimg = np.fft.ifft2(ishift)
iimg = np.abs(iimg)

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#显示原始图像和高通滤波处理图像
plt.subplot(121), plt.imshow(img, 'gray'), plt.title('(a)原始图像
')
plt.axis('off')
plt.subplot(122), plt.imshow(iimg, 'gray'), plt.title('(b)结果图
```

```
像')
plt.axis('off')
plt.show()
```

输出结果如图 32-10 所示，图(a)为原始“Lena”图，图(b)为高通滤波器提取的边缘轮廓图像。它通过傅里叶变换转换为频谱图像，再将中心的低频部分设置为 0，再通过傅里叶逆变换转换为最终输出图像。

图 32-10 高通滤波器提取边缘轮廓

（2）低通滤波器

低通滤波器是指通过低频的滤波器，衰减高频而通过低频，常用于模糊图像。低通滤波器与高通滤波器相反，当一个像素与周围像素的插值小于一个特定值时，平滑该像素的亮度，常用于去燥和模糊化处理。如 PS 软件中的高斯模糊，就是常见的模糊滤波器之一，属于削弱高频信号的低通滤波器。

图 32-7 展示了“Lena”图对应的频谱图像，其中心区域为低频部分。如果构造低通滤波器，则将频谱图像中心低频部分保留，其他部分替换为黑色 0，其处理过程如图 32-11 所示，最终得到的效果图为模糊图像。

图 32-11 低通滤波器

那么，如何构造该滤波图像呢？如图 32-12 所示，滤波图像是通过低通滤波器和频谱图像形成。其中低通滤波器中心区域为白色 255，其他区域为黑色 0。

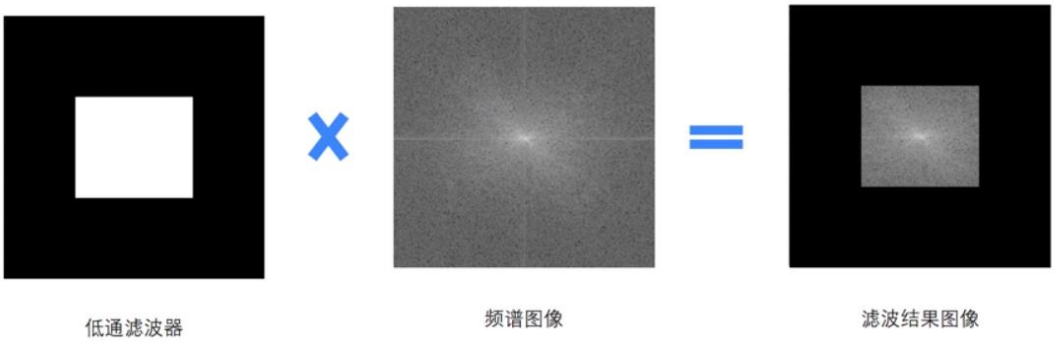


图 32-12 滤波图像构造过程

低通滤波器主要通过矩阵设置构造，其核心代码如下：

```
rows, cols = img.shape
crow,ccol = int(rows/2), int(cols/2)
mask = np.zeros((rows, cols, 2), np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1
```

通过低通滤波器将模糊图像的完整代码如下所示：

```
# -*- coding: utf-8 -*-
```

```
# By: Eastmount
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv2.imread('lena-hd.png', 0)

#傅里叶变换
dft = cv2.dft(np.float32(img), flags = cv2.DFT_COMPLEX_OUTPUT)
fshift = np.fft.fftshift(dft)

#设置低通滤波器
rows, cols = img.shape
crow,ccol = int(rows/2), int(cols/2) #中心位置
mask = np.zeros((rows, cols, 2), np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1

#掩膜图像和频谱图像乘积
```

```
f = fshift * mask
print(f.shape, fshift.shape, mask.shape)

#傅里叶逆变换
ishift = np.fft.ifftshift(f)
iimg = cv2.idft(ishift)
res = cv2.magnitude(iimg[:,:,0], iimg[:,:,1])

#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']

#显示原始图像和低通滤波处理图像
plt.subplot(121), plt.imshow(img, 'gray'), plt.title('(a)原始图像
')
plt.axis('off')
plt.subplot(122), plt.imshow(res, 'gray'), plt.title('(b)结果图像
')
plt.axis('off')
plt.show()
```

输出结果如图 32-13 所示，图(a)为原始“Lena”图，图(b)为低通滤波器模糊处理后的图像。

图 32-13 低通滤波器模糊处理

5.总结

本章主要讲解傅里叶变换。傅里叶变换主要用来进行图像除噪、图像增强处理，通过 Numpy 和 OpenCV 两种方法分别进行叙述，并结合代码加深了读者的印象。希望读者深入了解傅里叶变换的原理知识，该算法应用非常广。

参考文献：

[1] 冈萨雷斯著. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[4] 张铮，王艳平，薛桂香等. 数字图像处理与机器视觉——Visual C++与 Matlab 实现[M]. 北京：人民邮电出版社，2014.

[5] 百度百科. 傅里叶变换[EB/OL]. (2019.02.05). https://baike.baidu.com/item/傅里叶变换/7119029.

[6] Eastmount. [Python 图像处理] 三十二.傅里叶变换（图像去噪）与霍夫变换（特征识别）万字详细总结 [EB/OL]. (2020.12.02). https://blog.csdn.net/Eastmount/article/details/ 110487868.

[7] daduzimama. 图像的傅里叶变换的迷思----频谱居中[EB/OL]. (2018-06-07). https://blog.csdn.net/daduzimama/article/details/80597454.

[8] tenderwx. [数字图像处理] 傅里叶变换在图像处理中的应用[EB/OL]. (2016-03-05). https://www.cnblogs.com/tenderwx/p/5245859.html.

[9] 小小猫钓小小鱼. 深入浅出的讲解傅里叶变换（真正的通俗易懂）[EB/OL]. (2018-02-02). https://www.cnblogs.com/h2zZhou/p/8405717.html.

第 33 篇 图像霍夫变换详解

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

在数字图像处理中，有两个经典的变换被广泛应用——傅里叶变换和霍夫变换。其中，傅里叶变换主要是将时间域上的信号转变为频率域上的信号，用来进行图像除噪、图像增强等处理；霍夫变换主要用来辨别找出物件中的特征，用来进行特征检测、图像分析、数位影像处理等处理。本文主要讲解图像霍夫变换。

1.霍夫变换

霍夫变换(Hough Transform)是一种特征检测(Feature Extraction)，被广泛应用在图像分析、计算机视觉以及数位影像处理。霍夫变换是在 1959 年由气泡室 (Bubble Chamber) 照片的机器分析而发明，发明者 Paul Hough 在 1962 年获得美国专利。现在广泛使用的霍夫变换是由 Richard Duda 和 Peter Hart 在 1972 年发明，并称之为广义霍夫变换。经典的霍夫变换是检测图片中的直线，之后，霍夫变换不仅能识别直线，也能够识别任何形状，常见的有圆形、椭圆形。1981 年，因为 Dana H.Ballard 的一篇期刊论文“Generalizing the Hough transform to detect arbitrary shapes”，

让霍夫变换开始流行于计算机视觉界。

霍夫变换是一种特征提取技术，用来辨别找出物件中的特征，其目的是通过投票程序在特定类型的形状内找到对象的不完美实例。这个投票程序是在一个参数空间中进行的，在这个参数空间中，候选对象被当作所谓的累加器空间中的局部最大值来获得，累加器空间是由计算霍夫变换的算法明确地构建。霍夫变换主要优点是能容忍特征边界描述中的间隙，并且相对不受图像噪声的影响。

最基本的霍夫变换是从黑白图像中检测直线，它的算法流程大致如下：给定一个物件和要辨别的形状的种类，算法会在参数空间中执行投票来决定物体的形状，而这是由累加空间里的局部最大值来决定。假设存在直线公式如（33-1）所示，其中 m 表示斜率，b 表示截距。

$$y = m \cdot x + b \tag{33-1}$$

如果用参数空间表示，则直线为（b, m），但它存在一个问题，垂直线的斜率不存在（或无限大），使得斜率参数 m 值接近于无限。为此，为了更好的计算，Richard O. Duda 和 Peter E. Hart 在 1971 年 4 月，提出了 Hesse normal form（Hesse 法线式），如公式（33-2）所示，它转换为直线的离散极坐标公式。

$$r = x \cdot \cos\theta + y \cdot \sin\theta \tag{33-2}$$

其中 r 是原点到直线上最近点的距离，θ 是 x 轴与连接原点和最近点直线之间的夹角，如图 33-1 所示。

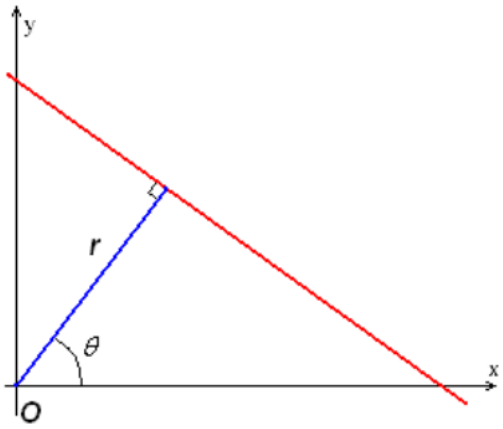


图 33-1 直线坐标

对于点(x_0, y_0)，可以将通过这个点的一族直线统一定义为公式（33-3）。因此，可以将图像的每一条直线与一对参数(r, θ)相关联，相当于每一对(r_0, θ)代表一条通过点的直线(x_0, y_0)，其中这个参数(r, θ)平面被称为霍夫空间。

$$r_0 = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta \qquad (33\text{-}3)$$

然而在实现的图像处理领域，图像的像素坐标 P(x, y)是已知的，而(r, θ)是需要寻找的变量。如果能根据像素点坐标 P(x, y)值绘制每个(r, θ)值，那么就从图像笛卡尔坐标系统转换到极坐标霍夫空间系统，这种从点到曲线的变换称为直线的霍夫变换。变换通过量化霍夫参数空间为有限个值间隔等分或者累加格子。

当霍夫变换算法开始，每个像素坐标点 P(x, y)被转换到(r, θ)的曲线点上面，累加到对应的格子数据点，当一个波峰出现时候，说明有直线存在。如图 33-2 所示，三条正弦曲线在平面相交于一点，该点坐标(r_0, θ)表示三个点组成的平面内的直线。这就是使用霍夫变换检测直线的过程，它追踪图像中每个点对应曲线间的交点，如果交于一点的曲线的数量超过了阈值，则认为该交点所代表的参数对(r_0, θ)在原图像中为一条直线。

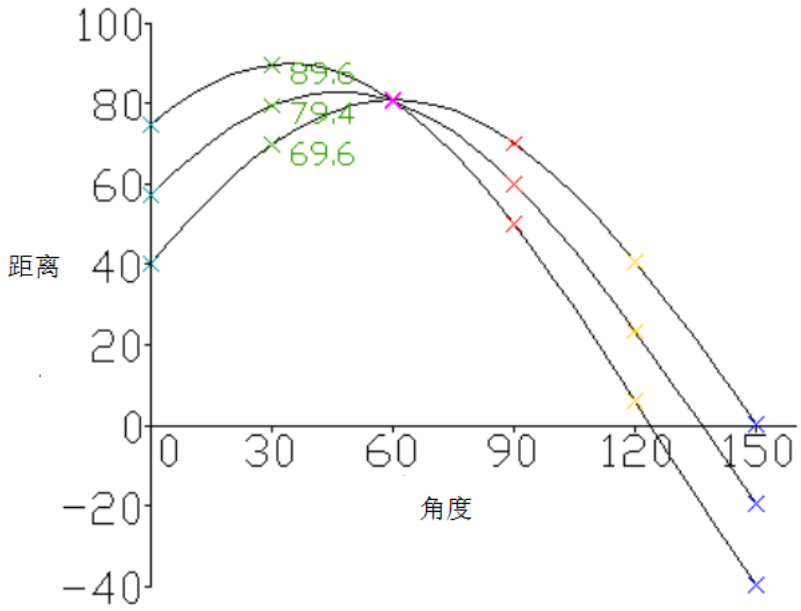


图 33-2 正弦曲线相交

同样的原理，可以用来检测圆，对于圆的参数方程变为如下等式：

$$(x-a)^2+(y-b)^2=r^2 \qquad (33\text{-}4)$$

其中(a, b)为圆的中心点坐标，r 圆的半径。这样霍夫参数空间就变成一个三维参数空间。给定圆半径转为二维霍夫参数空间，变换相对简单，也比较常用。

2.图像霍夫线变换操作

在 OpenCV 中，霍夫变换分为霍夫线变换和霍夫圆变换，其中霍夫线变换支持三种不同方法——标准霍夫变换、多尺度霍夫变换和累计概率霍夫变换[3]。

- 标准霍夫变换主要有 HoughLines()函数实现。
- 多尺度霍夫变换是标准霍夫变换在多尺度下的变换，可以通过 HoughLines()函数实现。

❖ 累计概率霍夫变换是标准霍夫变换的改进，它能在一定范围内进行霍夫变换，计算单独线段的方向及范围，从而减少计算量，缩短计算时间，可以通过 HoughLinesP()函数实现。

在 OpenCV 中，通过函数 HoughLines()检测直线，并且能够调用标准霍夫变换（SHT）和多尺度霍夫变换（MSHT），其函数原型如下所示[3]：

lines = HoughLines(image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]])

- image 表示输入的二值图像
- lines 表示经过霍夫变换检测到直线的输出矢量，每条直线为(r, θ)
- rho 表示以像素为单位的累加器的距离精度
- theta 表示以弧度为单位的累加器角度精度
- threshold 表示累加平面的阈值参数，识别某部分为图中的一条直线时它在累加平面中必须达到的值，大于该值线段才能被检测返回
- srn 表示多尺度霍夫变换中 rho 的除数距离，默认值为 0。粗略的累加器进步尺寸为 rho，而精确的累加器进步尺寸为 rho/srn
- stn 表示多尺度霍夫变换中距离精度 theta 的除数，默认值为 0,。如果 srn 和 stn 同时为 0，使用标准霍夫变换
- min_theta 表示标准和多尺度的霍夫变换中检查线条的最小角度。必须介于 0 和 max_theta 之间
- max_theta 表示标准和多尺度的霍夫变换中要检查线条的最大角度。必须介于 min_theta 和 π 之间

下面的代码是调用 HoughLines()函数检测图像中的直线，并将所有的直线绘制于原图像中。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv2.imread('lines.png')

#灰度变换
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#转换为二值图像
edges = cv2.Canny(gray, 50, 150)

#显示原始图像
plt.subplot(121), plt.imshow(edges, 'gray'), plt.title(u'(a)原始
图像')
```

```
plt.axis('off')

#霍夫变换检测直线
lines = cv2.HoughLines(edges, 1, np.pi / 180, 160)

#转换为二维
line = lines[:, 0, :]

#将检测的线在极坐标中绘制
for rho,theta in line[:]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho
    print(x0, y0)
    x1 = int(x0 + 1000 * (-b))
    y1 = int(y0 + 1000 * (a))
    x2 = int(x0 - 1000 * (-b))
    y2 = int(y0 - 1000 * (a))
    print(x1, y1, x2, y2)
```

```
    #绘制直线
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 1)


#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']


#显示处理图像
plt.subplot(122), plt.imshow(img, 'gray'), plt.title('(b)结果图像
')


plt.axis('off')
plt.show()
```

输出结果如图 33-3 所示，第一幅图为原始图像，第二幅检测出的直线。

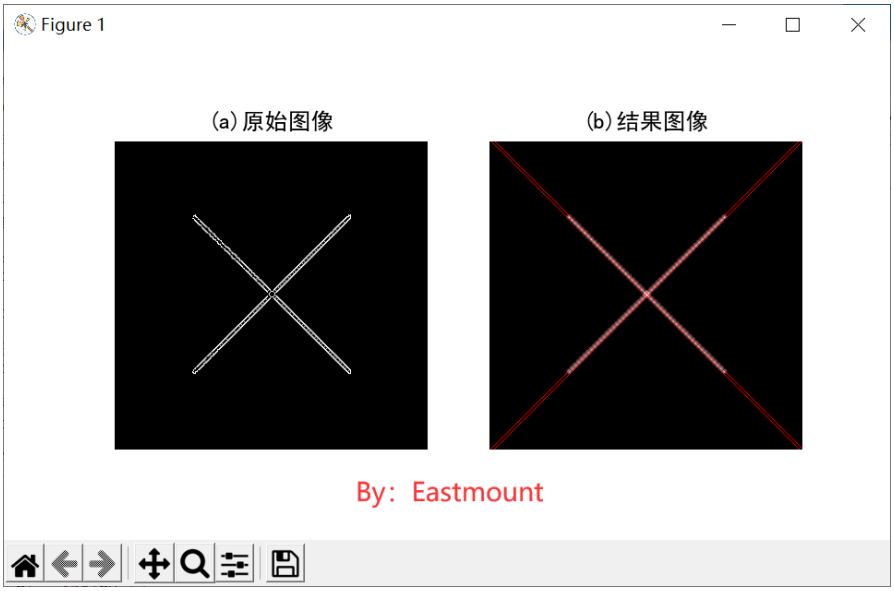


图 33-3 OpenCV 标准霍夫变换检测直线

使用该方法检测大楼图像中的直线如图 33-4 所示，可以发现直线会存在越界的情况。

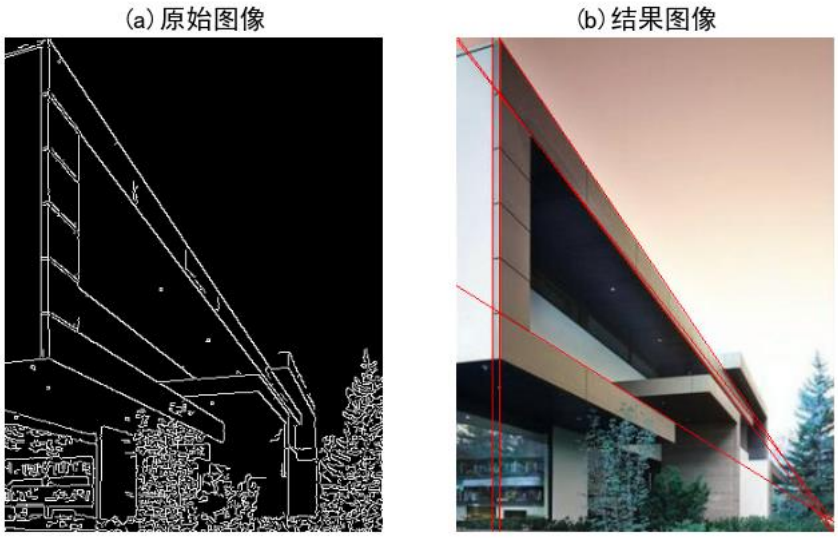


图 33-4 OpenCV 标准霍夫变换检测大楼直线

前面的标准霍夫变换会计算图像中的每一个点，计算量比较大，另外它得到的是整条线(r, θ)，并不知道原图中直线的端点。接下来使用累计概率霍夫变换，它是一种改进的霍夫变换，调用 HoughLinesP()函数实现。

lines = HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])

- image 表示输入的二值图像
- lines 表示经过霍夫变换检测到直线的输出矢量，每条直线具有 4 个元素的矢量，即（x_1, y_1）和（x_2, y_2）是每个检测线段的端点
- rho 表示以像素为单位的累加器的距离精度
- theta 表示以弧度为单位的累加器角度精度

- threshold 表示累加平面的阈值参数，识别某部分为图中的一条直线时它在累加平面中必须达到的值，大于该值线段才能被检测返回
- minLineLength 表示最低线段的长度，比这个设定参数短的线段不能被显示出来，默认值为 0
- maxLineGap 表示允许将同一行点与点之间连接起来的最大距离，默认值 0

下面的代码是调用 HoughLinesP()函数检测图像中的直线，并将所有的直线绘制于原图像中。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv2.imread('judge.png')

#灰度转换
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
#转换为二值图像
edges = cv2.Canny(gray, 50, 200)

#显示原始图像
plt.subplot(121), plt.imshow(edges, 'gray'), plt.title(u'(a)原始图像')
plt.axis('off')

#霍夫变换检测直线
minLineLength = 60
maxLineGap = 10
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 30, minLineLength, maxLineGap)

#绘制直线
lines1 = lines[:, 0, :]
for x1,y1,x2,y2 in lines1[:]:
    cv2.line(img, (x1,y1), (x2,y2), (255,0,0), 2)

res = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']


#显示处理图像
plt.subplot(122), plt.imshow(res), plt.title('(b)结果图像')
plt.axis('off')
plt.show()
```

输出结果如图 33-5 所示，图(a)为原始图像，图(b)检测出的直线，它有效地提取了线段的起点和终点。

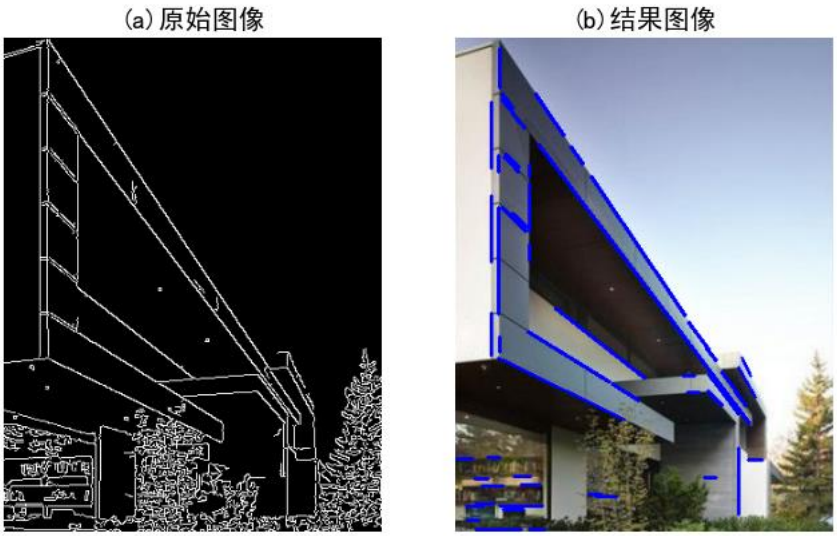


图 33-5 OpenCV 累计概率霍夫变换检测直线

3.图像霍夫圆变换操作

霍夫圆变换的原理与霍夫线变换很类似，只是将线的(r, θ)二维坐标提升为三维坐标，包括圆心点(x_center,y_center,r)和半径 r，其数学形式如公式

（33-5）。

$$(x-x_{center})^2+(y-y_{center})^2=r^2 \quad (33-5)$$

从而一个圆的确定需要三个参数，通过三层循环实现，接着寻找参数空间累加器的最大（或者大于某一阈值）的值。随着数据量的增大，导致圆的检测将比直线更耗时，所以一般使用霍夫梯度法减少计算量。在 OpenCV 中，提供了cv2.HoughCircles()函数检测圆，其原型如下所示：

circles = HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]])

- image 表示输入图像，8 位灰度单通道图像
- circles 表示经过霍夫变换检测到圆的输出矢量，每个矢量包括 3 个元素，即（x, y, radius）
- method 表示检测方法，包括 HOUGH_GRADIENT 值
- dp 表示用来检测圆心的累加器图像的分辨率于输入图像之比的倒数，允许创建一个比输入图像分辨率低的累加器
- minDist 表示霍夫变换检测到的圆的圆心之间的最小距离
- param1 表示参数 method 设置检测方法的对应参数，对当前唯一的方法霍夫梯度法 CV_HOUGH_GRADIENT，它表示传递给 Canny 边缘检测算子的高阈值，而低阈值为高阈值的一半，默认值 100
- param2 表示参数 method 设置检测方法的对应参数，对当前唯一的方法霍夫梯度法 CV_HOUGH_GRADIENT，它表示在检测

阶段圆心的累加器阈值，它越小，将检测到更多根本不存在的圆；它越大，能通过检测的圆就更接近完美的圆形

- minRadius 表示圆半径的最小值，默认值为 0
- maxRadius 表示圆半径的最大值，默认值 0

下列代码是检测图像中的圆。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

#读取图像
img = cv2.imread('test.png')

#灰度转换
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#显示原始图像
plt.subplot(121), plt.imshow(gray, 'gray'), plt.title(u'(a)原始图像')
```

```
plt.axis('off')


#霍夫变换检测圆
circles1 = cv2.HoughCircles(gray,
cv2.HOUGH_GRADIENT, 1, 20, param2=30)
print(circles1)


#提取为二维
circles = circles1[0, :, :]


#四舍五入取整
circles = np.uint16(np.around(circles))


#绘制圆
for i in circles[:]:
    cv2.circle(img, (i[0],i[1]), i[2], (255,0,0), 5) #画圆
    cv2.circle(img, (i[0],i[1]), 2, (255,0,255), 10) #画圆心


#设置字体
matplotlib.rcParams['font.sans-serif']=['SimHei']
plt.subplot(122), plt.imshow(img), plt.title('(b)结果图像')
```

```
plt.axis('off')
plt.show()
```

输出结果如图 33-6 所示，图(a)为原始图像，图(b)检测出的圆形，它有效地提取了圆形的圆心和轮廓。

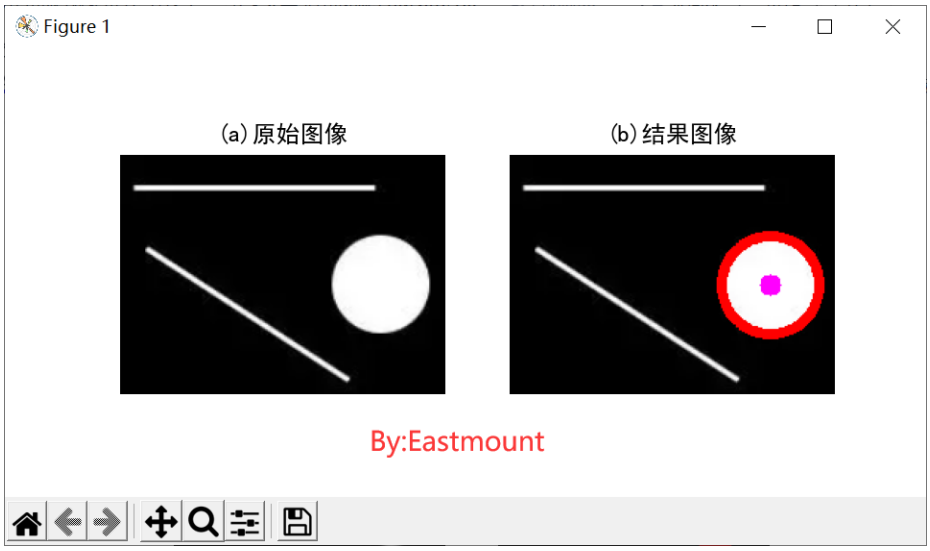


图 33-6 霍夫变换检测圆

使用下面的函数能有效提取人类眼睛的轮廓，核心函数如下：

```
circles1 = cv2.HoughCircles(gray,
cv2.HOUGH_GRADIENT, 1, 20,
                            param1=100, param2=30,
                            minRadius=160,
maxRadius=300)
```

输出结果如图 33-7 所示，它提取了三条圆形接近于人体的眼睛。

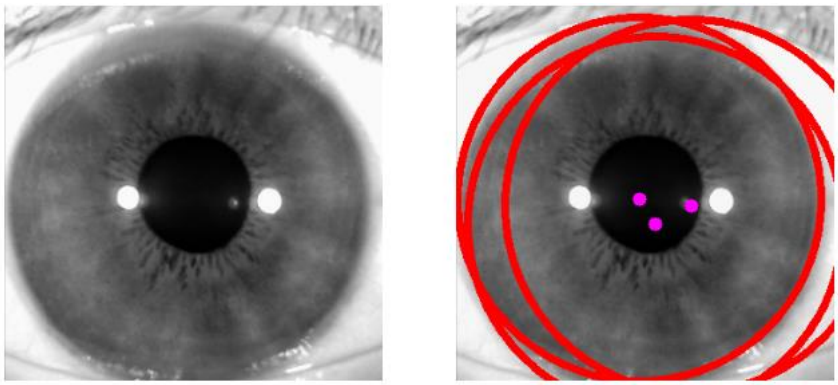


图 33-7 霍夫变换检测人体眼睛

图 33-7 中显示了三条曲线，通过不断优化最大半径和最小半径，比如将 minRadius 设置为 160，maxRadius 设置为 200，将提取更为精准的人体眼睛，如图 33-8 所示。

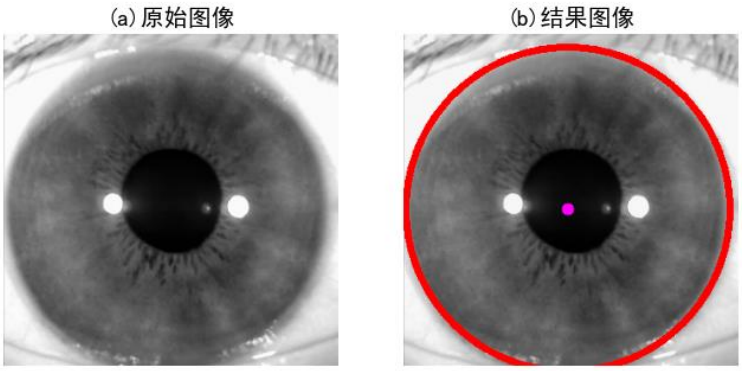


图 33-8 霍夫变换检测最终人体眼睛

4.总结

本章主要讲解霍夫变换。霍夫变换主要用来辨别找出物件中的特征，包括提取图像中的直线和圆，调用 cv2.HoughLines()、cv2.HoughLinesP()和 cv2.HoughCircles()实现。希望对大家有所帮助，这也将为后续图像识别和目标检测提供帮助。

参考文献：

[1] 冈萨雷斯著. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 阮秋琦. 数字图像处理学（第 3 版）[M]. 北京：电子工业出版社，2008.

[3] 毛星云，冷雪飞. OpenCV3 编程入门[M]. 北京：电子工业出版社，2015.

[4] 张铮，王艳平，薛桂香等. 数字图像处理与机器视觉——Visual C++与 Matlab 实现[M]. 北京：人民邮电出版社，2014.

[5] Eastmount. [Python 图像处理] 三十二.傅里叶变换（图像去噪）与霍夫变换（特征识别）万字详细总结 [EB/OL]. (2020.12.02). https://blog.csdn.net/Eastmount/article/details/ 110487868.

[6] 百度百科. 霍夫变换[EB/OL]. (2018-11-21). https://baike.baidu.com/item/霍夫变换/4647236.

[7] yuyuntan. 经典霍夫变换（Hough Transform）[EB/OL]. (2018-04-29). https://blog.csdn.net/yuyuntan/article/details/80141392.

[8] g20040733. 霍夫变换 [EB/OL]. (2016-12-07). https://blog.csdn.net/g200407331/article/details/53507784.

[9] 我 i 智能. Python 下 opencv 使用笔记（十一）（详解 hough 变换检测直线与圆）[EB/OL]. (2015-07-23). https://blog.csdn.net/on2way/article/details/47028969.

[10] ex2tron. Python+OpenCV 教程 17：霍夫变换[EB/OL]. (2018-01-10). https://www.jianshu.com/p/34d6dc466e81.

[11] Daetalus. OpenCV-Python 教程（9、使用霍夫变换检测直线）[EB/OL]. (2013-

07-12). https://blog.csdn.net/sunny2038/article/details/9253823.

第 34 篇 图像分类理论知识和基于机器学习的图像分类

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

图像分类是根据图像信息中所反映的不同特征，把不同类别的目标区分开来的图像处理方法。本文主要讲解常见的图像分类算法，并详细讲解了 Python 环境下的贝叶斯图像分类算法和基于 KNN 算法的图像分类等案例。

1.图像分类

图像分类（Image Classification）是对图像内容进行分类的问题，它利用计算机对图像进行定量分析，把图像或图像中的区域划分为若干个类别，以代替人的视觉判断。图像分类的传统方法是特征描述及检测，这类传统方法可能对于一些简单的图像分类是有效的，但由于实际情况非常复杂，传统的分类方法不

堪重负。现在，广泛使用机器学习和深度学习的方法来处理图像分类问题，其主要任务是给定一堆输入图片，将其指派到一个已知的混合类别中的某个标签[1]。

在图 34-1 中，图像分类模型将获取单个图像，并将为 4 个标签{cat，dog，hat，mug}分配对应的概率{0.6, 0.3, 0.05, 0.05}，其中 0.6 表示图像标签为猫的概率，其余类比。如图 16-1 所示，该图像被表示为一个三维数组。在这个例子中，猫的图像宽度为 248 像素，高度为 400 像素，并具有红绿蓝三个颜色通道（通常称为 RGB）。因此，图像由 248×400×3 个数字组成或总共 297600 个数字，每个数字是一个从 0（黑色）到 255（白色）的整数。图像分类的任务是将这接近 30 万个数字变成一个单一的标签，如“猫（cat）”[2]。

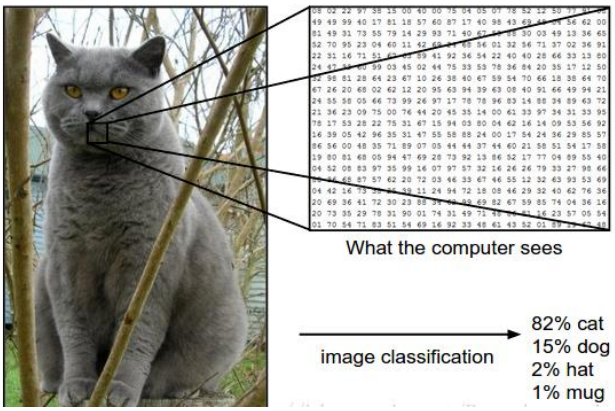


图 34-1 图像分类模型

那么，如何编写一个图像分类的算法呢？又怎么从众多图像中识别出猫呢？这里所采取的方法和教育小孩看图识物类似，给出很多图像数据，让模型不断去学习每个类的特征。在训练之前，首先需要对训练集的图像进行分类标注，如图 34-2 所示，包括 cat、dog、mug 和 hat 四类。在实际工程中，可能有成千上万类别的物体，每个类别都会有上百万张图像。

图 34-2 训练数据集

图像分类是输入一堆图像的像素值数组，然后给它分配一个分类标签，通过训练学习来建立算法模型，接着使用该模型进行图像分类预测，具体流程如下：

- **输入：**输入包含 N 个图像的集合，每个图像的标签是 K 种分类标签中的一种，这个集合称为训练集；
- **学习：**第二步任务是使用训练集来学习每个类的特征，构建训练分类器或者分类模型；
- **评价：**通过分类器来预测新输入图像的分类标签，并以此来评价分类器的质量。通过分类器预测的标签和图像真正的分类标签对比，从而评价分类算法的好坏。如果分类器预测的分类标签和图像真正的分类标签一致，表示预测正确，否则预测错误。

常见的分类算法包括朴素贝叶斯分类器、决策树、K 最近邻分类算法、支持向量机、神经网络和基于规则的分类算法等，同时还有用于组合单一类方法的集成学习算法，如 Bagging 和 Boosting 等[2]。

2.基于朴素贝叶斯的图像分类

朴素贝叶斯分类（Naive Bayes Classifier）发源于古典数学理论，利用Bayes 定理来预测一个未知类别的样本属于各个类别的可能性，选择其中可能性最大的一个类别作为该样本的最终类别。在朴素贝叶斯分类模型中，它将为每一个类别的特征向量建立服从正态分布的函数，给定训练数据，算法将会估计每一个类别的向量均值和方差矩阵，然后根据这些进行预测。

朴素贝叶斯分类模型的正式定义如下：

1) 设 $x=\{a_1,a_2,...,a_m\}$ 为一个待分类项，而每个 a 为 x 的一个特征属性；

2) 有类别集合 $C=\{y_1,y_2,...,y_n\}$；

3) 计算 $P(y_1|x),P(y_2|x),P(y_n|x)$；

4) 如果 $P(y_k|x)=\max\{P(y_1|x),P(y_2|x),...,P(y_n|x)\}$，则 $x\in y_k$。

该算法的特点为：如果没有很多数据，该模型会比很多复杂的模型获得更好的性能，因为复杂的模型用了太多假设，以致产生欠拟合。

本文主要使用 Scikit-Learn 包进行 Python 图像分类处理。Scikit-Learn 扩展包是用于 Python 数据挖掘和数据分析的经典、实用扩展包，通常缩写为 Sklearn。Scikit-Learn 中的机器学习模型是非常丰富的，包括线性回归、决策树、SVM、KMeans、KNN、PCA 等等，用户可以根据具体分析问题的类型选择该扩展包的合适模型，从而进行数据分析，其安装过程主要通过“pip install scikit-learn”实现。

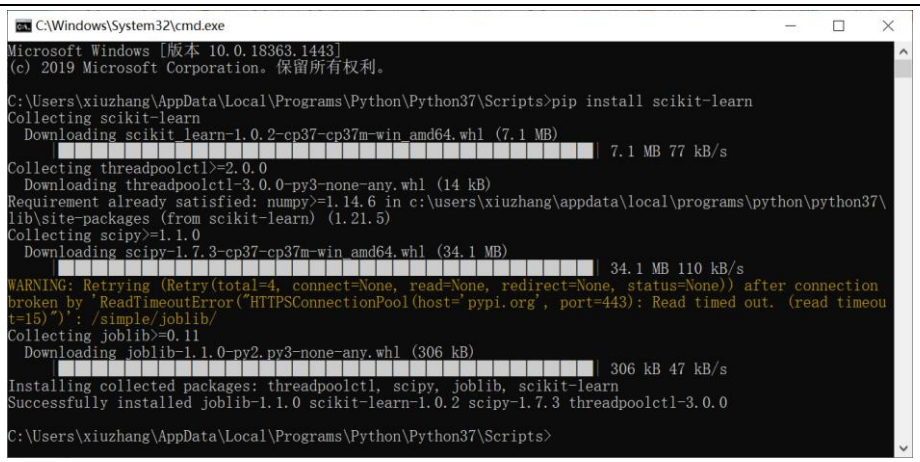


图 34-3 安装过程

实验所采用的数据集为 Sort_1000pics 数据集，该数据集包含了 1000 张图片，总共分为 10 大类，分别是人（第 0 类）、沙滩（第 1 类）、建筑（第 2 类）、大卡车（第 3 类）、恐龙（第 4 类）、大象（第 5 类）、花朵（第 6 类）、马（第 7 类）、山峰（第 8 类）和食品（第 9 类），每类 100 张。如图 34-4 所示。

图 34-4 数据集

接着将所有各类图像按照对应的类标划分至“0”至“9”命名的文件夹中，如图 34-5 所示，每个文件夹中均包含了 100 张图像，对应同一类别。

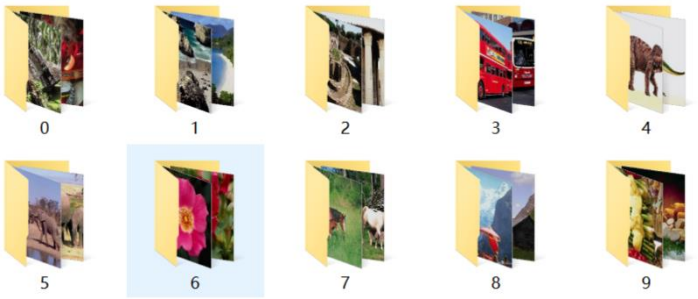


图 34-5 图像分类文件夹

比如，文件夹名称为“6”中包含了 100 张花的图像，如图 34-6 所示。

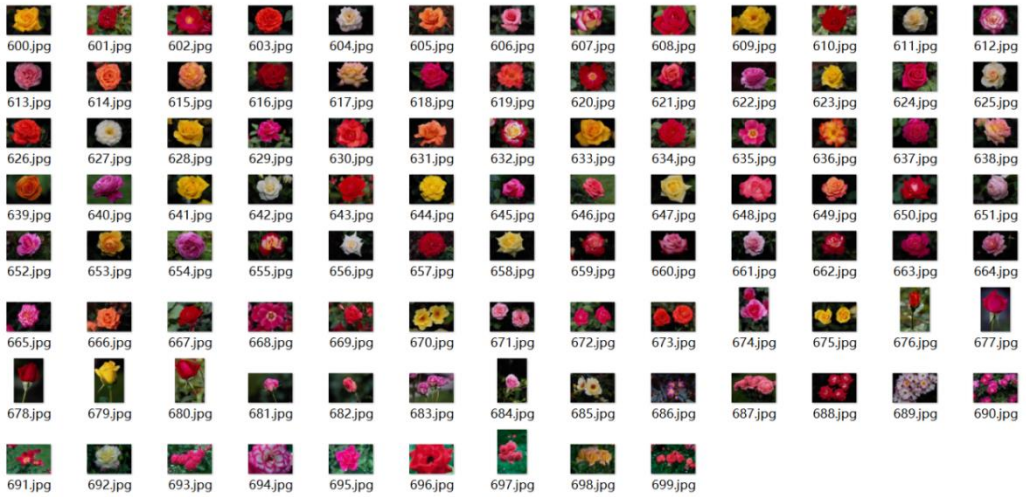


图 34-6 部分文件夹中的图像

下面是调用朴素贝叶斯算法进行图像分类的完整代码，调用 sklearn.naive_bayes 中的 BernoulliNB()函数进行实验。它将 1000 张图像按照训练集为 70%，测试集为 30%的比例随机划分，再获取每张图像的像素直方图，根据像素的特征分布情况进行图像分类分析。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import os
```

```
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,
classification_report

#----------------------------------------
----------------------------
# 第一步 切分训练集和测试集
#----------------------------------------
----------------------------
X = [] #定义图像名称
Y = [] #定义图像分类类标
Z = [] #定义图像像素

for i in range(0, 10):
    #遍历文件夹，读取图片
    for f in os.listdir("photo/%s" % i):
        #获取图像名称
        X.append("photo//" +str(i) + "//" + str(f))
        #获取图像类标即为文件夹名称
```

```
            Y.append(i)
X = np.array(X)
Y = np.array(Y)


#随机率为 100% 选取其中的 30%作为测试集
X_train, X_test, y_train, y_test = train_test_split(X, Y,


test_size=0.3, random_state=1)
print(len(X_train), len(X_test), len(y_train), len(y_test))


#------------------------------------------
----------------------------
# 第二步 图像读取及转换为像素直方图
#------------------------------------------
----------------------------
#训练集
XX_train = []
for i in X_train:
    #读取图像
    image = cv2.imread(i)
```

```
    #图像像素大小一致
    img = cv2.resize(image, (256,256),
                         interpolation=cv2.INTER_CUBIC)

    #计算图像直方图并存储至 X 数组
    hist = cv2.calcHist([img], [0,1], None,
                              [256,256],
[0.0,255.0,0.0,255.0])
    XX_train.append(((hist/255).flatten()))

#测试集
XX_test = []
for i in X_test:
    image = cv2.imread(i)
    img = cv2.resize(image, (256,256),
                         interpolation=cv2.INTER_CUBIC)
    hist = cv2.calcHist([img], [0,1], None,
                              [256,256],
[0.0,255.0,0.0,255.0])
    XX_test.append(((hist/255).flatten()))
```

```
#-----------------------------------------------
---------------------------
# 第三步 基于朴素贝叶斯的图像分类处理
#-----------------------------------------------
---------------------------
from sklearn.naive_bayes import BernoulliNB
clf = BernoulliNB().fit(XX_train, y_train)
predictions_labels = clf.predict(XX_test)
print('预测结果:')
print(predictions_labels)
print('算法评价:')
print((classification_report(y_test, predictions_labels)))

#输出前 10 张图片及预测结果
k = 0
while k<10:
    #读取图像
    print(X_test[k])
    image = cv2.imread(X_test[k])
    print(predictions_labels[k])
    #显示图像
```

```
cv2.imshow("img", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
k = k + 1
```

代码中对预测集的前十张图像进行了显示，其中“368.jpg”图像如图 34-7 所示，其分类预测的类标结果为“3”，表示第 3 类大卡车，预测结果正确。

图 34-7 分类预测大卡车图像

图 34-8 展示了“452.jpg”图像，其分类预测的类标结果为“4”，表示第 4 类恐龙，预测结果正确。

图 34-8 分类预测恐龙图像

图 34-9 展示了“507.jpg”图像，其分类预测的类标结果为“7”，错误

地预测为第 7 类恐龙，其真实结果应该是第 5 类大象。

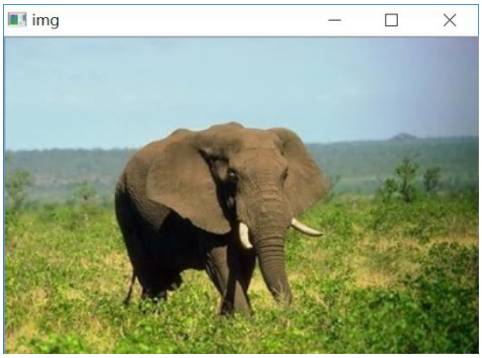


图 34-9 分类预测恐龙图像

使用朴素贝叶斯算法进行图像分类实验，最后预测的结果及算法评价准确率（Precision）、召回率（Recall）和 F 值（F1-score）如图 34-10 所示。

```
>>>
700 300 700 300
预测结果:
[7 8 4 3 2 9 2 4 3 9 4 9 0 3 0 8 8 5 7 4 9 4 2 5 4 1 2 7 2 3 9 7 7 4 8 2 2
 5 7 4 1 6 9 2 9 2 5 2 4 3 2 0 6 0 1 4 8 6 4 9 3 2 3 7 8 5 4 8 0 2 2 8 2 9
 4 2 1 8 3 5 2 7 7 7 9 9 4 8 2 5 6 1 5 9 4 8 5 8 2 8 3 2 9 8 4 5 2 5 4 9 4
 9 9 0 1 1 7 4 1 7 2 3 9 1 4 6 7 7 4 9 7 2 6 0 9 2 7 8 8 7 2 8 9 5 6 7 9 9
 5 2 3 9 1 0 3 5 7 8 0 2 8 2 1 6 5 4 2 5 7 8 2 2 8 4 5 2 1 9 8 9 2 0 7 6 2
 8 2 4 5 0 6 1 2 1 9 4 5 5 6 2 3 7 9 0 5 7 0 0 3 6 7 3 8 4 6 8 3 1 9 9 8 8
 8 9 5 7 0 7 9 8 2 3 8 4 5 9 0 7 2 0 8 5 3 4 4 8 8 4 8 7 2 0 4 7 0 6 9 8 8
 7 8 8 1 0 7 4 3 4 4 8 4 0 8 5 9 7 8 2 6 0 7 8 3 7 5 2 8 1 4 9 6 5 5 1 8 4
 4 0 0 3]
算法评价:
              precision    recall  f1-score   support

           0       0.50      0.39      0.44        31
           1       0.67      0.39      0.49        31
           2       0.48      0.77      0.59        26
           3       0.81      0.59      0.68        29
           4       0.79      0.94      0.86        32
           5       0.57      0.47      0.52        34
           6       0.81      0.43      0.57        30
           7       0.56      0.73      0.63        26
           8       0.48      0.68      0.56        31
           9       0.66      0.77      0.71        30

avg / total       0.63      0.61      0.60       300
```

图 34-10 朴素贝叶斯算法评价结果

3.基于 KNN 的图像分类

K 最近邻分类（K-Nearest Neighbor Classifier）算法是一种基于实例

的分类方法，是数据挖掘分类技术中最简单常用的方法之一[2]。该算法的核心思想如下：一个样本 x 与样本集中的 k 个最相邻的样本中的大多数属于某一个类别 yLabel，那么该样本 x 也属于类别 yLabel，并具有这个类别样本的特性。简而言之，一个样本与数据集中的 k 个最相邻样本中的大多数的类别相同。由其思想可以看出，KNN 是通过测量不同特征值之间的距离进行分类，而且在决策样本类别时，只参考样本周围 k 个“邻居”样本的所属类别。因此比较适合处理样本集存在较多重叠的场景，主要用于预测分析、文本分类、降维等处理。

该算法在建立训练集时，就要确定训练数据及其对应的类别标签；然后把待分类的测试数据与训练集数据依次进行特征比较，从训练集中挑选出最相近的 k 个数据，这 k 个数据中投票最多的分类，即为新样本的类别。KNN 分类算法的流程描述为如图 34-11 所示。

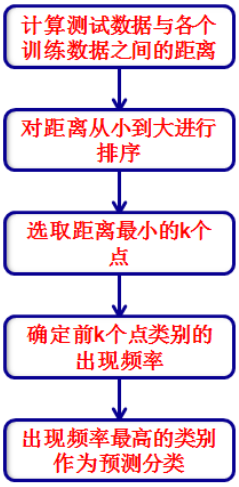


图 34-11 KNN 流程图

该算法的特点为：简单有效，但因为需要存储所有的训练集，占用很大内存，速度相对较慢，使用该方法前通常训练集需要进行降维处理。

下面是基于 KNN 算法的图像分类代码，调用 sklearn.neighbors 中的

KNeighborsClassifier()函数进行实验。核心代码如下：

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=11).fit(XX_train, y_train)
predictions_labels = clf.predict(XX_test)
```

完整代码如下。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

#-----------------------------------------------------------------
# 第一步 切分训练集和测试集
#-----------------------------------------------------------------
X = [] #定义图像名称
```

```
Y = [] #定义图像分类类标
Z = [] #定义图像像素


for i in range(0, 10):
    #遍历文件夹，读取图片
    for f in os.listdir("photo/%s" % i):
        #获取图像名称
        X.append("photo//" +str(i) + "//" + str(f))
        #获取图像类标即为文件夹名称
        Y.append(i)
X = np.array(X)
Y = np.array(Y)


#随机率为 100%  选取其中的 30%作为测试集
X_train, X_test, y_train, y_test = train_test_split(X, Y,


test_size=0.3, random_state=1)
print(len(X_train), len(X_test), len(y_train), len(y_test))


#----------------------------------------
--------------------------
```

```
# 第二步 图像读取及转换为像素直方图
#-------------------------------------------------------------------------
#训练集
XX_train = []
for i in X_train:
    #读取图像
    image = cv2.imread(i)

    #图像像素大小一致
    img = cv2.resize(image, (256,256),
                     interpolation=cv2.INTER_CUBIC)

    #计算图像直方图并存储至 X 数组
    hist = cv2.calcHist([img], [0,1], None,
                        [256,256],
[0.0,255.0,0.0,255.0])
    XX_train.append(((hist/255).flatten()))

#测试集
XX_test = []
```

```
for i in X_test:
    image = cv2.imread(i)
    img = cv2.resize(image, (256,256),
                         interpolation=cv2.INTER_CUBIC)
    hist = cv2.calcHist([img], [0,1], None,
                                [256,256],
[0.0,255.0,0.0,255.0])
    XX_test.append(((hist/255).flatten()))


#-----------------------------------------
--------------------------
# 第三步 基于 KNN 的图像分类处理
#-----------------------------------------
--------------------------
from sklearn.neighbors import KNeighborsClassifier
clf  =  KNeighborsClassifier(n_neighbors=11).fit(XX_train,
y_train)
predictions_labels = clf.predict(XX_test)
print('预测结果:')
print(predictions_labels)
print('算法评价:')
```

```
print((classification_report(y_test, predictions_labels)))


#输出前 10 张图片及预测结果
k = 0
while k<10:
    #读取图像
    print(X_test[k])
    image = cv2.imread(X_test[k])
    print(predictions_labels[k])
    #显示图像
    cv2.imshow("img", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    k = k + 1
```

代码中对预测集的前十张图像进行了显示，其中“818.jpg”图像如图 34-12 所示，其分类预测的类标结果为“8”，表示第 8 类山峰，预测结果正确。

图 34-12 分类预测山峰图像

图 34-13 展示了“929.jpg”图像，其分类预测的类标结果为“9”，正确地预测为第 9 类食品。

图 34-13 分类预测食品图像

使用 KNN 算法进行图像分类实验，最后算法评价的准确率(Precision)、召回率 (Recall) 和 F 值 (F1-score) 如图 34-14 所示，其中平均准确率为 0.64，平均召回率为 0.55，平均 F 值为 0.50，其结果略差于朴素贝叶斯的图像分类算法。

	precision	recall	f1-score	support
0	0.38	0.97	0.55	31
1	0.00	0.00	0.00	31
2	1.00	0.15	0.27	26
3	0.83	0.69	0.75	29
4	1.00	0.88	0.93	32
5	0.35	0.62	0.45	34
6	0.81	0.70	0.75	30
7	0.61	0.88	0.72	26
8	1.00	0.03	0.06	31
9	0.44	0.60	0.51	30
accuracy			0.55	300
macro avg	0.64	0.55	0.50	300
weighted avg	0.63	0.55	0.50	300

图 34-14 分类预测结果

5.总结

本文主要讲解 Python 环境下的图像分类算法，首先普及了常见的分类算法，包括朴素贝叶斯、KNN、SVM、随机森林、神经网络等，接着通过朴素贝叶斯和 KNN 分别实现了 1000 张图像的图像分类实验，希望对读者有一定帮助。此外，本文的代码存在很多缺陷，比如图像分类使用整幅图像的像素进行计算效果会更好，再如深度学习应用于图像分类的结果由于机器学习等，接下来我们将分享基于卷积神经网络的图像分类案例。

参考文献：

[1] 冈萨雷斯著. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 杨秀璋，颜娜. Python 网络数据爬取及分析从入门到精通（分析篇）[M]. 北京：北京航天航空大学出版社, 2018.

[3] gzq0723. 干货——图像分类（上）[EB/OL]. (2018-08-28). https://blog.csdn.net/gzq0723/article/details/82185832.

[4] sinat_34430765. OpenCV 分类器学习心得[EB/OL]. (2016-08-03). https://blog.csdn.net/sinat_34430765/article/details/52103189.

[5] baidu_28342107. 机器学习之贝叶斯算法图像分类[EB/OL]. (2018-10-10). https://blog.csdn.net/baidu_28342107/article/details/82999249.

[6] Eastmount. [Python 图像处理] 二十六.图像分类原理及基于 KNN、朴素贝叶斯算法的图像分类案例[EB/OL]. (2020-02-11). https://blog.csdn.net/Eastmount/article/details/104263641.

第 35 篇 基于卷积神经网络的 MNIST 图像分类

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

图像分类是根据图像信息中所反映的不同特征，把不同类别的目标区分开来的图像处理方法。上一篇文章主要讲解常见的图像分类算法，并介绍了 Python 环境下的贝叶斯图像分类算法和基于 KNN 算法的图像分类等案例。这篇文章将利用卷积神经网络实现 MNIST（手写数字）图像分类，这也是经典的图像分类案例。

1.图像分类

图像分类（Image Classification）是对图像内容进行分类的问题，它利用计算机对图像进行定量分析，把图像或图像中的区域划分为若干个类别，以代替人的视觉判断。图像分类的传统方法是特征描述及检测，这类传统方法可能对于一些简单的图像分类是有效的，但由于实际情况非常复杂，传统的分类方法不堪重负。现在，广泛使用机器学习和深度学习的方法来处理图像分类问题，其主要任务是给定一堆输入图片，将其指派到一个已知的混合类别中的某个标签。

在图 35-1 中，图像分类模型将获取单个图像，并将为 4 个标签{cat，

dog，hat，mug}分配对应的概率{0.6, 0.3, 0.05, 0.05}，其中 0.6 表示图像标签为猫的概率，其余类比。如图所示，该图像被表示为一个三维数组。在这个例子中，猫的图像宽度为 248 像素，高度为 400 像素，并具有红绿蓝三个颜色通道（通常称为 RGB）。因此，图像由 248×400×3 个数字组成或总共 297600 个数字，每个数字是一个从 0（黑色）到 255（白色）的整数。图像分类的任务是将这接近 30 万个数字变成一个单一的标签，如“猫（cat）”。

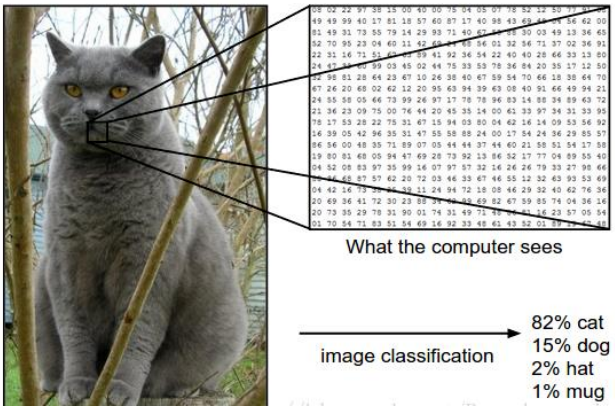


图 35-1 图像分类模型

那么，如何编写一个图像分类的算法呢？又怎么从众多图像中识别出猫呢？这里所采取的方法和教育小孩看图识物类似，给出很多图像数据，让模型不断去学习每个类的特征。在训练之前，首先需要对训练集的图像进行分类标注，如图 35-2 所示，包括 cat、dog、mug 和 hat 四类。在实际工程中，可能有成千上万类别的物体，每个类别都会有上百万张图像。

图 35-2 训练数据集

图像分类是输入一堆图像的像素值数组，然后给它分配一个分类标签，通过训练学习来建立算法模型，接着使用该模型进行图像分类预测。基于神经网络的图像分类流程如图 35-3 所示，参考网易云课程“莫烦”老师分享。

如下图所示，通常来说，计算机处理的东西和人类有所不同，无论是声音、图片还是文字，它们都只能以数字 0 或 1 出现在计算机神经网络里。神经网络看到的图片其实都是一堆数字，对数字的加工处理最终生成另一堆数字，并且具有一定认知上的意义，通过一点点的处理能够得知计算机到底判断这张图片是猫还是狗。

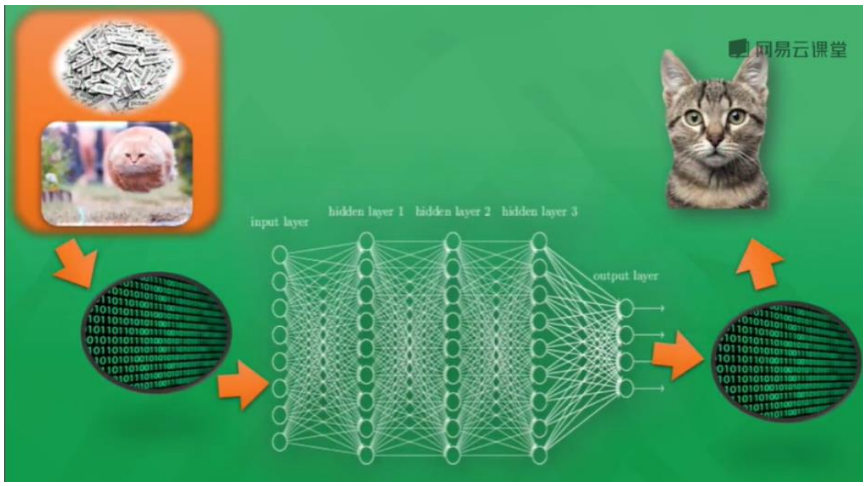


图 35-3 基于神经网络的图像分类

分类（Classification）属于有监督学习中的一类，它是数据挖掘、机器学习和数据科学中一个重要的研究领域。分类模型类似于人类学习的方式，通过对历史数据或训练集的学习得到一个目标函数，再用该目标函数预测新数据集的未知属性。分类模型主要包括两个步骤：

- 训练。给定一个数据集，每个样本都包含一组特征和一个类别信息，然后调用分类算法训练模型。
- 预测。利用生成的模型对新的数据集（测试集）进行分类预测，并判断其分类结果。

通常为了检验学习模型的性能会使用校验集。数据集会被分成不相交的训练集和测试集，训练集用来构造分类模型，测试集用来检验多少类标签被正确分类。

2.神经网络

神经网络（Neural Network）是对非线性可分数据的分类方法，通常包括输入层、隐藏层和输出层。其中，与输入直接相连的称为隐藏层（Hidden Layer），与输出直接相连的称为输出层（Output Layer）。神经网络算法的特点是有比较多的局部最优值，可通过多次随机设定初始值并运行梯度下降算法获得最优值。图像分类中使用最广泛的是 BP 神经网络和 CNN 神经网络。

BP 神经网络是一种多层的前馈神经网络，其主要的特点为：信号是前向传播的，而误差是反向传播的。BP 神经网络的过程主要分为两个阶段，第一阶段是信号的前向传播，从输入层经过隐含层，最后到达输出层；第二阶段是误差的反向传播，从输出层到隐含层，最后到输入层，依次调节隐含层到输出层的权重

和偏置，输入层到隐含层的权重和偏置，具体结构如图 35-4 所示。

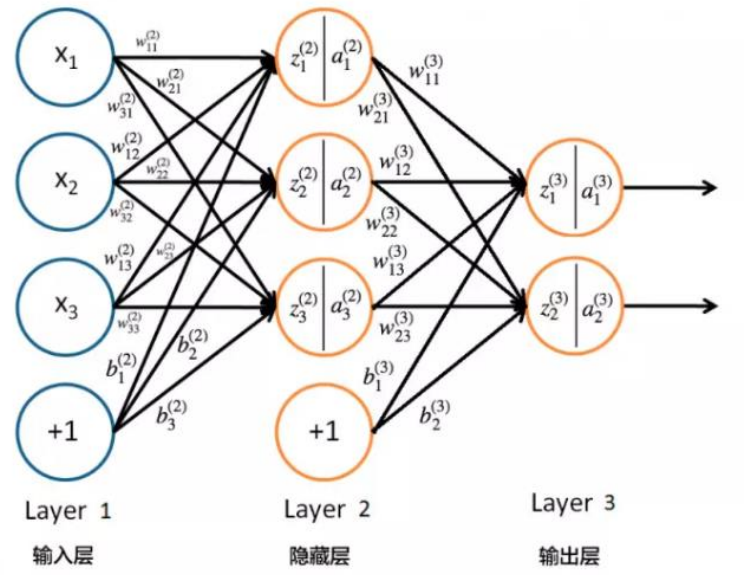


图 35-4 三层 BP 神经网络

神经网络的基本组成单元是神经元。神经元的通用模型如图 35-5 所示，其中常用的激活函数有阈值函数、Sigmoid 函数和双曲正切函数等。

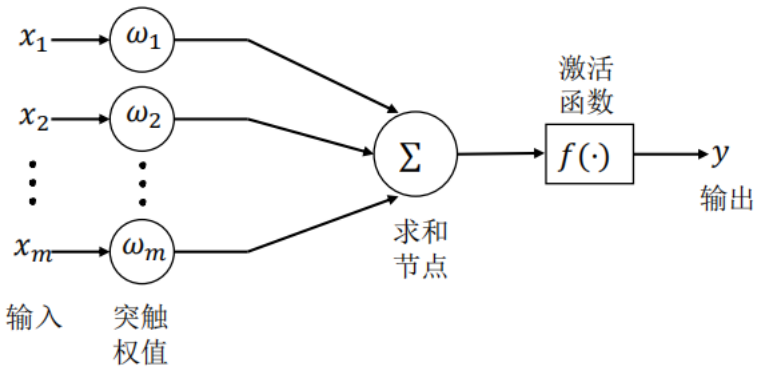


图 35-5 神经元模型

神经元的输出为：$y = f(\sum_{i=1}^{m} w_i x_i)$

3.卷积神经网络

卷积神经网络（Convolutional Neural Networks）是一类包含卷积计算且具有深度结构的前馈神经网络，是深度学习的代表算法之一。卷积神经网络的研究始于二十世纪 80 至 90 年代，时间延迟网络和 LeNet-5 是最早出现的卷积神经网络。在二十一世纪后，随着深度学习理论的提出和数值计算设备的改进，卷积神经网络得到了快速发展，并被大量应用于计算机视觉、自然语言处理等领域。

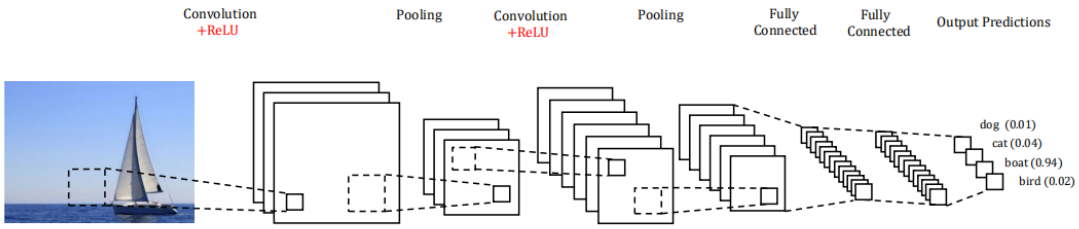


图 35-6 CNN 模型

图 35-6 是一个识别的 CNN 模型。最左边的图片是输入层二维矩阵，然后是卷积层，卷积层的激活函数使用 ReLU，即 $\mathrm{ReLU}(x) = \max(0, x)$。在卷积层之后是池化层，它和卷积层是 CNN 特有的，池化层中没有激活函数。卷积层和池化层的组合可以在隐藏层出现很多次，上图中循环出现了两次，而实际上这个次数是根据模型的需要而定。常见的 CNN 都是若干卷积层加池化层的组合，在若干卷积层和池化层后面是全连接层，最后输出层使用了 Softmax 激活函数来做图像识别的分类。

神经网络是由很多神经层组成，每一层神经层中存在很多神经元，这些神经元是识别事物的关键，当输入是图片时，其实就是一堆数字。卷积是指不在对每

个像素做处理，而是对图片区域进行处理，这种做法加强了图片的连续性，看到的是一个图形而不是一个点，也加深了神经网络对图片的理解。

下面结合 Google 推荐视频详细介绍 CNN 的原理知识。

假设你有一张小猫咪的照片，如下图所示，它可以被表示为一个薄饼，它有宽度(width)和高度(height)，并且由于天然存在红绿蓝三色，它还拥有 RGB 厚度（ depth ），此时你的输入深度为 3。

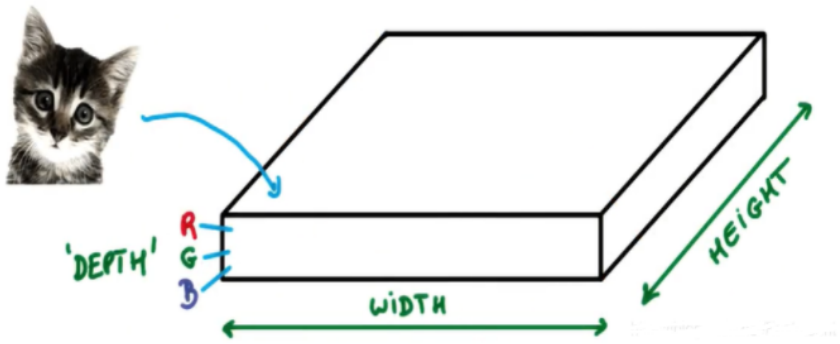


图 35-7 输入图像

假设我们现在拿出图片的一小块，运行一个具有 K 个输出的小神经网络，像图中一样把输出表示为垂直的一小列。

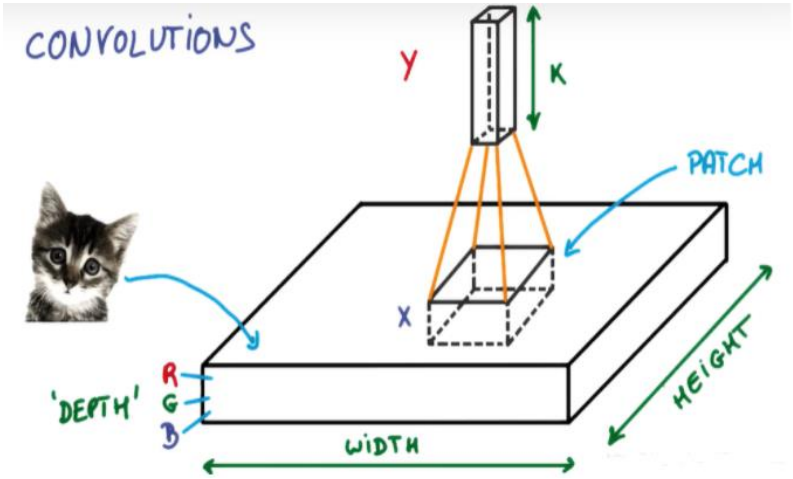


图 35-8 图像提取

在不改变权重的情况下，通过小神经网络滑动扫遍整个图片，就像我们拿着刷子刷墙一样水平垂直的滑动。

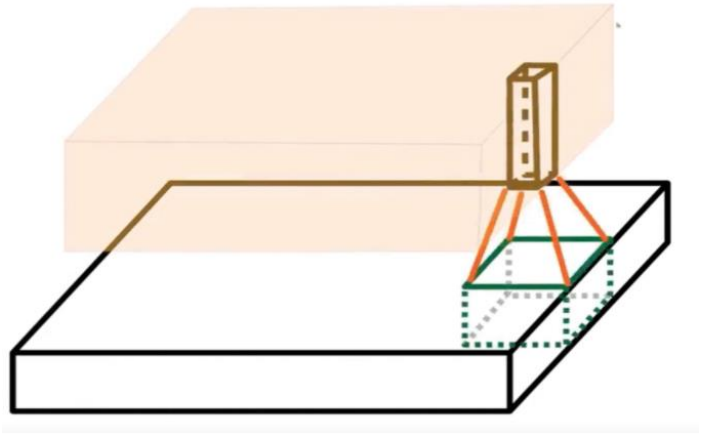


图 35-9 滑动窗口

此时，输出端画出了另一幅图像，如下图中红色区域所示。它与之前的宽度和高度不同，更重要的是它跟之前的深度不同，而不是仅仅只有红绿蓝，现在你得到了 K 个颜色通道，这种操作称为——**卷积**。

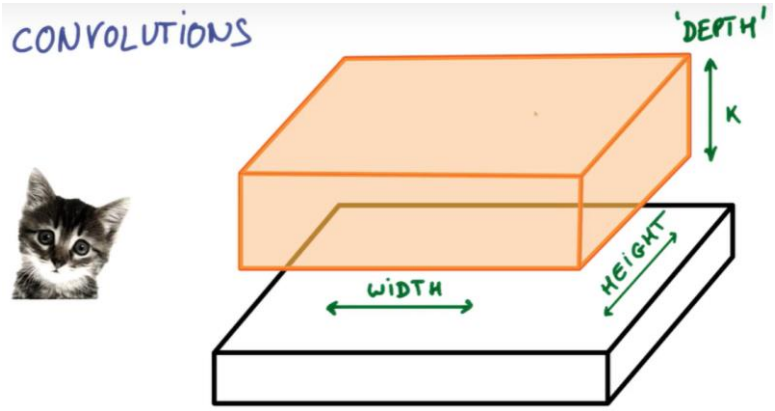


图 35-10 图像卷积处理

如果你的块大小是整张图片，那它跟普通的神经网络层没有任何区别，正是由于我们使用了小块，我们有很多小块在空间中共享较少的权重。卷积不在对每个像素做处理，而是对图片区域进行处理，这种做法加强了图片的连续性，也加

深了神经网络对图片的理解。

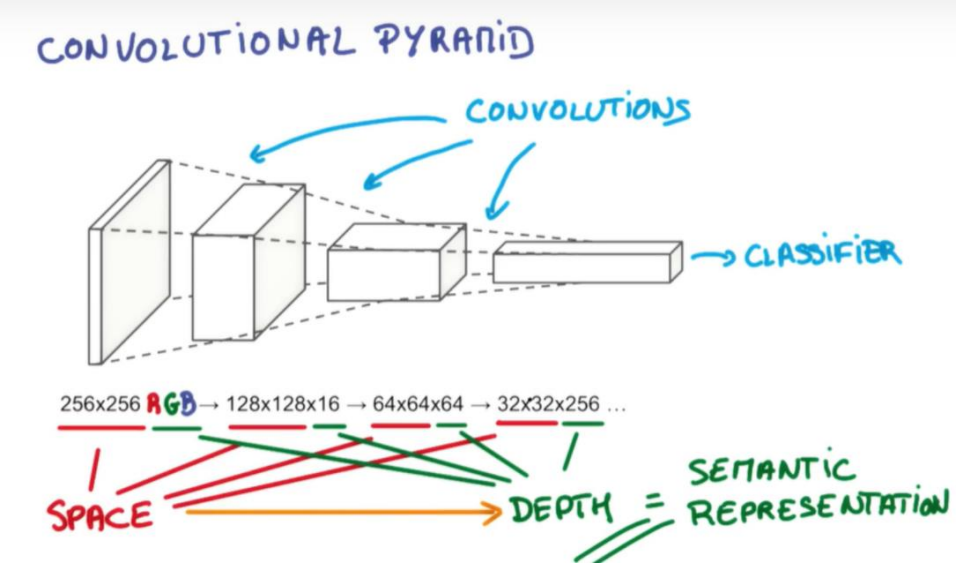


图 35-11 图像卷积整体流程

一个卷积网络是组成深度网络的基础，我们将使用数层卷积而不是数层的矩阵相乘。如下图所示，让它形成金字塔形状，金字塔底是一个非常大而浅的图片，仅包括红绿蓝，通过卷积操作逐渐挤压空间的维度，同时不断增加深度，使深度信息基本上可以表示出复杂的语义。同时，你可以在金字塔的顶端实现一个分类器，所有空间信息都被压缩成一个标识，只有把图片映射到不同类的信息保留，这就是 CNN 的总体思想。

上图的具体流程如下：

- 首先，这是有一张彩色图片，它包括 RGB 三原色分量，图像的长和宽为 256*256，三个层面分别对应红（R）、绿（G）、蓝（B）三个图层，也可以看作像素点的厚度。
- 其次，CNN 将图片的长度和宽度进行压缩，变成 12812816 的方块，压缩的方法是把图片的长度和宽度压小，从而增高厚度。

- 再次，继续压缩至 646464，直至 3232256，此时它变成了一个很厚的长条方块，我们这里称之为分类器 Classifier。该分类器能够将我们的分类结果进行预测，MNIST 手写体数据集预测结果是 10 个数字，比如[0,0,0,1,0,0,0,0,0,0]表示预测的结果是数字 3，Classifier 在这里就相当于这 10 个序列。
- 最后，CNN 通过不断压缩图片的长度和宽度，增加厚度，最终会变成了一个很厚的分类器，从而进行分类预测。

如果你想实现它，必须还要正确实现很多细节。此时，你已经接触到了块和深度的概念，块（PATCH）有时也叫做核（KERNEL），如下图所示，你堆栈的每个薄饼都被叫做特征图（Feature Map），这里把三个特性映射到 K 个特征图中，PATCH/KERNEL 的功能是从图片中抽离一小部分进行分析，每次抽离的小部分都会变成一个长度、一个宽度、K 个厚度的数列。

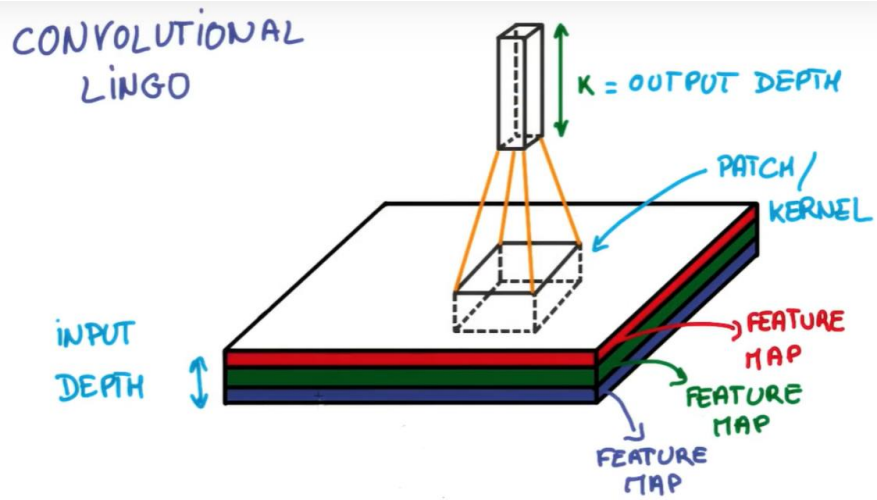


图 35-12 图像卷积处理

另一个你需要知道的概念是——步幅（STRIDE）。它是当你移动滤波器或

抽离时平移的像素的数量，每一次跨多少步去抽离图片中的像素点。

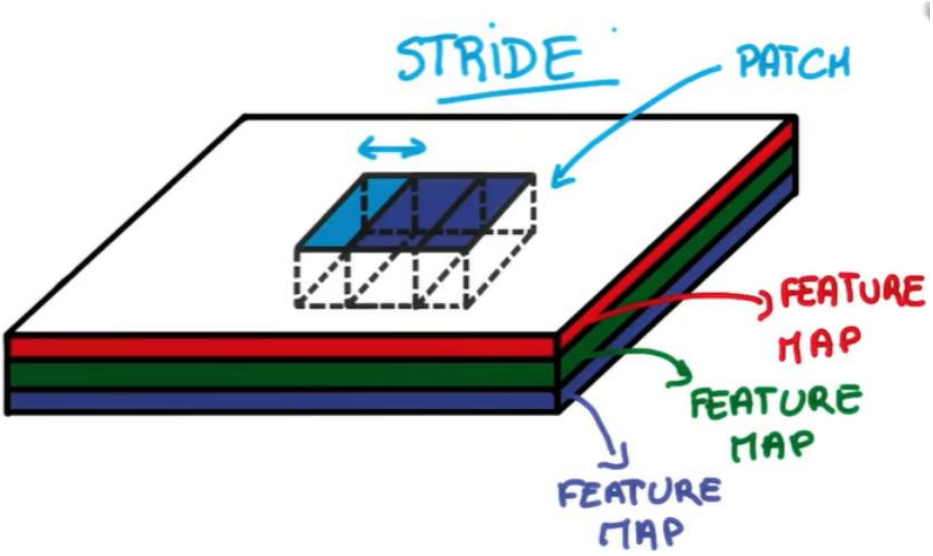


图 35-13 步幅处理

如果步幅 STRIDE 等于 1，表示每跨 1 个像素点抽离一次，得到的尺寸基本上和输入相同。

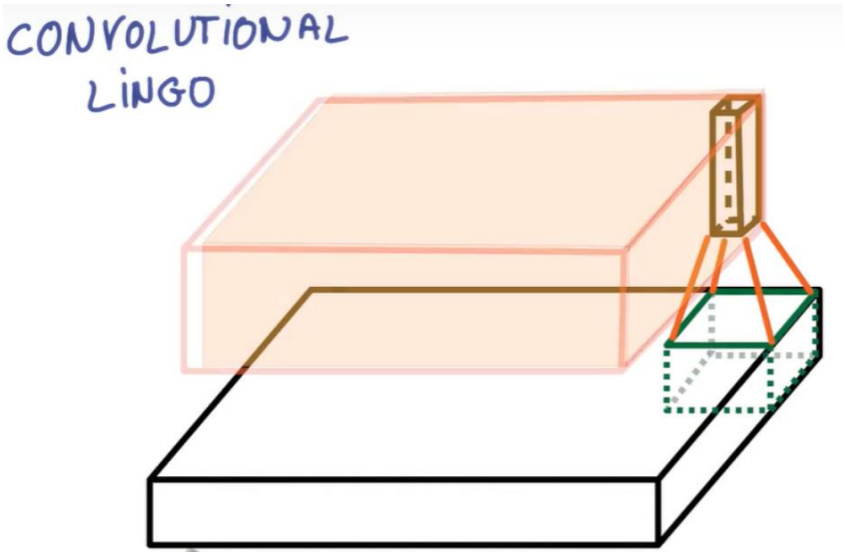


图 35-14 步幅为 1

如果步幅 STRIDE 等于 2，表示每次跨 2 个像素点抽离，意味着变为一半的尺寸。它收集到的信息就会被缩减，图片的长度和宽度被压缩了，压缩合并成

更小的一块立方体。

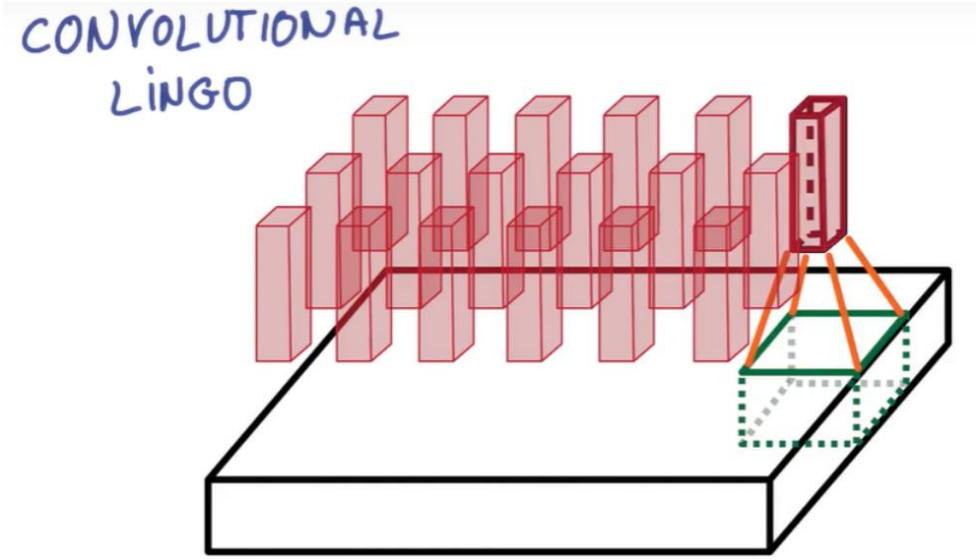


图 35-15 步幅为 2

压缩完之后再合并成一个立方体，它就是更小的一块立方体，包含了图片中的所有信息。

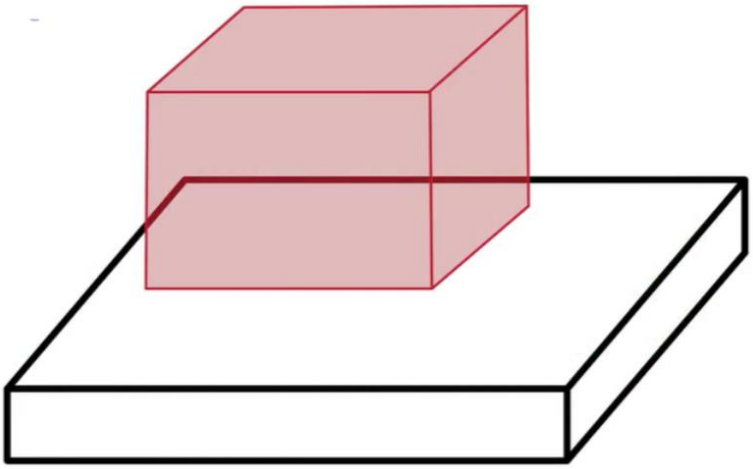


图 35-16 压缩图像

抽离图片信息的方式称为 PADDING（填充），一般分为两种：

- VALID PADDING：抽出来这层比原先那层图片宽和长裁剪了一点，抽取的内容全部是图片内的。

- SAME PADDING：抽离出的那层与之前的图片一样的长和宽，抽取的内容部分再图片外，图片外的值用 0 来填充。

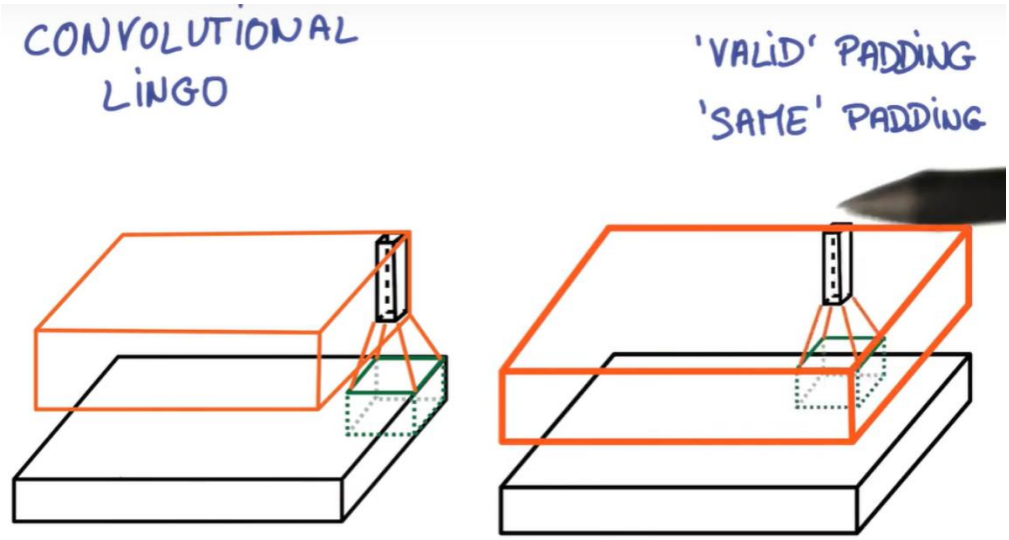


图 35-17 图像填充

研究发现，卷积过程会丢失一些信息，比如现在想跨 2 步去抽离原始图片的重要信息，形成长宽更小的图片，该过程中可能会丢失重要的图片信息。为了解决这个问题，通过 POOLING（持化）可以避免。其方法是：卷积时不再压缩长宽，尽量保证更多信息，压缩工作交给 POOLING。经过图片到卷积，持化处理卷积信息，再卷积再持化，将结果传入两层全连接神经层，最终通过分类器识别猫或狗。

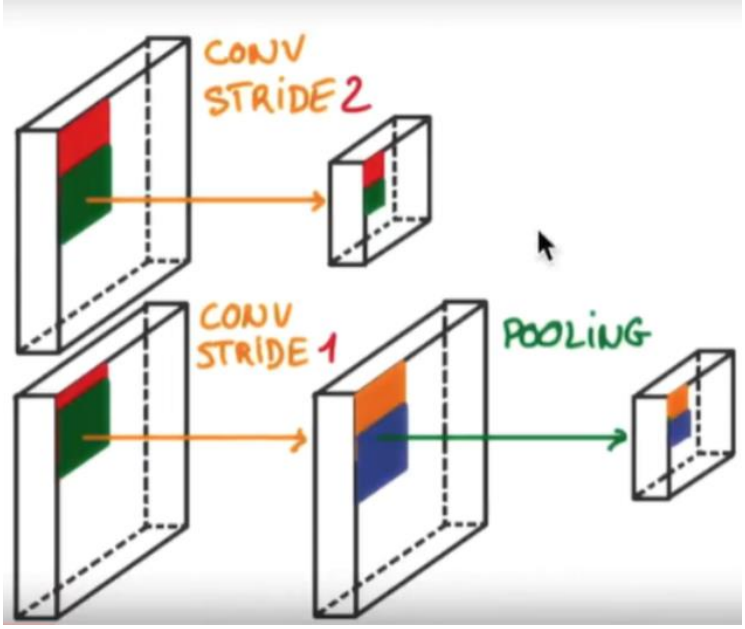


图 35-18 池化处理

总结：整个 CNN 从下往上依次经历“图片->卷积->持化->卷积->持化->结果传入两层全连接神经层->分类器”的过程，最终实现一个 CNN 的分类处理。

- IMAGE 图片
- CONVOLUTION 图层
- MAX POOLING 更好地保存原图片的信息
- CONVOLUTION 图层
- MAX POOLING 更好地保存原图片的信息
- FULLY CONNECTED 神经网络隐藏层
- FULLY CONNECTED 神经网络隐藏层
- CLASSIFIER 分类器

写到这里，CNN 的基本原理讲解完毕，希望大家对 CNN 有一个初步的理解。同时建议大家处理神经网络时，先用一般的神经网络去训练它，如果得到的结果非常好，就没必要去使用 CNN，因为 CNN 结构比较复杂。

4.MNIST 数据集

MNIST 是手写体识别数据集，它是非常经典的一个神经网络示例。MNIST 图片数据集包含了大量的数字手写体图片，如下图所示，我么可以尝试用它进行分类实验。

图 35-19 MNIST 数据集

MNIST 数据集是含标注信息的，上图分别表示数字 5、0、4 和 1。该数据集共包含三部分：

- 训练数据集：55,000 个样本，mnist.train
- 测试数据集：10,000 个样本，mnist.test
- 验证数据集：5,000 个样本，mnist.validation

通常，训练数据集用来训练模型，验证数据集用来检验所训练出来的模型的正确性和是否过拟合，测试集是不可见的（相当于一个黑盒），但我们最终的目的是使得所训练出来的模型在测试集上的效果（这里是准确性）达到最佳。

如图 35-20 所示，数据是以该形式被计算机所读取，比如 28*28=784 个

像素点，白色的地方都是 0，黑色的地方表示有数字的，总共有 55000 张图片。

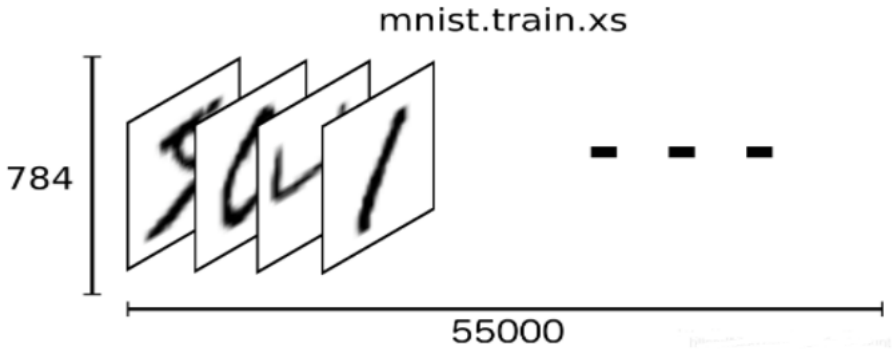


图 35-20 数据集介绍

MNIST 数据集中的一个样本数据包含两部分内容：手写体图片和对应的 label。这里我们用 xs 和 ys 分别代表图片和对应的 label，训练数据集和测试数据集都有 xs 和 ys，使用 mnist.train.images 和 mnist.train.labels 表示训练数据集中图片数据和对应的 label 数据。

如图 35-21 所示，它表示由 2828 的像素点矩阵组成的一张图片，这里的数字 784（2828）如果放在我们的神经网络中，它就是 x 输入的大小，其对应的矩阵如下图所示，类标 label 为 1。

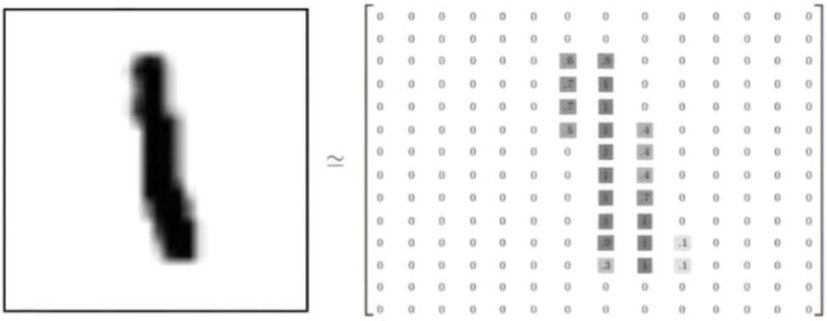


图 35-21 像素组成

最终MNIST的训练数据集形成了一个形状为55000*784位的tensor，也就是一个多维数组，第一维表示图片的索引，第二维表示图片中像素的索引

（tensor 中的像素值在 0 到 1 之间）。

这里的 y 值其实是一个矩阵，这个矩阵有 10 个位置，如果它是 1 的话，它在 1 的位置（第 2 个数字）上写 1，其他地方写 0；如果它是 2 的话，它在 2 的位置（第 3 个数字）上写 1，其他位置为 0。通过这种方式对不同位置的数字进行分类，例如用[0,0,0,1,0,0,0,0,0,0]来表示数字 3，如下图所示。

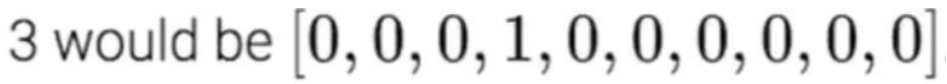


图 35-22 类标组成

mnist.train.labels 是一个 55000*10 的二维数组，如下图所示。它表示 55000 个数据点，第一个数据 y 表示 5，第二个数据 y 表示 0，第三个数据 y 表示 4，第四个数据 y 表示 1。

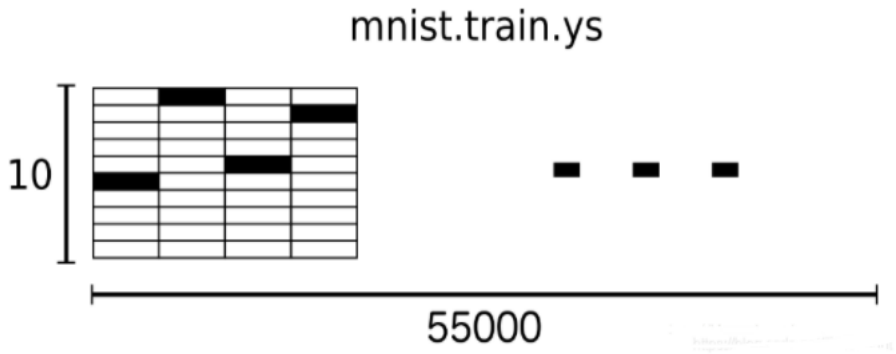


图 35-23 像素处理

知道 MNIST 数据集的组成，以及 x 和 y 具体的含义，我们就开始编写代码。

5.基于神经网络的图像分类

本文通过 Keras 搭建一个分类神经网络，再训练 MNIST 数据集。其中 X

表示图片，28*28，y 对应的是图像的标签。

第一步，导入扩展包。

```
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import RMSprop
```

第二步，载入 MNIST 数据及预处理。

该步骤的核心代码如下：

- X_train.reshape(X_train.shape[0], -1) / 255

将每个像素点进行标准化处理，从 0-255 转换成 0-1 的范围。

- np_utils.to_categorical(y_train, nb_classes=10)

调用 up_utils 将类标转换成 10 个长度的值，如果数字是 3，则会在对应的地方标记为 1，其他地方标记为 0，即{0,0,0,1,0,0,0,0,0,0}。

由于 MNIST 数据集是 Keras 或 TensorFlow 的示例数据，所以我们只需要下面一行代码，即可实现数据集的读取工作。如果数据集不存在它会在线下载，如果数据集已经被下载，它会被直接调用。

```
# 下载 MNIST 数据
# X shape(60000, 28*28) y shape(10000, )
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()


# 数据预处理
X_train = X_train.reshape(X_train.shape[0], -1) / 255   # normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255      # normalize


# 将类向量转化为类矩阵  数字 5 转换为 0 0 0 0 0 1 0 0 0 0 矩阵
y_train = np_utils.to_categorical(y_train, num_classes=10)
y_test = np_utils.to_categorical(y_test, num_classes=10)
```

第三步，创建神经网络层。

前面介绍创建神经网络层的方法是定义之后，利用 add()添加神经层。

- model = Sequential()
- model.add(Dense(output_dim=1, input_dim=1))

而这里采用另一种方法，在Sequential()定义的时候通过列表添加神经层。同时需要注意，这里增加了神经网络激励函数并调用 RMSprop 加速神经网络。

- from keras.layers import Dense, Activation
- from keras.optimizers import RMSprop

该神经网络层为：

- 第一层为 Dense(32, input_dim=784)，它将传入的 784 转换成 32 个输出
- 该数据加载一个激励函数 Activation（‘relu’），并转换成非线性化数据
- 第二层为 Dense(10)，它输出为 10 个单位。同时 Keras 定义神经层会默认其输入为上一层的输出，即 32（省略）
- 接着加载一个激励函数 Activation（‘softmax’)用于分类

对应代码如下：

```
# Another way to build your neural net
model = Sequential([
        Dense(32, input_dim=784),  # 输入值 784(28*28) => 输出值 32
        Activation('relu'),        # 激励函数 转换成非线性数据
        Dense(10),                 # 输出为 10 个单位的结果
        Activation('softmax')      # 激励函数 调用 softmax 进行分类
        ])

# Another way to define your optimizer
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08,
```

```
decay=0.0) #学习率 lr


# We add metrics to get more results you want to see
# 激活神经网络
model.compile(
        optimizer = rmsprop,                    # 加速神经网络
        loss = 'categorical_crossentropy',     # 损失函数
        metrics = ['accuracy'],                 # 计算误差或准
确率
        )
```

第四步，神经网络训练及预测。

```
print("Training")
model.fit(X_train, y_train, nb_epoch=2, batch_size=32)
# 训练次数及每批训练大小


print("Testing")
loss, accuracy = model.evaluate(X_test, y_test)


print("loss:", loss)
print("accuracy:", accuracy)
```

最终的完整代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 14 16:43:21 2020
@author: Eastmount YXZ
"""
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import RMSprop

#--------------------------载入数据及预处理-----
----------------------
# 下载 MNIST 数据
# X shape(60000, 28*28) y shape(10000, )
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 数据预处理
X_train = X_train.reshape(X_train.shape[0], -1) / 255   #
```

```
normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255       #
normalize

# 将类向量转化为类矩阵  数字 5 转换为 0 0 0 0 0 1 0 0 0 0 矩阵
y_train          =          np_utils.to_categorical(y_train,
num_classes=10)
y_test = np_utils.to_categorical(y_test, num_classes=10)

#--------------------------创建神经网络层-------------------
--------------------
# Another way to build your neural net
model = Sequential([
        Dense(32, input_dim=784),  # 输入值 784(28*28)
=> 输出值 32
        Activation('relu'),        # 激励函数 转换成非线性数据
        Dense(10),                 # 输出为 10 个单位的结果
        Activation('softmax')      # 激励函数 调用 softmax
进行分类
        ])
```

```
# Another way to define your optimizer
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08,
decay=0.0) #学习率 lr


# We add metrics to get more results you want to see
# 激活神经网络
model.compile(
        optimizer = rmsprop,                    # 加速神经网络
        loss = 'categorical_crossentropy',    # 损失函数
        metrics = ['accuracy'],                 # 计算误差或准
确率
        )


#------------------------------训练及预测-------
-----------------------
print("Training")
model.fit(X_train, y_train, nb_epoch=2, batch_size=32)
# 训练次数及每批训练大小
print("Testing")
loss, accuracy = model.evaluate(X_test, y_test)
```

```
print("loss:", loss)
print("accuracy:", accuracy)
```

运行代码，首先会下载 MNIT 数据集。

```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434
[==============================] - 18s 2us/step
```

接着输出两次训练的结果，可以看到误差不断减小、正确率不断增大。最终测试输出的误差 loss 为“0.185575”，正确率为“0.94690”。

```
Epoch 1/2
60000/60000 [==============================] - 4s 74us/step - loss: 0.3599 - accuracy: 0.9002
Epoch 2/2
60000/60000 [==============================] - 4s 69us/step - loss: 0.2071 - accuracy: 0.9413
Testing
10000/10000 [==============================] - 0s 38us/step
loss: 0.18557512020245195
accuracy: 0.9469000101089478
```

图 35-24 预测过程

如果读者想更直观地查看我们数字分类的图形，可以定义函数并显示。

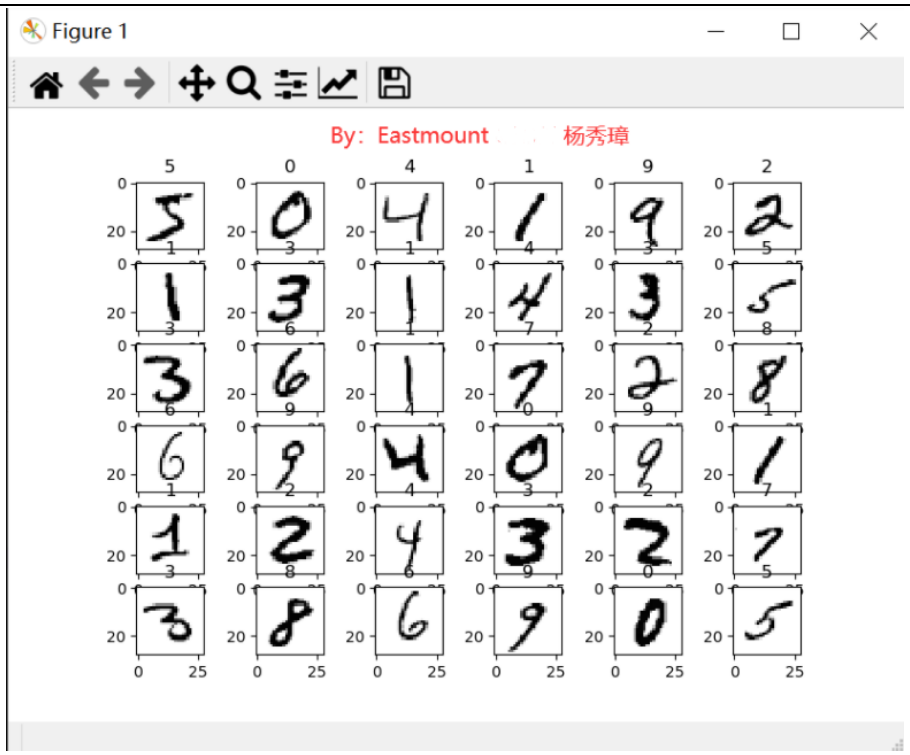


图 35-25 数字预测结果

此时的完整代码如下所示：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 14 16:43:21 2020
@author: Eastmount YXZ
"""
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation
```

```
from keras.optimizers import RMSprop
import matplotlib.pyplot as plt
from PIL import Image


#------------------------------载入数据及预处理------
----------------------
# 下载 MNIST 数据
# X shape(60000, 28*28) y shape(10000, )
(X_train, y_train), (X_test, y_test) = mnist.load_data()


#------------------------------显示图片--------
----------------------
def show_mnist(train_image, train_labels):
    n = 6
    m = 6
    fig = plt.figure()
    for i in range(n):
        for j in range(m):
            plt.subplot(n,m,i*n+j+1)
            index = i * n + j #当前图片的标号
            img_array = train_image[index]
```

```
            img = Image.fromarray(img_array)
            plt.title(train_labels[index])
            plt.imshow(img, cmap='Greys')
    plt.show()

show_mnist(X_train, y_train)

# 数据预处理
X_train = X_train.reshape(X_train.shape[0], -1) / 255   # normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255      # normalize

# 将类向量转化为类矩阵  数字 5 转换为 0 0 0 0 0 1 0 0 0 0 矩阵
y_train = np_utils.to_categorical(y_train, num_classes=10)
y_test = np_utils.to_categorical(y_test, num_classes=10)

#--------------------------创建神经网络层-------------------
# Another way to build your neural net
```

```
model = Sequential([
        Dense(32, input_dim=784),   # 输入值 784(28*28) => 输出值 32
        Activation('relu'),         # 激励函数 转换成非线性数据
        Dense(10),                  # 输出为 10 个单位的结果
        Activation('softmax')       # 激励函数 调用 softmax 进行分类
        ])

# Another way to define your optimizer
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0) #学习率 lr

# We add metrics to get more results you want to see
# 激活神经网络
model.compile(
        optimizer = rmsprop,                    # 加速神经网络
        loss = 'categorical_crossentropy',      # 损失函数
        metrics = ['accuracy'],                 # 计算误差或准确率
        )
```

```
#------------------------------训练及预测-------
-----------------------
print("Training")
model.fit(X_train, y_train, nb_epoch=2, batch_size=32)
# 训练次数及每批训练大小
print("Testing")
loss, accuracy = model.evaluate(X_test, y_test)

print("loss:", loss)
print("accuracy:", accuracy)
```

6.总结

写到这里，这篇文章就结束了。本文主要通过 Keras 实现了一个分类学习的案例，并详细介绍了 MNIST 手写体识别数据集。最后，希望这篇基础性文章对您有所帮助，如果文章中存在错误或不足之处，还请海涵。

参考文献：

[1] 冈萨雷斯著. 数字图像处理（第 3 版）[M]. 北京：电子工业出版社，2013.

[2] 杨秀璋，颜娜. Python 网络数据爬取及分析从入门到精通（分析篇）[M]. 北京：北京航天航空大学出版社, 2018.

[3] 网易云莫烦老师视频：

https://study.163.com/course/courseLearn.htm?courseId=1003209007

[4] 斯坦福机器学习视频 NG 教授： https://class.coursera.org/ml/class/index

[5] 机器学习实战—MNIST 手写体数字识别 - RunningSucks

[6] https://github.com/siucaan/CNN_MNIST

第 36 篇 图像特效之毛玻璃、浮雕、油漆和模糊特效变换

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面的文章围绕 Python 图像处理、图像运算和图像增强，从图像基础操作、几何变换、点运算、直方图、图像增强、图像平滑、图像锐化等方面进行讲解。接下来将详细讲解常见的图像特效处理，从而让读者实现各种各样的图像特殊效果，它们类似于 PS 或美图秀秀软件的特效处理，能有效地帮助我们理解这些特效背后的故事。第一篇文章将介绍图像毛玻璃、浮雕、油漆和模糊特效变换。

1.图像毛玻璃特效变换

图像毛玻璃特效如图 36-1 所示，左边为原始图像，右边为毛玻璃特效图像。

- **实现过程：**该特效是用图像邻域内随机一个像素点的颜色来替代当前像素点颜色的过程，从而为图像增加一个毛玻璃模糊的特效[1]。

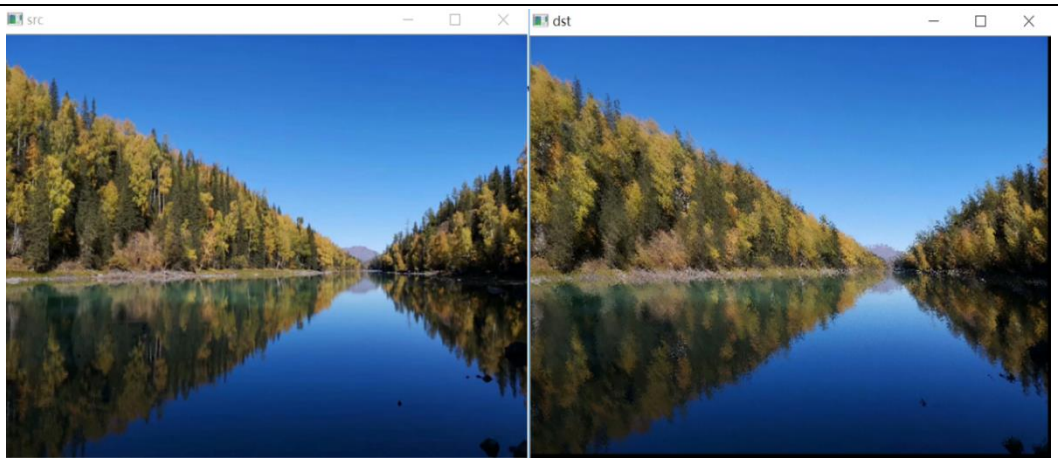


图 36-1 图像毛玻璃特效处理

Python 实现代码主要是通过双层循环遍历图像的各像素点，再用定义的随机数去替换各邻域像素点的颜色，具体代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np

#读取原始图像
src = cv2.imread('luo.png')

#新建目标图像
dst = np.zeros_like(src)

#获取图像行和列
```

```
rows, cols = src.shape[:2]


#定义偏移量和随机数
offsets = 5
random_num = 0


#毛玻璃效果：像素点邻域内随机像素点的颜色替代当前像素点的颜色
for y in range(rows - offsets):
    for x in range(cols - offsets):
        random_num = np.random.randint(0,offsets)
        dst[y,x] = src[y + random_num,x + random_num]


#显示图像
cv2.imshow('src',src)
cv2.imshow('dst',dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

图 36-2 显示了小珞珞图像的毛玻璃效果。

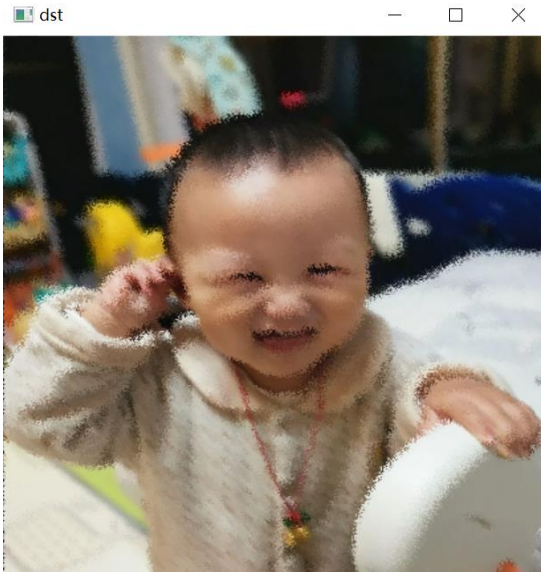


图 36-2 小珞珞的毛玻璃特效

2.图像浮雕特效变换

图像浮雕特效是仿造浮雕艺术而衍生的处理，它将要呈现的图像突起于石头表面，根据凹凸程度不同形成三维的立体效果。

- **实现过程：**Python 绘制浮雕图像是通过勾画图像的轮廓，并降低周围的像素值，从而产生一张具有立体感的浮雕效果图。传统的方法是设置卷积核，再调用 OpenCV 的 filter2D()函数实现浮雕特效。

该函数主要是利用内核实现对图像的卷积运算，其函数原型如下所示[1]：

```
dst = filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])
```

- src 表示输入图像
- dst 表示输出的边缘图，其大小和通道数与输入图像相同

- ddepth 表示目标图像所需的深度
- kernel 表示卷积核，一个单通道浮点型矩阵
- anchor 表示内核的基准点，其默认值为（-1，-1），位于中心位置
- delta 表示在储存目标图像前可选的添加到像素的值，默认值为 0
- borderType 表示边框模式

核心代码如下：

```
kernel = np.array([[-1,0,0],[0,1,0],[0,0,0]])
output = cv2.filter2D(src, -1, kernel)
```

本小节将直接对各像素点进行处理，采用相邻像素相减的方法来得到图像轮廓与平面的差，类似边缘的特征，从而获得这种立体感的效果。为了增强图片的主观感受，还可以给这个差加上一个固定值，如 150。实现效果如图 36-3 所示。

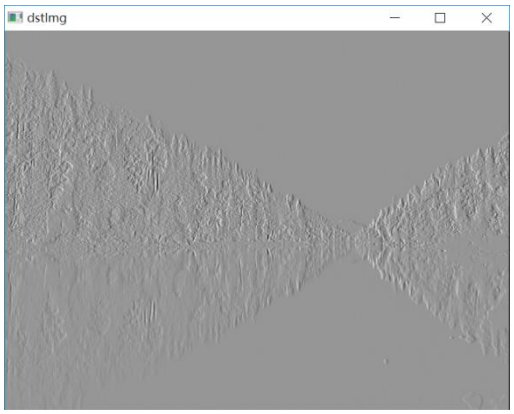


图 36-3 图像浮雕效果

Python 通过双层循环遍历图像的各像素点，使用相邻像素值之差来表示当前像素值，从而得到图像的边缘特征，最后加上固定数值 100 得到浮雕效果。

❖ newPixel = grayCurrentPixel - grayNextPixel + 100

具体代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np

#读取原始图像
img = cv2.imread('luo.png', 1)

#获取图像的高度和宽度
height, width = img.shape[:2]

#图像灰度处理
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#创建目标图像
dstImg = np.zeros((height,width,1),np.uint8)

#浮雕特效算法: newPixel = grayCurrentPixel - grayNextPixel
+ 100
for i in range(0,height):
```

```
    for j in range(0,width-1):
        grayCurrentPixel = int(gray[i,j])
        grayNextPixel = int(gray[i,j+1])
        newPixel = grayCurrentPixel - grayNextPixel + 100
        if newPixel > 255:
            newPixel = 255
        if newPixel < 0:
            newPixel = 0
        dstImg[i,j] = newPixel

#显示图像
cv2.imshow('src', img)
cv2.imshow('dst',dstImg)
cv2.waitKey()
cv2.destroyAllWindows()
```

图 36-4 显示了小珞珞图像的浮雕效果。

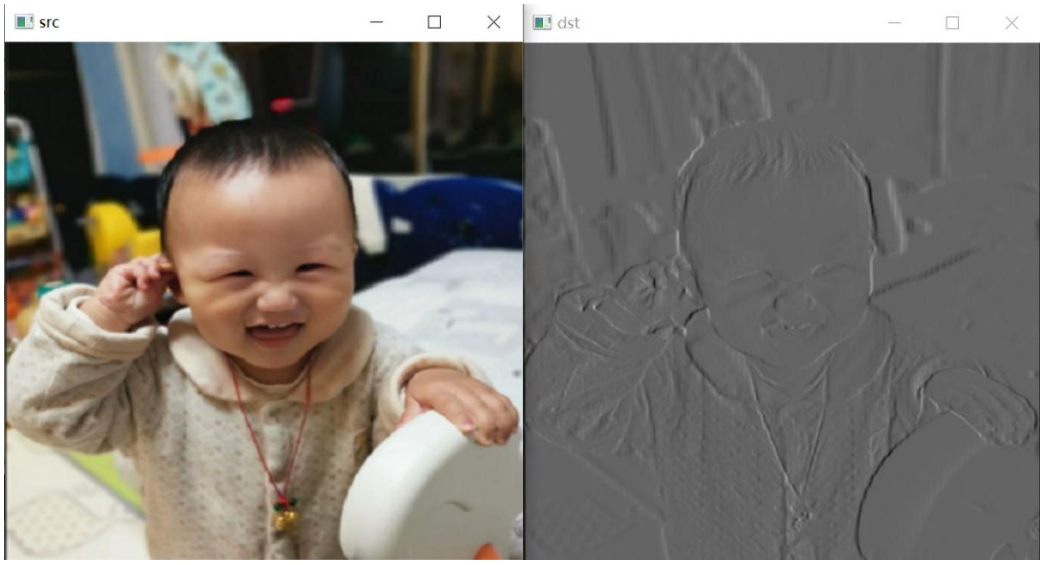


图 36-4 小珞珞的浮雕特效

3.图像油漆特效变换

图像油漆特效类似于油漆染色后的轮廓图形，其运行结果如图 36-5 所示。

❖ **实现过程：**主要采用自定义卷积核和 cv2.filter2D()函数实现。

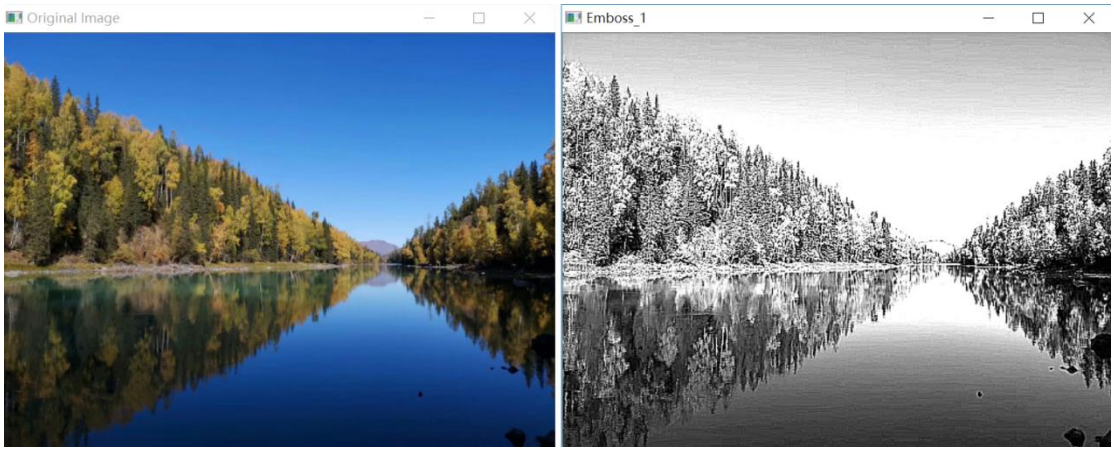


图 36-5 图像油漆特效处理

Python 实现代码主要通过 Numpy 定义卷积核，再进行特效处理，卷积核如公式（36-1）所示，其中心权重为 10，其余值均为-1。

$$K(3,3)=\begin{bmatrix}-1 & -1 & -1\\ -1 & 10 & -1\\ -1 & -1 & -1\end{bmatrix} \quad (36-1)$$

具体实现代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np

#读取原始图像
src = cv2.imread('luo.png')

#图像灰度处理
gray = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)

#自定义卷积核
kernel = np.array([[-1,-1,-1],[-1,10,-1],[-1,-1,-1]])

#图像浮雕效果
output = cv2.filter2D(gray, -1, kernel)
```

```
#显示图像
cv2.imshow('Original Image', src)
cv2.imshow('Emboss_1',output)
cv2.waitKey()
cv2.destroyAllWindows()
```

图 36-6 显示了小珞珞图像的油漆效果。

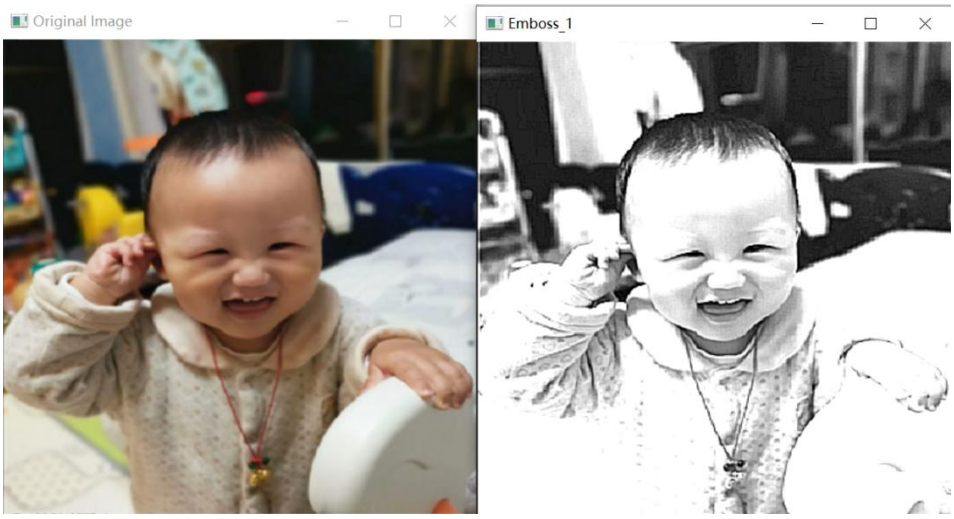


图 36-6 小珞珞的油漆特效

4.图像模糊特效变换

图像模糊特效可以通过第 23 篇和第 24 篇文章讲述的图像平滑方法实现，包括均值滤波、方框滤波、高斯滤波、中值滤波和双边滤波等，它能消除图像消除图像的噪声并保留图像的边缘轮廓。该部分主要采用高斯滤波进行模糊操作，如图 36-7 所示，左边为风景原始图像，右边为高斯滤波处理后的图像，它有效地将图像进行了模糊处理。

图 36-7 高斯滤波 11×11 核的处理

❖ **实现过程：**图像高斯滤波为图像不同位置的像素值赋予了不同的权重，距离越近的点权重越大，距离越远的点权重越小。下面是常用的 3×3 和 5×5 内核的高斯滤波模板。

$$K(3,3)=\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \tag{36-2}$$

$$K(5,5)=\frac{1}{273}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix} \tag{36-3}$$

Python 中 OpenCV 主要调用 GaussianBlur()函数实现高斯平滑处理，下面代码是使用 11×11 核模板进行高斯滤波处理。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
#读取图片
img = cv2.imread('luo.png')
source = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#高斯滤波
result = cv2.GaussianBlur(source, (11,11), 0)

#显示图形
plt.rcParams['font.sans-serif']=['SimHei']
titles = ['原始图像', '高斯滤波']
images = [source, result]
for i in range(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

图 36-8 显示了小珞珞图像的高斯滤波模糊效果。

图 36-6 小珞珞的模糊特效

5.总结

本文主要讲解了图像常见的特效处理，从处理效果图、算法原理、代码实现三个步骤进行详细讲解，涉及图像毛玻璃特效、浮雕特效、油漆特效和模糊特效等，这些知识点将为读者从事 Python 图像处理相关项目实践或科学研究提供一定基础。

参考文献：

[1] Eastmount. [Android] 通过 Menu 实现图片怀旧、浮雕、模糊、光照和素描效果 [EB/OL]. (2014-11-02). https://blog.csdn.net/Eastmount/article/details/40711317.

[2] Eastmount. [Python 图像处理] 三十三.图像各种特效处理及原理万字详解(毛玻璃、浮雕、素描、怀旧、流年、滤镜等) [EB/OL]. (2020-12-22).

https://blog.csdn.net/Eastmount/article/ details/111568397.

第 37 篇 图像特效之素描和卡通特效变换

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面的文章围绕 Python 图像处理、图像运算和图像增强，从图像基础操作、几何变换、点运算、直方图、图像增强、图像平滑、图像锐化等方面进行讲解。接下来将详细讲解常见的图像特效处理，从而让读者实现各种各样的图像特殊效果，它们类似于 PS 或美图秀秀软件的特效处理，能有效地帮助我们理解这些特效背后的故事。第一篇文章将介绍图像素描和卡通特效变换。

1.图像素描特效变换

图像素描特效会将图像的边界都凸显出来，通过边缘检测及阈值化处理能实现该功能。一幅图像的内部都具有相似性，而在图像边界处具有明显的差异，边缘检测利用数学中的求导来扩大这种变化。但是求导过程中会增大图像的噪声，所以边缘检测之前引入了高斯滤波降噪处理。本小节的图像素描特效主要经过以下几个步骤：

1) 调用 cv2.cvtColor()函数将彩色图像灰度化处理；

2) 通过 cv2.GaussianBlur()函数实现高斯滤波降噪；

3) 边缘检测采用 Canny 算子实现；

4) 最后通过 cv2.threshold()反二进制阈值化处理实现素描特效。

图 37-1 显示了风景图的素描特效结果。

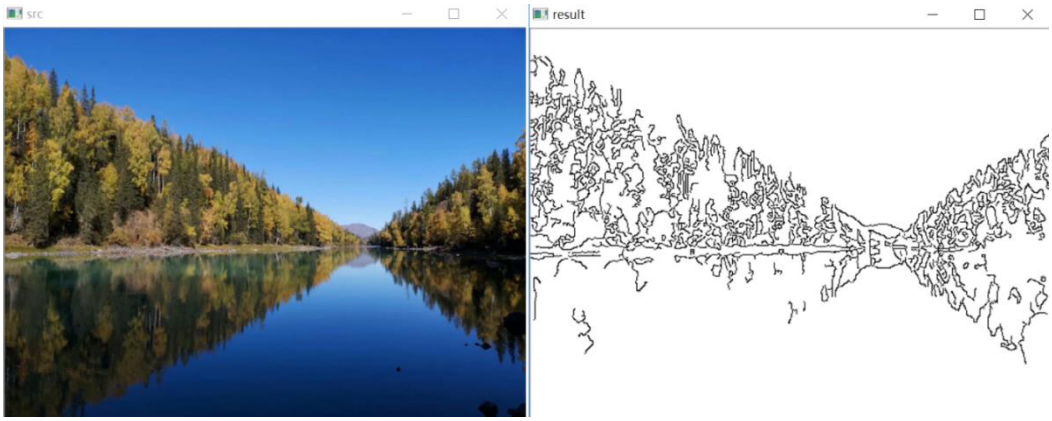


图 37-1 图像素描特效处理

完整运行代码如下所示。

```
# -*- coding: utf-8 -*-

# By: Eastmount

import cv2

import numpy as np


#读取原始图像

img = cv2.imread('luo.png')


#图像灰度处理

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
#高斯滤波降噪
gaussian = cv2.GaussianBlur(gray, (5,5), 0)

#Canny 算子
canny = cv2.Canny(gaussian, 50, 150)

#阈值化处理
ret, result = cv2.threshold(canny, 100, 255,
cv2.THRESH_BINARY_INV)

#显示图像
cv2.imshow('src', img)
cv2.imshow('result', result)
cv2.waitKey()
cv2.destroyAllWindows()
```

最终输出结果如图 37-2 所示，它将彩色图像素描处理。

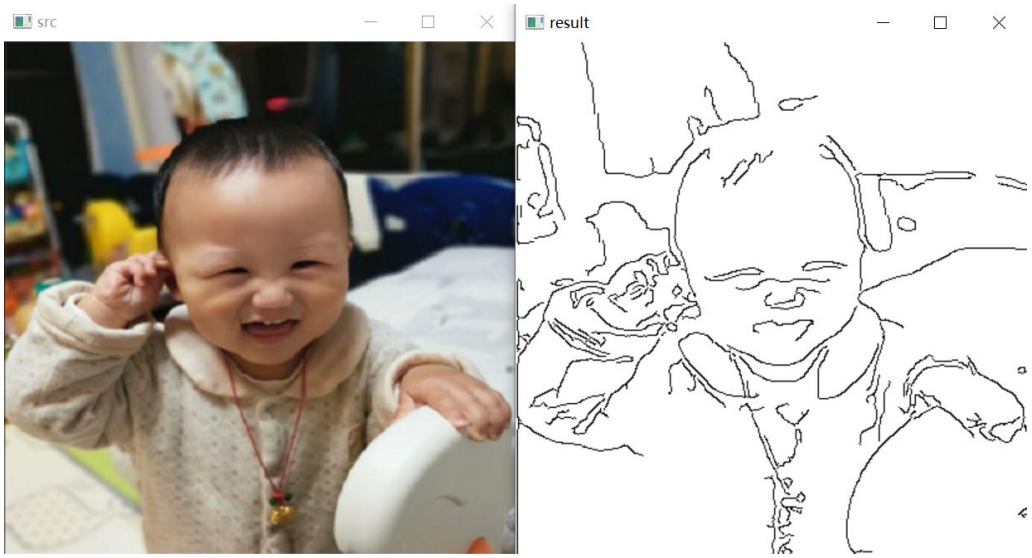


图 37-2 图像素描特效处理

图像的素描特效有很多种方法，本小节仅提供了一种方法，主要提取的是图像的边缘轮廓，还有很多更精细的素描特效方法，提取的轮廓更为清晰，如图 37-3 所示。这是小何老师分享的代码[3]。

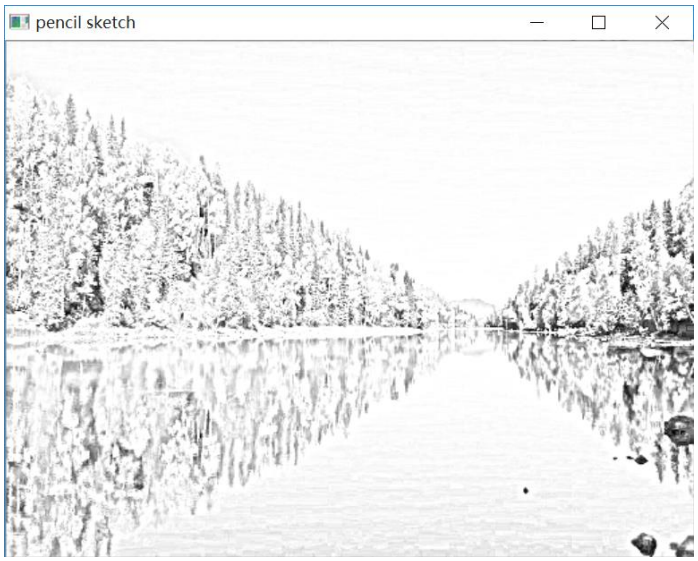


图 37-3 图像细节素描特效

该素描代码如下所示。

```
#coding:utf-8
```

```
#                                                    参
考 :https://blog.csdn.net/weixin_39059031/article/details/8
2724951
import cv2
import numpy as np


def dodgeNaive(image, mask):
    # determine the shape of the input image
    width, height = image.shape[:2]


    # prepare output argument with same size as image
    blend = np.zeros((width, height), np.uint8)


    for col in range(width):
        for row in range(height):
            # do for every pixel
            if mask[col, row] == 255:
                # avoid division by zero
                blend[col, row] = 255
            else:
                # shift image pixel value by 8 bits
```

```
                # divide by the inverse of the mask
                tmp = (image[col, row] << 8) / (255 - mask)
                # print('tmp={}'.format(tmp.shape))
                # make sure resulting value stays within
bounds
                if tmp.any() > 255:
                    tmp = 255
                    blend[col, row] = tmp
    return blend


def dodgeV2(image, mask):
    return cv2.divide(image, 255 - mask, scale=256)


def burnV2(image, mask):
    return 255 - cv2.divide(255 - image, 255 - mask,
scale=256)


def rgb_to_sketch(src_image_name, dst_image_name):
    img_rgb = cv2.imread(src_image_name)
    img_gray = cv2.cvtColor(img_rgb,
cv2.COLOR_BGR2GRAY)
```

```
    # 读取图片时直接转换操作
    # img_gray = cv2.imread('example.jpg',
cv2.IMREAD_GRAYSCALE)

    img_gray_inv = 255 - img_gray
    img_blur = cv2.GaussianBlur(img_gray_inv, ksize=(21,
21),
                                sigmaX=0, sigmaY=0)
    img_blend = dodgeV2(img_gray, img_blur)

    cv2.imshow('original', img_rgb)
    cv2.imshow('gray', img_gray)
    cv2.imshow('gray_inv', img_gray_inv)
    cv2.imshow('gray_blur', img_blur)
    cv2.imshow("pencil sketch", img_blend)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite(dst_image_name, img_blend)

if __name__ == '__main__':
    src_image_name = 'luo.png'
```

```
dst_image_name = 'sketch_example.png'
rgb_to_sketch(src_image_name, dst_image_name)
```

小珞珞图像的另一种素描效果显示如图 37-4 所示，其细节更加清晰。

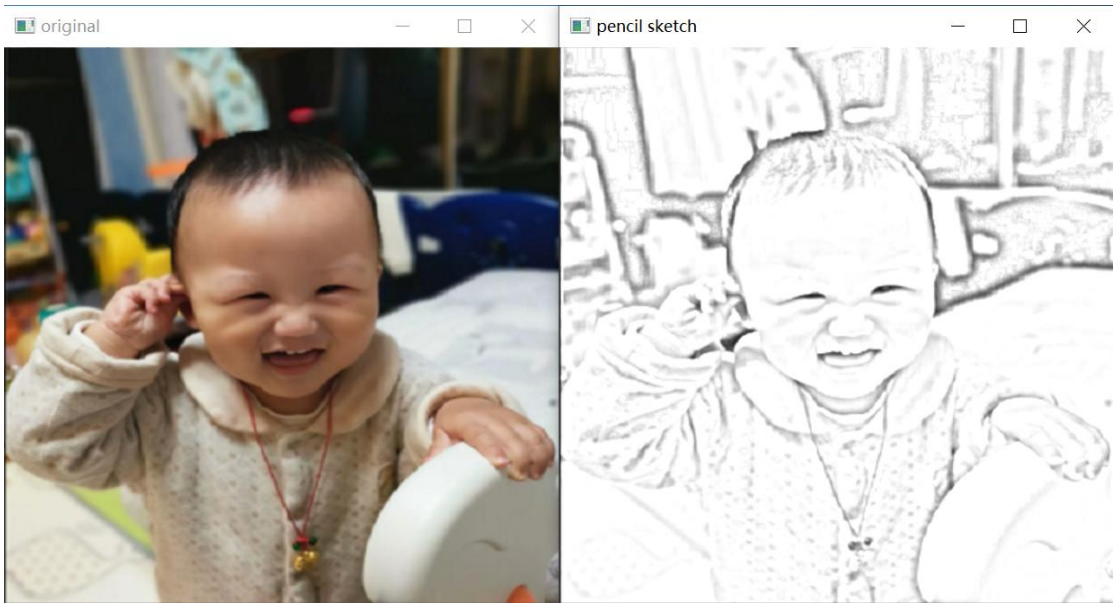


图 37-4 小珞珞图像的细节素描特效

2.图像卡通特效变换

图像卡通特效是将原始图像转换为具有卡通特色的效果图，本小节的算法主要包括以下几个步骤：

第一步，调用 cv2.bilateralFilter()函数对原始图像进行双边滤波处理。该滤波器可以在保证边界清晰的情况下有效的去掉噪声，将像素值缩短为每 7 个灰度级为一个值。它同时使用了空间高斯权重和灰度相似性高斯权重，确保边界不会被模糊掉。

第二步，调用 cv2.cvtColor()函数将原始图像转换为灰度图像，并进行中

值滤波处理，接着调用 cv2.adaptiveThreshold()函数进行自适应阈值化处理，并提取图像的边缘轮廓，将图像转换回彩色图像。此时显示的效果如图 37-5 所示。

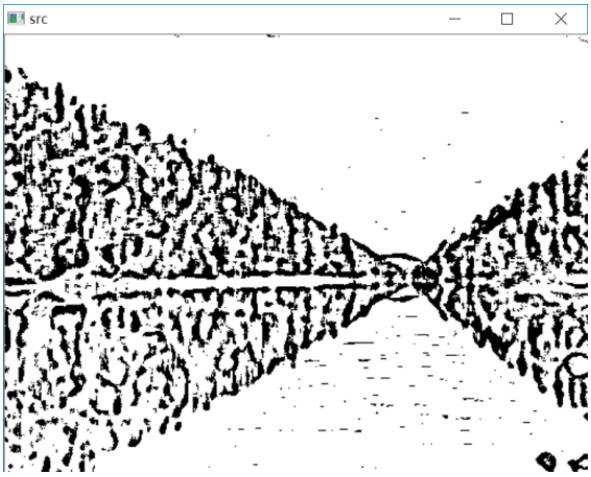


图 37-5 图像边缘轮廓

最后，调用 cv2.bitwise_and()函数将第 1 步和第 2 步产生的图像进行与运算，产生最终的卡通图像，如图 37-6 所示。

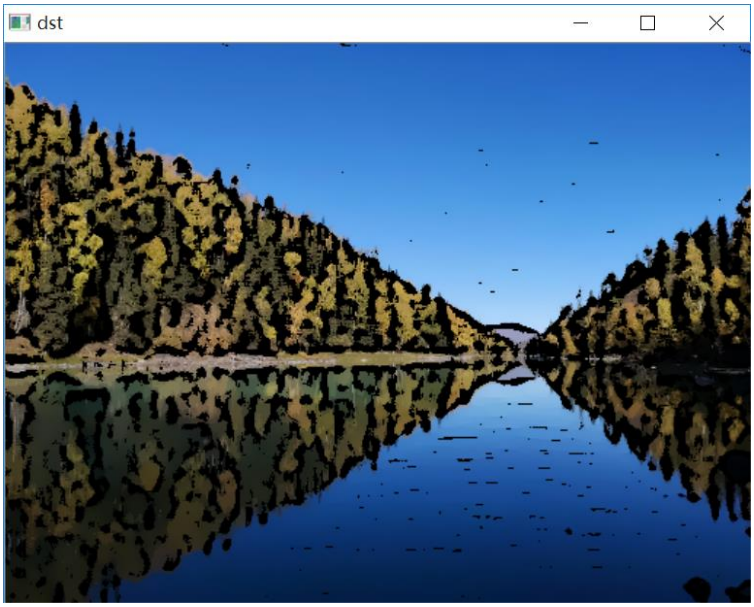


图 37-6 图像卡通特效

Python 实现代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np

#读取原始图像
img = cv2.imread('luo.png')

#定义双边滤波的数目
num_bilateral = 7

#用高斯金字塔降低取样
img_color = img

#双边滤波处理
for i in range(num_bilateral):
    img_color = cv2.bilateralFilter(img_color, d=9,
sigmaColor=9, sigmaSpace=7)

#灰度图像转换
```

```
img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

#中值滤波处理
img_blur = cv2.medianBlur(img_gray, 7)

#边缘检测及自适应阈值化处理
img_edge = cv2.adaptiveThreshold(img_blur, 255,

cv2.ADAPTIVE_THRESH_MEAN_C,
                                 cv2.THRESH_BINARY,
                                 blockSize=9,
                                 C=2)

#转换回彩色图像
img_edge = cv2.cvtColor(img_edge,
cv2.COLOR_GRAY2RGB)

#与运算
img_cartoon = cv2.bitwise_and(img_color, img_edge)

#显示图像
```

```
cv2.imshow('src', img)
cv2.imshow('dst', img_cartoon)
cv2.waitKey()
cv2.destroyAllWindows()
```

小珞珞卡通效果显示如下图所示。

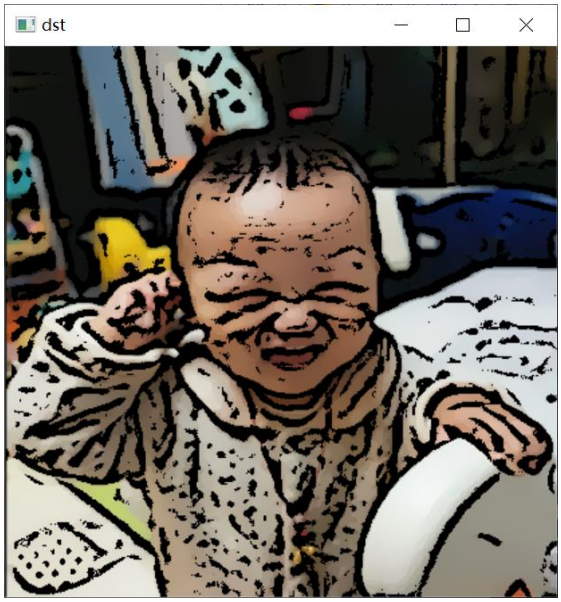


图 37-7 小珞珞图像卡通特效

3.总结

本文主要讲解了图像常见的特效处理，从处理效果图、算法原理、代码实现三个步骤进行详细讲解，涉及图像素描特效和卡通特效等，这些知识点将为读者从事 Python 图像处理相关项目实践或科学研究提供一定基础。

参考文献：

[1] Eastmount. [Android] 通过 Menu 实现图片怀旧、浮雕、模糊、光照和素描效果 [EB/OL]. (2014-11-02). https://blog.csdn.net/Eastmount/article/details/40711317.

[2] Eastmount. [Python 图像处理] 三十三.图像各种特效处理及原理万字详解(毛玻璃、浮雕、素描、怀旧、流年、滤镜等) [EB/OL]. (2020-12-22). https://blog.csdn.net/Eastmount/article/ details/111568397.

[3] 小小何先生. 使用 python 和 opencv 将图片转化为素描图-python 代码解析 [EB/OL]. (2018-09-16). https://blog.csdn.net/weixin_39059031/article/details/82724951.

第 38 篇 图像特效之怀旧、流年、光照和水波特效变换

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面的文章围绕 Python 图像处理、图像运算和图像增强，从图像基础操作、几何变换、点运算、直方图、图像增强、图像平滑、图像锐化等方面进行讲解。接下来将详细讲解常见的图像特效处理，从而让读者实现各种各样的图像特殊效果，它们类似于 PS 或美图秀秀软件的特效处理，能有效地帮助我们理解这些特效背后的故事。第一篇文章将介绍图像怀旧、流年、光照和水波特效变换。

1.图像怀旧特效变换

图像怀旧特效是指图像经历岁月的昏暗效果，如图 38-1 所示，左边“src”为原始图像，右边“dst”为怀旧特效图像。

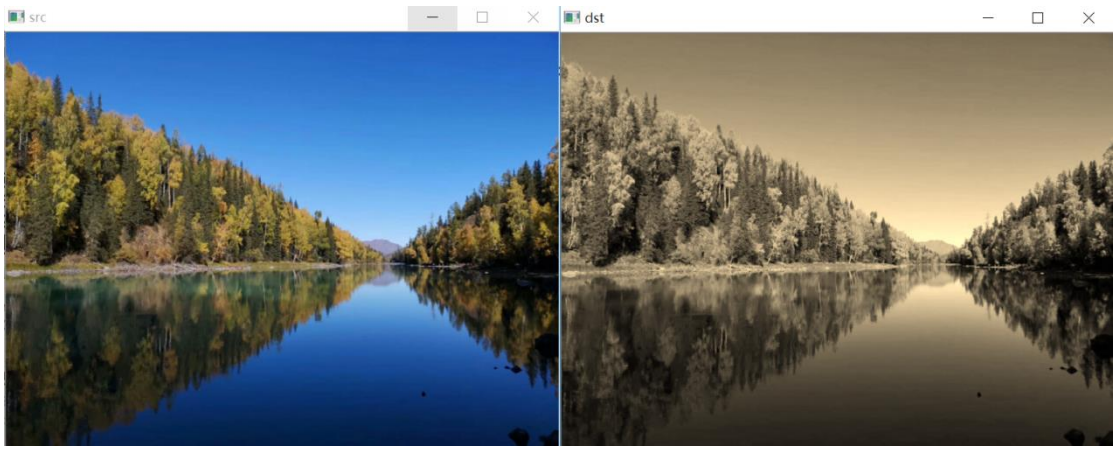


图 38-1 图像怀旧特效处理

怀旧特效是将图像的 RGB 三个分量分别按照一定比例进行处理的结果，其怀旧公式如（38-1）所示[1]。

$$\begin{aligned} R &= 0.393*r + 0.769*g + 0.189*b \\ G &= 0.349*r + 0.686*g + 0.168*b \\ B &= 0.272*r + 0.534*g + 0.131*b \end{aligned} \quad (38-1)$$

Python 实现代码主要通过双层循环遍历图像的各像素点，再结合该公式计算各颜色通道的像素值，其完整代码如下。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np

#读取原始图像
img = cv2.imread('luo.png')
```

```
#获取图像行和列
rows, cols = img.shape[:2]


#新建目标图像
dst = np.zeros((rows, cols, 3), dtype="uint8")


#图像怀旧特效
for i in range(rows):
    for j in range(cols):
        B = 0.272*img[i,j][2] + 0.534*img[i,j][1] + 0.131*img[i,j][0]
        G = 0.349*img[i,j][2] + 0.686*img[i,j][1] + 0.168*img[i,j][0]
        R = 0.393*img[i,j][2] + 0.769*img[i,j][1] + 0.189*img[i,j][0]
        if B>255:
            B = 255
        if G>255:
            G = 255
        if R>255:
            R = 255
```

```
            dst[i,j] = np.uint8((B, G, R))


#显示图像
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

最终生成如图 38-2 所示的效果图。

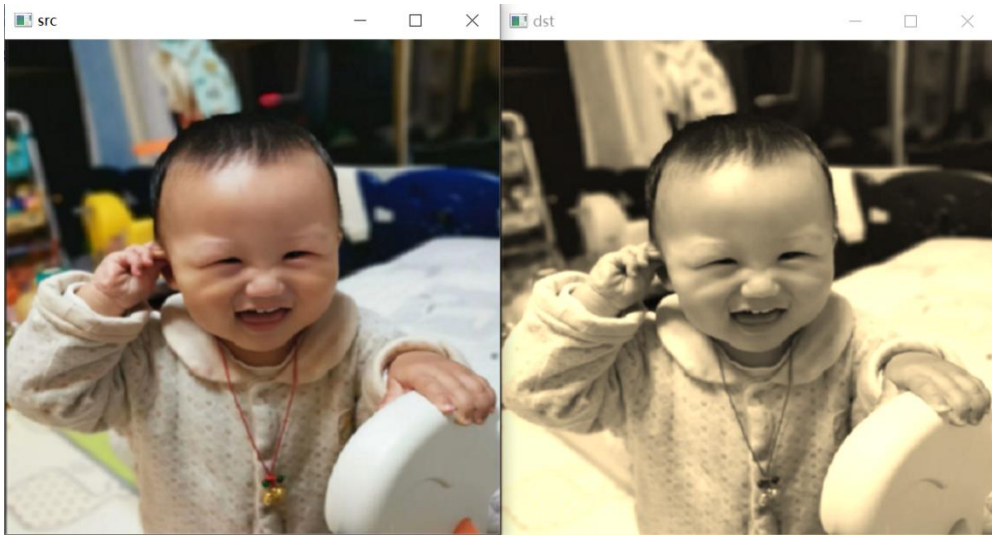


图 38-2 小珞珞图像的怀旧特效

2.图像流年特效变换

流年是用来形容如水般流逝的光阴或年华，图像处理中特指将原图像转换为具有时代感或岁月沉淀的特效，其效果如图 38-3 所示。

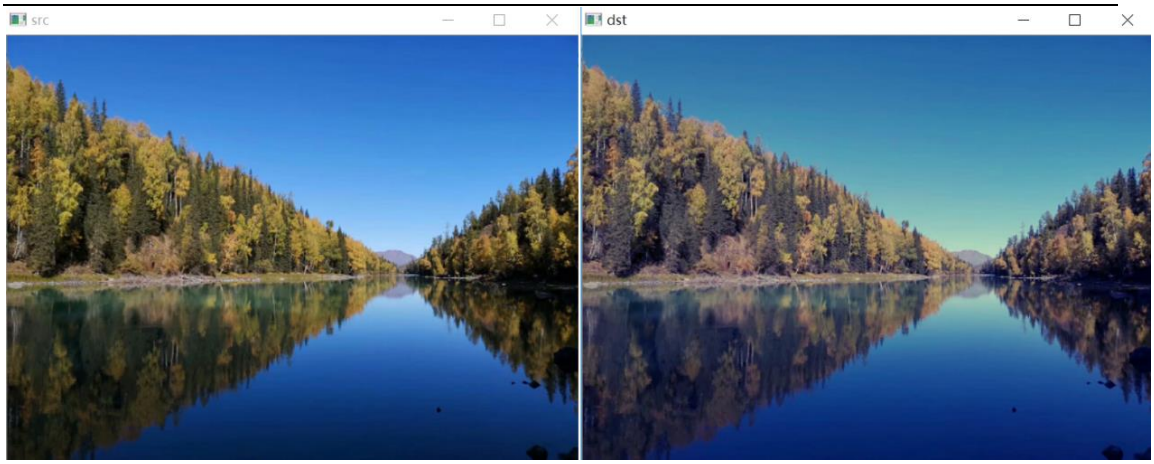


图 38-3 图像流年特效处理

Python 实现代码如下，它将原始图像的蓝色（B）通道的像素值开根号，再乘以一个权重参数，产生最终的流年效果。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import math
import numpy as np

#读取原始图像
img = cv2.imread('luo.png')

#获取图像行和列
rows, cols = img.shape[:2]
```

```
#新建目标图像
dst = np.zeros((rows, cols, 3), dtype="uint8")


#图像流年特效
for i in range(rows):
    for j in range(cols):
        #B 通道的数值开平方乘以参数 12
        B = math.sqrt(img[i,j][0]) * 12
        G =  img[i,j][1]
        R =  img[i,j][2]
        if B>255:
            B = 255
        dst[i,j] = np.uint8((B, G, R))


#显示图像
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

最终生成如图 38-4 所示的效果图。

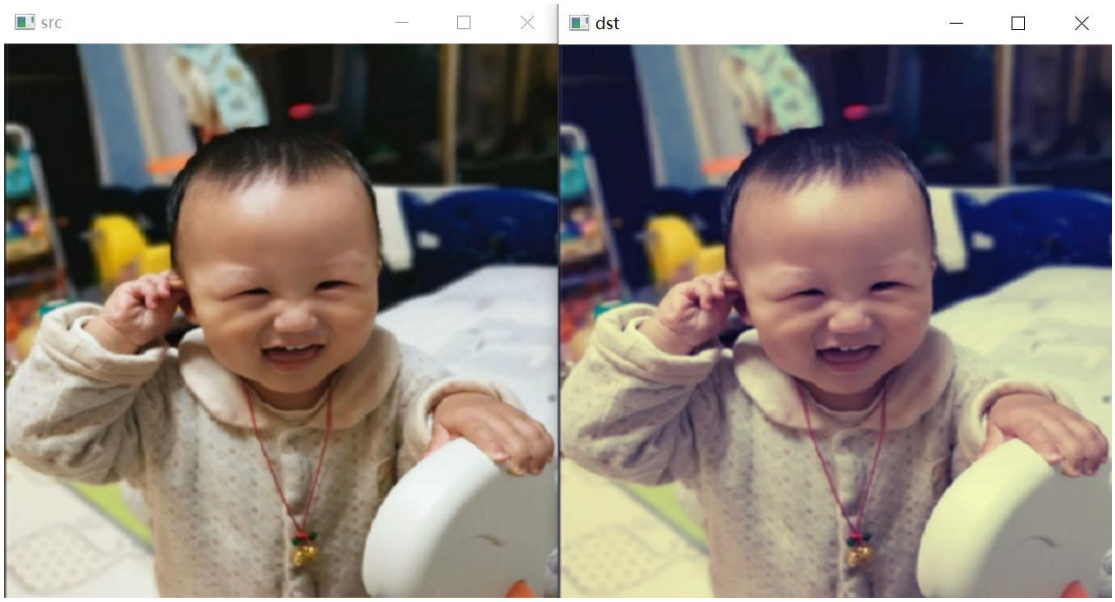


图 38-4 小珞珞图像的流年特效

3.图像光照特效变换

图像光照特效是指图像存在一个类似于灯光的光晕特效，图像像素值围绕光照中心点呈圆形范围内的增强。如图 38-5 所示，该图像的中心点为（192，192），光照特效之后中心圆范围内的像素增强了 200。

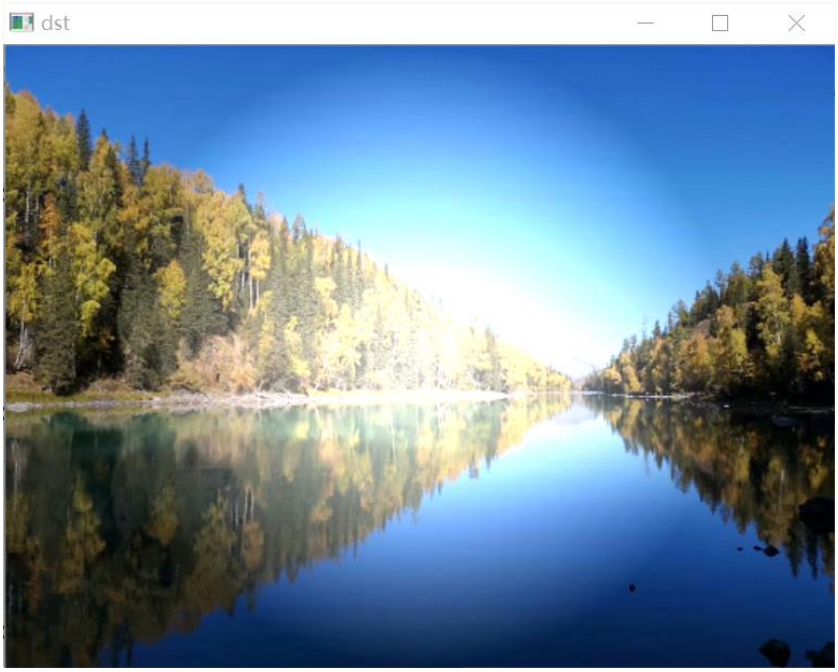


图 38-5 图像光照特效处理

Python 实现代码主要是通过双层循环遍历图像的各像素点，寻找图像的中心点，再通过计算当前点到光照中心的距离（平面坐标系中两点之间的距离），判断该距离与图像中心圆半径的大小关系，中心圆范围内的图像灰度值增强，范围外的图像灰度值保留，并结合边界范围判断生成最终的光照效果[1]。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import math
import numpy as np


#读取原始图像
```

```
img = cv2.imread('luo.png')


#获取图像行和列
rows, cols = img.shape[:2]


#设置中心点
centerX = rows / 2
centerY = cols / 2
radius = min(centerX, centerY)


#设置光照强度
strength = 200


#新建目标图像
dst = np.zeros((rows, cols, 3), dtype="uint8")


#图像光照特效
for i in range(rows):
    for j in range(cols):
        #计算当前点到光照中心距离(平面坐标系中两点之间的距离)
        distance = math.pow((centerY-j), 2) +
```

```
math.pow((centerX-i), 2)
            #获取原始图像
            B =   img[i,j][0]
            G =   img[i,j][1]
            R = img[i,j][2]
            if (distance < radius * radius):
                #按照距离大小计算增强的光照值
                result = (int)(strength*( 1.0 -
math.sqrt(distance) / radius ))
                B = img[i,j][0] + result
                G = img[i,j][1] + result
                R = img[i,j][2] + result
                #判断边界 防止越界
                B = min(255, max(0, B))
                G = min(255, max(0, G))
                R = min(255, max(0, R))
                dst[i,j] = np.uint8((B, G, R))
            else:
                dst[i,j] = np.uint8((B, G, R))

#显示图像
```

```
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

最终生成如图 38-6 所示的效果图。

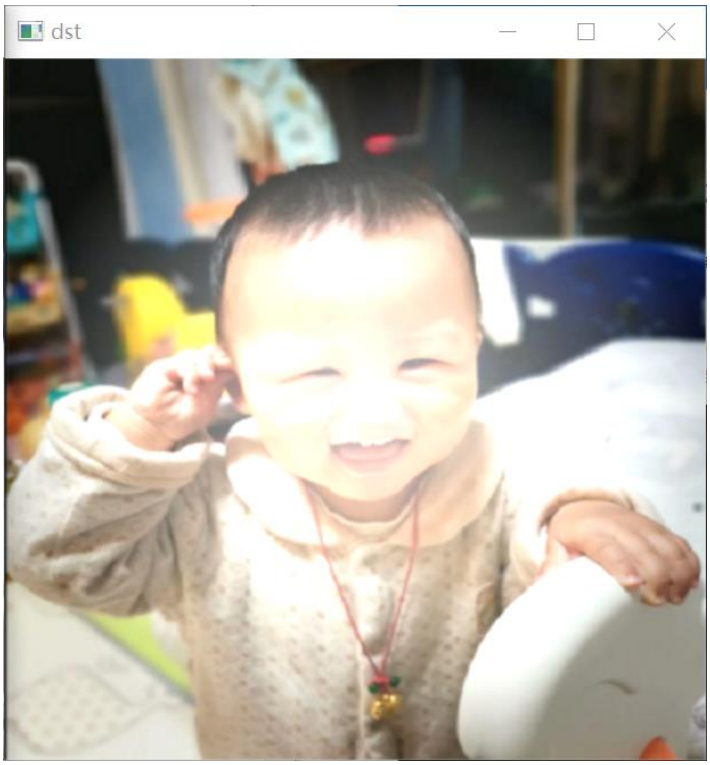


图 38-6 小珞珞图像的流年特效

4.图像水波特效变换

图像水波特效是将图像转换为波浪的效果，围绕水波中心点进行波纹涟漪传递，如图 38-7 所示。

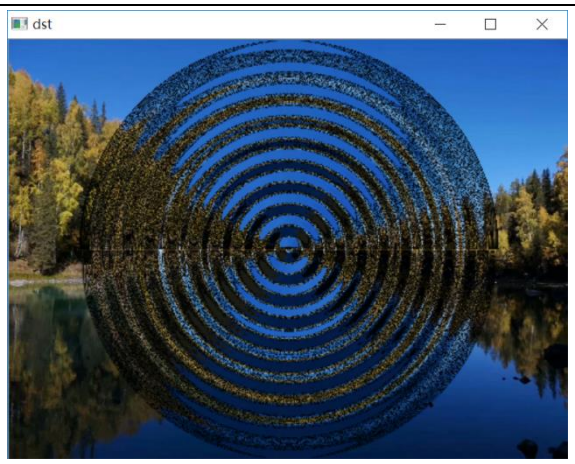


图 38-7 图像水波特效处理

Python 实现代码如下所示，它通过计算水波中心位置，然后调用 np.sin() 函数计算水波传递函数，最终形成水波特效。该小节的代码是依次计算图像所有像素点并进行相关运算，具有一定难度，希望读者能实现对应的效果。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import math
import numpy as np

#读取原始图像
img = cv2.imread('luo.png')

#获取图像行和列
rows, cols = img.shape[:2]
```

```
#新建目标图像
dst = np.zeros((rows, cols, 3), dtype="uint8")

#定义水波特效参数
wavelength = 20
amplitude = 30
phase = math.pi / 4
#获取中心点
centreX = 0.5
centreY = 0.5
radius = min(rows, cols) / 8

#设置水波覆盖面积
icentreX = cols*centreX
icentreY = rows*centreY

#图像水波特效
for i in range(rows):
    for j in range(cols):
        dx = j - icentreX
```

```
            dy = i - icentreY
            distance = dx*dx + dy*dy

            if distance>radius*radius:
                x = j
                y = i
            else:
                #计算水波区域
                distance = math.sqrt(distance)
                amount = amplitude * math.sin(distance / wavelength * 2*math.pi - phase)
                amount = amount *  (radius-distance) / radius
                amount = amount * wavelength / (distance+0.0001)
                x = j + dx * amount
                y = i + dy * amount

            #边界判断
            if x<0:
                x = 0
            if x>=cols-1:
```

```
            x = cols - 2
        if y<0:
            y = 0
        if y>=rows-1:
            y = rows - 2

        p = x - int(x)
        q = y - int(y)

        #图像水波赋值
        dst[i, j, :] = (1-p)*(1-q)*img[int(y),int(x),:] + p*(1-
q)*img[int(y),int(x),:]
        + (1-p)*q*img[int(y),int(x),:] + p*q*img[int(y),int(x),:]

#显示图像
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

5.总结

本文主要讲解了图像常见的特效处理，从处理效果图、算法原理、代码实现三个步骤进行详细讲解，涉及图像怀旧特效、流年特效、光照特效和水波特效等，这些知识点将为读者从事 Python 图像处理相关项目实践或科学研究提供一定基础。

参考文献：

[1] Eastmount. [Android] 通过 Menu 实现图片怀旧、浮雕、模糊、光照和素描效果 [EB/OL]. (2014-11-02). https://blog.csdn.net/Eastmount/article/details/40711317.

[2] Eastmount. [Python 图像处理] 三十三.图像各种特效处理及原理万字详解(毛玻璃、浮雕、素描、怀旧、流年、滤镜等) [EB/OL]. (2020-12-22). https://blog.csdn.net/Eastmount/article/ details/111568397.

第 39 篇 图像特效之滤镜和均衡化特效变换

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面的文章围绕 Python 图像处理、图像运算和图像增强，从图像基础操作、几何变换、点运算、直方图、图像增强、图像平滑、图像锐化等方面进行讲解。接下来将详细讲解常见的图像特效处理，从而让读者实现各种各样的图像特殊效果，它们类似于 PS 或美图秀秀软件的特效处理，能有效地帮助我们理解这些特效背后的故事。第一篇文章将介绍图像滤镜和均衡化特效变换。

1.图像滤镜特效变换

滤镜主要是用来实现图像的各种特殊效果，它在 Photoshop 中具有非常神奇的作用。滤镜通常需要同通道、图层等联合使用，才能取得最佳艺术效果。本小节将讲述一种基于颜色查找表（Look up Table）的滤镜处理方法，它通过将每一个原始颜色进行转换之后得到新的颜色。比如，原始图像的某像素点为红色（R-255, G-0, B-0），进行转换之后变为绿色（R-0, G-255, B-0），之后所有是红色的地方都会被自动转换为绿色，而颜色查找表就是将所有的颜色进行一次（矩阵）转换，很多的滤镜功能就是提供了这么一个转换的矩阵，在原

始色彩的基础上进行颜色的转换。

假设现在存在一张新的滤镜颜色查找表，如图 39-1 所示，它是一张 512×512 大小，包含各像素颜色分布的图像。

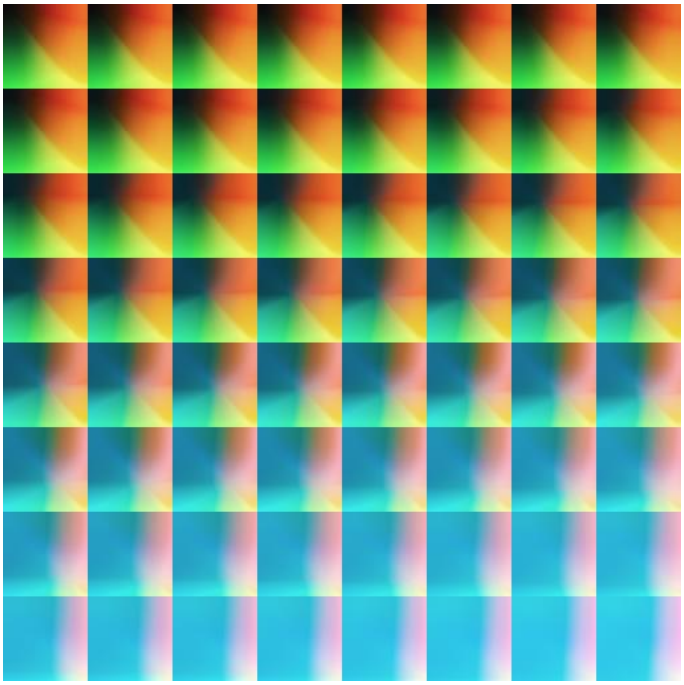


图 39-1 滤镜颜色查找表

滤镜特效实现的 Python 代码如下所示，它通过自定义 getBRG()函数获取颜色查找表中映射的滤镜颜色，再依次循环替换各颜色。

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np


#获取滤镜颜色
```

```
def getBGR(img, table, i, j):
    #获取图像颜色
    b, g, r = img[i][j]
    #计算标准颜色表中颜色的位置坐标
    x = int(g/4 + int(b/32) * 64)
    y = int(r/4 + int((b%32) / 4) * 64)
    #返回滤镜颜色表中对应的颜色
    return lj_map[x][y]

#读取原始图像
img = cv2.imread('scenery.png')
lj_map = cv2.imread('table.png')

#获取图像行和列
rows, cols = img.shape[:2]

#新建目标图像
dst = np.zeros((rows, cols, 3), dtype="uint8")

#循环设置滤镜颜色
for i in range(rows):
```

```
        for j in range(cols):
            dst[i][j] = getBGR(img, lj_map, i, j)


#显示图像
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

滤镜特效的运行结果如图 39-2 所示，其中左边“src”为原始风景图像，右边“dst”为滤镜处理后的图像，其颜色变得更为鲜艳，对比度更强。

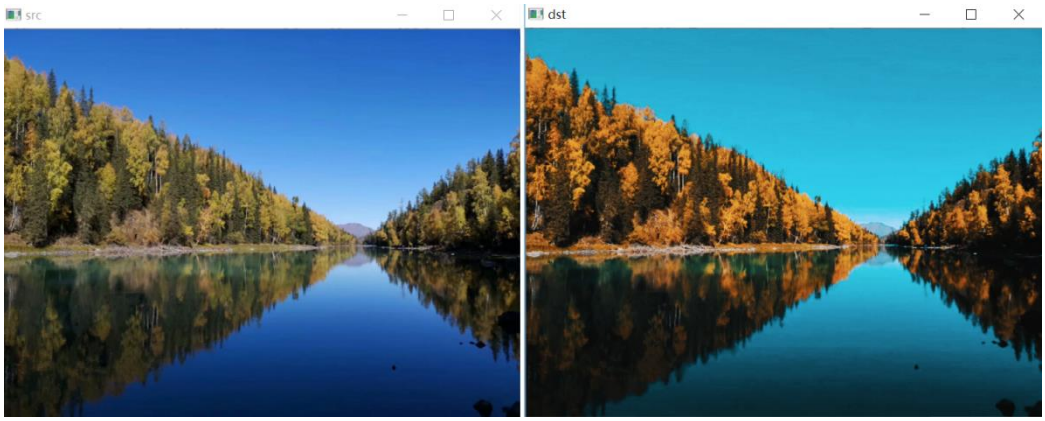


图 39-2 图像滤镜特效

小珞珞的滤镜特效如下图所示。

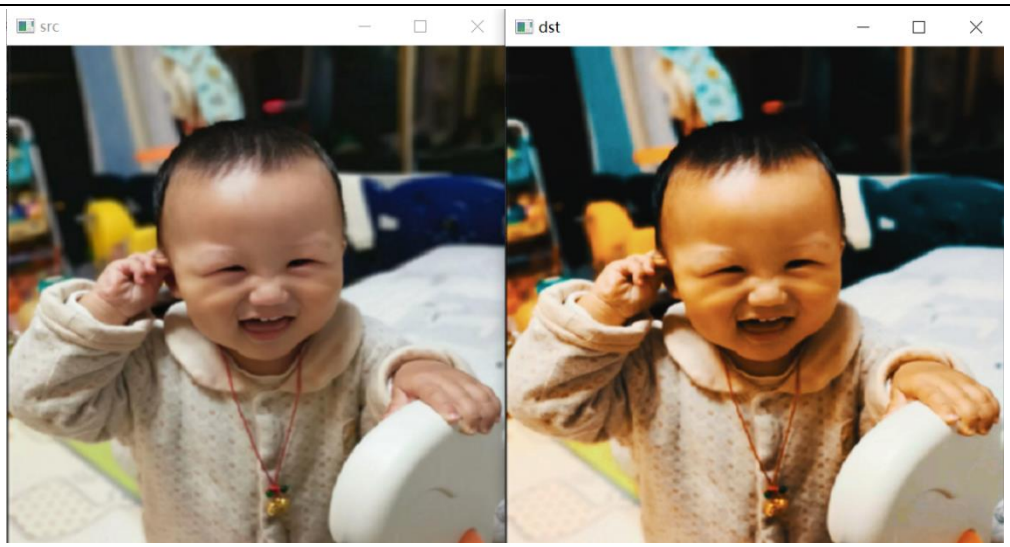


图 39-3 小珞珞图像滤镜特效

2.图像均衡化特效变换

图像直方图均衡化是图像处理领域中利用图像直方图对对比度进行调整的方法，其目的是使输入图像转换为在每一灰度级上都有相同的像素点（即输出的直方图是平的），它可以产生一幅灰度级分布概率均衡的图像，它是增强图像的有效手段之一。

本小节主要是对彩色图像的直方图均衡化特效处理，它将彩色图像用 split()函数拆分成 BGR 三个通道，然后调用 equalizeHist()函数分别进行均衡化处理，最后使用 merge()方法将均衡化之后的三个通道进行合并，生成最终的效果图。实现代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
```

```
import cv2
import numpy as np

#读取原始图像
img = cv2.imread('scenery.png')

#获取图像行和列
rows, cols = img.shape[:2]

#新建目标图像
dst = np.zeros((rows, cols, 3), dtype="uint8")

#提取三个颜色通道
(b, g, r) = cv2.split(img)

#彩色图像均衡化
bH = cv2.equalizeHist(b)
gH = cv2.equalizeHist(g)
rH = cv2.equalizeHist(r)
#合并通道
dst = cv2.merge((bH, gH, rH))
```

```
#显示图像
cv2.imshow('src', img)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

最终生成如图 39-4 所示的图像。

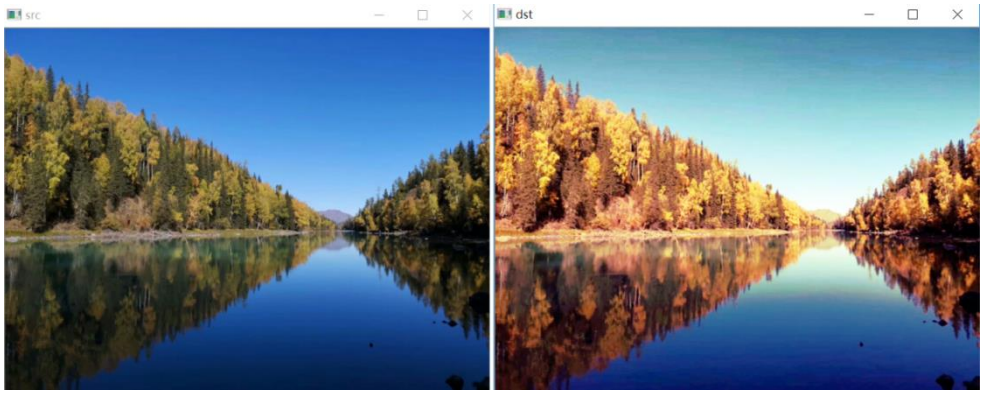


图 39-4 图像直方图均衡化特效

小珞珞的均衡化特效如下图所示。

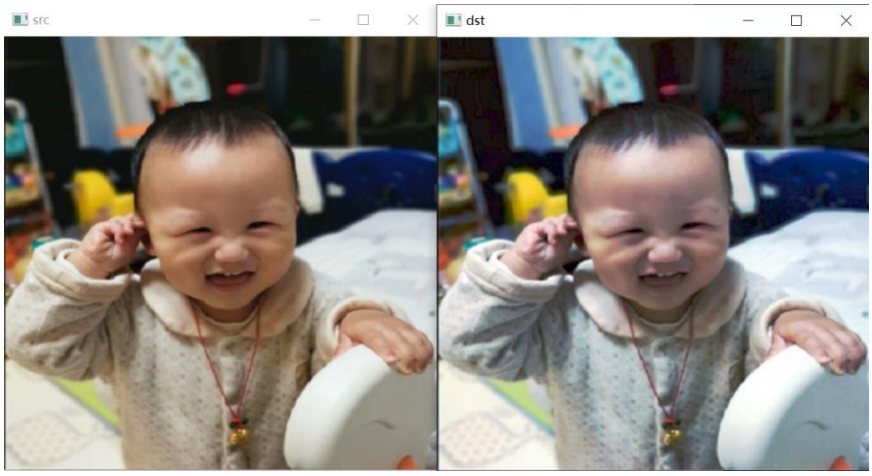


图 39-5 小珞珞图像均衡化特效

3.总结

本文主要讲解了图像常见的特效处理，从处理效果图、算法原理、代码实现三个步骤进行详细讲解，涉及图像滤镜特效和均衡化特效等，这些知识点将为读者从事 Python 图像处理相关项目实践或科学研究提供一定基础。

参考文献：

[1] Eastmount. [Android] 通过 Menu 实现图片怀旧、浮雕、模糊、光照和素描效果 [EB/OL]. (2014-11-02). https://blog.csdn.net/Eastmount/article/details/40711317.

[2] Eastmount. [Python 图像处理] 三十三.图像各种特效处理及原理万字详解(毛玻璃、浮雕、素描、怀旧、流年、滤镜等) [EB/OL]. (2020-12-22). https://blog.csdn.net/Eastmount/article/ details/111568397.

第 40 篇 OpenGL 入门及绘制基本图形和 3D 图

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面介绍的图像处理操作都是基于平面图像的，这篇文章将补充 3D 图形绘制和 OpenGL 的基础入门用法，包括安装、语法、基本图形绘制等，希望对大家有所帮助。

1.什么是 OpenGL

OpenGL(Open Graphics Library，译为“开放式图形库”）是用于渲染 2D、3D 矢量图形的跨语言、跨平台的应用程序编程接口（API）。这个接口由近 350 个不同的函数调用组成，用来绘制从简单的图形元件到复杂的三维景象。OpenGL 常用于 CAD、虚拟现实、科学可视化程序和电子游戏开发。

OpenGL 可用于设置所需的对象、图像和操作，以便开发交互式的 3 维计算机图形应用程序。OpenGL 被设计为一个现代化的、硬件无关的接口，因此我们可以在不考虑计算机操作系统或窗口系统的前提下，在多种不同的图形硬件系统上，或者完全通过软件的方式实现 OpenGL 的接口。OpenGL 的高效实现（利用了图形加速硬件）存在于 Windows，部分 UNIX 平台和 Mac OS。

这些实现一般由显示设备厂商提供，而且非常依赖于该厂商提供的硬件。

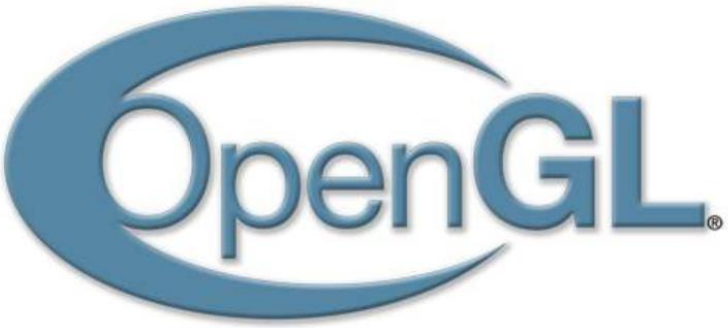


图 40-1 OpenGL 图标

OpenGL 规范由 1992 年成立的 OpenGL 架构评审委员会(ARB)维护。ARB 由一些对创建一个统一的、普遍可用的 API 特别感兴趣的公司组成。到了今天已经发布了非常多的 OpenGL 版本，以及大量构建于 OpenGL 之上以简化应用程序开发过程的软件库。这些软件库大量用于视频游戏、科学可视化和医学软件的开发，或者只是用来显示图像。

一个用来渲染图像的 OpenGL 程序需要执行的主要操作如下：

- 从 OpenGL 的几何图元中设置数据，用于构建形状；
- 使用不同的着色器（ shader ）对输入的图元数据执行计算操作，判断它们的位置、颜色，以及其他渲染属性；
- 将输入图元的数学描述转换为与屏幕位置对应的像素片元(fragment)，这一步也称作光栅化（ rasterization ）；
- 最后，针对光栅化过程产生的每个片元，执行片元着色器（ fragment shader ），从而决定这个片元的最终颜色和位置。

如果有必要，还需要对每个片元执行一些额外的操作，例如判断片元对应的

对象是否可见，或者将片元的颜色与当前屏幕位置的颜色进行融合等。

作者的电脑环境为 Win10 和 Python3.7，打开 CMD 调用 pip 工具进行安装，如下图所示。

- pip install pyopengl

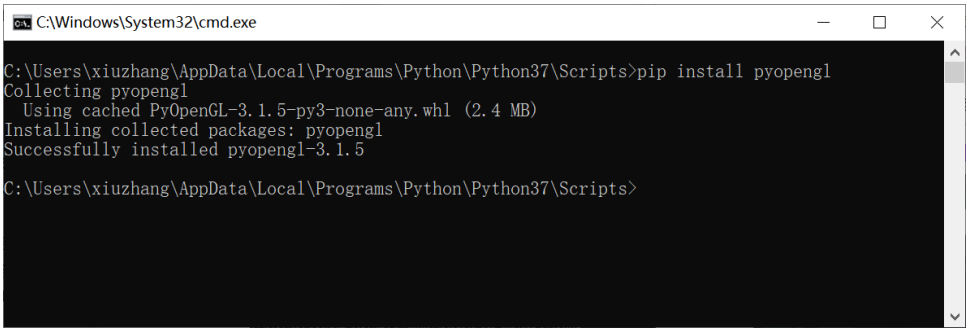


图 40-2 安装过程

如果运行代码会报错“OpenGL.error.NullFunctionError: Attempt to call an undefined function glutInit, check for bool(glutInit) before calling”，说明版本错误，需要去“https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyopengl”网站下载适合自己的版本。

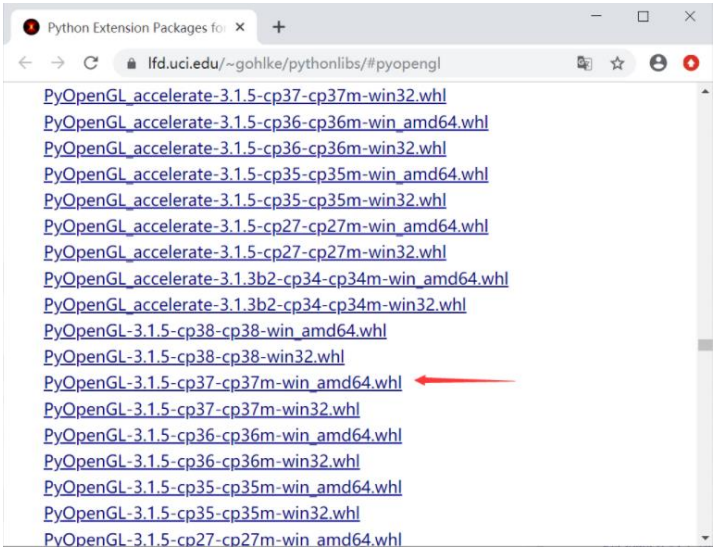


图 40-3 下载指定版本

此时的安装流程如下所示：

- pip install D:\PyOpenGL-3.1.5-cp37-cp37m-win_amd64.whl

```
C:\Users\xiuzhang\AppData\Local\Programs\Python\Python37\Scripts>pip install D:\\PyOpenGL-3.1.5-cp37
-cp37m-win_amd64.whl
Processing d:\pyopengl-3.1.5-cp37-cp37m-win_amd64.whl
Installing collected packages: PyOpenGL
Successfully installed PyOpenGL-3.1.5

C:\Users\xiuzhang\AppData\Local\Programs\Python\Python37\Scripts>
```

图 40-4 安装指定版本

写到这里，我们 Python 的 OpenGL 库就安装成功了！接下来开始实例讲解。

2.OpenGL 绘制几何图形

我们首先介绍两个入门的案例，然后再进行深入讲解。

（1）OpenGL 绘制正方形

第一个案例是绘制正方形，具体代码如下：

```
# -*- coding: utf-8 -*-
# By: Eastmount
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *


# 绘制图像函数
def display():
    # 清除屏幕及深度缓存


glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_
BIT)
    # 设置红色
    glColor3f(1.0, 0.0, 0.0)
    # 开始绘制四边形
    glBegin(GL_QUADS)
    # 绘制四个顶点
    glVertex3f(-0.5, -0.5, 0.0)
    glVertex3f(0.5, -0.5, 0.0)
    glVertex3f(0.5, 0.5, 0.0)
    glVertex3f(-0.5, 0.5, 0.0)
```

```
    # 结束绘制四边形
    glEnd()
    # 清空缓冲区并将指令送往硬件执行
    glFlush()


# 主函数
if __name__ == "__main__":
    # 使用 glut 库初始化 OpenGL
    glutInit()
    # 显示模式 GLUT_SINGLE 无缓冲直接显示|GLUT_RGBA
采用 RGB(A 非 alpha)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)
    # 设置窗口位置大小
    glutInitWindowSize(400, 400)
    # 创建窗口
    glutCreateWindow("eastmount")
    # 调用 display()函数绘制图像
    glutDisplayFunc(display)
    # 进入 glut 主循环
    glutMainLoop()
```

运行结果如下图所示：

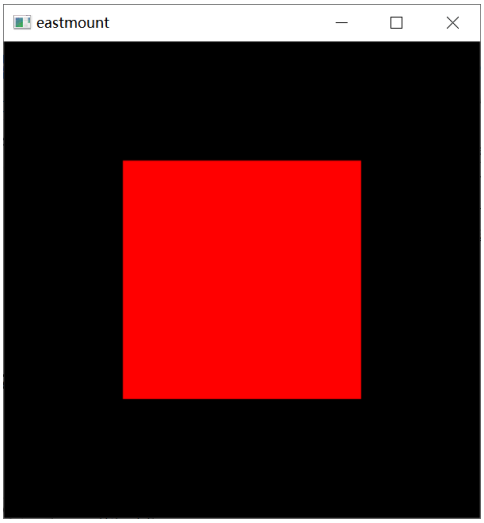


图 40-5 绘制正方形

接下来介绍该部分代码的核心流程。具体如下：

- 主函数使用 glut 库初始化 OpenGL
  - glutInit()
- 设置显示模式并初始化 glut 窗口（画布）
  - glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)
  - glutInitWindowSize(400, 400)
  - glutCreateWindow(“eastmount”)
- 注册绘制图像的回调函数，如 display()
  - glutDisplayFunc(display)
- 绘制图像 display 函数，包括清除画布、设置颜色、绘制图元、设置定点、结束绘制、刷新执行
  - glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)

- glColor3f(1.0, 0.0, 0.0)
- glBegin(GL_QUADS)
- glVertex3f(-0.5, -0.5, 0.0)
- glVertex3f(0.5, -0.5, 0.0)
- glVertex3f(0.5, 0.5, 0.0)
- glVertex3f(-0.5, 0.5, 0.0)
- glEnd()
- glFlush()

❖ 进入 glut 主循环

- glutMainLoop()

（2）OpenGL 绘制多个图形

接下来的案例是绘制一个包含四个图形的坐标系，并设置不同颜色，代码如下所示。

```
# -*- coding: utf-8 -*-
# By: Eastmount
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *


# 绘制图像函数
def display():
```

```
    # 清除屏幕及深度缓存

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_
BIT)

    # 绘制线段
    glBegin(GL_LINES)
    glVertex2f(-1.0, 0.0)        # 左下角顶点
    glVertex2f(1.0, 0.0)         # 右下角顶点
    glVertex2f(0.0, 1.0)         # 右上角顶点
    glVertex2f(0.0, -1.0)        # 左上角顶点
    glEnd()

    # 绘制顶点
    glPointSize(10.0)
    glBegin(GL_POINTS)
    glColor3f(1.0, 0.0, 0.0)     # 红色
    glVertex2f(0.3, 0.3)
    glColor3f(0.0, 1.0, 0.0)     # 绿色
    glVertex2f(0.5, 0.6)
    glColor3f(0.0, 0.0, 1.0)     # 蓝色
```

```
glVertex2f(0.9, 0.9)
glEnd()

# 绘制四边形
glColor3f(1.0, 1.0, 0)
glBegin(GL_QUADS)
glVertex2f(-0.2, 0.2)
glVertex2f(-0.2, 0.5)
glVertex2f(-0.5, 0.5)
glVertex2f(-0.5, 0.2)
glEnd()

# 绘制多边形
glColor3f(0.0, 1.0, 1.0)
glPolygonMode(GL_FRONT, GL_LINE)
glPolygonMode(GL_BACK, GL_FILL)
glBegin(GL_POLYGON)
glVertex2f(-0.5, -0.1)
glVertex2f(-0.8, -0.3)
glVertex2f(-0.8, -0.6)
glVertex2f(-0.5, -0.8)
```

```
    glVertex2f(-0.2, -0.6)
    glVertex2f(-0.2, -0.3)
    glEnd()

    # 绘制三角形
    glColor3f(1.0, 1.0, 1.0)
    glPolygonMode(GL_FRONT, GL_FILL)
    glPolygonMode(GL_BACK, GL_LINE)
    glBegin(GL_TRIANGLES)
    glVertex2f(0.5, -0.5)
    glVertex2f(0.3, -0.3)
    glVertex2f(0.2, -0.6)

    # 结束绘制四边形
    glEnd()
    # 清空缓冲区并将指令送往硬件执行
    glFlush()

# 主函数
if __name__ == "__main__":
    # 使用 glut 库初始化 OpenGL
```

```
    glutInit()
    # 显示模式 GLUT_SINGLE 无缓冲直接显示|GLUT_RGBA 采用 RGB(A 非 alpha)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)
    # 设置窗口位置及大小
    glutInitWindowSize(400, 400)
    glutInitWindowPosition(500, 300)
    # 创建窗口
    glutCreateWindow("By: Eastmount")
    # 调用 display()函数绘制图像
    glutDisplayFunc(display)
    # 进入 glut 主循环
    glutMainLoop()
```

最终显示效果如下图所示。

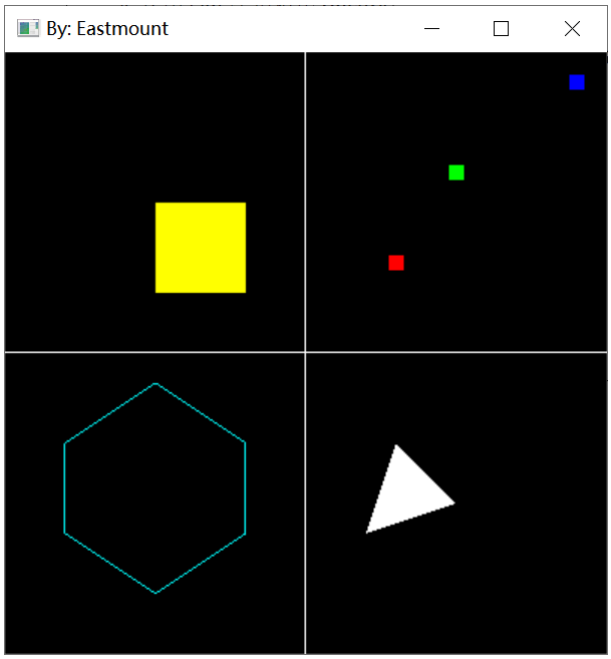


图 40-6 绘制多种图形

3.OpenGL 绘图基本原理

作者先让大家先看代码及其运行效果，从而提升 OpenGL 编程兴趣，再深入分析其原理，这种倒叙的方式希望您们喜欢。

（1）核心函数

上述代码中，以 glut 开头的函数都是 GLUT 工具包所提供的函数。

- **glutInit()**：对 GLUT 进行初始化，该函数必须在其它的 GLUT 使用之前调用一次。其格式比较死板，一般 glutInit()直接调用即可。
- **glutInitDisplayMode()**：设置显示方式，其中 GLUT_RGB 表示使用 RGB 颜色，与之对应的是 GLUT_INDEX（表示使用索引颜色）；GLUT_SINGLE 表示使用单缓冲，与之对应的是 GLUT_DOUBLE（表示使用双缓冲）。更多参数请读者阅读官方网站或 Google。

- glutInitWindowPosition()：设置窗口在屏幕中的位置。
- glutInitWindowSize()：设置窗口的大小，两个参数表示长度和宽度。
- glutCreateWindow()：根据当前设置的信息创建窗口，参数将作为窗口的标题。需要注意的是，当窗口被创建后，并不是立即显示到屏幕上，需要调用 glutMainLoop()才能看到窗口。
- glutDisplayFunc()：设置一个函数，当需要进行画图时，这个函数就会被调用，通常用来调用绘制图形函数。
- glutMainLoop()：进行一个消息循环，大家需要知道这个函数可以显示窗口，并且等待窗口关闭后才会返回。

以 gl 开头的函数是 OpenGL 的标准函数。

- glClear()：清除，其中参数 GL_COLOR_BUFFER_BIT 表示清除颜色，GL_DEPTH_BUFFER_BIT 表示清除深度。
- glRectf()：画一个矩形，四个参数分别表示位于对角线上的两个点的横、纵坐标。
- glFlush()：刷新显示图像，保证前面的 OpenGL 命令立即执行，而不是让它们在缓冲区中等待。
- OpenGL 要求指定顶点的命令（glVertex2f）必须包含在 glBegin()函数和 glEnd()函数之间执行。

（2）绘制顶点

顶点（vertex）是 OpengGL 中非常重要的概念，描述线段、多边形都离不开顶点。它们都是以 glVertex 开头，后面跟一个数字和 1~2 个字母，比如：

- glVertex2d
- glVertex2f
- glVertex3f
- glVertex3fv

数字表示参数的个数，2 表示有 2 个参数（xy 坐标），3 表示三个（xyz 坐标），4 表示四个（齐次坐标 w）。字母表示参数的类型，s 表示 16 位整数（OpenGL 中将这个类型定义为 GLshort），i 表示 32 位整数（OpenGL 中将这个类型定义为 GLint 和 GLsizei），f 表示 32 为浮点数（OpenGL 中将这个类型定义为 GLfloat 和 GLclampf），d 表示 64 位浮点数（OpenGL 中将这个类型定义为 GLdouble 和 GLclampd）。例如：

- glVertex2i(1, 3)
- glVertex2f(1.0, 3.0)
- glVertex3f(1.0, 3.0, 1.0)
- glVertex4f(1.0, 3.0, 0.0, 1.0)

（3）设置颜色

在 OpenGL 中，设置颜色函数以 glColor 开头，后面跟着参数个数和参数类型。参数可以是 0 到 255 之间的无符号整数，也可以是 0 到 1 之间的浮点数。三个参数分别表示 RGB 分量，第四个参数表示透明度（其实叫不透明度更恰当）。以下最常用的两个设置颜色的方法：

- glColor3f(1.0，0.0，0.0) #红色
- glColor3f(0.0，1.0，0.0) #绿色

- glColor3f(0.0，0.0，1.0) #蓝色
- glColor3f(1.0，1.0，1.0) #白色
- glColor4f(0.0，1.0，0.0，0.0) #红色且不透明度
- glColor3ub(255, 0, 0) #红色

注意，OpenGL 是使用状态机模式，颜色是一个状态变量，设置颜色就是改变这个状态变量并一直生效，直到再次调用设置颜色的函数。除了颜色，OpenGL 还有很多的状态变量或模式。

（4）绘制基本图形

前面我们介绍了各种图像，下表展示了常见的图像元件。

- GL_POINTS：绘制顶点
- GL_LINES：绘制线段
- GL_LINE_STRIP：绘制连续线段
- GL_LINE_LOOP：绘制闭合的线段
- GL_POLYGON：绘制多边形
- GL_TRIANGLES：绘制三角形
- GL_TRIANGLE_STRIP：绘制连续三角形
- GL_TRIANGLE_FAN：绘制多个三角形组成的扇形
- GL_QUADS：绘制四边形
- GL_QUAD_STRIP：绘制连续四边形

详见下图所示。

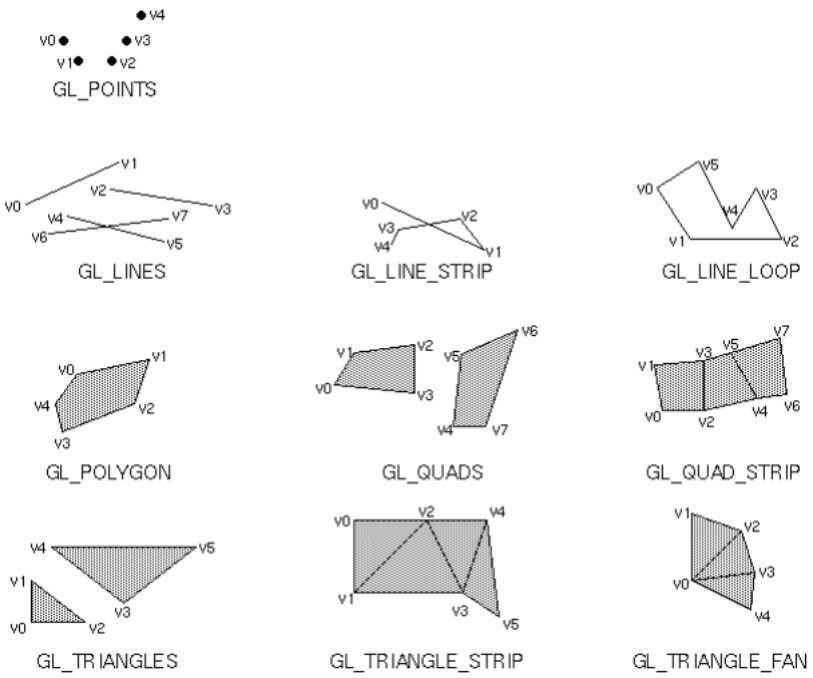


图 40-7 绘制图形

4.OpenGL 基础语法

OpenGL 程序的基本结构通常包括初始化物体渲染所对应的状态、设置需要渲染的物体。渲染（render）表示计算机从模型创建最终图像的过程，OpenGL 只是其中一种渲染系统。模型（model）或者场景对象是通过几何图元，比如点、线和三角形来构建的，而图元与模型的顶点（vertex）也存在着各种对应的关系。

OpenGL 另一个最本质的概念叫着色器，它是图形硬件设备所执行的一类特色函数。可以将着色器理解为专为图形处理单元（GPU）编译的一种小型程序。在 OpenGL 中，会用到始终不同的着色阶段（shader stage），最常用的包括顶点着色器（vertex shader）以及片元着色器，前者用于处理顶点数

据，后者用于处理光栅化后的片元数据。所有的 OpenGL 程序都需要用到这两类着色器。最终生成的图像包含了屏幕上绘制的所有像素点。像素（pixel）是显示器上最小的可见单元。计算机系统将所有的像素保存到帧缓存（framebuffer）当中，后者是由图形硬件设备管理的一块独立内存区域，可以直接映射到最终的显示设备上。

图 40-8 图像像素组成

正如前面您看到的，OpenGL 库中所有的函数都会以字符“gl”作为前缀，然后是一个或者多个大写字母开头的词组，以此来命令一个完整的函数（例如 glBindVertexArray()）。OpenGL 的所有函数都是这种格式，上面看到的“glut”开头的函数，它们来自第三方库 OpenGL Utility Toolkit(GLUT)，可以用来显示窗口、管理用户输入以及执行其他一些操作。

与函数命名约定类似，OpenGL 库中定义的常量也是 GL_COLOR_BUFFER_BIT 的形式，常量以 GL_作为前缀，并且使用下划

线来分割单词。这些常量的定义是通过#define 来完成的，它们基本可以在 OpenGL 的头文件 glcorearb.h 和 glext.h 中找到。

为了能够方便地在不同的操作系统之间移植 OpenGL 程序，它还为函数定义了不同的数据类型，例如 GLfloat 是浮点数类型。此外，比如 glVertex*() 的函数，它有多种变化形式，如 glVertex2d、glVertex2f。在函数名称的“核心”部分之后，通过后缀的变化来提示函数应当传入的参数，通常由一个数字和 1~2 个字母组成。glVertex2f()中的“2”表示需要传入 2 个参数，f 表示浮点数。

后缀	数据类型	通常对应的 C 语言数据类型	OpenGL 类型定义
b	8 位整型	signed char	GLbyte
s	16 位整型	signed short	GLshort
i	32 位整型	int	GLint、GLsizei
f	32 位浮点型	float	GLfloat、GLclampf
d	64 位浮点型	double	GLdouble、GLclampd
ub	8 位无符号整型	unsigned char	GLubyte
us	16 位无符号整型	unsigned short	GLushort
ui	32 位无符号整型	unsigned int	GLuint、GLenum、GLbitfield

图 40-9 OpenGL 类型定义

（1）老式 OpenGL

在大多数计算机图形系统中，绘图的方式是将一些顶点发送给处理管线，管线由一系列功能模块互相连接而成。最近，OpenGL 应用编程接口（API）从固定功能的图形管线转换为可编程的图形管线。

如图 40-10 绘制正方形的代码，它使用的是老式 OpenGL，要为三维图元（在这个代码中，是一个 GL_QUADS 即矩形）指定各个顶点，但随后每个顶点需要被分别发送到 GPU，这是低效的方式。这种老式编程模式伸缩性不好，如果几何图形变得复杂，程序就会很慢。对于屏幕上的顶点和像素如何变换，它

只提供了有限的控制。后续我们将专注于现代的 OpenGL，但是网络上也会有老式 OpenGL 的例子。

```
# -*- coding: utf-8 -*-
# By: Eastmount
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

# 绘制图像函数
def display():
    # 清除屏幕及深度缓存
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    # 设置红色
    glColor3f(1.0, 0.0, 0.0)
    # 开始绘制四边形
    glBegin(GL_QUADS)
    # 绘制四个顶点
    glVertex3f(-0.5, -0.5, 0.0)
    glVertex3f(0.5, -0.5, 0.0)
    glVertex3f(0.5, 0.5, 0.0)
    glVertex3f(-0.5, 0.5, 0.0)
    # 结束绘制四边形
    glEnd()
    # 清空缓冲区并将指令送往硬件执行
    glFlush()

# 主函数
if __name__ == "__main__":
    # 使用glut库初始化OpenGL
    glutInit()
    # 显示模式 GLUT_SINGLE无缓冲直接显示|GLUT_RGBA采用RGB(A非alpha)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)
    # 设置窗口位置大小
    glutInitWindowSize(400, 400)
    # 创建窗口
    glutCreateWindow("eastmount")
    # 调用display()函数绘制图像
    glutDisplayFunc(display)
    # 进入glut主循环
    glutMainLoop()
```

图 40-10 老式 OpenGL 代码

（2）现代 OpenGL

现代 OpenGL 利用一系列的操作，即通过“三维图形管线”绘制图形，其基本流程如图 40-11 所示。

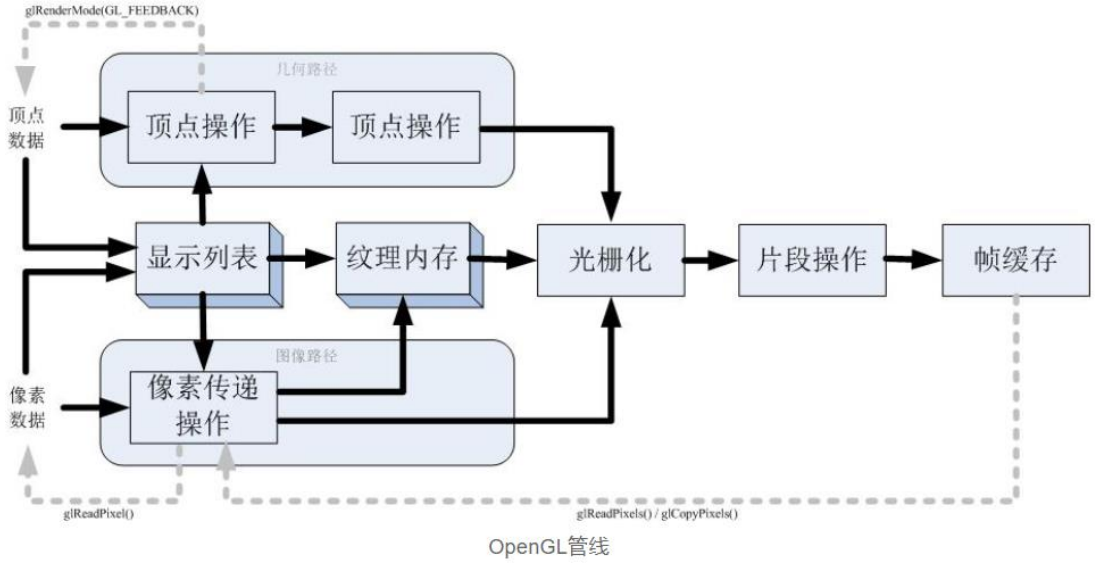


图 40-11 新式 OpenGL 代码

简化三维图形管线分为 6 步：

- **三维几何图形定义（VBO 等）**。在第一步，通过定义在三维空间中的三角形的顶点，并指定每个顶点相关联的颜色，我们定义了三维几何图形。
- **顶点着色器**。接下来，变换这些顶点：第一次变换将这些顶点放在三维空间中，第二次变换将三维坐标投影到二维空间。根据照明等因素，对应顶点的颜色值也在这一步中计算，这在代码中通常称为“顶点着色器”。
- **光栅化**。将几何图形“光栅化”（从几何物体转换为像素）。
- **片段着色器**。针对每个像素，执行一个名为“片段着色器”的代码块。正如顶点着色器作用于三维顶点，片段着色器作用于光栅化后的二维像素。
- **帧缓冲区操作（深度测试、混合等）**。最后，像素经过一系列帧缓冲区操作，其中，它经过“深度缓冲区检验”（检查一个片段是否遮挡另一

个）、“混合”（用透明度混合两个片段）以及其他操作，其当前的颜色与帧缓冲区中该位置已有的颜色结合。

- **帧缓冲区**。这些变化最终体现在最后的帧缓冲区上，通常显示在屏幕上。

该部分参考 Mahesh Venkitachalam 大神编写的《Python 极客项目编程》，代码可以在 github 中查看，对应网址如下：

- https://github.com/electronut/pp

5.OpenGL 绘制 3D 水壶

接着补充一段经典的水壶代码，所有计算机试卷、计算机图形学、3D 图像领域都会绘制它。

```
# -*- coding: utf-8 -*-
# By: Eastmount
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *


# 绘制图像函数
def drawFunc():
    # 清除屏幕及深度缓存


glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_
```

```
BIT)
    # 设置绕轴旋转(角度,x,y,z)
    glRotatef(0.1, 5, 5, 0)
    # 绘制实心茶壶
    # glutSolidTeapot(0.5)
    # 绘制线框茶壶
    glutWireTeapot(0.5)
    # 刷新显示图像
    glFlush()
# 主函数
if __name__ == "__main__":
    # 使用 glut 库初始化 OpenGL
    glutInit()
    # 显示模式 GLUT_SINGLE 无缓冲直接显示|GLUT_RGBA
采用 RGB(A 非 alpha)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)
    # 设置窗口位置及大小
    glutInitWindowPosition(0, 0)
    glutInitWindowSize(400, 400)
    # 创建窗口
    glutCreateWindow("CSDN Eastmount")
```

```
    # 调用 display()函数绘制图像
    glutDisplayFunc(drawFunc)
    # 设置全局的回调函数
    # 当没有窗口事件到达时,GLUT 程序功能可以执行后台处理任务或连续动画
    glutIdleFunc(drawFunc)
    # 进入 glut 主循环
    glutMainLoop()
```

运行结果如图 40-12 所示，它主要调用 glutWireTeapot(0.5)函数绘制线框茶壶。

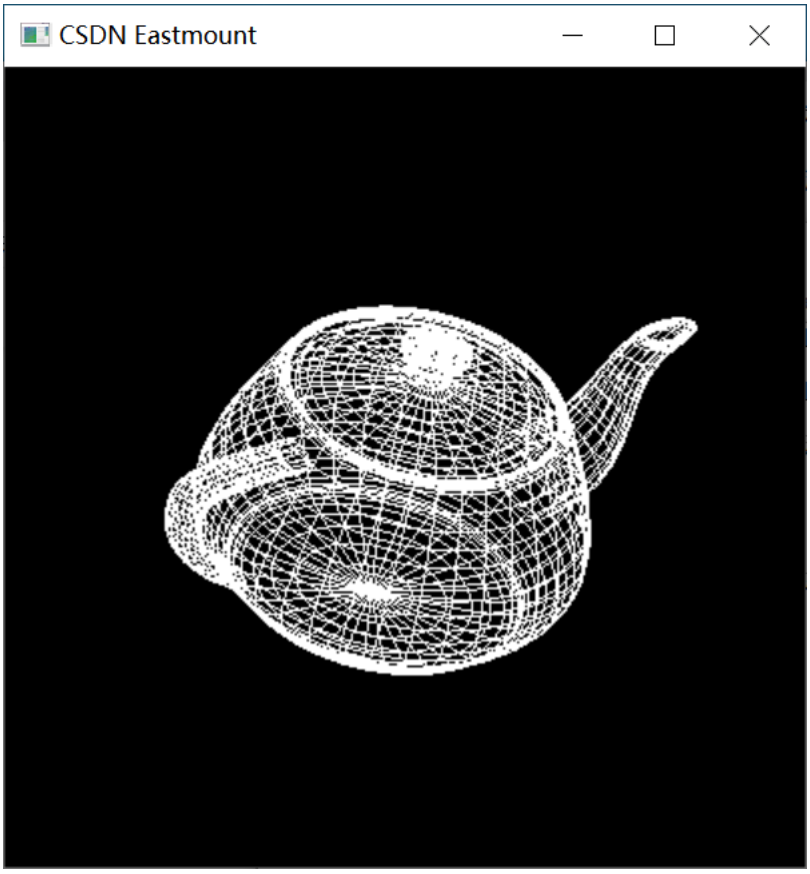


图 40-12 OpenGL 绘制线框水壶

同时，可以调用 glutSolidTeapot(0.5)函数绘制实现实心茶壶。

图 40-13 OpenGL 绘制实心水壶

注意，glut 提供了一些现成的绘制立体的 API，如 glutWireSphere 绘制球、glutWireCone 绘制椎体、glutWireCube 绘制立体、glutWireTorus 绘制甜圈、glutWireTeapot 绘制茶壶、glutWireOctahedron 绘制八面体，请读者自行提升。

6.OpenGL 绘制时钟

最后补充“xiaoge2016 老师”的一段趣味代码，通过 OpenGL 绘制时钟，注意它是跳动的。

```
# -*- coding: utf-8 -*-
```

```
# By: Eastmount & xiaoge2016 老师
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import math
import time


h = 0
m = 0
s = 0


# 绘制图像函数
def Draw():
    PI = 3.1415926
    R = 0.5
    TR = R - 0.05
    glClear(GL_COLOR_BUFFER_BIT)
    glLineWidth(5)
    glBegin(GL_LINE_LOOP)
    for i in range(100):
        glVertex2f(R * math.cos(2 * PI / 100 * i), R *
```

```
math.sin(2 * PI / 100 * i))
        glEnd()
        glLineWidth(2)
        for i in range(100):
            glBegin(GL_LINES)
            glVertex2f(TR * math.sin(2 * PI / 12 * i), TR *
math.cos(2 * PI / 12 * i))
            glVertex2f(R * math.sin(2 * PI / 12 * i), R *
math.cos(2 * PI / 12 * i))
            glEnd()
        glLineWidth(1)

        h_Length = 0.2
        m_Length = 0.3
        s_Length = 0.4
        count = 60.0
        s_Angle = s / count
        count *= 60
        m_Angle = (m * 60 + s) / count
        count *= 12
        h_Angle = (h * 60 * 60 + m * 60 + s) / count
```

```
    glLineWidth(1)
    glBegin(GL_LINES)
    glVertex2f(0.0, 0.0)
    glVertex2f(s_Length * math.sin(2 * PI * s_Angle),
s_Length * math.cos(2 * PI * s_Angle))
    glEnd()
    glLineWidth(5)
    glBegin(GL_LINES)
    glVertex2f(0.0, 0.0)
    glVertex2f(h_Length * math.sin(2 * PI * h_Angle),
h_Length * math.cos(2 * PI * h_Angle))
    glEnd()
    glLineWidth(3)
    glBegin(GL_LINES)
    glVertex2f(0.0, 0.0)
    glVertex2f(m_Length * math.sin(2 * PI * m_Angle),
m_Length * math.cos(2 * PI * m_Angle))
    glEnd()
    glLineWidth(1)
    glBegin(GL_POLYGON)
    for i in range(100):
```

```
            glVertex2f(0.03 * math.cos(2 * PI / 100 * i), 0.03 *
math.sin(2 * PI / 100 * i));
    glEnd()
    glFlush()


# 更新时间函数
def Update():
    global h, m, s
    t = time.localtime(time.time())
    h = int(time.strftime('%H', t))
    m = int(time.strftime('%M', t))
    s = int(time.strftime('%S', t))
    glutPostRedisplay()


# 主函数
if __name__ == "__main__":
    glutInit()
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)
    glutInitWindowSize(400, 400)
    glutCreateWindow("My clock")
    glutDisplayFunc(Draw)
```

```
glutIdleFunc(Update)
glutMainLoop()
```

其运行结果如图 40-14 所示。

图 40-14 OpenGL 绘制时钟

7.总结

本文主要讲解 Python 和 OpenGL 基础知识，包括安装、基础语法、绘制图形等，让大家进一步体会 3D 图形。希望对读者有一定帮助，也希望这些知识点为读者从事 Python 图像处理相关项目实践或科学研究提供一定基础。

参考文献：

[1]《OpenGL 编程指南（第 8 版）》 作者：Dave Shreiner Granham Sellers 等，王锐 译

[2]《Python 极客项目编程》 作者：Mahesh Venkitachalam，王海鹏 译

[3]《OpenGL 编程精粹》杨柏林，陈根浪，徐静 编著

[4] [Python 图像处理] 二十七.OpenGL 入门及绘制基本图形——Eastmount

[5] 写给 python 程序员的 OpenGL 教程——许老师(天元浪子)

[6] Python 之 OpenGL 笔记(2)：现代 OpenGL 编程常用的几个通用函数——大龙老师

[7] python3+OpenGL 环境配置——GraceSkyer 老师

[8] VS2012 下基于 Glut OpenGL 显示一些立体图形示例程序——yearafteryear 老师

[9] Python——OpenGL——白季飞龙老师

[10] python+opengl 显示三维模型小程序——xiaoge2016

[11] https://github.com/electronut/pp

第 41 篇 图像去雾之 ACE 算法和暗通道先验去雾算法实现

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前面介绍了 OpenGL 绘制图形相关知识。这篇文章继续补充图像处理的高阶应用，即图像去雾算法。本文主要讲解 ACE 去雾算法、暗通道先验去雾算法以及雾化生成算法，并且参考了两位计算机视觉大佬(Rizzi 和何恺明)的论文，希望对大家有所帮助。

1.图像去雾

随着社会的发展，环境污染逐渐加剧，越来越多的城市频繁出现雾霾，这不仅给人们的身体健康带来危害，还给那些依赖图像信息的计算机视觉系统造成了不良影响，因为在雾天采集到的图像对比度和饱和度均较低，颜色易发生偏移与失真等。因此，寻找一种简单有效的图像去雾方法，对计算机视觉的后续研究至关重要。

图像增强（Image Enhancement）是指按照某种特定的需求，突出图像中有用的信息，去除或者削弱无用的信息。图像增强的目的是使处理后的图像更适合人眼的视觉特性或易于机器识别。 在医学成像、遥感成像、人物摄影等领

域，图像增强技术都有着广泛的应用。图像增强同时可以作为目标识别、目标跟踪、特征点匹配、图像融合、超分辨率重构等图像处理算法的预处理算法[1-3]。

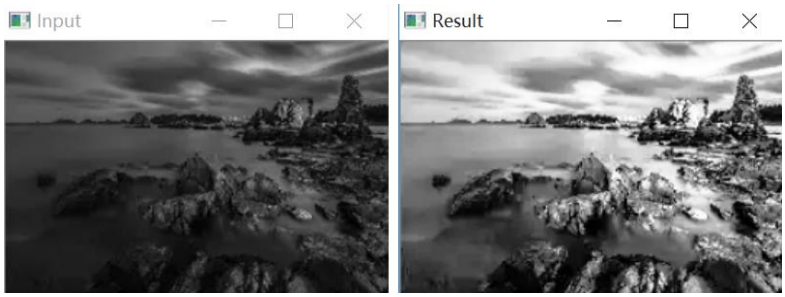


图 41-1 图像增强效果

近些年来，出现了众多的单幅图像去雾算法，应用比较广泛的有：

- 直方图均衡化去雾算法
- Retinex 去雾算法
- 暗通道先验去雾算法
- 基于卷积神经网络的 DehazeNet 去雾算法

其主要可以分为 3 类：基于图像增强的去雾算法、基于图像复原的去雾算法和基于 CNN 的去雾算法。

（1）基于图像增强的去雾算法

通过图像增强技术突出图像细节，提升对比度，使之看起来更加清晰，这类算法的适用性较广。具体的算法有：

- Retinex 算法：根据成像原理消除反射分量影响，达到图像增强去雾效果。
- 直方图均衡化算法：使图像的像素分布更加均匀，放大图像的细节。

- 偏微分方程算法：将图像视作一个偏微分方程，计算梯度场提高对比度。
- 小波变换算法：对图像进行分解，放大有用的部分。

此外，在这类算法的基础上出现了众多的基于图像增强原理的改进算法。

图 41-2 基于图像增强的去雾算法

（2）基于图像复原的去雾算法

主要是基于大气散射物理学模型，通过对大量有雾图像和无雾图像进行观察总结，得到其中存在的一些映射关系，然后根据有雾图像的形成过程来进行逆运算，从而恢复清晰图像。其中最经典的要属何恺明大佬提出的：

- 暗通道先验去雾算法

通过对大量无雾图像进行特征分析，找到了无雾图像与大气散射模型中某些参数的先验关系。该算法复杂度低，去雾效果好，因此在其基础上出现了大量基于暗通道先验的改进算法[4-6]。

图 41-3 传统暗通道和混合暗通道对比

（3）基于 CNN 的去雾算法

使用卷积神经网络（CNN）建立一个端到端的模型，通过有雾图像恢复出无雾图像，目前使用神经网络进行去雾的算法主要有两种思路：

- 使用 CNN 生成大气散射模型的某些参数，再根据大气散射模型来恢复无雾图像
- 使用 CNN 或 GAN 根据模糊图像生成无雾的清晰图像

CNN 因其强大的学习能力在多个领域得到应用，因此也出现了采用 CNN 进行去雾的算法。2016 年 CAI 等首次提出了一种名为 DehazeNet 的去雾网络，用于估计有雾图像的透射率。DehazeNet 将有雾的模糊图像作为输入，输出其透射率，基于大气散射模型理论恢复出无雾的清晰图像[1-2]。

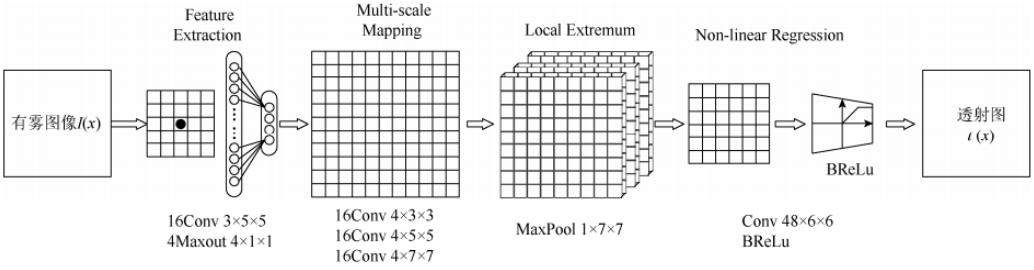


图 41-4 DehazeNet 结构图

图 41-5 是分别对直方图均衡化、暗通道先验去雾、DehazeNet 和 AOD-Net 去雾算法进行测试，实验结果如图所示。由图可知，基于图像增强的直方图均衡化算法的去雾图像对比度明显增强，由于不考虑降质原因，在增加对比度的同时也对噪声进行了放大，出现细节丢失与色彩偏差现象。基于物理模型的暗通道去雾算法、基于神经网络的 DehazeNet 和 AOD-Net 算法的去雾效果较直方图均衡化算法更佳。

图 41-5 图像去雾算法对比图

其他去雾算法对比结果如图 41-6 所示，比如城市和道路有无图像去雾效果。

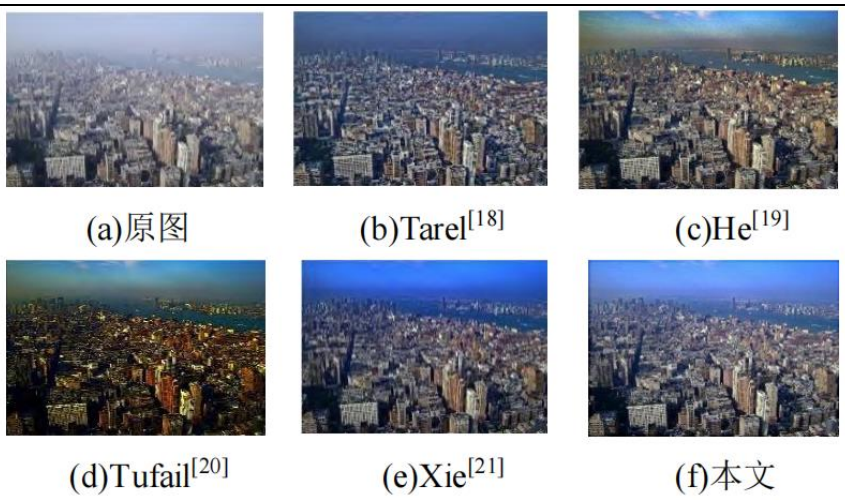


图 41-6 各算法城市图像去雾对比图

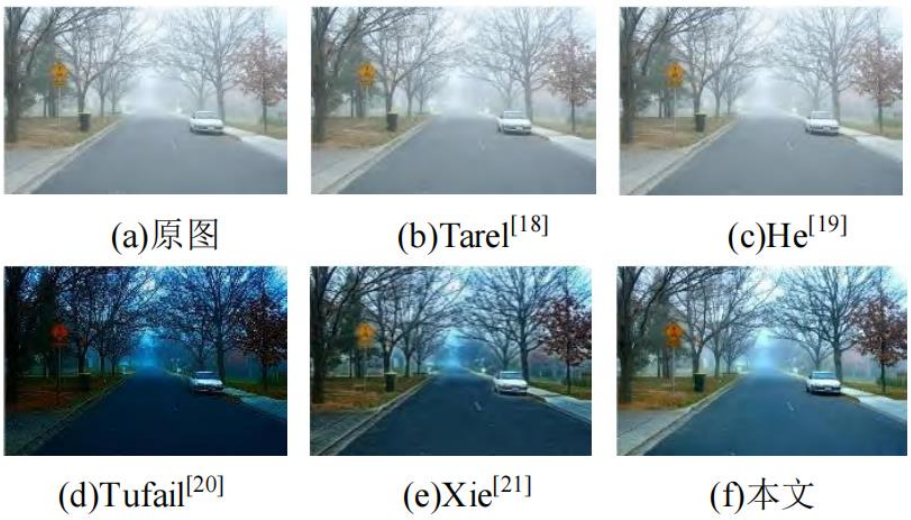


图 41-5 各算法道路图像去雾对比图

正如总结王道累老师[2]总结的一样，目前针对有雾图像去雾的算法主要是从基于图像增强、图像复原和卷积神经网络 3 个方向进行的。

- 基于图像增强的方法不考虑有雾图像的形成过程，而是直接通过突出图像的细节，提高对比度等方式，从而使有雾图像看上去更加清晰。
- 基于图像复原的方法则是追寻图像降质的物理过程，通过物理模型还原出清晰的图像。

- 基于 CNN 的方法则是利用神经网络强大的学习能力，寻找有雾图像与图像复原物理模型中某些系数的映射关系或者使用 GAN，根据有雾图像还原出无雾的清晰图像。

上述 3 类去雾算法对于雾天图像都有着明显的去雾效果，尽管其在实际生活中已经得到了广泛的应用，但下述几点仍有可能是今后图像去雾领域的研究重点和难点：

- 更加真实的雾天图像数据集

采用神经网络进行去雾的算法在效果上好于图像增强和复原的方法，但是由于在自然界中很难拍摄到一组背景相同的有雾图像和无雾图像，因此目前训练神经网络所采用的数据集均是通过合成得到的，虽然能够在一定程度上拟合自然环境，但是仍然存在着一些差距。所以目前急需一种由在真实环境中获取到的具有相同背景的有雾图像和无雾图像构建的数据集，来提高神经网络去雾算法的鲁棒性和稳定性。

- 更加简便的去雾算法

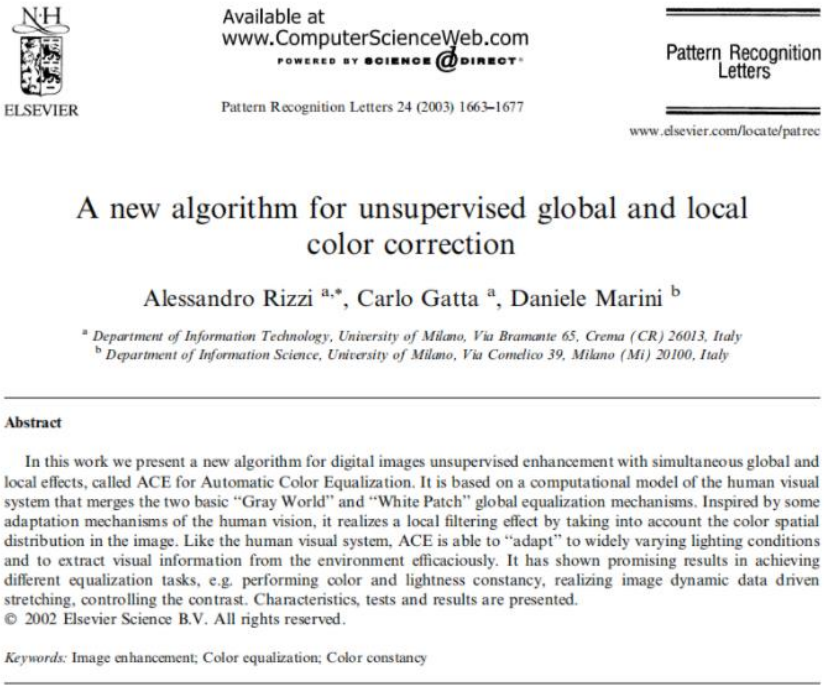
目前各类算法能够有效去除单幅图像上的雾霾，但相对较好的算法都存在着时间复杂度高的问题，很难应用到视频去雾或者需求较多的复杂任务中去。

- 鲁棒性更强的去雾算法

上述算法都只对图像上存在的均匀的薄雾有较好的去雾效果，对于浓雾或者分布不均的团雾则效果较差，因此找到一种适用范围更广的去雾方法将会是一个极具挑战性的课题。

2.ACE 去雾算法

该部分主要介绍参考相关论文进行叙述，简单介绍 ACE 算法的原理知识。如果读者想详细了解其原理，推荐阅读这些大佬的英文原文[7-8]。



ELSEVIER

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Pattern Recognition Letters

Pattern Recognition Letters 24 (2003) 1663–1677

www.elsevier.com/locate/patrec

A new algorithm for unsupervised global and local color correction

Alessandro Rizzi [a,*], Carlo Gatta [a], Daniele Marini [b]

[a] Department of Information Technology, University of Milano, Via Bramante 65, Crema (CR) 26013, Italy
[b] Department of Information Science, University of Milano, Via Comelico 39, Milano (Mi) 20100, Italy

Abstract

In this work we present a new algorithm for digital images unsupervised enhancement with simultaneous global and local effects, called ACE for Automatic Color Equalization. It is based on a computational model of the human visual system that merges the two basic "Gray World" and "White Patch" global equalization mechanisms. Inspired by some adaptation mechanisms of the human vision, it realizes a local filtering effect by taking into account the color spatial distribution in the image. Like the human visual system, ACE is able to "adapt" to widely varying lighting conditions and to extract visual information from the environment efficaciously. It has shown promising results in achieving different equalization tasks, e.g. performing color and lightness constancy, realizing image dynamic data driven stretching, controlling the contrast. Characteristics, tests and results are presented.

Keywords: Image enhancement; Color equalization; Color constancy

图 41-5 ACE 算法论文

图像对比度增强的算法在很多场合都有用处，特别是在医学图像中，这是因为在众多疾病的诊断中，医学图像的视觉检查时很有必要的。Retinex 算法是代表性的图像增强算法，它根据人的视网膜和大脑皮层模拟对物体颜色的波长光线反射能力而形成，对复杂环境下的一维条码具有一定范围内的动态压缩，对图像边缘有着一定自适应的增强。

自动色彩均衡(Automatic Color Enhancement，ACE)算法是 Rizzi[8] 大神在 Retinex 算法的理论上提出的，它通过计算图像目标像素点和周围像素

点的明暗程度及其关系来对最终的像素值进行校正，实现图像的对比度调整，产生类似人体视网膜的色彩恒常性和亮度恒常性的均衡，具有很好的图像增强效果。

ACE 算法包括两个步骤：

- 一是对图像进行色彩和空域调整，完成图像的色差校正，得到空域重构图像。模仿视觉系统的侧抑制性和区域自适应性，进行色彩的空域调整。侧抑制性是一个生理学概念，指在某个神经元受到刺激而产生兴奋时，再刺激相近的神经元，后者所发生的兴奋对前者产生的抑制作用。
- 二是对校正后的图像进行动态扩展。对图像的动态范围进行全局调整，并使图像满足灰度世界理论和白斑点假设。算法针对单通道，再延伸应用到 RGB 彩色空间 3 通道图像，即对 3 个通道分别处理再进行整合完成。

（1）区域自适应滤波

输入图像 I(灰度图为例)，该步是对单通道图像 I 中所有点 p 的区域自适应滤波，得到完成色差校正，空域重构后的中间结果图像，计算公式如下：

$$R_c(p) = \sum_{j \in Subset, j \neq p} \frac{S_\alpha(I_c(p) - I_c(j))}{d(p,j)}$$

式中：$I_c(p)-I_c(j)$为 p、j 两个像素点间灰度差值，表达拟生物学上的侧抑制性；d(p,j)表示距离度量函数，使用两点间的欧氏距离，作用上控制点 j 对 p 的影响权重，映射出滤波的区域适应性；$S_\alpha(x)$是亮度表现函数(奇函数)，本文算法选择经典 Saturation 函数。

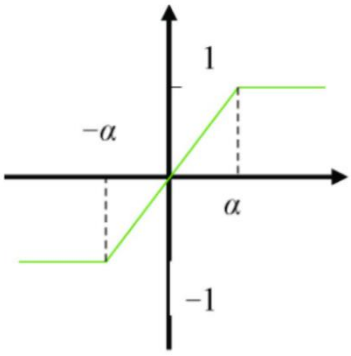


图 41-6 Saturation 函数

不同亮度函数和参数的选择控制了对比度增强的程度，经典的 Saturation 函数在饱和前取越大的斜率，结果的对比度增强越明显。极限情况是 sign 函数形式，而 Sign 函数由于无差别过度增强放大，导致噪声同样得到放大效果不佳，最终选择 Saturation 函数作为相对亮度表现函数。公式如下：

$$S_\alpha(x)=\begin{cases}1, & x<-\alpha\\ x/\alpha, & -\alpha\le x\le\alpha\\ -1, & x>\alpha\end{cases}$$

（2）色调重整拉伸，对图像动态扩展

将上式得到的中间量拉伸映射到[0, 255]中，占满动态范围[0, 255]（8 位灰度图像），计算公式如下，式中：[minR,maxR]是中间量 L(x)的全部定义域，该项使图像达到全局白平衡。

$$L(x)=\frac{R(x)-\min R}{\max R-\min R}$$

下图是条形码图像进行 ACE 图像增强后的效果图，通过图像增强后的图(b)对比度更强，改善了原图像的明暗程度，增强的同时保持了图像的真实性[6]。

图 41-7 条形码 ACE 图像增强

最后，再次推荐大家去阅读 ACE 算法英文论文。其实验对比效果如图 41-8 所示，大家在写该主题论文的时候，也需要注意和传统方法对比。

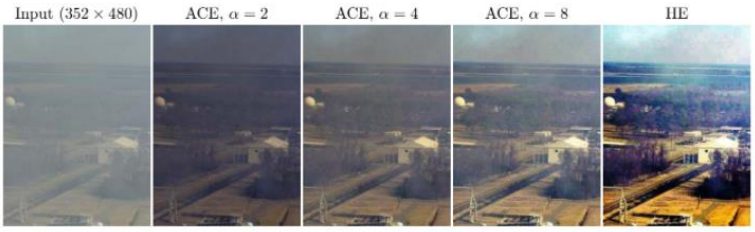


图 41-8 各种算法对比

由于 OpenCV 中暂时没有 ACE 算法包，下面代码是借鉴“zmshy2128”老师的文章[9-11]，修改实现的彩色直方图均衡化处理，以及被我扩展为 Python3 版本。推荐读者结合论文详细分析其代码实现过程。

```
# -*- coding: utf-8 -*-
# By: zmshy2128 老师 & Eastmount 修改成（Python3）
# 参 考 :
https://www.cnblogs.com/zmshy2128/p/6135551.html
import cv2
import numpy as np
```

```
import math
import matplotlib.pyplot as plt


#线性拉伸处理
#去掉最大最小 0.5%的像素值 线性拉伸至[0,1]
def stretchImage(data, s=0.005, bins = 2000):
    ht = np.histogram(data, bins);
    d = np.cumsum(ht[0])/float(data.size)
    lmin = 0; lmax=bins-1
    while lmin<bins:
        if d[lmin]>=s:
            break
        lmin+=1
    while lmax>=0:
        if d[lmax]<=1-s:
            break
        lmax-=1
    return np.clip((data-ht[1][lmin])/(ht[1][lmax]-
ht[1][lmin]), 0,1)


#根据半径计算权重参数矩阵
```

```
g_para = {}
def getPara(radius = 5):
    global g_para
    m = g_para.get(radius, None)
    if m is not None:
        return m
    size = radius*2+1
    m = np.zeros((size, size))
    for h in range(-radius, radius+1):
        for w in range(-radius, radius+1):
            if h==0 and w==0:
                continue
            m[radius+h, radius+w] = 1.0/math.sqrt(h**2+w**2)
    m /= m.sum()
    g_para[radius] = m
    return m


#常规的 ACE 实现
def zmIce(I, ratio=4, radius=300):
    para = getPara(radius)
```

```
height,width = I.shape
zh = []
zw = []
n = 0
while n < radius:
    zh.append(0)
    zw.append(0)
    n += 1
for n in range(height):
    zh.append(n)
for n in range(width):
    zw.append(n)
n = 0
while n < radius:
    zh.append(height-1)
    zw.append(width-1)
    n += 1
#print(zh)
#print(zw)

Z = I[np.ix_(zh, zw)]
```

```
    res = np.zeros(I.shape)
    for h in range(radius*2+1):
        for w in range(radius*2+1):
            if para[h][w] == 0:
                continue
            res += (para[h][w] * np.clip((I-Z[h:h+height,
w:w+width])*ratio, -1, 1))
    return res


#单通道 ACE 快速增强实现
def zmIceFast(I, ratio, radius):
    print(I)
    height, width = I.shape[:2]
    if min(height, width) <=2:
        return np.zeros(I.shape)+0.5
    Rs = cv2.resize(I, (int((width+1)/2), int((height+1)/2)))
    Rf = zmIceFast(Rs, ratio, radius)                    #递归调用
    Rf = cv2.resize(Rf, (width, height))
    Rs = cv2.resize(Rs, (width, height))


    return Rf+zmIce(I,ratio, radius)-zmIce(Rs,ratio,radius)
```

```
#rgb 三通道分别增强 ratio 是对比度增强因子 radius 是卷积模板半径
def zmIceColor(I, ratio=4, radius=3):
    res = np.zeros(I.shape)
    for k in range(3):
        res[:,:,k] = stretchImage(zmIceFast(I[:,:,k], ratio, radius))
    return res

#主函数
if __name__ == '__main__':
    img = cv2.imread('car.png')
    res = zmIceColor(img/255.0)*255
    cv2.imwrite('car-Ice.jpg', res)
```

运行结果如图所示，ACE 算法能有效进行图像去雾处理，实现图像的细节增强。图 41-9 显示行驶中的车辆去雾效果。

图 41-9 行驶中的车辆去雾对比

图 41-10 显示了大雾天游动的鸭群去雾效果。

图 41-10 水中游动鸭群的去雾对比

图 41-11 显示了城市大雾天气的去雾对比效果。

图 41-11 城市去雾对比

3.暗通道先验去雾算法

该算法是计算机视觉领域何恺明大佬于 2009 年提出的图像去雾经典算法，并获取当年 CVPR 最佳论文。论文题目为《Single Image Haze Removal Using Dark Channel Prior》，何老师的很多文章都值得大家学习，是真的非常厉害。如果您是研究图像去雾或计算机视觉领域的读者，建议大家认真阅读这篇英文原文，能在 2009 年提出该算法真的很惊艳[12-13]。

暗通道先验（Dark Channel Prior, DCP）去雾算法依赖大气散射模型进行去雾处理，通过对大量有雾图像和无雾图像进行观察总结，得到其中存在的一些映射关系，然后根据有雾图像的形成过程来进行逆运算，从而恢复清晰图像[14-15]。

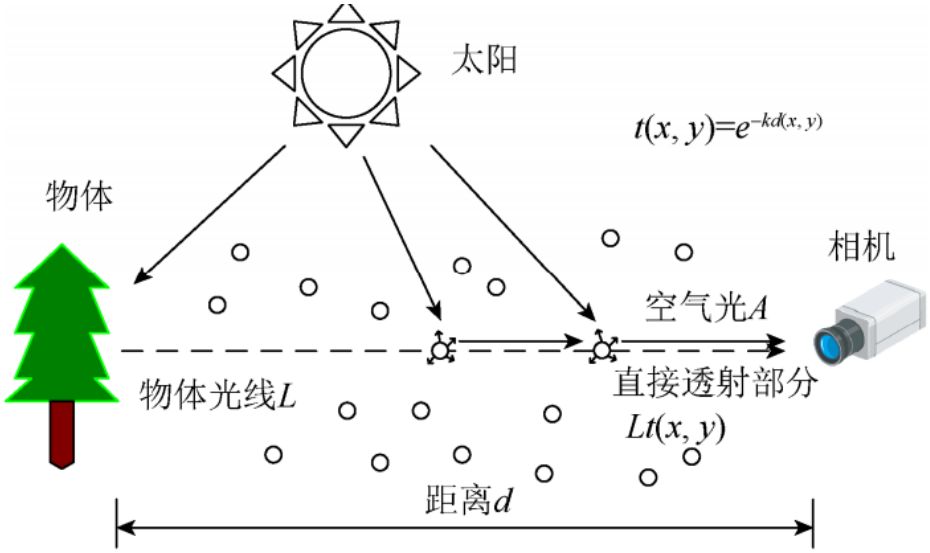


图 41-12 大气散射模型

算法实现过程及原理如下，参考何恺明老师和何涛老师的论文[12-15]。

（1）大气散射模型

在计算机视觉和计算机图形学中，下列方程所描述的大气散射模型被广泛使用。方程右边第一项为场景直接衰减项，第二项为环境光项。

$$\mathbf{I}(\mathbf{x}) = \mathbf{J}(\mathbf{x})t(\mathbf{x}) + \mathbf{A}(1 - t(\mathbf{x}))$$

具体参数解释如下：

- x 是图像的空间坐标
- I(x)代表有雾图像（待去雾图像）
- J(x)代表无雾图像（待恢复图像）
- A 代表全球大气光值
- t(x)代表透射率

（2）暗通道定义

在绝大多数非天空的局部区域中，某些像素总会至少有一个颜色通道的值很低。对于一幅图像 J(x)，其暗通道的数学定义表示如下：

$$J^{\text{dark}}(\mathbf{x}) = \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_{c \in \{r,g,b\}} J^c(\mathbf{y}) \right)$$

其中，Ω(x)表示以 x 为中心的局部区域，上标 c 表示 RGB 三个通道。该公式的意义用代码表达也很简单，首先求出每个像素 RGB 分量中的最小值，存入一副和原始图像大小相同的灰度图中，然后再对这幅灰度图进行最小值滤波，滤波的半径由窗口大小决定。

（3）暗通道先验理论

暗通道先验理论指出：对于非天空区域的无雾图像 J(x)的暗通道趋于 0，即：

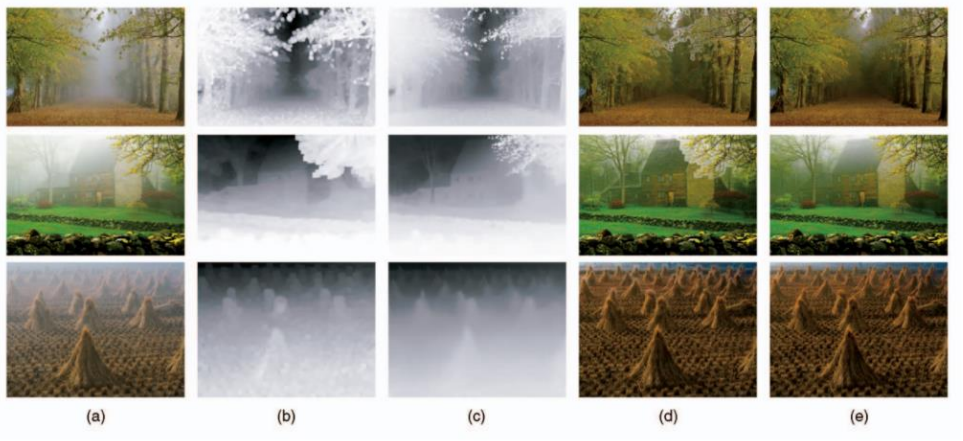
$$J^{\text{dark}} \to 0$$

实际生活中造成暗原色中低通道值主要有三个因素：

- 汽车、建筑物和城市中玻璃窗户的阴影，或者是树叶、树与岩石等自然景观的投影；
- 色彩鲜艳的物体或表面，在 RGB 的三个通道中有些通道的值很低（比如绿色的草地 / 树 / 植物，红色或黄色的花朵 / 叶子，或者蓝色的水面）；
- 颜色较暗的物体或者表面，例如灰暗色的树干和石头。

图 41-13 暗色图像

总之，自然景物中到处都是阴影或者彩色，这些景物的图像的暗原色总是很灰暗的，而有雾的图像较亮。因此，可以明显的看到暗通道先验理论的普遍性。



Haze removal. (a) Input hazy images. (b) Estimated transmission maps before soft matting. (c) Refined transmission maps after soft matting. (d), (e) Recovered images using (b) and (c), respectively.

图 41-14 暗色图像处理示例

（4）公式变形

根据大气散射模型，将第一个公式稍作处理，变形为下式：

$$\frac{I^c(\mathbf{x})}{A^c} = t(\mathbf{x})\frac{J^c(\mathbf{x})}{A^c} + 1 - t(\mathbf{x})$$

假设每一个窗口的透射率 t(x)为常数，记为 t’(x)，并且 A 值已给定，对式两边同时进行两次最小值运算，可得：

$$\min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c \frac{I^c(\mathbf{y})}{A^c}\right) = \tilde{t}(\mathbf{x}) \min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c \frac{J^c(\mathbf{y})}{A^c}\right) + 1 - \tilde{t}(\mathbf{x})$$

其中，J(x)是要求的无雾图像，根据前述的暗通道先验理论可知：

$$J^{\text{dark}}(\mathbf{x}) = \min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c J^c(\mathbf{y})\right) = 0$$

因此可推导出：

$$\min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c \frac{J^c(\mathbf{y})}{A^c}\right) = 0$$

（5）透射率计算

将上式带入可得到透射率 t’(x)的预估值，如下所示：

$$\tilde{t}(\mathbf{x}) = 1 - \min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_{c}\frac{I^c(\mathbf{y})}{A^c}\right)$$

现实生活中，即便晴空万里，空气中也会存在一些颗粒，在眺望远处的景物时，人们还是能感觉到雾的存在。另外，雾的存在让人们感受到景深，因此在去雾的同时有必要保留一定程度的雾。可以通过引入一个 0 到 1 之间的因子 w(一般取 0.95 ）对预估透射率进行修正，如式所示：

$$\tilde{t}(\mathbf{x}) = 1 - \omega\min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_{c}\frac{I^c(\mathbf{y})}{A^c}\right)$$

以上的推导过程均假设大气光值 A 是已知的，在实际中，可以借助暗通道图从原始雾图中求取。具体步骤如下：

- 先求取暗通道图，在暗通道图中按照亮度大小提取最亮的前 0.1%的像素
- 在原始雾图 I(x)中找对应位置上具有最高亮度的点的值，作为大气光值 A

此外，由于透射率 t 偏小时，会造成 J 偏大，恢复的无雾图像整体向白场过度，因此有必要对透射率设置一个下限值 t0（一般取值为 0.1），当 t 值小于 t_0 时，取 $t=t_0$。将以上求得的透射率和大气光值代入公式，最终整理得到图像的恢复公式如下：

$$\mathbf{J}(\mathbf{x}) = \frac{\mathbf{I}(\mathbf{x}) - \mathbf{A}}{\max(t(\mathbf{x}), t_0)} + \mathbf{A}$$

这就是暗通道先验去雾算法的原理过程，下面简单补充论文中的处理效果图。

再次推荐大家阅读何老师的原文。

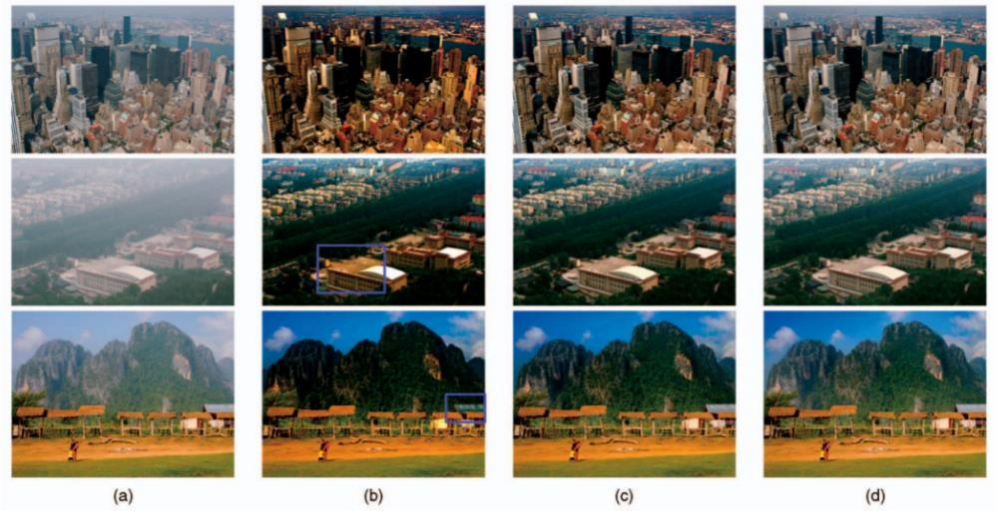


图 41-15 暗通道先验去雾效果

实现代码引用木盏老师的，感觉比作者的写得好[16-17]，核心代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 11 00:16:07 2021
@author: xiuzhang
参考资料：
https://blog.csdn.net/leviopku/article/details/83898619
"""
import sys
import cv2
import math
```

```
import numpy as np


def DarkChannel(im,sz):
    b,g,r = cv2.split(im)
    dc = cv2.min(cv2.min(r,g),b)
    kernel =
cv2.getStructuringElement(cv2.MORPH_RECT,(sz,sz))
    dark = cv2.erode(dc,kernel)
    return dark


def AtmLight(im,dark):
    [h,w] = im.shape[:2]
    imsz = h*w
    numpx = int(max(math.floor(imsz/1000),1))
    darkvec = dark.reshape(imsz,1)
    imvec = im.reshape(imsz,3)


    indices = darkvec.argsort()
    indices = indices[imsz-numpx::]


    atmsum = np.zeros([1,3])
```

```
    for ind in range(1,numpx):
        atmsum = atmsum + imvec[indices[ind]]

    A = atmsum / numpx;
    return A

def TransmissionEstimate(im,A,sz):
    omega = 0.95
    im3 = np.empty(im.shape,im.dtype)

    for ind in range(0,3):
        im3[:,:,ind] = im[:,:,ind]/A[0,ind]

    transmission = 1 - omega*DarkChannel(im3,sz)
    return transmission

def Guidedfilter(im,p,r,eps):
    mean_I = cv2.boxFilter(im,cv2.CV_64F,(r,r))
    mean_p = cv2.boxFilter(p, cv2.CV_64F,(r,r))
    mean_Ip = cv2.boxFilter(im*p,cv2.CV_64F,(r,r))
    cov_Ip = mean_Ip - mean_I*mean_p
```

```
    mean_II = cv2.boxFilter(im*im,cv2.CV_64F,(r,r))
    var_I    = mean_II - mean_I*mean_I

    a = cov_Ip/(var_I + eps)
    b = mean_p - a*mean_I

    mean_a = cv2.boxFilter(a,cv2.CV_64F,(r,r))
    mean_b = cv2.boxFilter(b,cv2.CV_64F,(r,r))

    q = mean_a*im + mean_b
    return q

def TransmissionRefine(im,et):
    gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    gray = np.float64(gray)/255
    r = 60
    eps = 0.0001
    t = Guidedfilter(gray,et,r,eps)
    return t
```

```
def Recover(im,t,A,tx = 0.1):
    res = np.empty(im.shape,im.dtype)
    t = cv2.max(t,tx)
    for ind in range(0,3):
        res[:,:,ind] = (im[:,:,ind]-A[0,ind])/t + A[0,ind]
    return res

if __name__ == '__main__':

    fn = 'city.png'
    src = cv2.imread(fn)
    I = src.astype('float64')/255

    dark = DarkChannel(I,15)
    A = AtmLight(I,dark)
    te = TransmissionEstimate(I,A,15)
    t = TransmissionRefine(src,te)
    J = Recover(I,t,A,0.1)

    arr = np.hstack((I, J))
    cv2.imshow("contrast", arr)
```

```
cv2.imwrite("car-dehaze.png", J*255 )
cv2.imwrite("car-contrast.png", arr*255)
cv2.waitKey();
```

城市雾天的去雾效果如图 41-16 所示，可以发现暗通道先验去雾算法的效果非常棒。

图 41-15 图像去雾效果对比

如果想和后续目标汽车检测结合，同样可以先去雾再进行检测，如图 41-16 所示。

图 41-16 驾驶汽车的图像去雾效果对比

4.图像噪声和雾生成

图像处理总少不了噪声添加或生成，最后补充两个简单的椒盐噪声和雾气模拟生成的代码。这与本文的实验紧密相关，能为我们提供更多的 GAN 生成样本。后面人工智能系列文章，我们看看能不能利用 GAN 学习真实雾化场景图像。

（1）加盐噪声

原图是一张风景图像如图 41-17 所示。

图 41-17 风景图像

图像增加椒盐噪声的代码如下：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np
```

```
#读取图片
img = cv2.imread("fj.png", cv2.IMREAD_UNCHANGED)
rows, cols, chn = img.shape

#加噪声
for i in range(50000):
    x = np.random.randint(0, rows)
    y = np.random.randint(0, cols)
    img[x,y,:] = 210

cv2.imshow("noise", img)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite('fj-res.png',img)
```

输出结果如下图所示。

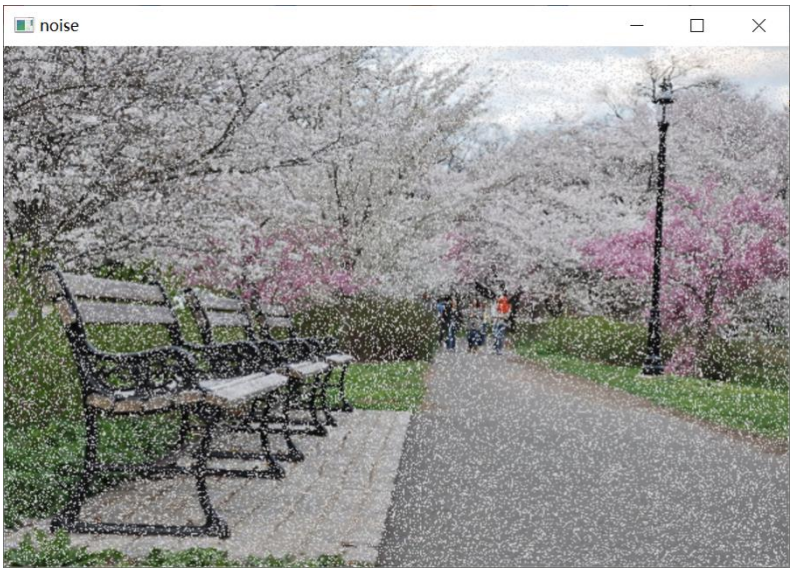


图 41-18 噪声图像

（2）雾的模拟生成

该部分代码如下：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import numpy as np
import cv2 as cv
import os
import random

file = ['fj.png']
output = 'fj-wu.png'

for file_img in file:
```

```
    #打开图像
    img = cv.imread(file_img)
    mask_img = cv.imread(file_img)

    #雾的颜色
    mask_img[:, :] = (166, 178, 180)

    #里面参数可调，主要调整雾的浓度
    image = cv.addWeighted(img,
                           round(random.uniform(0.03,
0.28), 2),
                           mask_img, 1, 0)

    #保存的文件夹
    cv.imwrite(output, image)
```

输出结果如图 41-19 所示，效果还不错。

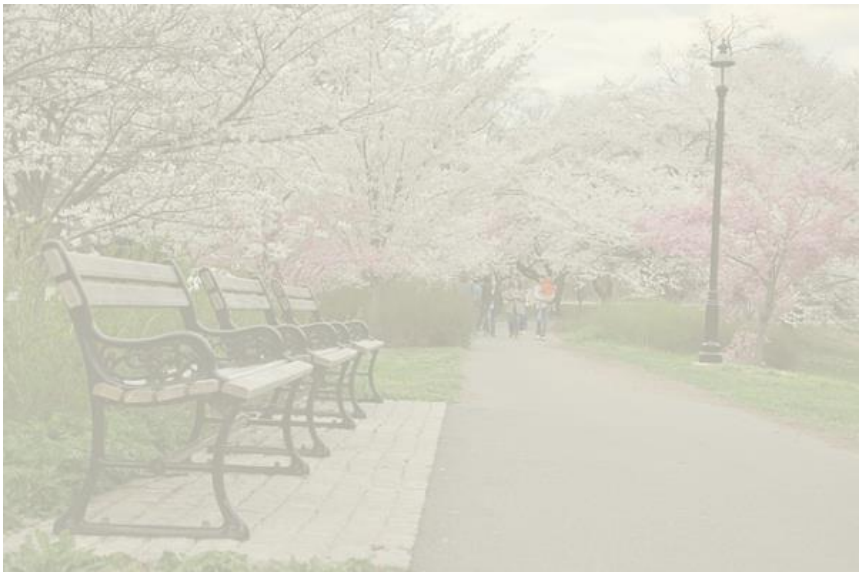


图 41-19 生成雾的图像

5.总结

本文主要讲解两种经典的图像去雾算法，分别是 ACE 去雾算法和暗通道先验去雾算法，同时补充了雾化生成的算法。本文推荐大家学习 Rizzi 和何恺明两位计算机视觉大佬的论文。

参考文献：

[1] 魏红伟，等. 图像去雾算法研究综述[J]. 软件导刊, 2021.

[2] 王道累，等. 图像去雾算法的综述及分析[J]. 图学学报, 2021.

[3] OpenCV 图像增强万字详解（直方图均衡化、自动色彩均衡化）—— Eastmount

[4] 尹胜楠，等. 基于快速 ACE 算法的视觉里程计图像增强方法[J]. 电子测量与仪器学报, 2021.

[5] 李景文，等. 基于暗通道先验改进的自动色彩均衡算法[J]. 科学技术与工程, 2019.

[6] 杨秀璋，等. 一种改进的复杂环境下条形码图像增强和定位算法[J]. 现代计算机，2020.

[7] Pascal Gertreuer. Automatic Color Enhancement (ACE) and its Fast Implementation. https://www.ipol.im/pub/art/2012/g-ace/?utm_source=doi.

[8] Rizzi A., et al. A new algorithm for unsupervised global and local color correction.

https://www.sciencedirect.com/science/article/abs/pii/S0167865502003239.

[9] 自动色彩均衡（ACE）快速算法 - zmshy2128 老师

[10] OpenCV—python 自动色彩均衡（ACE）- SongpingWang

[11] [Python 图像识别] 四十六.图像预处理之图像去雾详解——Eastmount.

[12] He Kaiming, et al. Single Image Haze Removal Using Dark Channel Prior. https://ieeexplore.ieee.org/ document/5567108.

[13] He Kaiming, et al.Single image haze removal using dark channel prior. https://ieeexplore.ieee.org/ document/5206515.

[14] 何涛，等. 基于暗通道先验的单幅图像去雾新算法[J]. 计算机科学, 2021.

[15] 王蓉，等. 基于改进加权融合暗通道算法的图像去雾研究[J]. 浙江科技学院学报，2021.

[16] 图像去雾算法的原理、实现、效果（速度可实时）—— 挚爱图像处理

[17] 图像去雾之何凯明暗通道先验去雾算法原理及 c++代码实现 —— Do it !

第 42 篇 文字图像区域定位及提取分析

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前文介绍了两种经典的图像去雾算法，这篇文章将详细讲解文字图像区域定位和提取分析。图像识别（Image Recognition）是通过算法和函数提取像素中的某些特征，并对图像进行识别和分类的过程。本文提供的案例将为读者提供深入的理解，也是最简单的图像识别案例。

1.OpenCV 文字识别基本步骤

一个完整的文字图像识别系统是一个复杂的系统，主要包括图像灰度处理、图像平滑、图像增强、阈值分割、形态学处理、文字提取、边缘检测、锐化处理等步骤。本文提出了一种改进的图像增强及识别方法，其框架如图 42-1 所示。

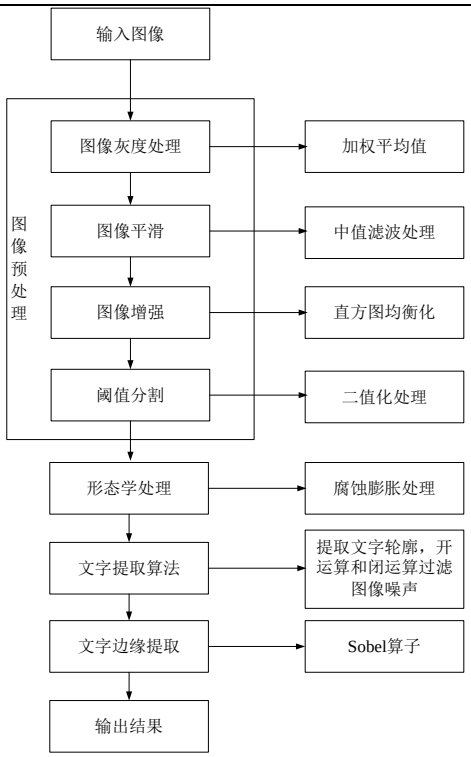


图 42-1 文字图像识别框架图

具体步骤如下：

（1）读取原始数据，经过图像预处理操作，使用加权平均法将彩色图像转换为灰度图像，中值滤波算法降低图像噪声，直方图均衡化方法扩展灰度值的范围，再经过二值化处理提取轮廓。

（2）通过形态学腐蚀处理和膨胀处理细化和扩张文字图像背景。

（3）最后采用改进的文字提取算法凸显文字轮廓，Sobel 算子锐化突出图像的边缘细节，改善图像的对比度，使模糊的图像变得更清晰，并输出是别的结果。

注意，在 OpenCV 中，通常采用 findContours()函数寻找文字轮廓，定位并提取目标文字，接着调用 drawContours()函数绘制相关轮廓，输出最终图像。

2.图像灰度和平滑处理

图像灰度转化是将彩色图像转换为灰度化图像的过程。灰度图像中每个像素仅具有一种样本颜色，其灰度是位于黑色与白色之间的多级色彩深度。式(42-1)是本文所采用的灰度处理公式。

$$Gray=R*0.299+G*0.587+B*0.114 \quad (42-1)$$

式中，R、G、B 分别为图像中每个像素点的三原色红、绿、蓝分量，Gray 为灰度处理后该像素点的灰度值。它将原始 RGB(R,G,B)颜色均匀地替换成新颜色 RGB(Gray,Gray,Gray)，从而将彩色图像转化为灰度图像。由于人类的眼睛感官蓝色的敏感度最低，感官绿色的敏感度最高，因此将 RGB 三分量按照 0.299、0.587、0.114 比例加权平均能得到较合理的灰度图。

该部分实现代码如下：

```
# coding:utf8

import cv2

import numpy as np

import matplotlib.pyplot as plt


#读取原始图像

img = cv2.imread("word.png" )


#转换成灰度图像
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#显示图像
cv2.imshow('Gray Image', gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图 42-2 所示。

图 42-2 文字图像灰度转换

中值滤波（Median Blur）是一种非线性平滑方法，能够在消除噪声的同时保留边界信息。该方法通过计算每一个像素点某邻域范围内所有像素点灰度值的中值，来替换该像素点的灰度值，从而让周围的像素值更接近真实情况，消除孤立的噪声。

中值滤波算法的计算过程如图 42-3 所示。选择含有五个点的窗口，依次扫

描该窗口中的像素，每个像素点所对应的灰度值按照升序或降序排列，然后获取最中间的值来替换该点的灰度值。

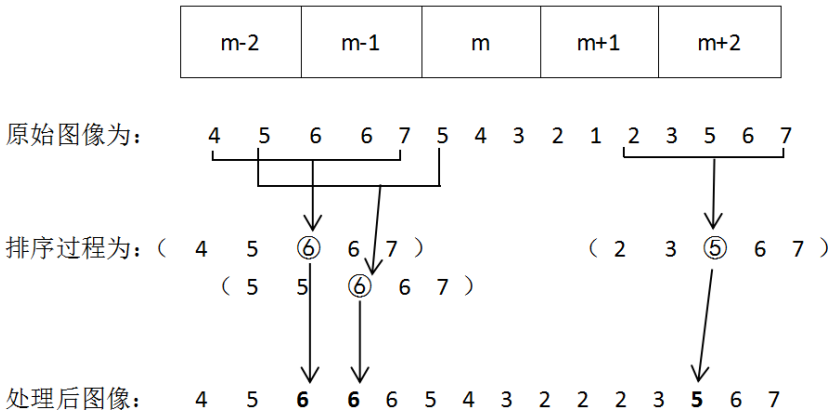


图 42-3 中值滤波算法计算过程

上图展示的是矩形窗口，常用的窗口还包括正方形、十字形、环形和圆形等，不同形状的窗口会带来不同的过滤效果，其中正方形和圆形窗口适合于外轮廓边缘较长的图像，十字形窗口适合于带尖角形状的图像。实现代码如下：

```
# coding:utf8

import cv2

import numpy as np

import matplotlib.pyplot as plt


#读取原始图像

img = cv2.imread("word.png" )


#转换成灰度图像
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


#中值滤波去除噪声
median = cv2.medianBlur(gray, 3)


#显示图像
cv2.imshow('Gray Image', gray)
cv2.imshow('Median Blur', median)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

处理后的结果如图 42-4 所示，它是中值滤波处理后的图像，它有效地过滤掉了图像的噪声。

图 42-4 中值滤波处理结果

3.图像增强处理

直方图均衡化是图像灰度变化的一个重要处理，被广泛应用于图像增强领域。它是指通过某种灰度映射将原始图像的像素点均匀地分布在每一个灰度级上，其结果将产生一幅灰度级分布概率均衡的图像。直方图均衡化的中心思想是把原始图像的灰度直方图从比较集中的某个灰度区间转变为全范围均匀分布的灰度区间，通过该处理，增加了像素灰度值的动态范围，从而达到增强图像整体对比度的效果。该算法的处理过程如下：

首先，计算原始图像直方图的概率密度，如公式(42-2)所示。其中，表示第 k 个灰度级（k=0,1,2,..,L-1），L 最大值为 256；表示图像中灰度级为的像素个数；N 表示图像中像素的总个数；为图像中第 k 个灰度级占总像素数的比例。

$$P(r_k)=\frac{n_k}{N} \tag{42-2}$$

其次，通过灰度变换函数 T 计算新图像灰度级的概率密度。新图像灰度级的概率密度是原始图像灰度级概率密度的累积，如公式(42-3)所示。其中，k 是新图像的灰度级，k=0,1,2,..,L-1；表示原始图像的第 k 个灰度级；为直方图均衡化处理后的第 k 个灰度级。

$$s_k=T(r_k)=\sum_{j=0}^{k}P(r_j) \tag{42-3}$$

最后，计算新图像的灰度值。由于公式(42-3)计算所得的位于 0 至 1 之间，需要乘以图像的最大灰度级 L，转换为最终的灰度值 res，如公式(42-4)所示。

$$res = s_k \times L \tag{42-4}$$

如果图像清晰度较差或存在噪声，可以进行图像直方图均衡化处理，图 42-5 表示原始图像及其直方图。

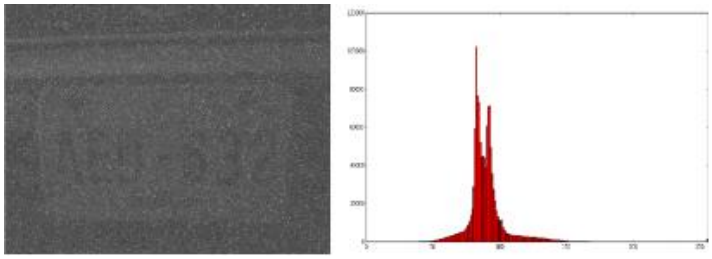


图 42-5 原始图像及其直方图

图 42-6 表示直方图均衡化处理后的图像及其直方图。从效果图可以看出，经过直方图均衡化处理，图像变得更加清晰，图像的灰度级分布也更加均匀。

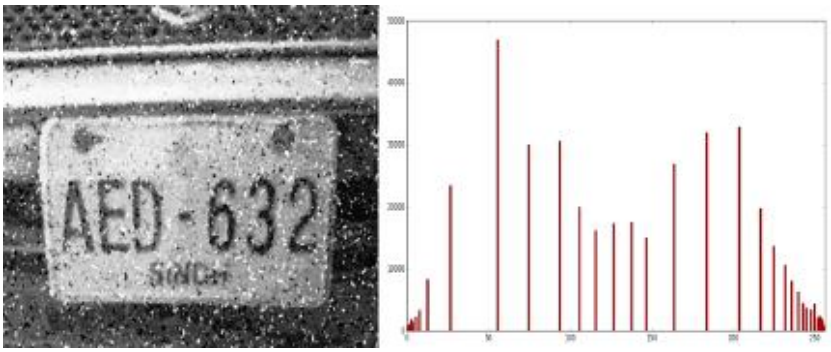


图 42-6 直方图均衡化后的图像及其直方图

核心代码如下：

- equalize = cv2.equalizeHist(median)

4.文字边缘提取

图像数据在收集或传输过程中，受一些外界因素影响，会使得图像存在模糊和有噪声的情况，从而影响图像识别工作。此时需要通过图像锐化处理，加强原图像的高频部分，锐化突出图像的边缘细节，从而改善图像的对比度，使模糊的

图像变得更加清晰。

Sobel 算子是一种计算图像明暗程度近似值的差分算子，它根据图像边缘附近的明暗程度把该区域内超过某个值的特定点记为边缘。Sobel 算子根据相邻点的距离远近为当前像素点赋予不同的权重，距离越近的像素点对当前像素的影响越大，反之越小，从而实现图像锐化并突出边缘轮廓。Sobel 算子的边缘定位更加准确，常用于噪声较多、灰度渐变的图像。其模板如公式(42-5)所示：

$$d_x=\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} \quad d_y=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \tag{42-5}$$

其中，dx 表示水平方向，dy 表示垂直方向，Sobel 算子的最终计算如公式(42-6)所示：

$$S=\sqrt{\left(d_x(i,j)^2+d_y(i,j)^2\right)} \tag{42-6}$$

图 42-7 为 Sobel 算子的处理结果，由于其考虑各方面因素，对噪声较多的图像处理效果更好。

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt


#读取原始图像
img = cv2.imread("word.png" )
```

```
#转换成灰度图像
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#中值滤波去除噪声
median = cv2.medianBlur(gray, 3)

#图像直方图均衡化
equalize = cv2.equalizeHist(median)

#Sobel 算子锐化处理
sobel = cv2.Sobel(median, cv2.CV_8U, 1, 0, ksize = 3)

#显示图像
cv2.imshow('Sobel Image', sobel)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行结果如下图所示。

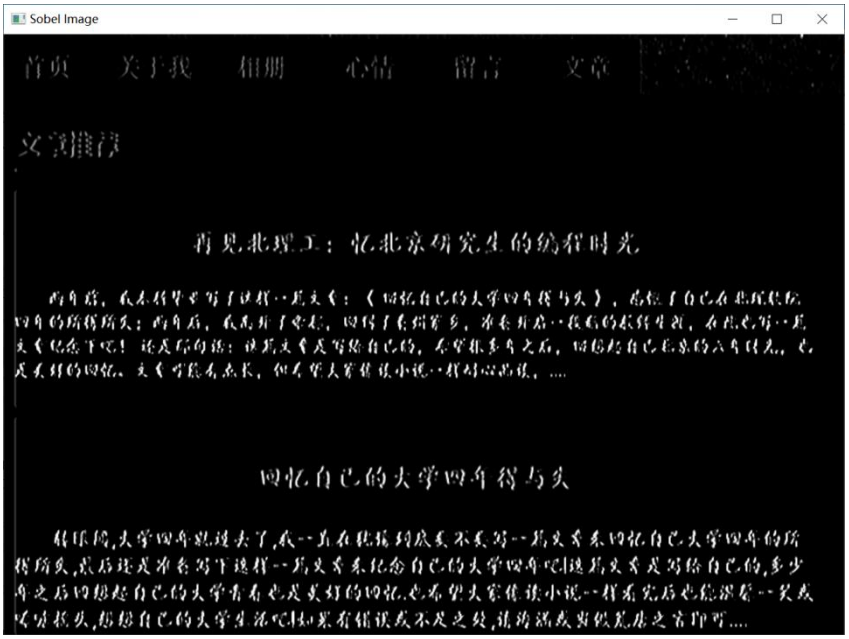


图 42-7 Sobel 算子提取边缘轮廓

5.阈值分割处理

图像二值化（Binarization）旨在提取图像中的目标物体，将背景以及噪声区分开来。通常会设置一个临界值，将图像的像素划分为两类，常见的二值化算法如公式(42-5)所示。

$$Gray(i,j)=\begin{cases}255, & Gray(i,j)\geq T\\ 0, & Gray(i,j)<T\end{cases} \tag{42-5}$$

当某个像素点的灰度 Gray(i,j)小于阈值 T 时，其像素设置为 0，表示黑色；当灰度 Gray(i,j)大于或等于阈值 T 时，其像素值为 255，表示白色。Python OpenCV 中提供了阈值函数 threshold()实现二值化处理，当像素大于等于 127 时灰度值设置为最大值 255，小于 127 的像素点灰度值设置为 0。二值化处理为后续的水族文字识别提供有效支撑。

完整代码如下所示：

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取原始图像
img = cv2.imread("word.png" )

#转换成灰度图像
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#中值滤波去除噪声
median = cv2.medianBlur(gray, 3)

#图像直方图均衡化
equalize = cv2.equalizeHist(median)

#Sobel 算子锐化处理
sobel = cv2.Sobel(median, cv2.CV_8U, 1, 0, ksize = 3)
```

```
#图像二值化处理
ret, binary = cv2.threshold(sobel, 0, 255,

cv2.THRESH_OTSU+cv2.THRESH_BINARY)

#显示图像
cv2.imshow('Binary Image', binary)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图 42-8 表示图像二值化处理后的效果图。

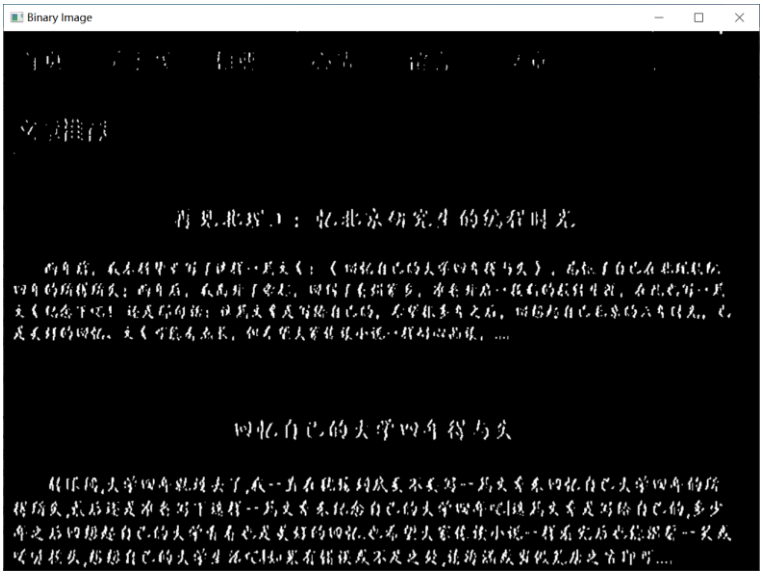


图 42-8 二值化处理结果

6.形态学处理

图像的腐蚀（Erosion）和膨胀（Dilation）是两种基本的形态学运算，主要用来寻找图像中的极小区域和极大区域。其中腐蚀类似于“领域被蚕食”，它将图像中的高亮区域或白色部分进行缩减细化，其运行结果比原图的高亮区域更小；膨胀类似于“领域扩张”，它将图像中的高亮区域或白色部分进行扩张，其运行结果比原图的高亮区域更大。

腐蚀的运算符是“-”，其定义如下：

$$A\text{-}B=\{x|B_x \subseteq A\} \tag{42-6}$$

该公式表示图像 A 用卷积模板 B 来进行腐蚀处理，通过模板 B 与图像 A 进行卷积计算，得出 B 覆盖区域的像素点最小值，并用这个最小值来替代参考点的像素值。如图 42-9 所示，将左边的原始图像 A 腐蚀处理为右边的效果图 A-B。

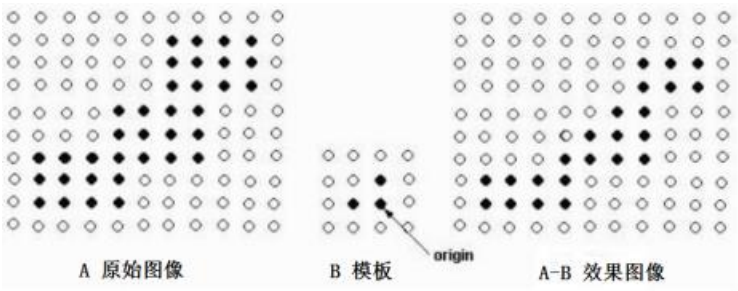


图 42-9 腐蚀处理原理图

膨胀的运算符是“⊕”，其定义如下：

$$A \oplus B=\{x|B_x \cap A \neq \Theta\} \tag{42-7}$$

该公式表示用 B 来对图像 A 进行膨胀处理，其中 B 是一个卷积模板，其形状可以为正方形或圆形，通过模板 B 与图像 A 进行卷积计算，扫描图像中的每

一个像素点，用模板元素与二值图像元素做“与”运算，如果都为 0，那么目标像素点为 0，否则为 1。从而计算 B 覆盖区域的像素点最大值，并用该值替换参考点的像素值实现图像膨胀。图 42-10 是将左边的原始图像 A 膨胀处理为右边的效果图 A⊕B。

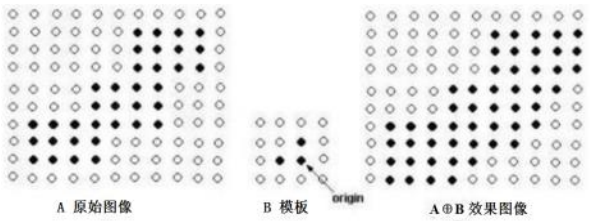


图 42-10 膨胀处理原理图

图 42-11 和图 42-12 是经过膨胀和腐蚀处理后的效果图，图像经过腐蚀处理过滤掉文字部分，再通过膨胀处理提取图像的背景轮廓。

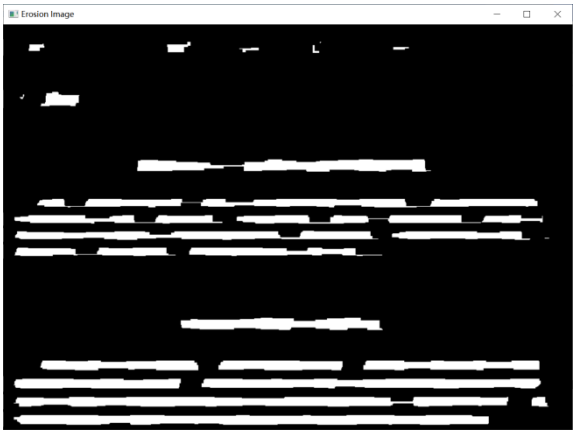


图 42-11 图像腐蚀处理

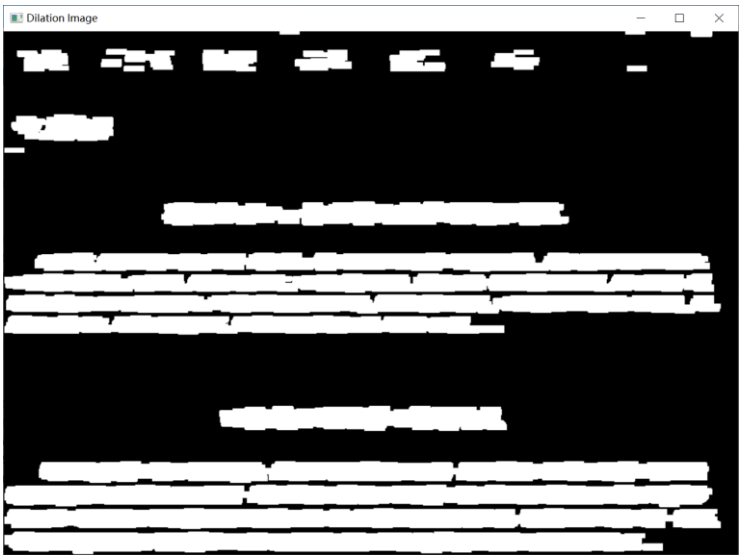


图 42-12 图像膨胀处理

7.文字区域提取

经过图像腐蚀和膨胀处理后，得到如图 10 所示的效果图，它将图像的背景轮廓提取出来，接着利用文字提取算法识别目标文字。算法流程如下：

- 用腐蚀膨胀处理后的图像减去二值化处理图像，提取图像中的文字轮廓。
- 通过开运算处理去除图像的噪声并保留文字内容。
- 通过闭运算处理过滤掉图像文字中的黑点噪声，提取更清晰的文字轮廓。

完整代码如下：

```
# -*- coding: utf-8 -*-

# By:Eastmount

import cv2

import numpy as np

import matplotlib.pyplot as plt
```

```
#读取原始图像
img = cv2.imread("word.png" )

#转换成灰度图像
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#中值滤波去除噪声
median = cv2.medianBlur(gray, 3)

#图像直方图均衡化
equalize = cv2.equalizeHist(median)

#Sobel 算子锐化处理
sobel = cv2.Sobel(median, cv2.CV_8U, 1, 0, ksize = 3)

#图像二值化处理
ret, binary = cv2.threshold(sobel, 0, 255,

cv2.THRESH_OTSU+cv2.THRESH_BINARY)
```

```
#膨胀和腐蚀处理 设置膨胀和腐蚀操作的核函数
element1 =
cv2.getStructuringElement(cv2.MORPH_RECT, (30, 9))
element2 =
cv2.getStructuringElement(cv2.MORPH_RECT, (24, 6))

#膨胀突出轮廓
dilation = cv2.dilate(binary, element2, iterations = 1)

#腐蚀去掉细节
erosion = cv2.erode(dilation, element1, iterations = 1)

#查找文字轮廓
region = []
contours, hierarchy = cv2.findContours(erosion,
                                       cv2.RETR_TREE,

cv2.CHAIN_APPROX_SIMPLE)

#筛选面积
for i in range(len(contours)):
```

```
#遍历所有轮廓
cnt = contours[i]

#计算轮廓面积
area = cv2.contourArea(cnt)

#寻找最小矩形
rect = cv2.minAreaRect(cnt)

#轮廓的四个点坐标
box = cv2.boxPoints(rect)
box = np.int0(box)

# 计算高和宽
height = abs(box[0][1] - box[2][1])
width = abs(box[0][0] - box[2][0])

#过滤太细矩形
if(height > width * 1.5):
    continue
```

```
        region.append(box)


#定位的文字用绿线绘制轮廓
for box in region:
        print(box)
        cv2.drawContours(img, [box], 0, (0, 255, 0), 2)


#显示图像
cv2.imshow('Gray Image', gray)
cv2.imshow('Median Blur', median)
cv2.imshow('Sobel Image', sobel)
cv2.imshow('Binary Image', binary)
cv2.imshow('Dilation Image', dilation)
cv2.imshow('Erosion Image', erosion)
cv2.imshow('Result Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

最终输出结果如图 42-13 所示，它是调用 findContours()函数寻找轮廓，并过滤掉面积异常区域，采用函数 drawContours()绘制文字轮廓，最终输出图像有效地将原图中所有文字区域定位并提取出来。

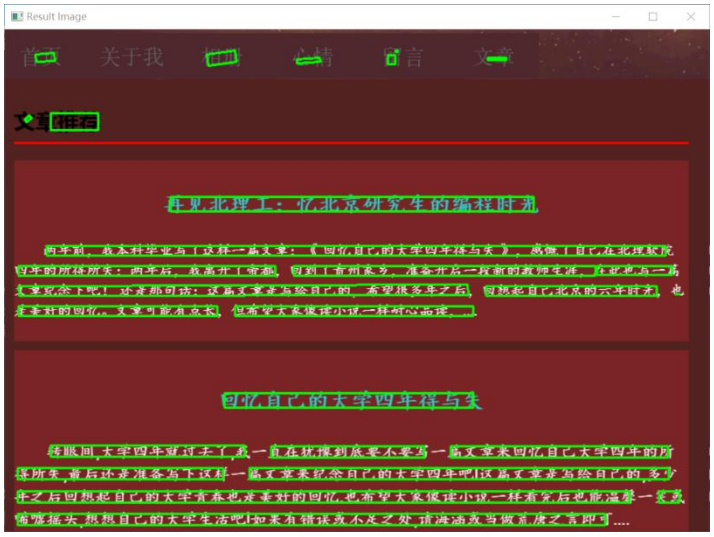


图 42-13 图像文字提取处理

8.总结

本文讲解了文字图像区域定位及提取案例，主要采用 OpenCV 图像处理算法组合实现，包括图像灰度处理、图像平滑处理、图像增强处理、文字边缘提取、阈值分割处理、形态学处理和文字区域提取，这是非常经典的案例。此外，该方法是图像分割和图像识别前的重要环节，可以广泛应用于文字识别、车牌提取、区域定位等领域。

参考文献：

[1] Eastmount. [图像处理] Python+OpenCV 实现车牌区域识别及 Sobel 算子.

https://blog.csdn.net/Eastmount/article/details/81461679.

[2] 杨秀璋，等. 一种基于水族濒危文字的图像增强及识别方法研究[J]. 计算机科学, 2019.

第 43 篇 OpenCV 实现车牌检测及区域识别

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前文介绍了文字图像区域定位和提取分析，但是提取的效果仍然存在一定的误差。这篇文章将详细讲解车牌检测及区域识别，通过融合车牌颜色更精准识别汽车的车牌，并分割对应的车牌字母或数字。图像识别(Image Recognition) 是通过算法和函数提取像素中的某些特征，并对图像进行识别和分类的过程。本文提供的案例将为读者提供深入的理解，希望您喜欢。

1.车牌预处理及区域定位

前文介绍了 OpenCV 识别文字区域的过程，这篇文章将结合颜色识别车牌。其核心过程包括九个步骤。

（1）形状变换和压缩图像

将图像转换成规定的大小，便于后续识别处理。核心代码如下，

```
m = 400*img.shape[0]/img.shape[1]
img                                                    =
```

```
cv2.resize(img,(400,int(m)),interpolation=cv2.INTER_CUBIC)
```

（2）灰度转换

将彩色图像转换为灰度图。核心代码如下：

```
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

（3）灰度拉伸

通过自定义 stretch 函数实现图像灰度拉伸，旨在改善图像的对比图，将灰度拉伸到整个 0 至 255 区间，从而增强对比度。本文采用的灰度拉伸公式如下所示：

$$I(x,y) = \frac{I(x,y) - Imin}{Imax - Imin}(MAX - MIN) + MIN$$

函数实现如下：

```
def stretch(img):
    maxi=float(img.max())
    mini=float(img.min())
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            img[i,j]=(255/(maxi-mini)*img[i,j]-(255*mini)/(maxi-mini))
    return img
```

（4）开运算

采用开运算（先腐蚀后膨胀）去除图像的噪声，核心代码如下：

```
kernel = np.zeros((h,w),np.uint8)
cv2.circle(kernel,(r,r),r,1,-1)
    openimg =
cv2.morphologyEx(stretchedimg,cv2.MORPH_OPEN,kernel)
```

（5）获取差分图像

将拉伸后的图像和开运算的图像做差分处理，即将两幅图像的对应像素值相减，以削弱图像的相似部分，突出显示图像的变化部分。

```
diffimg = cv2.absdiff(stretchedimg,openimg)
```

（6）图像二值化

图像二值化处理就是将图像上点的灰度置为 0 或 255，即整个图像呈现出明显的黑白效果。将 256 个亮度等级的灰度图像通过适当的阀值选取而获得仍然可以反映图像整体和局部特征的二值化图像。通常像素大于 127 设置为 255，小于设置为 0，这里提取图像的最大值和最小值，其平均值作为阈值。核心处理过程如下：

```
maxi=float(diffimg.max())
mini=float(diffimg.min())
x = maxi-((maxi-mini)/2)
ret,binary =
cv2.threshold(diffimg,x,255,cv2.THRESH_BINARY)
```

（7）Canny 边缘检测

边缘检测的目的是找到图像中亮度变化剧烈的像素点构成的集合，表现出来往往是轮廓。边缘检测有很多检测器，其中最常用的是 canny 边检测器，不容易受到噪声的影响。核心代码如下：

```
canny =
cv2.Canny(binary,binary.shape[0],binary.shape[1])
```

（8）消除异常区域

通过多次图像膨胀和图像腐蚀处理来消除较小区域，保留较大区域。具体操作依次经过闭运算、开运算和再次开运算。

- 开运算可以消除亮度较高的细小区域，在纤细点处分离物体，对于较大物体，可以在不明显改变其面积的情况下平滑其边界。
- 闭运算具有填充白色物体内细小黑色空洞的区域、连接临近物体、平滑边界等作用。

具体实现代码如下：

```
#闭运算
kernel = np.ones((5,19),np.uint8)
closeimg =
cv2.morphologyEx(canny,cv2.MORPH_CLOSE,kernel)
#开运算
openimg2 =
cv2.morphologyEx(closeimg,cv2.MORPH_OPEN,kernel)
#再次开运算
```

```
kernel = np.ones((11,5),np.uint8)
openimg3 =
cv2.morphologyEx(openimg2,cv2.MORPH_OPEN,kernel)
```

（9）基于颜色的车牌定位

通过自定义函数 LocateLicense，结合颜色特征来定位车牌位置并消除小区域。具体流程包括：

- 调用函数 cv2.findContours 定位车牌号
- 寻找矩形轮廓（找出轮廓的左上点和右下点）
- 计算面积和长度比
- 选出可能存在的面积最大的 3 个区域
- 使用颜色识别判断找出最像车牌的区域
- 根据阈值构建掩膜并确定最终车牌区域
- 定位并显示最终识别的车牌

完整代码如下所示：

```
# -*- coding: utf-8 -*-
# By:Eastmount
import cv2
import numpy as np
import matplotlib.pyplot as plt

"""
```

```
函数：stretch
功能：图像灰度拉伸
参数：原始图像
"""
def stretch(img):
    maxi=float(img.max())
    mini=float(img.min())
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            img[i,j]=(255/(maxi-mini)*img[i,j]-
(255*mini)/(maxi-mini))
    return img

"""
函数：LocateLicense
功能：结合颜色定位车牌图像
参数：处理后的图像、原始图像
"""
def LocateLicense(img, sourceimg):
    #定位车牌号
    contours,hierarchy = cv2.findContours(img,
```

```
cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)
    #找出最大的三个区域
    block = []
    for contour in contours:
        #寻找矩形轮廓（找出轮廓的左上点和右下点）
        y,x = [],[]
        for p in contour:
            y.append(p[0][0])
            x.append(p[0][1])
        r = [min(y),min(x),max(y),max(x)]

        #计算面积和长度比
        a = (r[2]-r[0])*(r[3]-r[1])    #面积
        s = (r[2]-r[0])*(r[3]-r[1])    #长度比
        block.append([r,a,s])

    #选出面积最大的 3 个区域
    block = sorted(block,key=lambda b: b[1])[-3:]
```

```
#使用颜色识别判断找出最像车牌的区域
maxweight,maxindex = 0,-1
for i in range(len(block)):
    b = sourceimg[block[i][0][1]:block[i][0][3],
                  block[i][0][0]:block[i][0][2]]
    #BGR 转 HSV
    hsv = cv2.cvtColor(b,cv2.COLOR_BGR2HSV)
    #蓝色车牌的范围
    lower = np.array([100,50,50])
    upper = np.array([140,255,255])
    #根据阈值构建掩膜
    mask = cv2.inRange(hsv,lower,upper)
    #统计权值
    w1 = 0
    for m in mask:
        w1 += m/255
    w2 = 0
    for n in w1:
        w2 += n
    #选出最大权值的区域
```

```
            if w2 > maxweight:
                maxindex = i
                maxweight = w2
    return block[maxindex][0]


"""
函数：ShowImage
功能：显示图像
参数：9 种处理步骤的图像
"""
def
ShowImage(img,gray,stretchedimg,openimg,diffimg,binar
y,canny,openimg3,result):
    titles = ['Source Image','Gray Image', 'Stretched Image',
'Open Image',
              'Diff Image', 'Binary Image', 'Canny Image',
'Locate Image', 'Result Image']
    images = [img,gray,stretchedimg,openimg,
                diffimg,binary,canny,openimg3,result]
    for i in range(9):
        plt.subplot(3,3,i+1),plt.imshow(images[i],'gray')
```

```
        plt.title(titles[i])
        plt.xticks([]),plt.yticks([])
    plt.show()

"""
函数：ImagePreprocessing
功能：图像预处理
参数：原始图像
"""
def ImagePreprocessing(img):

    source = img

    #1.形状变换和压缩图像
    m = 400*img.shape[0]/img.shape[1]
    img = cv2.resize(img,(400,int(m)),interpolation=cv2.INTER_CUBIC)

    #2.BGR 转换为灰度图像
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
    #3.灰度拉伸
    stretchedimg = stretch(gray)

    #4.开运算：去除噪声
    r = 16
    h = w = r*2+1
    kernel = np.zeros((h,w),np.uint8)
    cv2.circle(kernel,(r,r),r,1,-1)
    openimg =
cv2.morphologyEx(stretchedimg,cv2.MORPH_OPEN,ker
nel)

    #5.获取差分图：两幅图像做差
    diffimg = cv2.absdiff(stretchedimg,openimg)

    #6.图像二值化
    maxi=float(diffimg.max())
    mini=float(diffimg.min())
    x = maxi-((maxi-mini)/2)
    ret,binary =
```

```
cv2.threshold(diffimg,x,255,cv2.THRESH_BINARY)


    #7.Canny 边缘检测
    canny                                                =
cv2.Canny(binary,binary.shape[0],binary.shape[1])


    #8.消除小区域保留大区域
    #闭运算
    kernel = np.ones((5,19),np.uint8)
    closeimg                                             =
cv2.morphologyEx(canny,cv2.MORPH_CLOSE,kernel)
    #开运算
    openimg2                                             =
cv2.morphologyEx(closeimg,cv2.MORPH_OPEN,kernel)
    #再次开运算
    kernel = np.ones((11,5),np.uint8)
    openimg3                                             =
cv2.morphologyEx(openimg2,cv2.MORPH_OPEN,kernel)


    #9.结合颜色定位车牌位置并消除小区域
    rect = LocateLicense(openimg3,img)
```

```
    print(rect)


    #10.框出车牌号
    result                                                    =
cv2.rectangle(img,(rect[0],rect[1]),(rect[2],rect[3]),(0,255,
0),2)
    cv2.imshow('afterimg',result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


    #显示图像
    ShowImage(source,gray,stretchedimg,openimg,
              diffimg,binary,canny,openimg3,result)


    return rect,img


#-------------------------------------------
----------------------------
#主函数
if __name__=='__main__':
    #读取图片
```

```
imagePath = 'car.png'
img = cv2.imread(imagePath, cv2.IMREAD_COLOR)


#图像预处理：车牌位置 处理图像
rect, source = ImagePreprocessing(img)
```

运行结果如下图所示，图 43-1 分别显示各个阶段的处理效果图。分别包括原始图像、灰度图像、拉伸图像、开运算图像、差分处理图像、二值化图像、Canny 边缘检测图像、定位区域和识别车牌。

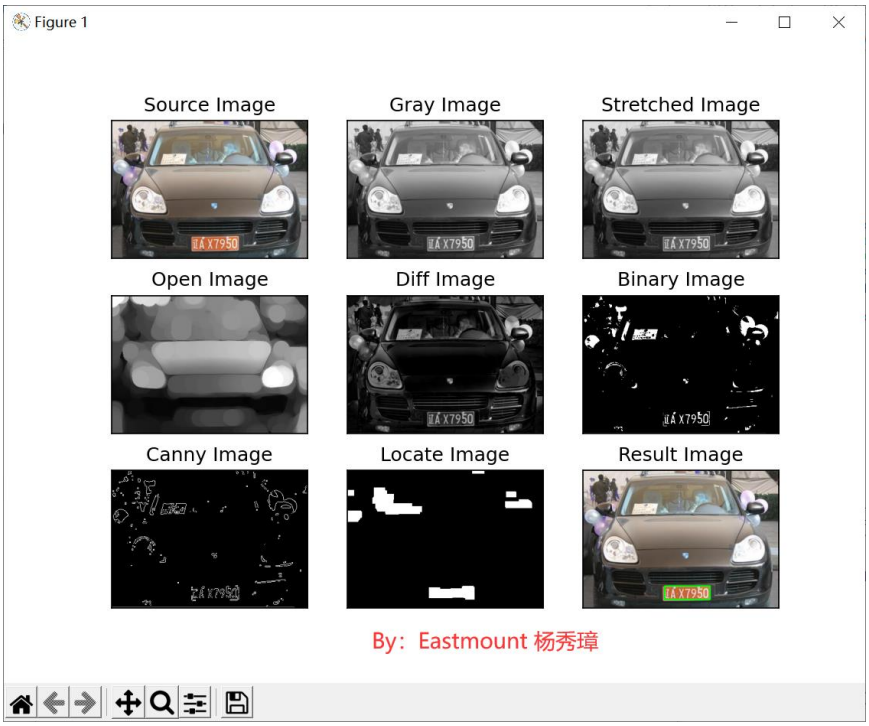


图 43-1 基于颜色特征的车牌识别预处理

最终识别的车牌如图 43-2 所示。

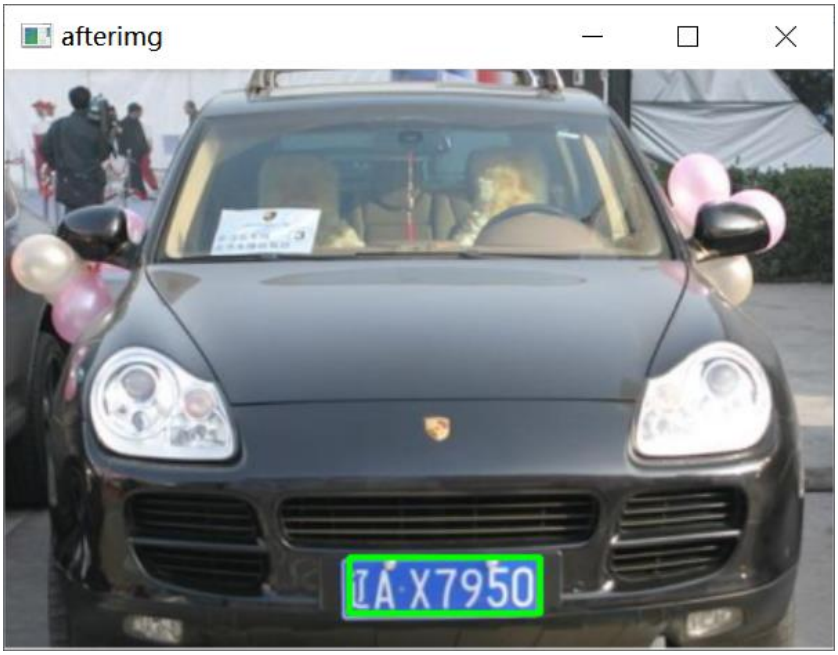


图 43-2 识别车牌

图 43-3 显示 SUV 车型对应车牌的识别效果。

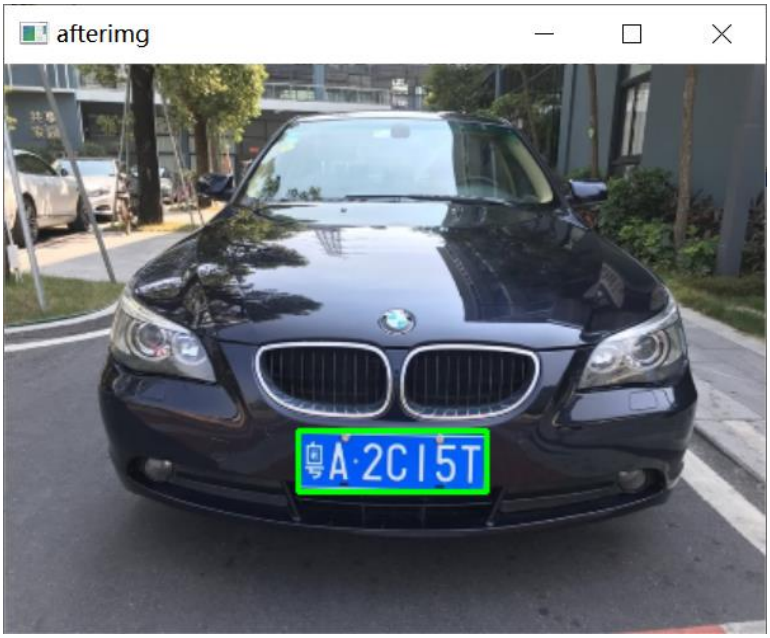


图 43-3 SUV 车型识别的车牌

下图展示其预处理过程。

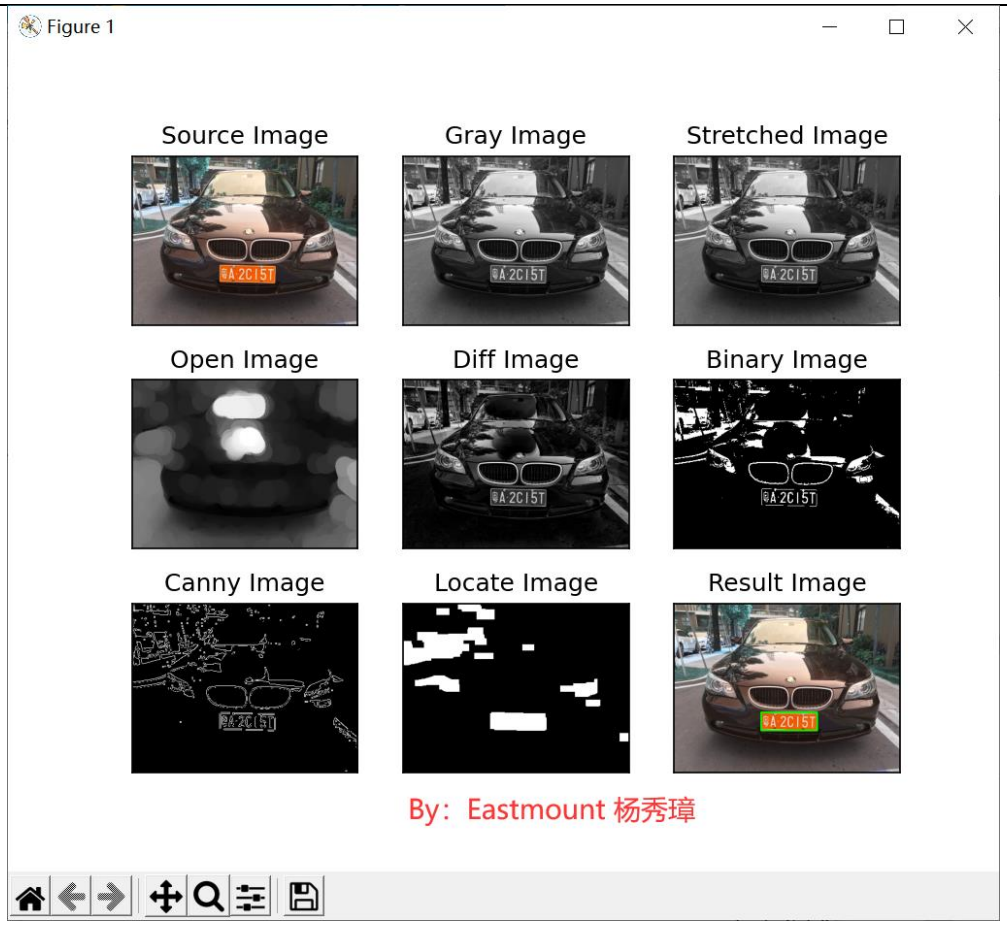


图 43-4 基于颜色特征的车牌识别预处理

2.车牌图像分割

当我们确定了车牌区域的四个坐标点，就可以对其进行分割，将车牌识别出来。对应的核心代码如下：

```
'''
函数：CutLicense
功能：图像分割
参数：预处理后图像 识别区域
'''
```

```
def CutLicense(result, rect):
    CutImg = result[rect[1]:rect[3], rect[0]:rect[2]]
    return CutImg


#主函数
if __name__=='__main__':

    #读取图片
    imagePath = 'car-03.png'
    img = cv2.imread(imagePath, cv2.IMREAD_COLOR)

    #图像预处理：车牌位置 处理图像
    rect, result = ImagePreprocessing(img)

    #车牌图像分割
    cutimg = CutLicense(result, rect)
    cv2.imshow('cutimg', cutimg)
    cv2.imwrite('save-'+imagePath, cutimg)
```

运行结果如下图所示。

图 43-5 车牌分割

3.车牌上下边缘去噪

当我们分割车牌后，通常还需要对车牌的噪声进行过滤，比如车牌中的螺丝，再如复杂环境下的车牌去雾。结合“苹果的芳菲”老师的文章，我国汽车牌照一般由七个字符和一个点组成（如图 43-6 所示），车牌字符的高度和宽度是固定的，分别为 90mm 和 45mm，七个字符之间的距离也是固定的 12mm，中间分割符圆点的直径是 10mm，但是真实车牌图像会因为透视原因造成字符间的距离变化。在民用车牌中，字符排列位置遵循以下规律：

- 第一个字符通常是我国各省区的简称，共 31 个，用汉字表示；
- 第二个字符通常是发证机关的代码号；
- 最后五个字符由英文字母和数字组合而成，字母是 24 个大写字母（除去 I 和 O）的组合，数字用“0-9”之间的数字表示。

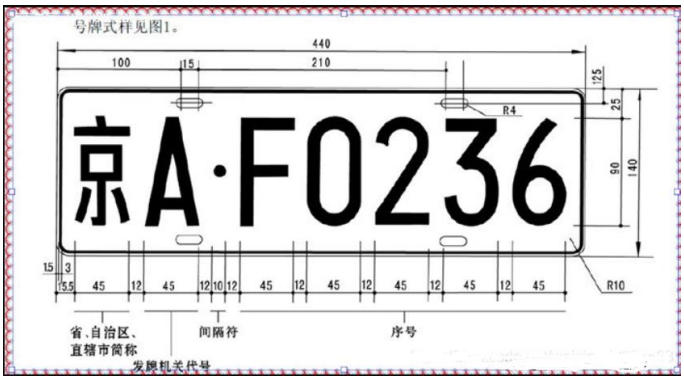


图 43-6 车牌示例

接下来我们需要对车牌上下边缘进行去噪。主要是去除车牌上下不规范的边界区域，以及过滤铆螺母的位置。核心代码如下：

```
"""
函数：FindWaves
功能：寻找波峰：通过设定的阈值和图片直方图寻找波峰，从而分别字符
参数：阈值 图像直方图
"""
def FindWaves(threshold, histogram):
    up_point = -1      # 上升点
    is_peak = False
    if histogram[0] > threshold:
        up_point = 0
        is_peak = True
    wave_peaks = []

    for i, x in enumerate(histogram):
        if is_peak and x < threshold:
            if i - up_point > 2:
                is_peak = False
                wave_peaks.append((up_point, i))
        elif not is_peak and x >= threshold:
            is_peak = True
```

```
                up_point = i
        if is_peak and up_point != -1 and i - up_point > 4:
            wave_peaks.append((up_point, i))

    return wave_peaks

"""
函数：RemoveUpdownBorder
功能：图像上下边缘去噪，去除车牌上下无用的边缘部分并确定上下边界
参数：分割后的图像
"""
def RemoveUpdownBorder(img):
    #灰度转换
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #二值化处理
    ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    #数组的每一行求和
```

```
    row_histogram = np.sum(binary, axis=1)
    row_min = np.min(row_histogram)
    row_average = np.sum(row_histogram) / binary.shape[0]
    row_threshold = (row_min + row_average) / 2

    #寻找波峰：通过设定的阈值和图片直方图寻找波峰，从而分别字符
    wave_peaks = FindWaves(row_threshold, row_histogram)
    print(wave_peaks)

    #挑选跨度最大的波峰
    wave_span = 0.0
    selected_wave = []
    for wave_peak in wave_peaks:
        span = wave_peak[1] - wave_peak[0]
        if span > wave_span:
            wave_span = span
            selected_wave = wave_peak
    result = binary[selected_wave[0]:selected_wave[1], :]
```

```
    return   result


#主函数
if __name__=='__main__':


    #读取图片
    imagePath = 'car-03.png'
    img = cv2.imread(imagePath, cv2.IMREAD_COLOR)


    #图像预处理：车牌位置 处理图像
    rect, result = ImagePreprocessing(img)


    #车牌图像分割
    cutimg = CutLicense(result, rect)
    cv2.imshow('cutimg', cutimg)
    cv2.imwrite('save-'+imagePath, cutimg)


    #去除上下边缘噪声
    cleanimg = RemoveUpdownBorder(cutimg)
    cv2.imshow('cleanimg', cleanimg)
    cv2.imwrite('save-qz-'+imagePath, cleanimg)
```

注意，这里自定义了一个 FindWaves 函数，重要用于寻找波峰：通过设定的阈值和图片直方图寻找波峰，从而分别字符。其原因是：

- 我国车牌的灰度分布呈现出连续的“波谷-波峰-波谷”分布，由于我国车牌颜色单一，字符直线排列。
- 车牌直方图呈现出双峰状的特点，即车牌直方图中可以看到双个波峰，比如图片中输出的波峰为“[(0, 4), (8, 26)]”。

此时的输出结果如图 43-7 所示。

图 43-7 去噪后的车牌

4.车牌最终识别

最后，需要对车牌的每个字符进行分割，本文从左往右检测字符，若宽度（end－start）大于 5 则认为是字符，将其裁剪并保存下来。更好的方法还有一个基于神经网络的字符识别过程，从而自动识别车牌对应的文字，这里请读者结合后续的文本分类综合实现。该部分核心代码如下：

```
"""
函数：FindEnd
功能：寻找结束位置
"""
def FindEnd(start, arg, black, white, width, black_max,
white_max):
```

```
        end = start + 1
        for m in range(start + 1, width - 1):
            if (black[m] if arg else white[m]) > (0.95*black_max
if arg else 0.95*white_max):
                end = m
                break
        return end


"""
函数：CharsSegmentation
功能：分割字符：左往右开始检测匹配字符，若宽度（end - start）
大于 5 则认为是字符
参数：去噪处理后的图像
"""
def CharsSegmentation(img):
        #list 记录每一列的黑/白色像素总和
        white, black = [], []
        height, width = img.shape
        white_max = 0       #仅保存每列，取列中白色最多的像素总数
        black_max = 0       #仅保存每列，取列中黑色最多的像素总数
```

```
#计算每一列的黑白像素总和
for i in range(width):
    line_white = 0      #这一列白色总数
    line_black = 0      #这一列黑色总数
    for j in range(height):
        if img[j][i] == 255:
            line_white += 1
        if img[j][i] == 0:
            line_black += 1
    white_max = max(white_max, line_white)
    black_max = max(black_max, line_black)
    white.append(line_white)
    black.append(line_black)

#arg 为 true 表示黑底白字  False 为白底黑字
arg = True
if black_max < white_max:
    arg = False

#分割车牌字符
n = 1
```

```
    while n < width - 2:
        n += 1
        #判断是白底黑字还是黑底白字  0.05 参数对应上面的 0.95 可作调整
        if (white[n] if arg else black[n]) > (0.05 * white_max if arg else 0.05 * black_max):  # 这点没有理解透彻
            start = n
            end = FindEnd(start, arg, black, white, width, black_max, white_max)
            n = end
            if end - start > 5 or end > (width * 3 / 7):
                cropImg = img[0:height, start-1:end+1]
                #对分割出的数字、字母进行 resize 并保存
                cropImg = cv2.resize(cropImg, (34, 56))
                cv2.imwrite("chars" + '\\{}.bmp'.format(n), cropImg)
                cv2.imshow('Char_{}'.format(n), cropImg)

#------------------------------------------------------------------------
#主函数
```

```
if __name__=='__main__':

    #读取图片
    imagePath = 'car-03.png'
    img = cv2.imread(imagePath, cv2.IMREAD_COLOR)

    #图像预处理：车牌位置 处理图像
    rect, result = ImagePreprocessing(img)

    #车牌图像分割
    cutimg = CutLicense(result, rect)
    cv2.imshow('cutimg', cutimg)
    cv2.imwrite('save-'+imagePath, cutimg)

    #去除上下边缘噪声
    cleanimg = RemoveUpdownBorder(cutimg)
    cv2.imshow('cleanimg', cleanimg)
    cv2.imwrite('save-qz-'+imagePath, cleanimg)

    #分割字符
    CharsSegmentation(cleanimg)
```

```
    cv2.waitKey(0)

    cv2.destroyAllWindows()
```

最终运行结果如图 43-8 所示，将不同的字符进行分割。

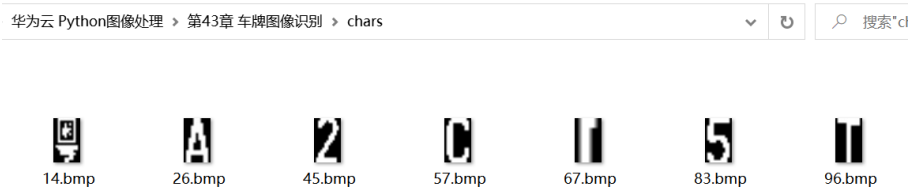


图 43-8 识别车牌字符

对应的原图如下所示。

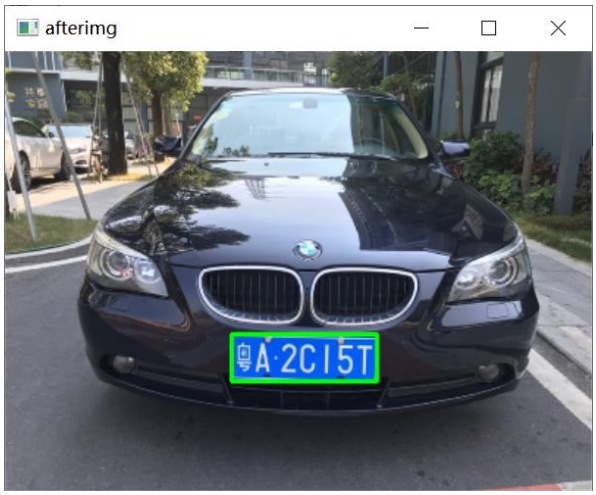


图 43-9 识别车牌

5.完整代码

最后给出完整代码：

```
# -*- coding: utf-8 -*-

# By:Eastmount

import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt


"""
函数：stretch
功能：图像灰度拉伸
参数：原始图像
"""
def stretch(img):
    maxi=float(img.max())
    mini=float(img.min())
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            img[i,j]=(255/(maxi-mini)*img[i,j]-
(255*mini)/(maxi-mini))
    return img


"""
函数：LocateLicense
功能：结合颜色定位车牌图像
参数：处理后的图像、原始图像
```

```
"""
def LocateLicense(img, sourceimg):
    #定位车牌号
    contours,hierarchy = cv2.findContours(img,


cv2.RETR_EXTERNAL,


cv2.CHAIN_APPROX_SIMPLE)
    #找出最大的三个区域
    block = []
    for contour in contours:
        #寻找矩形轮廓（找出轮廓的左上点和右下点）
        y,x = [],[]
        for p in contour:
            y.append(p[0][0])
            x.append(p[0][1])
        r = [min(y),min(x),max(y),max(x)]


        #计算面积和长度比
        a = (r[2]-r[0])*(r[3]-r[1])    #面积
        s = (r[2]-r[0])*(r[3]-r[1])    #长度比
```

```
        block.append([r,a,s])


#选出面积最大的 3 个区域
block = sorted(block,key=lambda b: b[1])[-3:]


#使用颜色识别判断找出最像车牌的区域
maxweight,maxindex = 0,-1
for i in range(len(block)):
        b = sourceimg[block[i][0][1]:block[i][0][3],
                      block[i][0][0]:block[i][0][2]]
        #BGR 转 HSV
        hsv = cv2.cvtColor(b,cv2.COLOR_BGR2HSV)
        #蓝色车牌的范围
        lower = np.array([100,50,50])
        upper = np.array([140,255,255])
        #根据阈值构建掩膜
        mask = cv2.inRange(hsv,lower,upper)
        #统计权值
        w1 = 0
        for m in mask:
            w1 += m/255
```

```
            w2 = 0
            for n in w1:
                w2 += n
            #选出最大权值的区域
            if w2 > maxweight:
                maxindex = i
                maxweight = w2
        return block[maxindex][0]


"""
函数：ShowImage
功能：显示图像
参数：9 种处理步骤的图像
"""
def
ShowImage(img,gray,stretchedimg,openimg,diffimg,binar
y,canny,openimg3,result):
    titles = ['Source Image','Gray Image', 'Stretched Image',
'Open Image',
              'Diff Image', 'Binary Image', 'Canny Image',
'Locate Image', 'Result Image']
```

```
    images = [img,gray,stretchedimg,openimg,
                diffimg,binary,canny,openimg3,result]
    for i in range(9):
        plt.subplot(3,3,i+1),plt.imshow(images[i],'gray')
        plt.title(titles[i])
        plt.xticks([]),plt.yticks([])
    plt.show()


"""
函数：ImagePreprocessing
功能：图像预处理
参数：原始图像
"""
def ImagePreprocessing(img):


    source = img


    #1.形状变换和压缩图像
    m = 400*img.shape[0]/img.shape[1]
    img                                                    =
cv2.resize(img,(400,int(m)),interpolation=cv2.INTER_CUB
```

```
IC)

    #2.BGR 转换为灰度图像
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    #3.灰度拉伸
    stretchedimg = stretch(gray)

    #4.开运算：去除噪声
    r = 16
    h = w = r*2+1
    kernel = np.zeros((h,w),np.uint8)
    cv2.circle(kernel,(r,r),r,1,-1)
    openimg =
cv2.morphologyEx(stretchedimg,cv2.MORPH_OPEN,ker
nel)

    #5.获取差分图：两幅图像做差
    diffimg = cv2.absdiff(stretchedimg,openimg)

    #6.图像二值化
```

```
    maxi=float(diffimg.max())
    mini=float(diffimg.min())
    x = maxi-((maxi-mini)/2)
    ret,binary =
cv2.threshold(diffimg,x,255,cv2.THRESH_BINARY)

    #7.Canny 边缘检测
    canny =
cv2.Canny(binary,binary.shape[0],binary.shape[1])

    #8.消除小区域保留大区域
    #闭运算
    kernel = np.ones((5,19),np.uint8)
    closeimg =
cv2.morphologyEx(canny,cv2.MORPH_CLOSE,kernel)
    #开运算
    openimg2 =
cv2.morphologyEx(closeimg,cv2.MORPH_OPEN,kernel)
    #再次开运算
    kernel = np.ones((11,5),np.uint8)
    openimg3 =
```

```
cv2.morphologyEx(openimg2,cv2.MORPH_OPEN,kernel)

    #9.结合颜色定位车牌位置并消除小区域
    rect = LocateLicense(openimg3,img)
    print(rect)

    #10.框出车牌号
    result =
cv2.rectangle(img,(rect[0],rect[1]),(rect[2],rect[3]),(0,255,
0),2)
    cv2.imshow('afterimg',result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    #显示图像
    ShowImage(source,gray,stretchedimg,openimg,
              diffimg,binary,canny,openimg3,result)

    return rect,img

"""
```

```
函数：CutLicense
功能：图像分割
参数：预处理后图像  识别区域
"""
def CutLicense(result, rect):
    CutImg = result[rect[1]:rect[3], rect[0]:rect[2]]
    return CutImg


"""
函数：FindWaves
功能：寻找波峰：通过设定的阈值和图片直方图寻找波峰，从而分别字
符
参数：阈值  图像直方图
"""
def FindWaves(threshold, histogram):
    up_point = -1      #  上升点
    is_peak = False
    if histogram[0] > threshold:
        up_point = 0
        is_peak = True
    wave_peaks = []
```

```
    for i, x in enumerate(histogram):
        if is_peak and x < threshold:
            if i - up_point > 2:
                is_peak = False
                wave_peaks.append((up_point, i))
        elif not is_peak and x >= threshold:
            is_peak = True
            up_point = i
    if is_peak and up_point != -1 and i - up_point > 4:
        wave_peaks.append((up_point, i))

    return wave_peaks

"""
函数：RemoveUpdownBorder
功能：图像上下边缘去噪，去除车牌上下无用的边缘部分并确定上下边界
参数：分割后的图像
"""
def RemoveUpdownBorder(img):
```

```
    #灰度转换
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #二值化处理
    ret, binary = cv2.threshold(gray, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    #数组的每一行求和
    row_histogram = np.sum(binary, axis=1)
    row_min = np.min(row_histogram)
    row_average = np.sum(row_histogram) /
binary.shape[0]
    row_threshold = (row_min + row_average) / 2

    #寻找波峰：通过设定的阈值和图片直方图寻找波峰，从而分别字
符
    wave_peaks = FindWaves(row_threshold,
row_histogram)
    print(wave_peaks)

    #挑选跨度最大的波峰
```

```
    wave_span = 0.0
    selected_wave = []
    for wave_peak in wave_peaks:
        span = wave_peak[1] - wave_peak[0]
        if span > wave_span:
            wave_span = span
            selected_wave = wave_peak
    result = binary[selected_wave[0]:selected_wave[1], :]
    return  result

"""
函数：FindEnd
功能：寻找结束位置
"""
def FindEnd(start, arg, black, white, width, black_max,
white_max):
    end = start + 1
    for m in range(start + 1, width - 1):
        if (black[m] if arg else white[m]) > (0.95*black_max
if arg else 0.95*white_max):
            end = m
```

```
                break
    return end


"""
函数：CharsSegmentation
功能：分割字符：左往右开始检测匹配字符，若宽度（end - start）
大于 5 则认为是字符
参数：去噪处理后的图像
"""
def CharsSegmentation(img):
    #list 记录每一列的黑/白色像素总和
    white, black = [], []
    height, width = img.shape
    white_max = 0      #仅保存每列，取列中白色最多的像素总数
    black_max = 0      #仅保存每列，取列中黑色最多的像素总数

    #计算每一列的黑白像素总和
    for i in range(width):
        line_white = 0      #这一列白色总数
        line_black = 0      #这一列黑色总数
        for j in range(height):
```

```
            if img[j][i] == 255:
                line_white += 1
            if img[j][i] == 0:
                line_black += 1
        white_max = max(white_max, line_white)
        black_max = max(black_max, line_black)
        white.append(line_white)
        black.append(line_black)

    #arg 为 true 表示黑底白字  False 为白底黑字
    arg = True
    if black_max < white_max:
        arg = False

    #分割车牌字符
    n = 1
    while n < width - 2:
        n += 1
        #判断是白底黑字还是黑底白字  0.05 参数对应上面的 0.95
可作调整
        if (white[n] if arg else black[n]) > (0.05 * white_max
```

```
if arg else 0.05 * black_max):   # 这点没有理解透彻
                start = n
                end = FindEnd(start, arg, black, white, width,
black_max, white_max)
                n = end
                if end - start > 5 or end > (width * 3 / 7):
                    cropImg = img[0:height, start-1:end+1]
                    #对分割出的数字、字母进行 resize 并保存
                    cropImg = cv2.resize(cropImg, (34, 56))
                    cv2.imwrite("chars"  +  '\\{}.bmp'.format(n),
cropImg)
                    cv2.imshow('Char_{}'.format(n), cropImg)

#------------------------------------------
----------------------------
#主函数
if __name__=='__main__':

    #读取图片
    imagePath = 'car-03.png'
    img = cv2.imread(imagePath, cv2.IMREAD_COLOR)
```

```
#图像预处理：车牌位置 处理图像
rect, result = ImagePreprocessing(img)

#车牌图像分割
cutimg = CutLicense(result, rect)
cv2.imshow('cutimg', cutimg)
cv2.imwrite('save-'+imagePath, cutimg)

#去除上下边缘噪声
cleanimg = RemoveUpdownBorder(cutimg)
cv2.imshow('cleanimg', cleanimg)
cv2.imwrite('save-qz-'+imagePath, cleanimg)

#分割字符
CharsSegmentation(cleanimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

其它车牌的识别结果。

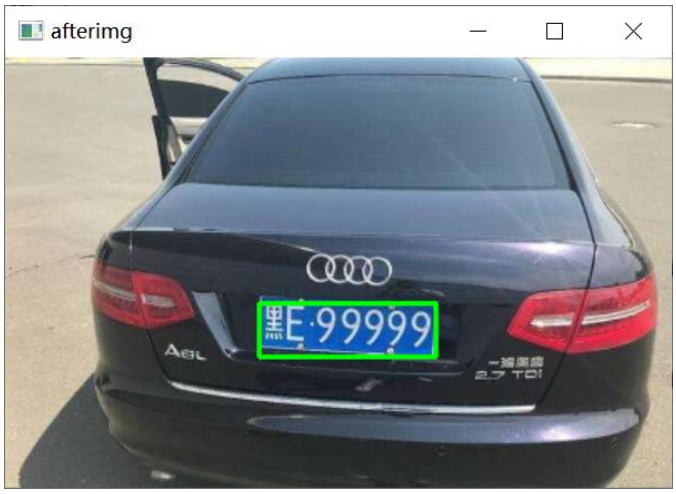


图 43-10 识别车牌

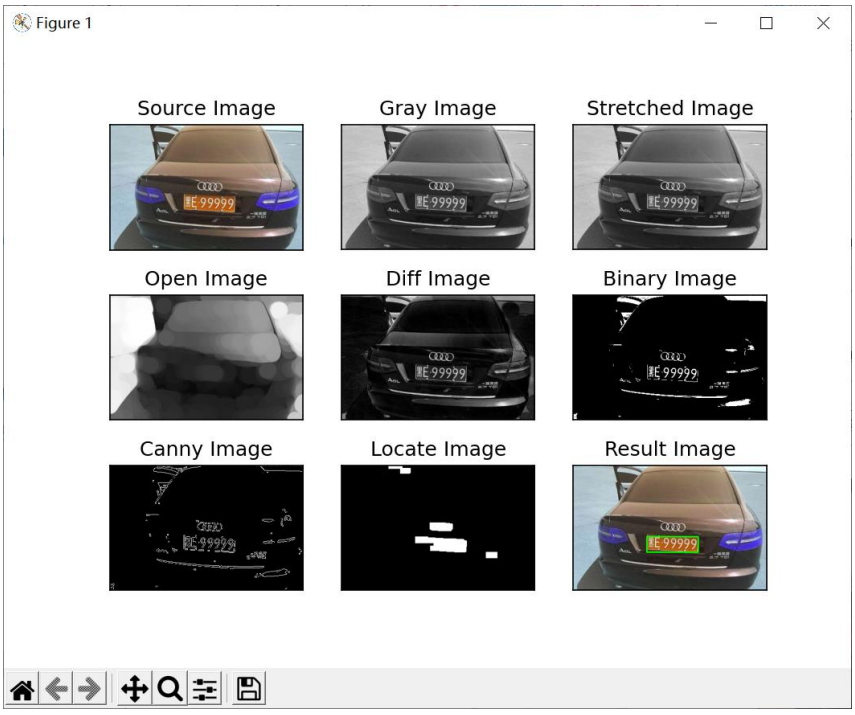


图 43-11 识别车牌处理过程

6.总结

本文详细讲解了车牌检测及区域识别，通过融合车牌颜色更精准识别汽车的

车牌，并分割对应的车牌字母或数字。主要采用 OpenCV 图像处理算法组合实现，包括图像灰度处理、图像平滑处理、图像增强处理、文字边缘提取、阈值分割处理、形态学处理和文字区域提取，这是非常经典的案例。

此外，该方法是图像分割和图像识别前的重要环节，可以广泛应用于文字识别、车牌提取、区域定位等领域。最后再次推荐大家结合后面的图像分类文章对分割后的车牌字符进行自动识别，识别出对应的车牌号码，加油喔！

参考文献：

[1] Eastmount. [图像处理] Python+OpenCV 实现车牌区域识别及 Sobel 算子. https://blog.csdn.net/Eastmount/article/details/81461679.

[2] 杨秀璋, 等. 一种基于水族濒危文字的图像增强及识别方法研究[J]. 计算机科学, 2019.

[3] 壳 壳 . opencv+python 车 牌 识 别 . https://www.cnblogs.com/kekexxr/p/11574589.html.

[4] 苹 果 の 芳 菲 . OpenCV 实 现 车 牌 定 位 和 字 符 分 割 . https://blog.csdn.net/weixin_43397593/article/details/102451527.

第 44 篇 OpenCV 快速实现人脸检测及视频人脸动态识别

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前文介绍了车牌检测及区域识别，通过融合车牌颜色更精准识别汽车的车牌，并分割对应的车牌字母或数字。这篇文章将详细讲解人脸检测的应用案例，通过 OpenCV 快速实现人脸检测，涉及图像、视频、摄像头。本文提供的案例将为读者提供深入的理解，也是最简单的人脸识别案例，希望您喜欢。

1.图像单人脸检测

人脸识别的常见步骤如下，如果想要将人脸准确地找出来，需要通过建立人脸模型，获取准确区分人脸的分类器，这里我们使用网上公开的扩展包或已经训练好的分类器。

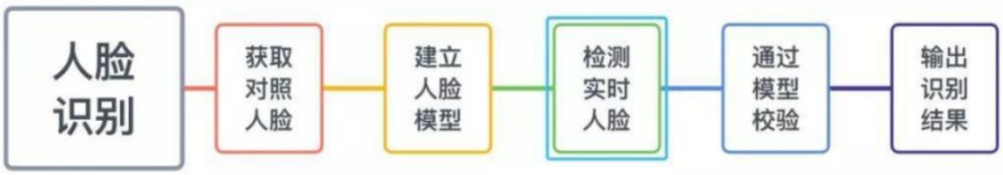


图 44-1 人脸识别基本流程

第一步，下载人脸识别算法，这里使用 OpenCV 发布在 github 上的代码。

下载地址如下：

- https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

我们将“haarcascade_frontalface_default.xml”文件下载至本地，后续调用它辅助我们进行人脸识别。

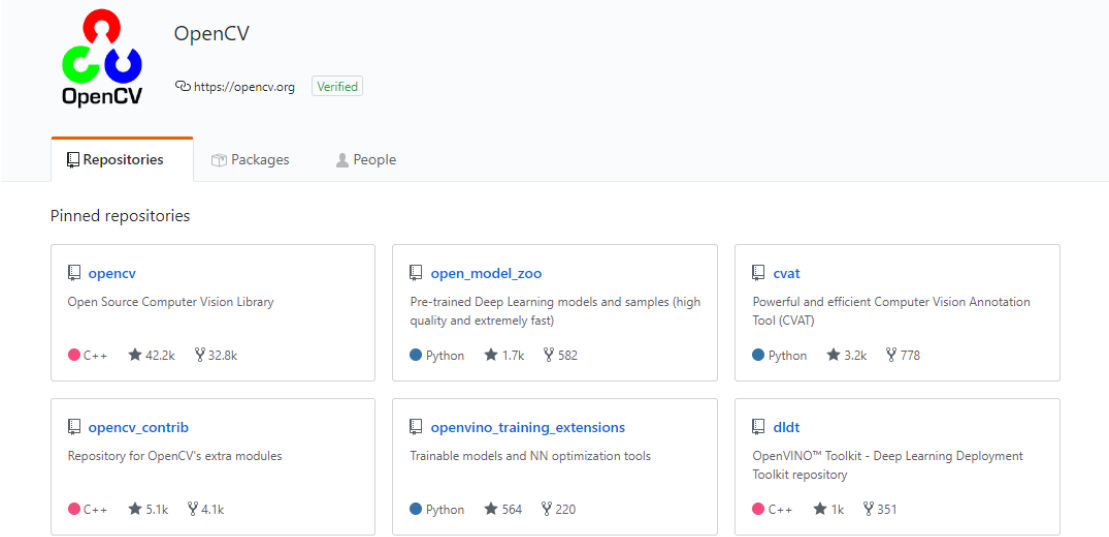


图 44-2 OpenCV 官方资源

下载过程如图 44-3 所示。

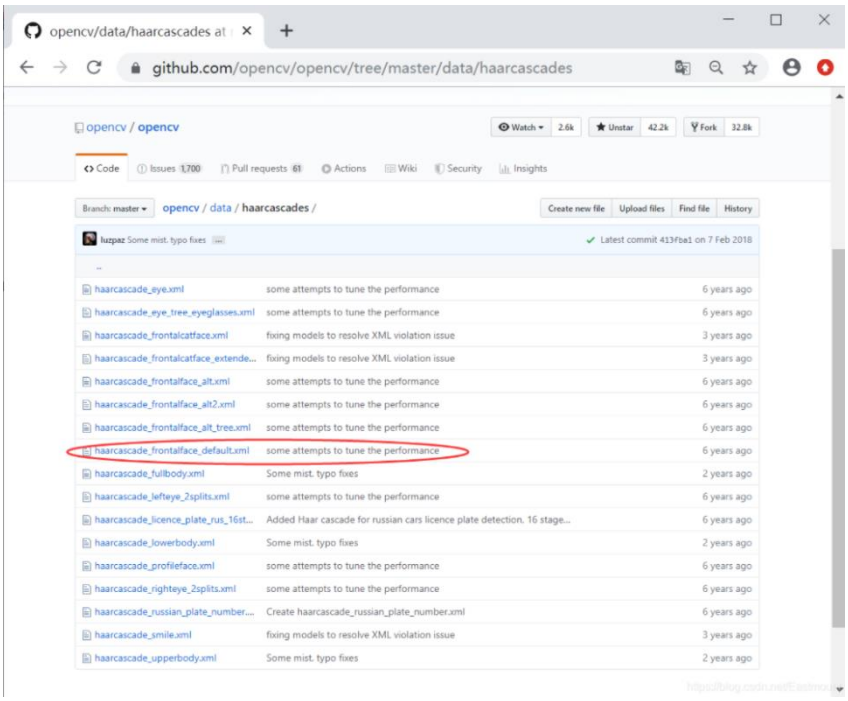


图 44-3 下载人脸识别文件

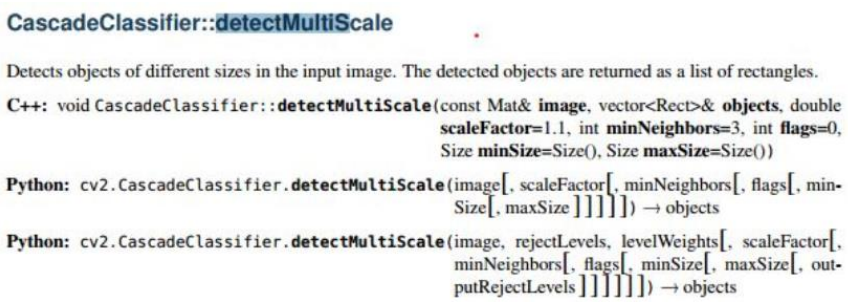
第二步，调用 cv2.CascadeClassifier()和 detectMultiScale()函数，根据人脸特征进行识别。

- **CascadeClassifier**：是 OpenCV 中人脸检测的一个级联分类器，既可以使用 Haar，也可以使用 LBP 特征。以 Haar 特征分类器为基础的对象检测技术是一种非常有效的技术。它是基于机器学习且使用大量的正负样本训练得到分类器。
- **Haar-like 矩形特征**：是用于物体检测的数字图像特征。这类矩形特征模板由两个或多个全等的黑白矩形相邻组合而成，而矩形特征值是白色矩形的灰度值的和减去黑色矩形的灰度值的和，矩形特征对一些简单的图形结构，如线段、边缘比较敏感。如果把这样的矩形放在一个非人脸区域，那么计算出的特征值应该和人脸特征值不一样，所以这些矩形就

是为了把人脸特征量化，以区分人脸和非人脸。

- **LBP**：是一种特征提取方式，能提取出图像的局部的纹理特征，最开始的 LBP 算子是在 3X3 窗口中，取中心像素的像素值为阀值，与其周围八个像素点的像素值比较，若像素点的像素值大于阀值，则此像素点被标记为 1，否则标记为 0。这样就能得到一个八位二进制的码，转换为十进制即 LBP 码，于是得到了这个窗口的 LBP 值，用这个值来反映这个窗口内的纹理信息。LBPH 是在原始 LBP 上的一个改进，在 opencv 支持下我们可以直接调用函数直接创建一个 LBPH 人脸识别的模型。比如：cv2.face.LBPHFaceRecognizer_create()。
- **detectMultiScale**：检测人脸算法，其参数：
  - image 表示要检测的输入图像
  - objects 表示检测到的人脸目标序列
  - scaleFactor 表示每次图像尺寸减小的比例
  - minNeighbors 表示每一个目标至少要被检测到 3 次才算是真的目标，因为周围的像素和不同的窗口大小都可以检测到人脸
  - minSize 表示目标的最小尺寸
  - minSize 表示目标的最大尺寸

算法原型如下图所示。



CascadeClassifier::detectMultiScale

Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

C++: void CascadeClassifier::**detectMultiScale**(const Mat& **image**, vector<Rect>& **objects**, double **scaleFactor**=1.1, int **minNeighbors**=3, int **flags**=0, Size **minSize**=Size(), Size **maxSize**=Size())

Python: cv2.CascadeClassifier.**detectMultiScale**(image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]]) → objects

Python: cv2.CascadeClassifier.**detectMultiScale**(image, rejectLevels, levelWeights[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize[, outputRejectLevels]]]]]]) → objects

图 44-4 函数原型

第三步，绘制矩形或圆形检测人脸，并将人脸识别的最终图像显示出来

完整代码如下所示：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2

# 读取原始图像
img = cv2.imread('yxz.png')

# 调用熟悉的人脸分类器 识别特征类型
# 人脸 - haarcascade_frontalface_default.xml
# 人眼 - haarcascade_eye.xm
# 微笑 - haarcascade_smile.xml
face_detect =
cv2.CascadeClassifier('haarcascade_frontalface_default.
xml')
```

```
# 检查人脸 按照 1.1 倍放到 周围最小像素为 5
face_zone = face_detect.detectMultiScale(img, scaleFactor=1.1, minNeighbors=5)
print ('识别人脸的信息：',face_zone)

# 绘制矩形和圆形检测人脸
for x, y, w, h in face_zone:
    # 绘制矩形人脸区域 thickness 表示线的粗细
    cv2.rectangle(img, pt1=(x, y), pt2=(x+w, y+h),color=[0,0,255], thickness=2)
    # 绘制圆形人脸区域 radius 表示半径
    cv2.circle(img, center=(x+w//2, y+h//2), radius=w//2, color=[0,255,0], thickness=2)

# 设置图片可以手动调节大小
cv2.namedWindow("Eastmount", 0)

# 显示图片
cv2.imshow("Eastmount", img)
```

```
# 等待显示 设置任意键退出程序
cv2.waitKey(0)
cv2.destroyAllWindows()
```

检测人脸如图 44-5 所示，同时输出“识别人脸的信息：[[318 115 151 151]]”。作者大一在北京十渡的照片，笑容灿烂啊！哈哈。

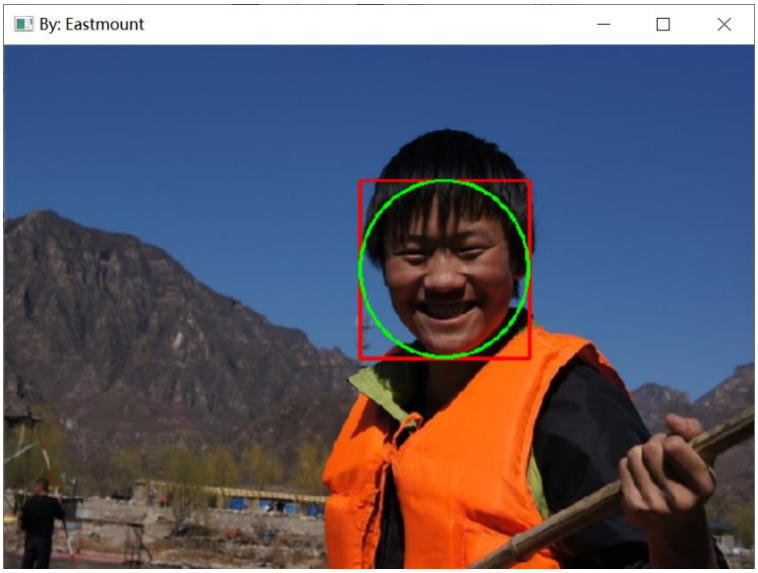


图 44-5 人脸识别结果

同样的方法可以检测图像处理经典的 Lena 图的人脸，如下所示。

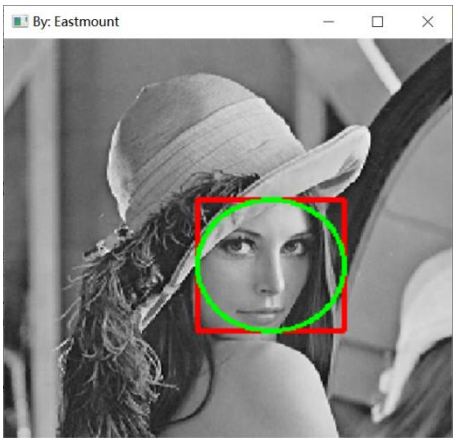


图 44-6 Lena 图的人脸识别结果

注意，此时的算法只能检测正脸，比如下图所示女神的侧脸就无法检测。

图 44-7 无法有效识别侧脸

在 github 提供的资源中，本篇文章主要介绍人脸识别，还有眼睛和微笑识别，对应的开源文件如下所示，它是通过调用熟悉的人脸分类器识别特征类型，推荐读者进行其它类型的尝试。

- 人脸：haarcascade_frontalface_default.xml
- 人眼：haarcascade_eye.xml
- 微笑 – haarcascade_smile.xml

2.图像多人脸检测

检测图像多张人脸，其方法和前面非常类似，基本流程如下：

- 将图像转换为灰度图像
- 调用 cv2.CascadeClassifier()和 detectMultiScale()函数，根据人脸识别特征

■ 绘制矩形或圆形检测人脸，并将人脸识别的最终图像显示出来

完整代码如下图所示：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2

# 读取原始图像
img = cv2.imread('test02.jpg')

# 调用熟悉的人脸分类器 识别特征类型
# 人脸 - haarcascade_frontalface_default.xml
# 人眼 - haarcascade_eye.xm
# 微笑 - haarcascade_smile.xml
face_detect =
cv2.CascadeClassifier('haarcascade_frontalface_default.
xml')

# 灰度处理
gray = cv2.cvtColor(img, code=cv2.COLOR_BGR2GRAY)

# 检查人脸 按照 1.1 倍放到 周围最小像素为 5
```

```
face_zone = face_detect.detectMultiScale(gray,
scaleFactor = 1.1, minNeighbors = 4) # maxSize = (55,55)
print ('识别人脸的信息：\n',face_zone)


# 绘制矩形和圆形检测人脸
for x, y, w, h in face_zone:
    # 绘制矩形人脸区域
    cv2.rectangle(img, pt1 = (x, y), pt2 = (x+w, y+h), color =
[0,0,255], thickness=2)
    # 绘制圆形人脸区域 radius 表示半径
    cv2.circle(img, center = (x + w//2, y + h//2), radius = w//2,
color = [0,255,0], thickness = 2)


# 设置图片可以手动调节大小
cv2.namedWindow("By: Eastmount", 0)


# 显示图片
cv2.imshow("By: Eastmount", img)


# 等待显示 设置任意键退出程序
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

如图 44-8 所示，将两人的脸庞识别出来。注意，该算法会根据人脸的不同尺寸，绘制不同人脸的矩形或圆形框出。

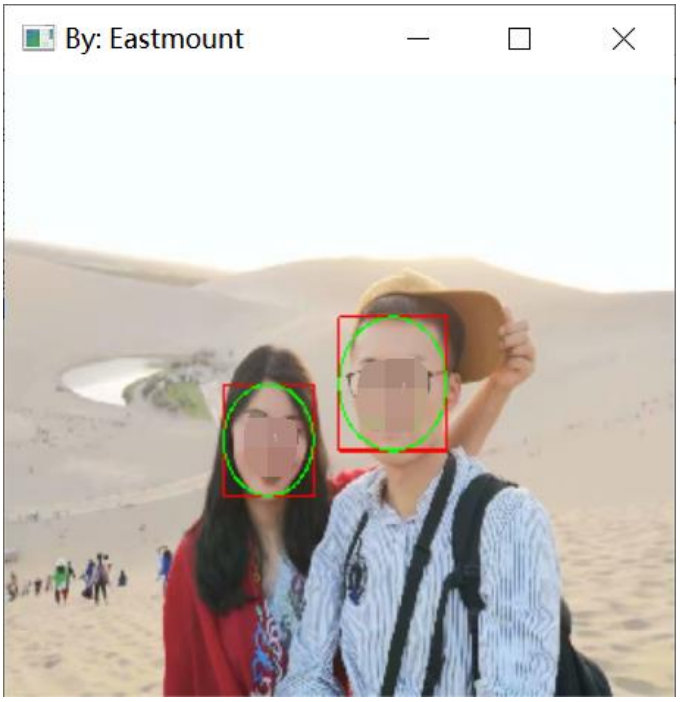


图 44-8 两人人脸识别结果

识别人脸的信息：

```
[[479 281 154 154]
 [314 359 129 129]]
```

再举一个北影毕业照的示例，可以看到人脸被成功的识别。

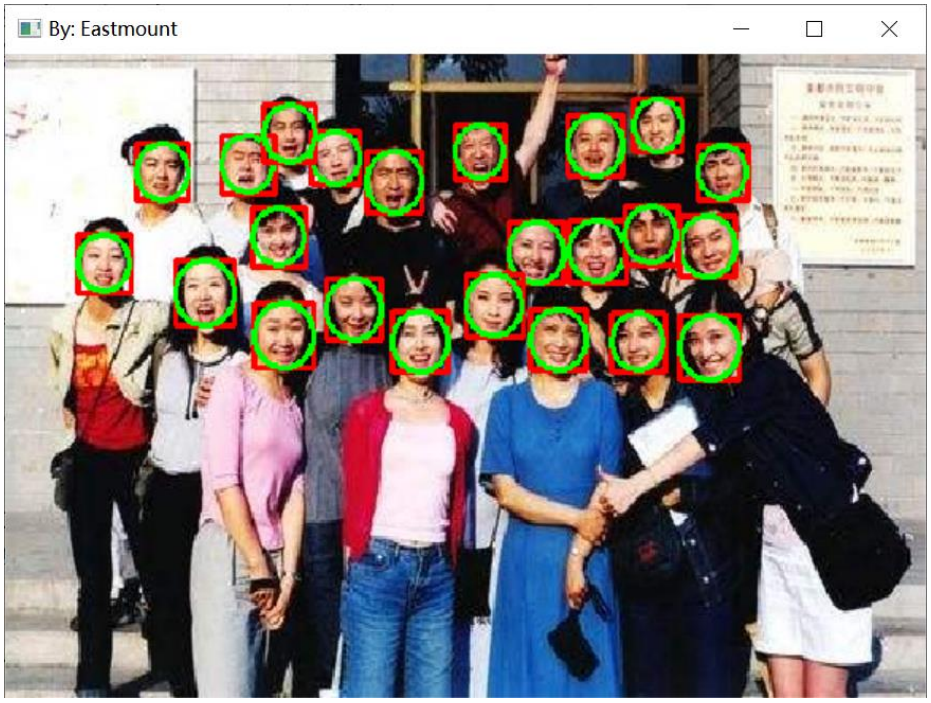


图 44-9 多人人脸准确识别

识别人脸的坐标信息如图 44-10 所示。

```
 RESTART: C:\Users\xiuzhang\Desktop\华为云 Python图像处理\第44章 人脸识别\44-02-mulfaces.py
识别人脸的信息:
 [[166  38  26  26]
 [366  80  30  30]
 [273  83  30  30]
 [210 127  30  30]
 [117  40  29  29]
 [ 39  89  29  29]
 [285 126  30  30]
 [ 71  44  28  28]
 [329 128  29  29]
 [376  45  27  27]
 [366 129  32  32]
 [196  48  30  30]
 [306  82  31  31]
 [135 123  32  32]
 [134  76  29  29]
 [ 92 102  32  32]
 [250 109  31  31]
 [341  22  26  26]
 [140  25  28  28]
 [305  30  30  30]
 [244  35  27  27]
 [174 111  30  30]
 [336  75  29  29]]
>>>
```

图 44-10 人脸识别对应的坐标

此外，任何算法都不会 100%识别准确，由于噪声、误差、算法、训练集等

影响，某些时候也会出现错误识别。一方面 OpenCV 无法识别侧脸，另一方面某些图案也可能被错误地识别成人脸。

3.检测视频人脸

这里需要调用 cv2.VideoCapture()函数导入视频，然后读取视频中的数据，最后调用之前的算法识别人脸。核心代码如下：

（1）读取视频数据

- cap = cv2.VideoCapture（‘shipin.mp4’）#导入视频
- flag, frame = cap.read()
- flag：返回值为 True 和 False，True 和图片一起返回，当读完视频后返回 False
- frame：接受返回来的图片

（2）人脸像素检测识别

- face_zone = face_detect.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3)
- scaleFactor：检测像素点扩散的面积
- minNeighbors：像素点邻近值

（3）设置视频展示频度

为了让视频的展示速度和原视频一致，我们需要调用 “cap.get(propId

= cv2.CAP_PROP_FPS)” 函数将其设置为 25，即每秒 25 帧。

完整代码如下：

```
# -*- coding: utf-8 -*-
# By: Eastmount
import cv2
import numpy as np

# 加载视频
cap = cv2.VideoCapture('shipin01.mp4')

# 调用熟悉的人脸分类器 识别特征类型
# 人脸 - haarcascade_frontalface_default.xml
# 人眼 - haarcascade_eye.xm
# 微笑 - haarcascade_smile.xml
face_detect                                                    =
cv2.CascadeClassifier('haarcascade_frontalface_default.
xml')

while True:
    # 读取视频片段
    flag, frame = cap.read()
```

```
    if flag == False:
        break

    # 灰度处理
    gray = cv2.cvtColor(frame, code=cv2.COLOR_BGR2GRAY)

    # 检查人脸 按照 1.1 倍放到 周围最小像素为 5
    face_zone = face_detect.detectMultiScale(gray, scaleFactor = 1.1, minNeighbors = 4)

    # 绘制矩形和圆形检测人脸
    for x, y, w, h in face_zone:
        cv2.rectangle(frame, pt1 = (x, y), pt2 = (x+w, y+h), color = [0,0,255], thickness=2)
        cv2.circle(frame, center = (x + w//2, y + h//2), radius = w//2, color = [0,255,0], thickness = 2)

    # 显示图片
    cv2.imshow('video', frame)
```

```
    # 设置退出键和展示频率
    if ord('q') == cv2.waitKey(40):
        break


# 释放资源
cv2.destroyAllWindows()
cap.release()
```

女神跳舞输出视频如下图所示，能够实时获取人脸并显示。但是当人物跳舞运动时，由于抖动会导致识别的人脸不准确，这也是更多算法要进行优化的原因。本文当前仅分享基础知识，后续随着自己深入研究，可能会分享更深更好的文章。

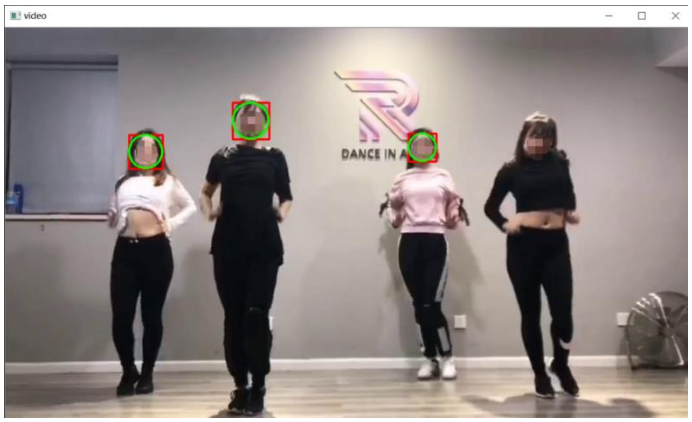


图 44-11 视频的人脸识别

4.摄像头人脸检测

识别电脑摄像头的流程和识别视频中的是一样的，只是加载源的方式不一样。当 VideoCapture(0)时，自动调用摄像头捕捉视频。

```
# -*- coding: utf-8 -*-
```

```
# By: Eastmount
import cv2


# 识别电脑摄像头并打开
cap = cv2.VideoCapture(0)


# 调用熟悉的人脸分类器 识别特征类型
# 人脸 - haarcascade_frontalface_default.xml
# 人眼 - haarcascade_eye.xm
# 微笑 - haarcascade_smile.xml
face_detect                                                =
cv2.CascadeClassifier('haarcascade_frontalface_default.
xml')


while True:
    # 读取视频片段
    flag, frame = cap.read()
    if flag == False:
        break


    # 灰度处理
```

```
    gray = cv2.cvtColor(frame, code=cv2.COLOR_BGR2GRAY)

    # 检查人脸 按照 1.1 倍放到 周围最小像素为 5
    face_zone = face_detect.detectMultiScale(gray, scaleFactor = 1.2, minNeighbors = 5)

    # 绘制矩形和圆形检测人脸
    for x, y, w, h in face_zone:
        cv2.rectangle(frame, pt1 = (x, y), pt2 = (x+w, y+h), color = [0,0,255], thickness=2)
        cv2.circle(frame, center = (x + w//2, y + h//2), radius = w//2, color = [0,255,0], thickness = 2)

    # 显示图片
    cv2.imshow('video', frame)

    # 设置退出键和展示频率
    if ord('q') == cv2.waitKey(40):
        break
```

```
# 释放资源
cv2.destroyAllWindows()
cap.release()
```

输出结果如图 44-12 所示，下图展示了作者“Eastmount”在实验室的工作环境。注意，同样需要注意识别正脸和静止状态效果更好。

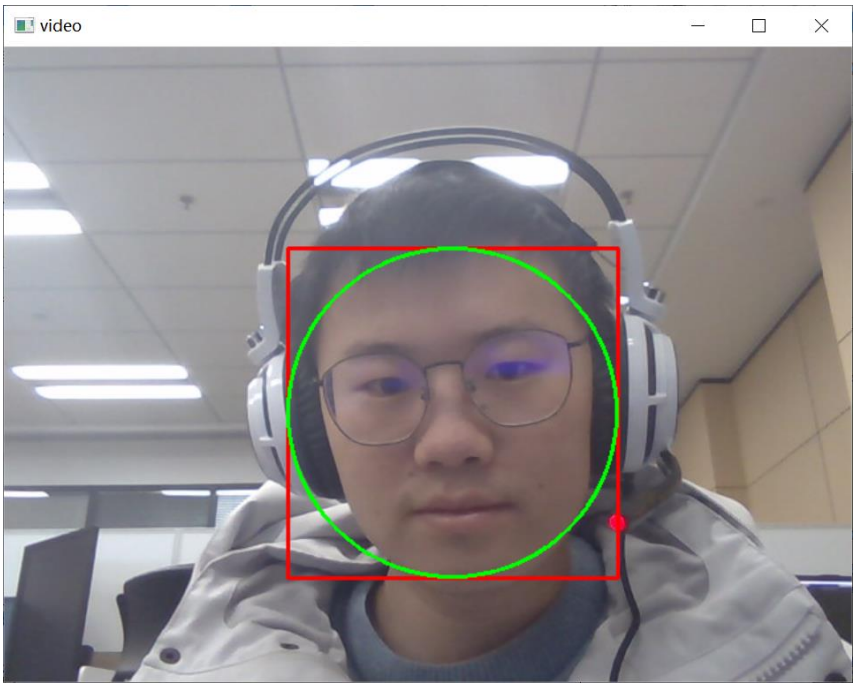


图 44-12 视频直播的人脸识别

5.总结

本篇文章主要讲解 Python 和 OpenCV 基础知识，主要调用 github 算法进行人脸识别，包括识别图像、视频和摄像头等。但该算法也存在缺陷，比如侧脸识别效果不佳、抖动视频识别不好等，后续随着作者深入，希望能够分享更好的代码。希望这篇基础性文章对读者有一定帮助，也希望这些知识点为读者从

事 Python 图像处理相关项目实践或科学研究提供一定基础。

参考文献：

[1] Eastmount. [图像处理] Python+OpenCV 实现车牌区域识别及 Sobel 算子. https://blog.csdn.net/Eastmount/article/details/81461679.

[2] https://github.com/opencv/opencv/blob/master/data/haarcascades

[3] 阿优乐扬. 图像处理之 opencv 识别图片和视频中人脸. https://blog.csdn.net/ayouleyang/article/details/104091979.

[4] Eastmount. [Python 图像处理] 二十八.OpenCV 快速实现人脸检测及视频中的人脸. https://blog.csdn.net/Eastmount/article/details/104463173

第 45 篇 目标检测入门普及及 ImageAI 实现对象检测详解

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前文介绍了人脸检测的应用案例，通过 OpenCV 快速实现人脸检测。这篇文章将详细讲解目标检测基础知识，并通过 ImageAI 实现对象检测的经典案例。本文主要介绍目标检测原理，通过七个问题来普及什么是目标检测。然后利用 ImageAI 实现最简单的目标检测案例，加深读者的印象，本文提供的案例将为读者提供深入的理解，希望您喜欢。

1.目标检测入门普及

该部分结合自己多年图像识别的经验，并参考图像算法 AI（yegeli）、Zhengxia Zou、牛戈和 SIGAI 老师们的文章进行总结。主要通过七个核心问题带领初学者了解什么是目标检测。

（1）什么是目标检测?

目标检测（Object Detection）旨在寻找出图像中所有感兴趣的目标物体或对象，包含物体定位和物体分类两个子任务，同时确定它们的类别和位置。如

下图所示，通过卷积神经网络有效定位猫的轮廓及类别。

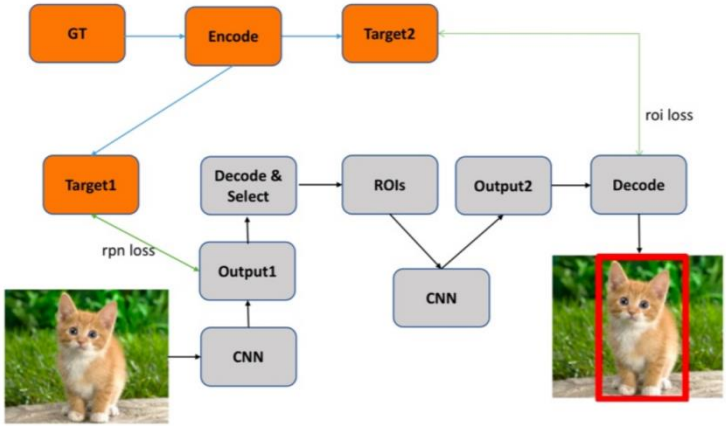


图 45-1 目标检测流程

目标检测是计算机视觉领域中最基本、最具挑战性的问题之一，近年来受到了广泛的关注。它在过去二十年的发展可以说是计算机视觉历史的缩影。由于各类物体有不同的外观、形状和姿态，加上成像时光照、遮挡、天气、分辨率、景深等因素的干扰，目标检测一直是计算机视觉领域最具有挑战性的问题，并且目标检测的结果将直接影响后续的跟踪、动作识别和行为描述的效果。因此，目标检测发展到今天仍然是非常具有挑战且存在很大提升空间的问题。

（2）目标检测的核心问题是什么?

在计算机视觉领域，图像识别主要包括四大类任务：

❖ 分类（Classification）

解决“是什么？”问题，即给定一张图片或视频判断其包含什么类别的目标。

❖ 定位（Location）

解决“在哪里？”问题，即定位出这个目标的的位置。

❖ 检测（Detection）

解决“在哪里？是什么？”问题，即定位出目标位置并知道目标物是什么。

❖ 分割（Segmentation）

包括实例分割（Instance-level）和场景分割（Scene-level），解决“每一个像素属于哪个目标物或场景”的问题。

引用 yegeli 老师的文章，如图 45-2 所示，目标检测主要是实现分类问题和定位问题的叠加。

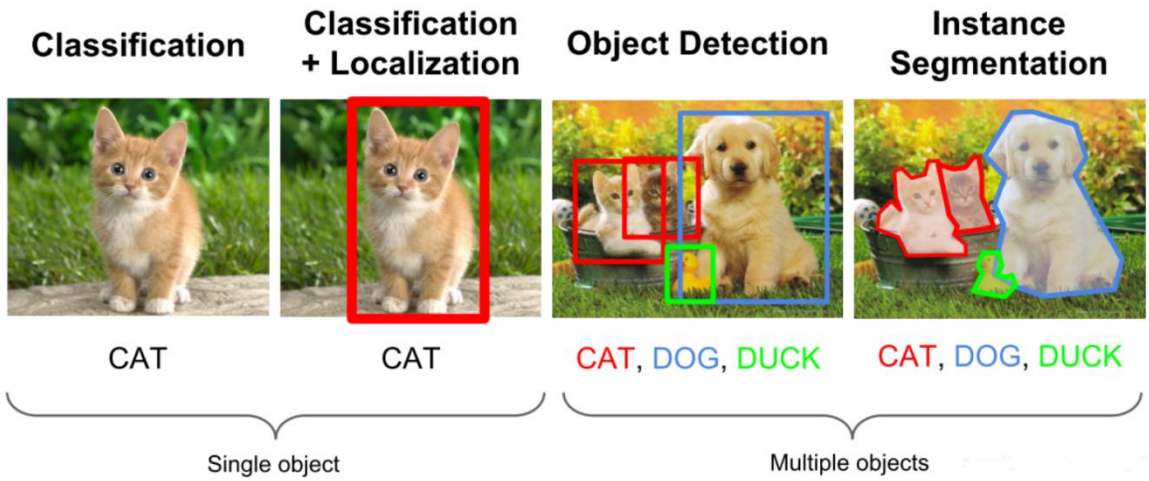


图 45-2 目标检测分类

其核心问题分为：

❖ 分类问题：确定图片（或某个区域）中的图像属于哪个类别

❖ 定位问题：目标可能出现在图像的任何位置

同时，需要解决大小问题（目标存在各种不同的大小）和形状问题（目标可能有各种不同的形状）。

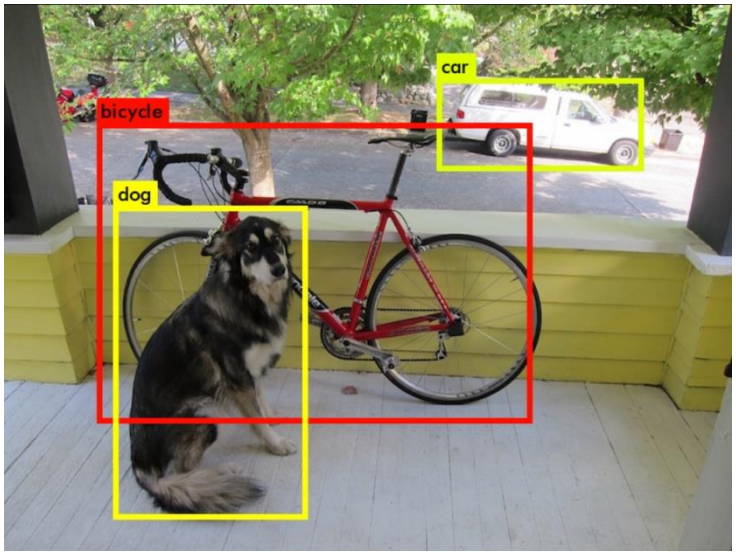


图 45-3 目标检测示例

（3）目标检测算法常用分类

每个领域的分类方法通常各式各样，这里主要介绍两种分类方法。

❖ 基于应用程序的角度分类

目标检测作为计算机视觉的基本问题之一，是许多其他计算机视觉任务的基础，如实例分割、图像字幕、对象跟踪等。从应用程序的角度来看，目标检测可以被分为两个研究主题：

- general object detection：旨在探索统一框架下检测不同类型物体的方法，以模拟人类的视觉和认知。
- detection applications：指特定应用场景下的检测，如行人检测、人脸检测、文本检测等。

近年来，随着深度学习技术的快速发展，为目标检测注入了新的血液，取得了显著的突破，将其推向了一个前所未有的研究热点。目前，目标检测已广泛应用于自主驾驶、机器人视觉、视频监控等领域。下图显示了过去二十年中与“目

标检测”相关的出版物数量的增长。

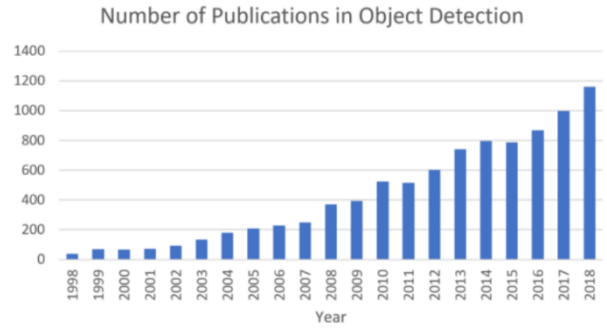


Fig. 1. The increasing number of publications in object detection from 1998 to 2018. (Data from Google scholar advanced search: *allintitle: “object detection” AND “detecting objects”*.)

图 45-4 目标检测相关研究情况

❖ 基于深度学习的目标检测分类

基于深度学习的目标检测算法主要分为两类：Two stage 和 One stage。

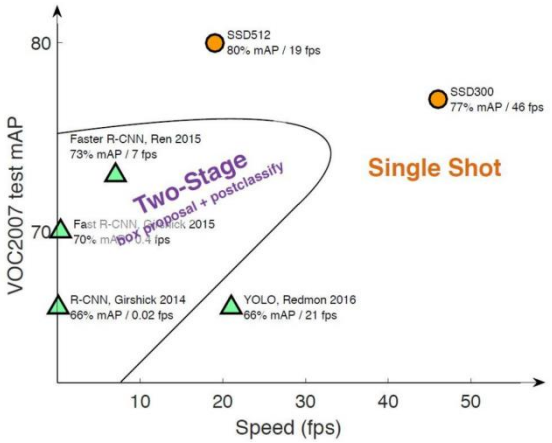


图 45-5 目标检测算法分类

➢ Two stage

Two-stage Detection 将检测框定为一个“从粗到细”的过程。先进行区域生成，该区域称之为 Region Proposal（简称 RP，一个有可能包含待检物体的预选框），再通过卷积神经网络进行样本分类。基

本任务流程为：**特征提取=>生成 RP=>分类/定位回归**。常见 Two Stage 目标检测算法有：

- ✧ R-CNN
- ✧ SPP-Net
- ✧ Fast R-CNN
- ✧ Faster R-CNN
- ✧ R-FCN
- ✧ ...

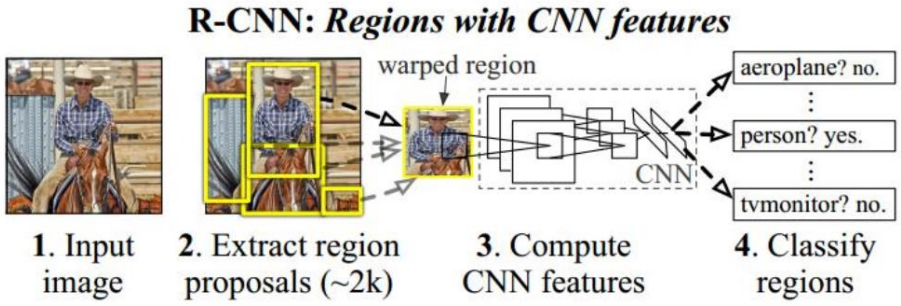


图 45-6 RCNN 算法流程

➢ One stage

One-stage Detection 定义为“一步完成”过程。不用 RP，直接在网络中提取特征来预测物体分类和位置。基本任务流程为：**特征提取=>分类/定位回归**。常见 One Stage 目标检测算法有：

- ✧ OverFeat
- ✧ YOLOv1
- ✧ YOLOv2

- ✧ YOLOv3
- ✧ SSD
- ✧ RetinaNet
- ✧ ...

One-Stage 基本流程如图 45-7 所示。

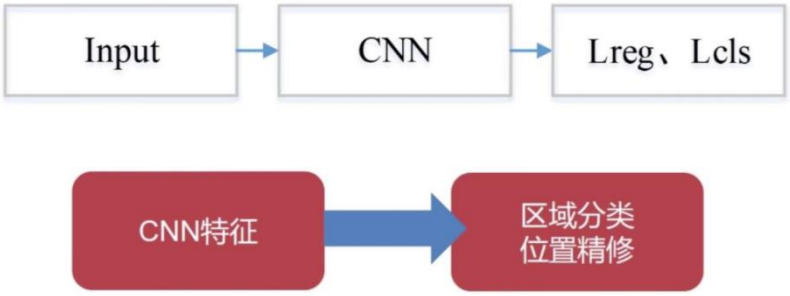


图 45-7 One-Stage 基本流程

（4）目标检测常见算法

在过去二十年中，人们普遍认为，目标检测的发展大致经历了两个历史时期：

- ■ 传统的目标检测时期（2014 年以前）
- ■ 基于深度学习的检测时期（2014 年以后）

其发展历程如图 45-8 所示。

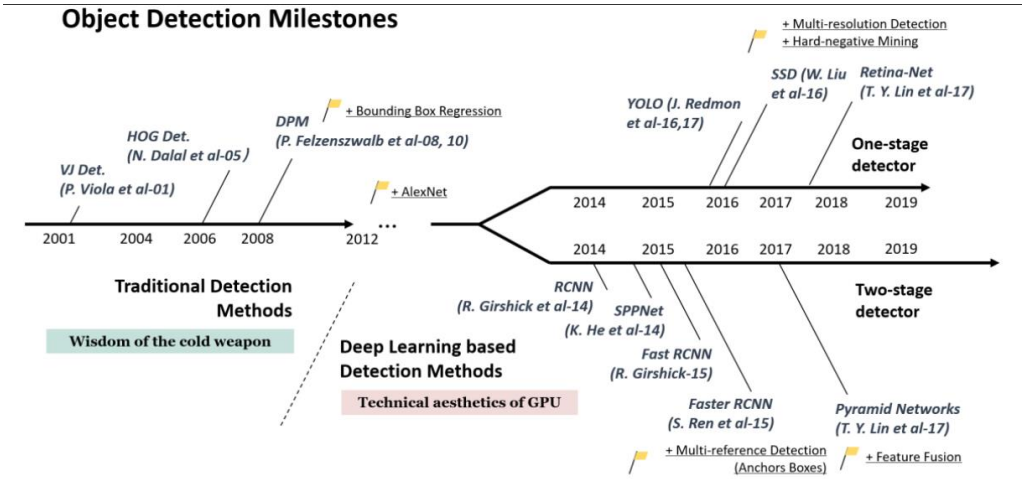


Fig. 2. A road map of object detection. Milestone detectors in this figure: VJ Det. [10, 11], HOG Det. [12], DPM [13-15], RCNN [16], SPPNet [17], Fast RCNN [18], Faster RCNN [19], YOLO [20], SSD [21], Pyramid Networks [22], Retina-Net [23].

图 45-8 目标检测发展历史

下面分别列举了各里程碑式的目标检测算法，后续博客会分别进行详细介绍及编程案例实现。

❖ 传统检测器（Traditional Detectors）

早期的目标检测算法大多是基于手工特征构建的。由于当时缺乏有效的图像表示，人们只能设计复杂的特征表示，以及各种加速技术来用尽有限的计算资源。

① Viola Jones Detectors

2001 年，P. Viola 和 M. Jones 在没有任何约束条件（如肤色分割）的情况下首次实现了人脸的实时检测。VJ 检测器采用滑动窗口最直接的检测方法，查看图像中所有可能的位置和比例，看看是否有窗口包含人脸。其检测器结合了“积分图像”“特征选择”和“检测级联”三种重要技术，大大提高了检测速度。

② HOG Detector

方向梯度直方图（HOG）特征描述符最初是由 N. Dalal 和 B.Triggs 在 2005 年提出的。HOG 可以被认为是对当时的尺度不变特征变换和形状上下文

的重要改进。多年来，HOG 检测器一直是许多目标检测器和各种计算机视觉应用的重要基础。

③ Deformable Part-based Model（基于可变形部件的模型，DPM）

DPM 最初是由 P. Felzenszwalb 提出的，于 2008 年作为 HOG 检测器的扩展。DPM 遵循“分而治之”的检测思想，训练可以简单地看作是学习一种正确的分解对象的方法，推理可以看作是对不同对象部件的检测的集合。例如，检测汽车的问题可以看作是检测它的窗口、车身和车轮。

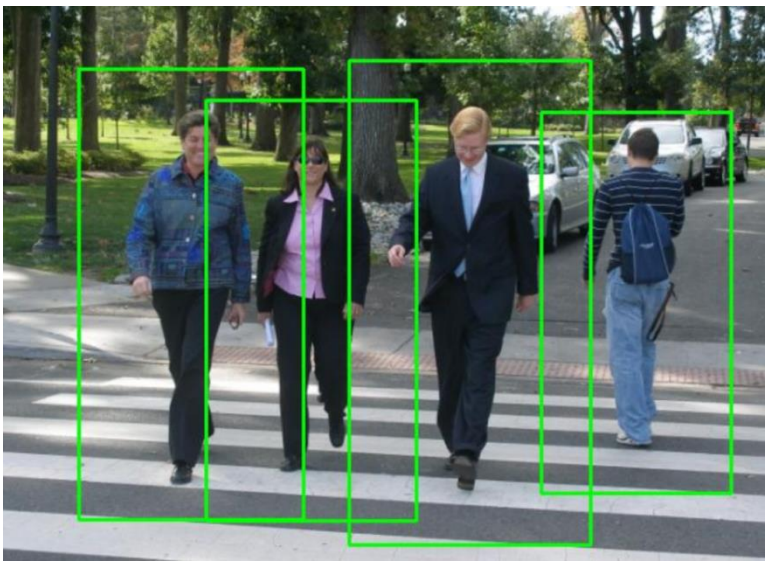


图 45-9 目标检测结果

❖ CNN based Two-stage Detectors

随着手工特征的性能趋于饱和，目标检测在 2010 年之后达到了一个稳定的水平。2012 年卷积神经网络在世界范围内重生。由于深度卷积网络能够学习图像的鲁棒性和高层次特征表示，一个自然的问题是能否将其应用到目标检测中?

在 2014 年，R. Girshick 等人率先打破僵局，提出了具有 CNN 特征的区域（RCNN）用于目标检测。从那时起，目标检测开始以前所未有的速度发

展。在深度学习时代，目标检测可以分为两类，即 two-stage detection 和 one-stage detection，分别对应“从粗到细”的检测过程和“一步完成”的检测过程。

① RCNN

RCNN 首先通过选择性搜索提取一组对象候选框。然后，每个提案都被重新调整成一个固定大小的图像，并输入到一个在 ImageNet 上训练得到的 CNN 模型（如 AlexNet）来提取特征。最后，利用线性 SVM 分类器对每个区域内的目标进行预测，识别目标类别。

② SPPNet

2014 年，K. He 等人提出了空间金字塔池化网络（Spatial Pyramid Pooling Networks，SPPNet）。以前的 CNN 模型需要固定大小的输入，例如，AlexNet 需要 224x224 图像。SPPNet 主要贡献是引入了空间金字塔池化(SPP)层，它使 CNN 能够生成固定长度的表示，而不需要重新缩放图像/感兴趣区域的大小。利用 SPPNet 进行目标检测时，只对整个图像进行一次特征映射计算，然后生成任意区域的定长表示，训练检测器，避免了卷积特征的重复计算。

③ Fast RCNN

2015 年，R. Girshick 提出了 Fast RCNN 检测器，这是对 R-CNN 和 SPPNet 的进一步改进。Fast-RCNN 融合了 R-CNN 和 SPPNet 的优点，使我们能够在相同的网络配置下同时训练检测器和边界框回归器。

④ Faster RCNN

2015 年，S. Ren 等人提出了 Faster RCNN 检测器，在 Fast RCNN 之后不久。Faster RCNN 是第一个端到端的，也是第一个接近实时的深度学习检测器。Faster RCNN 主要贡献是引入了区域建议网络(RPN)，使几乎 cost-free 的区域建议成为可能。从 RCNN 到 Faster RCNN，一个目标检测系统中的大部分独立块，如提案检测、特征提取、边界框回归等，都已逐渐集成到一个统一的端到端学习框架中。

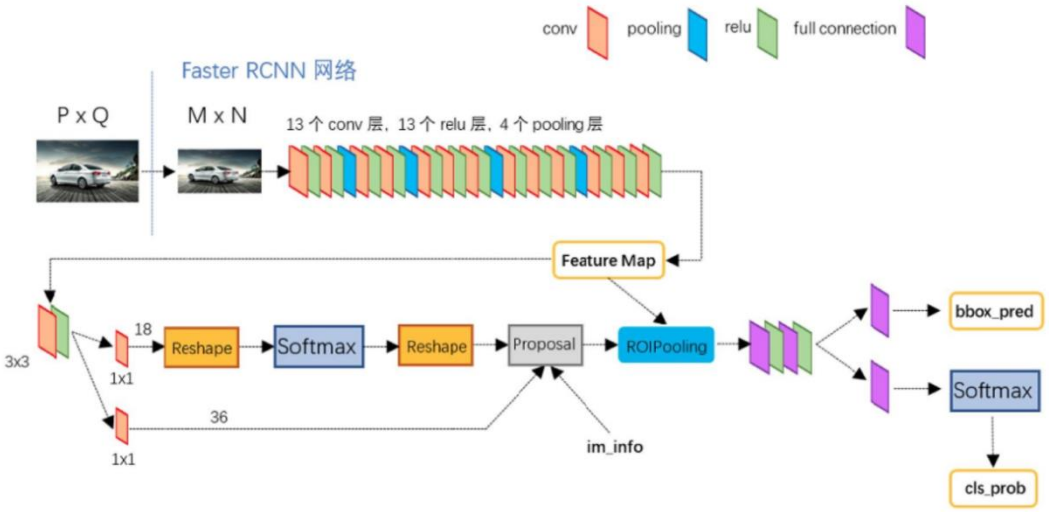


图 45-10 Faster RCNN 网络

⑤ Feature Pyramid Networks（FPN）

2017 年，T.-Y.Lin 等人基于 Faster RCNN 提出了特征金字塔网络(FPN)。在 FPN 之前，大多数基于深度学习的检测器只在网络的顶层进行检测。虽然 CNN 较深层的特征有利于分类识别，但不利于对象的定位。为此，开发了具有横向连接的自顶向下体系结构，用于在所有级别构建高级语义。由于 CNN 通过它的正向传播，自然形成了一个特征金字塔，FPN 在检测各种尺度的目标方面显示出了巨大的进步。FPN 现在已经成为许多最新探测器的基本组成部分。

❖ CNN based One-stage Detectors

① You Only Look Once (YOLO)

YOLO 由 R. Joseph 等人于 2015 年提出，它是深度学习时代的第一个单级检测器。YOLO 是“You Only Look Once”的缩写，其速度较快。从它的名字可以看出，作者完全抛弃了之前的“提案检测+验证”的检测范式。相反，它遵循一个完全不同的哲学：将单个神经网络应用于整个图像。该网络将图像分割成多个区域，同时预测每个区域的边界框和概率。后来 R. Joseph 在 YOLO 的基础上进行了一系列改进，提出了其 v2 和 v3 版本，在保持很高检测速度的同时进一步提高了检测精度。尽管与两级探测器相比，它的探测速度有了很大的提高，但是 YOLO 的定位精度有所下降，特别是对于一些小目标。

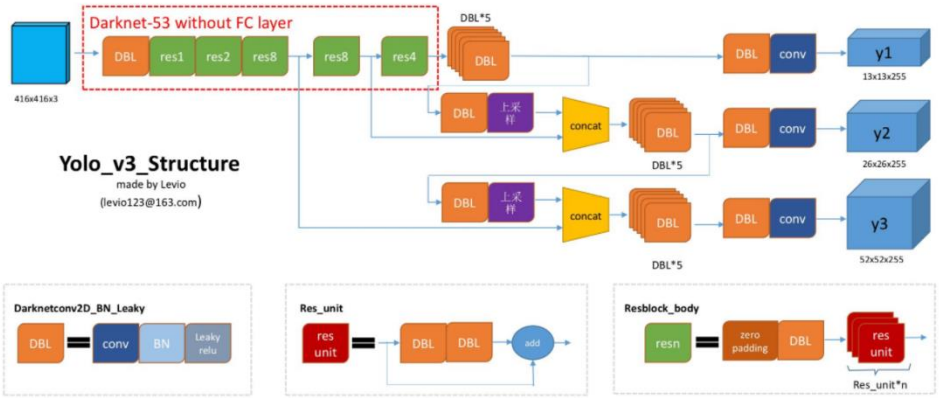


图 45-11 YOLOv3 网络

② Single Shot MultiBox Detector（SSD）

SSD 由 W. Liu 等人于 2015 年提出，这是深度学习时代的第二款单级探测器。SSD 的主要贡献是引入了多参考和多分辨率检测技术，这大大提高了单级检测器的检测精度，特别是对于一些小目标。SSD 与以往任何检测器的主要

区别在于，前者在网络的不同层检测不同尺度的对象，而后者仅在其顶层运行检测。

③ RetinaNet

单级检测器速度快、结构简单，但多年来一直落后于两级检测器的精度。T.-Y.Lin 等人发现了背后的原因，并在 2017 年提出了 RetinaNet。他们声称，在密集探测器训练过程中所遇到的极端的前景-背景阶层不平衡(the extreme foreground-background class imbalance ）是主要原因。为此，在 RetinaNet 中引入了一个新的损失函数“焦损失（focal loss）”，通过对标准交叉熵损失的重构，使检测器在训练过程中更加关注难分类的样本。焦损耗使得单级检测器在保持很高的检测速度的同时，可以达到与两级检测器相当的精度。

建立具有更少偏置的大数据集，是开发先进的计算机视觉算法的关键。在目标检测方面，在过去 10 年中，已经发布了许多著名的数据集和基准测试，包括 PASCAL VOC 挑战的数据集（如 VOC2007、VOC2012）、ImageNet 大尺度视觉识别挑战（如 ILSVRC2014）、MS-COCO 检测挑战等。下图给出了这些数据集的统计数据以及数据集对应的一些图像示例。

Dataset	train		validation		trainval		test	
	images	objects	images	objects	images	objects	images	objects
VOC-2007	2,501	6,301	2,510	6,307	5,011	12,608	4,952	14,976
VOC-2012	5,717	13,609	5,823	13,841	11,540	27,450	10,991	-
ILSVRC-2014	456,567	478,807	20,121	55,502	476,688	534,309	40,152	-
ILSVRC-2017	456,567	478,807	20,121	55,502	476,688	534,309	65,500	-
MS-COCO-2015	82,783	604,907	40,504	291,875	123,287	896,782	81,434	-
MS-COCO-2018	118,287	860,001	5,000	36,781	123,287	896,782	40,670	-
OID-2018	1,743,042	14,610,229	41,620	204,621	1,784,662	14,814,850	125,436	625,282

TABLE 1
Some well-known object detection datasets and their statistics.

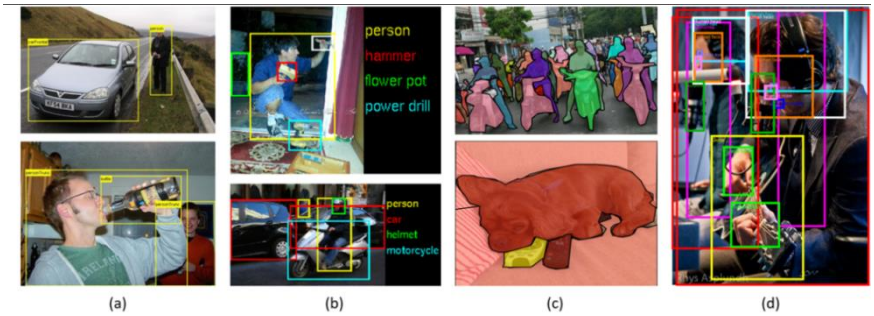


图 45-12 性能对比

（5）目标检测应用

■ 车辆检测

包括自动驾驶、违章查询、关键通道检测、广告检测（检测广告中的车辆类型，弹出链接）等。

图 45-13 车辆检测示例

■ 人脸检测

包括人脸支付、智能门控、考勤签到、智慧超市、车站机场实名认证、公共安全（逃犯抓捕、走失人员检测）等。

图 45-14 人脸检测示例

■ 行人检测

包括步态识别、智能辅助驾驶、智能监控、暴恐检测（根据面相识别暴恐倾向）、移动侦测、区域入侵检测、安全帽/安全带检测等。

图 45-15 行人检测示例

■ 遥感检测

包括大地遥感（如土地使用、公路、水渠、河流监控）、农作物监控、工业检测、JS 检测等。

图 45-16 遥感检测示例

（6）目标检测未来挑战

近 20 年来，目标检测取得了显著的成，包括里程碑检测器、关键技术、加速方法、检测应用、数据集和指标，如 VJ、HOG、DPM、Faster-RCNN、YOLO、SSD 等。未来的目标检测研究可能会集中在以下几个方面：

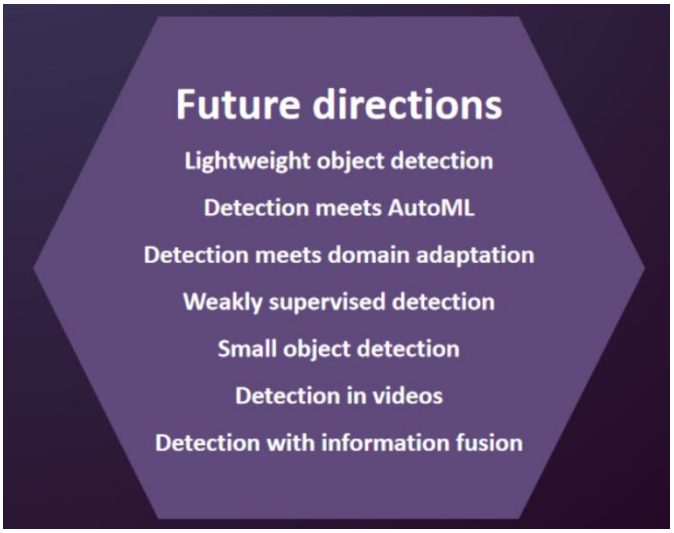


图 45-17 目标检测未来研究方向

① 加快检测算法的速度（Lightweight object detection）

使其能够在移动设备上平稳运行。一些重要的应用包括移动增强现实、智能

摄像头、人脸验证等。虽然近年来已经做了很大的努力，但机器和人眼之间的速度差距仍然很大，特别是在检测一些小物体时。

② 自动机器学习目标检测（Detection meets AutoML）

近年来，基于深度学习的检测器变得越来越复杂，严重依赖于经验。未来的方向是在使用神经结构搜索设计检测模型时减少人为干预 ，譬如如何设计引擎和如何设置锚框)。AutoML 可能是未来的目标检测。

③ 领域自适应检测（Detection meets domain adaptation）

任何目标检测器的训练过程本质上都可以看作是一个假设数据独立且同分布(i.i.d)时的似然估计过程。使用非 i.i.d 数据的目标检测，特别是对一些实际应用程序来说，仍然是一个挑战。GAN 在领域自适应方面显示出良好的应用前景，对未来的目标检测具有重要的指导意义。

④ 弱监督检测（Weakly supervised detection）

基于深度学习的检测器的训练通常依赖于大量注释良好的图像。注释过程耗时、开销大且效率低。开发弱监督检测技术，只使用图像级标注或部分使用边界框标注对检测器进行训练，对于降低人工成本和提高检测灵活性具有重要意义。

⑤ 小目标检测（Small object detection）

在大场景中检测小物体一直是一个挑战。该研究方向的一些潜在应用包括利用遥感图像计算野生动物的数量和检测一些重要 JS 目标的状态。进一步的方向可能包括视觉注意机制的集成和高分辨率轻量级网络的设计。

⑥ 视频目标检测（Detection in videos）

高清视频中的实时目标检测/跟踪对于视频监控和自主驾驶具有重要意义。

传统的目标检测器通常设计为基于图像的检测，而忽略了视频帧之间的相关性。通过探索时空相关性来改进检测是一个重要的研究方向。

⑦ 含信息融合的目标检测（Detection with information fusion）

RGB-D 图像、三维点云、激光雷达等多数据源/多模式的目标检测对自主驾驶和无人机应用具有重要意义。目前存在的问题包括：如何将训练有素的检测器移植到不同的数据模式，如何进行信息融合以提高检测能力等。

（7）目标检测原理简述

目标检测分为两大系列——RCNN 系列和 YOLO 系列。RCNN 系列是基于区域检测的代表性算法，YOLO 是基于区域提取的代表性算法，另外还有著名的 SSD 是基于前两个系列的改进。目标检测的基本原理包括：

① 候选区域产生

目标检测技术大都会涉及候选框（bounding boxes）的生成，物体候选框获取当前主要使用图像分割与区域生长技术。区域生长（合并）主要由于检测图像中存在的物体具有局部区域相似性（颜色、纹理等）。目标识别与图像分割技术的发展进一步推动有效提取图像中信息。

- 滑动窗口

首先对输入图像进行不同窗口大小的滑窗进行从左往右、从上到下的滑动。每次滑动时候对当前窗口执行分类器（分类器是事先训练好的）。如果当前窗口得到较高的分类概率，则认为检测到了物体。其次，对每个不同窗口大小的滑窗都进行检测后，会得到不同窗口检测到的物体标记，这些窗口大小会存在重复较高的部分，接着采用非极大值抑制（Non-Maximum Suppression, NMS）

的方法进行筛选。最后，经过 NMS 筛选后获得检测到的物体。

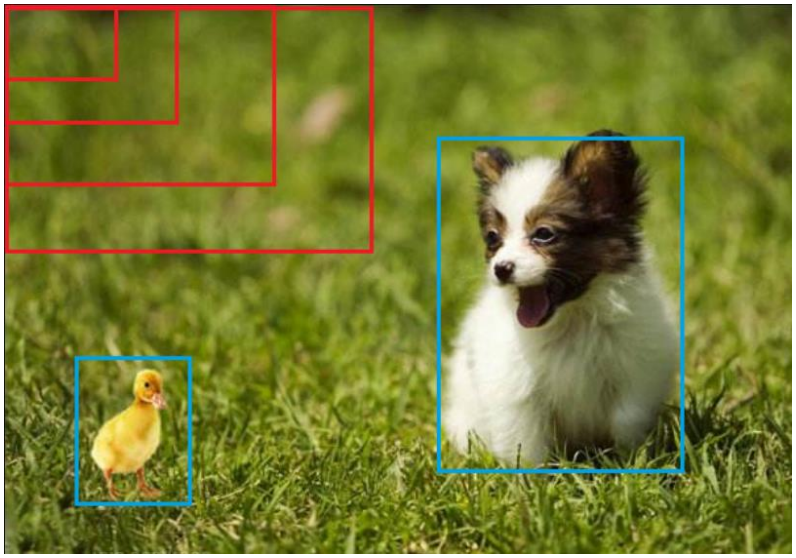


图 45-18 滑动窗口

■ 选择性搜索

图像中物体可能存在的区域应该是有某些相似性或者连续性区域的。因此，选择搜索基于上面的想法采用子区域合并的方法进行提取 bounding boxes。首先，对输入图像进行分割算法产生许多小的子区域。其次，根据这些子区域之间相似性（相似性标准主要有颜色、纹理、大小等）进行区域合并，不断的进行区域迭代合并。每次迭代过程中对这些合并的子区域做 bounding boxes（外切矩形），这些子区域外切矩形就是通常所说的候选框。最终提高检测物体的概率。

图 45-19 选择性搜索

② 数据表示

经过标记后的样本数据如下所示：

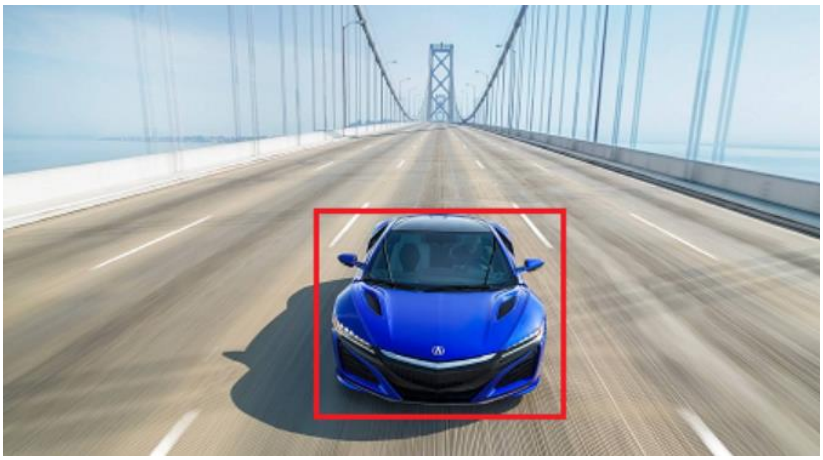


图 45-20 数据标注

预测输出可以表示如下，其中 pc 为预测结果的置信概率，(bx,by,bw,bh) 是边框坐标，Ci 为属于某个类别的概率。通过预测结果、实际结果，构建损失函数。

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_w \\ b_h \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}, y_{true} = \begin{bmatrix} 1 \\ 40 \\ 45 \\ 80 \\ 60 \\ 0 \\ 1 \\ 0 \end{bmatrix}, y_{pred} = \begin{bmatrix} 0.88 \\ 41 \\ 46 \\ 82 \\ 59 \\ 0.01 \\ 0.95 \\ 0.04 \end{bmatrix}$$

③ 效果评估

使用交并比（Intersection over Union，IoU）来判断模型的好坏。交并比是指预测边框、实际边框交集和并集的比率，一般约定 0.5 为一个可以接收的值。

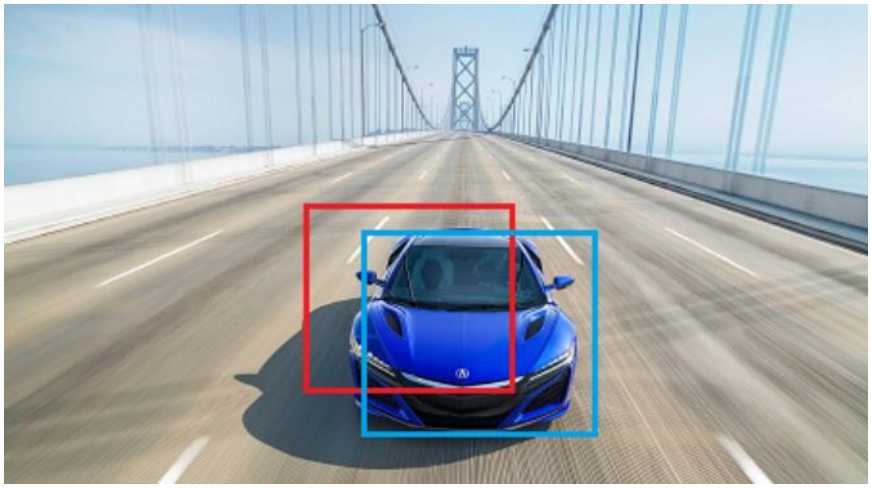


图 45-21 效果评估

④ 非极大值抑制

预测结果中，可能多个预测结果间存在重叠部分，需要保留交并比最大的、去掉非最大的预测结果，这就是非极大值抑制（Non-Maximum Suppression，简写作 NMS）。如下图所示，对同一个物体预测结果包含三

个概率 0.8、0.9、0.95，经过非极大值抑制后，仅保留概率最大的预测结果。

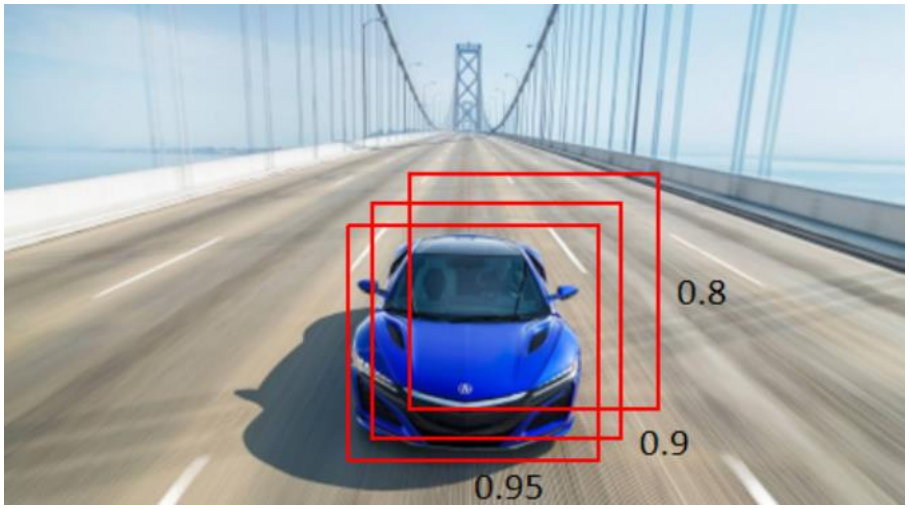


图 45-22 预测结果显示

写到这里，一个简单的目标检测过程介绍完毕。接下来我们重点使用 ImageAI 库实现最简单的目标检测或对象检测案例。这部分原理引用了 yegeli 老师的博客。推荐大家学习，文章最后给出了对应的参考文献。

2.ImageAI 简介

ImageAI 是一个开源 Python 库，旨在使开发人员能够使用简单的几行代码构建具有包含深度学习和计算机视觉功能的应用程序和系统。这个 AI Commons 项目由 Moses Olafenwa 和 John Olafenwa 开发和维护。官方源码及文档如下：

- https://github.com/OlafenwaMoses/ImageAI
- https://imageai-cn.readthedocs.io/zh_CN/latest/

图 45-23 ImageAI 图标

对应的中文文档如下图所示。

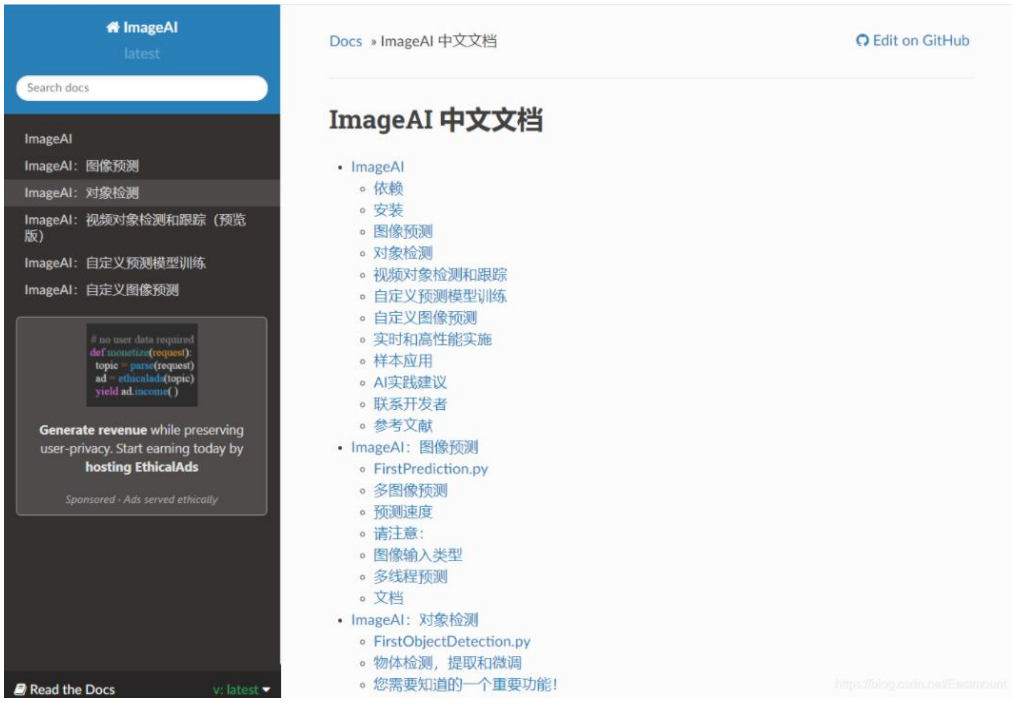


图 45-24 ImageAI 中文文档

ImageAI 本着简洁的原则，支持最先进的机器学习算法，用于图像预测、自定义图像预测、物体检测、视频检测、视频对象跟踪和图像预测训练。ImageAI 目前支持使用在 ImageNet-1000 数据集上训练的 4 种不同机器学习算法进

行图像预测和训练。ImageAI 还支持使用在 COCO 数据集上训练的 RetinaNet、YOLOv3 和 TinyYOLOv3 进行对象检测、视频检测和对象跟踪。最终，ImageAI 将为计算机视觉提供广泛和专业化的支持，包括但不限于特殊环境和特殊领域的图像识别。

- 新版本：ImageAI 2.1.6

新添加功能如下：

- 添加了 SqueezeNet、ResNet50、InceptionV3 和 DenseNet121 模型进行自定义图像预测训练
- 添加了自定义训练模型和 json 文件进行导入和导出自定义图像
- 预览版：添加视频对象检测和视频自定义对象检测（对象跟踪）
- 为所有图像预测和对象检测任务添加文件，numpy 数组和流输入类型
- 添加文件和 numpy 数组输出类型，用于图像中的对象检测和自定义对象检测
- 引入 4 种速度模式（normal, fast, faster 和 fastest）进行图像预测，在 fastest 速度模式下预测时间将缩短 50%，同时保持预测精准度
- 为图像所有物体检测和视频物体检测任务引入 5 种速度模式（normal, fast, faster, fastest 和 flash）
- 引入帧检测率，允许开发人员调整视频中的检测间隔 frame_detection_interval，有利于达到特定效果

ImageAI 核心功能如下：

① 图像检测

ImageAI 提供 4 种不同的算法和模型类型来执行图像预测，并在 ImageNet-1000 数据集上进行训练。为图像预测提供的 4 种算法包括：

- MobileNetV2
- ResNet50
- InceptionV3
- DenseNet121

图 45-25 数据集中的图像示例

② 对象检测

ImageAI 提供了非常方便和强大的方法来对图像进行对象检测并从图像中提取每个对象。对象检测类提供对三种模型的支持，并提供针对最先进性能或实时处理进行调整的选项。

- RetinaNet

- YOLOv3
- TinyYOLOv3

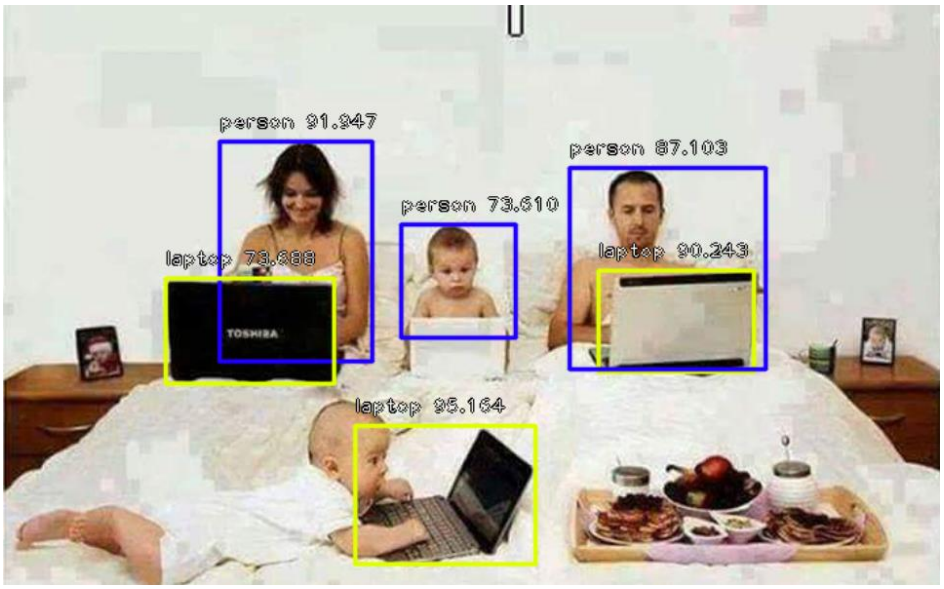


图 45-26 对象检测示例

比如上图的检测结果如下所示：

```
person : 91.946941614151

--------------------------------

person : 73.61021637916565

--------------------------------

laptop : 90.24320840835571

--------------------------------

laptop : 73.6881673336029

--------------------------------

laptop : 95.16398310661316

--------------------------------
```

```
person : 87.10319399833679

--------------------------------
```

③ 视频对象检测和跟踪

ImageAI 提供了非常方便和强大的方法来执行视频中的对象检测和跟踪特定对象。提供的视频对象检测类仅支持当前最先进的 RetinaNet，但可以选择调整最先进的性能或实时处理。图 45-27 展示了对人、自行车和摩托车的视频快照检测效果。

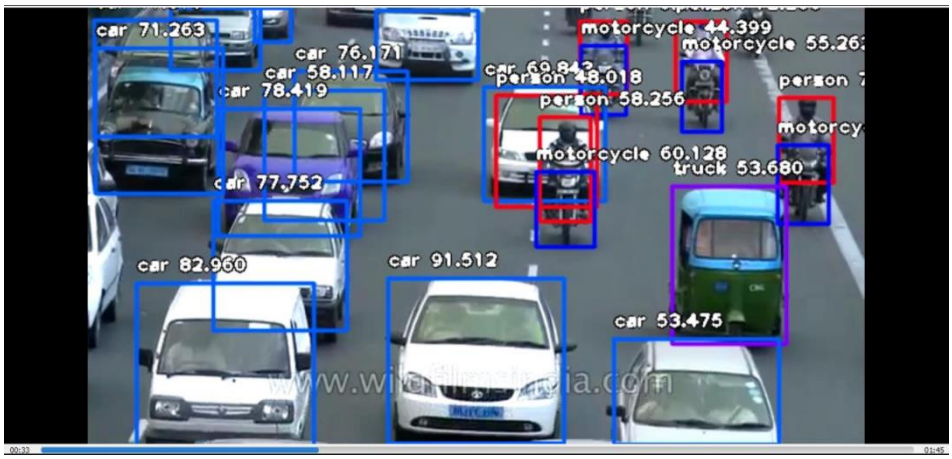


图 45-27 视频快照的对象检测效果

下面是 ImageAI 返回到 per_second 函数中的视频分析的可视化。

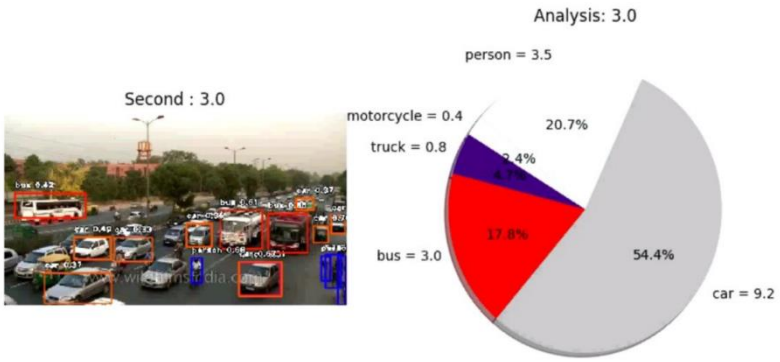


图 45-28 视频可视化分析

④ 自定义模型训练

ImageAI 为您提供了用于训练新模型的类和方法，该模型可用于对您自己的自定义对象进行预测。您可以在 5 行代码中使用算法训练您的自定义模型 。

- SqueezeNet
- ResNet50
- InceptionV3
- DenseNet

下图展示了来自 IdenProf 数据集的样本，用于训练预测专业人士的模型。

图 45-29 IdenProf 数据集

自定义图像预测：来自在 IdenProf 上训练的样本模型的预测，用于预测专业人士。

图 45-29 IdenProf 数据集预测

其预测结果如下所示：

- mechanic : 76.82620286941528
- chef : 10.106072574853897
- waiter : 4.036874696612358
- police : 2.6663416996598244
- pilot : 2.2393483668565 75

ImageAI 提供了一些类和方法，供您使用通过 ImageAI 模型训练类训练的自己的模型来运行图像预测您自己的自定义对象。您可以使用 SqueezeNet、ResNet50、InceptionV3 和 DenseNet 训练的自定义模型以及包含自定义对象名称映射的 JSON 文件。

⑤ 自定义检测模型训练

ImageAI 提供类和方法供您在自定义数据集上训练新的 YOLOv3 对象检测模型。这意味着您可以通过提供图像、注释和使用 ImageAI 进行训练来训练模型以检测任何感兴趣的对象。

图 45-40 自定义检测模型训练

下图展示了自定义 YOLOv3 模型的检测结果，该模型经过训练以检测 Hololens 耳机。

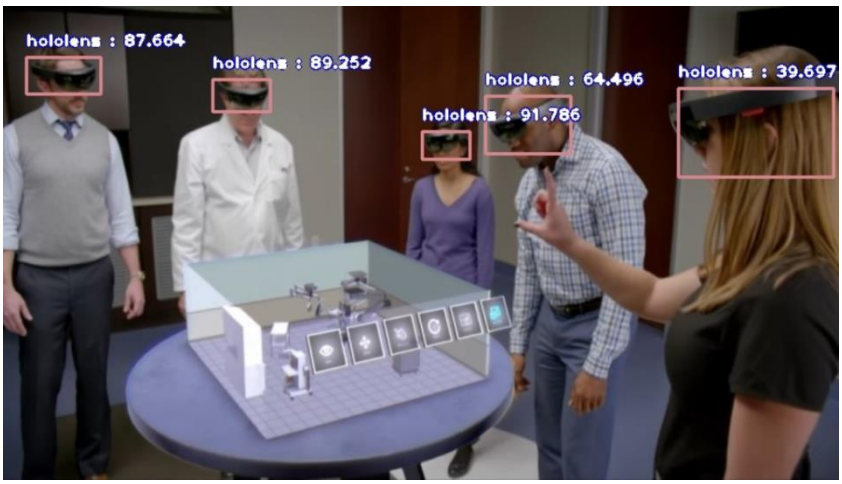


图 45-41 耳机检测

```
hololens  :  39.69653248786926  :  [611, 74, 751, 154]
hololens  :  87.6643180847168  :  [23, 46, 90, 79]
hololens  :  89.25175070762634  :  [191, 66, 243, 95]
hololens  :  64.49641585350037  :  [437, 81, 514, 133]
hololens  :  91.78624749183655  :  [380, 113, 423, 138]
```

ImageAI 现在提供类和方法，供您使用通过 DetectionModelTraining 类训练的自己的模型检测和识别图像中的自定义对象。您可以使用自定义训练的 YOLOv3 模式和训练期间生成的 detection_config.json 文件。

⑥ 自定义视频对象检测和分析

来自自定义 YOLOv3 模型的视频检测结果，训练用于检测视频中的 Hololens 耳机。

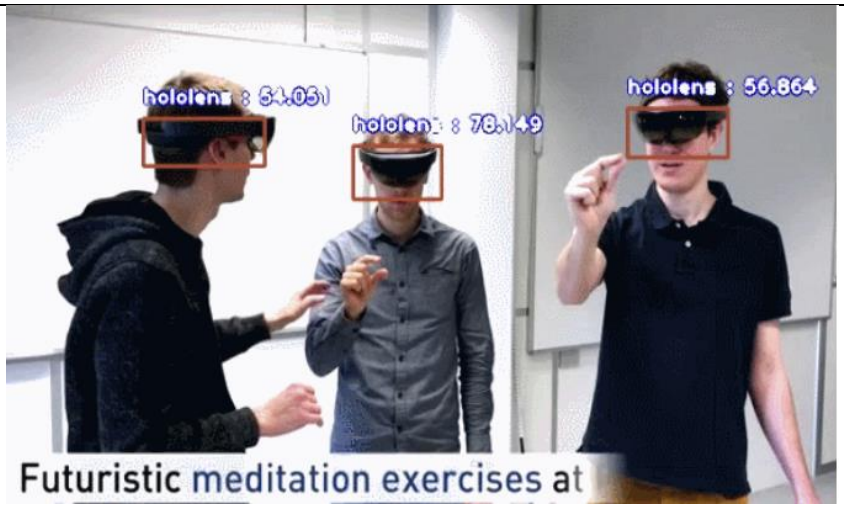


图 45-42 自定义视频中的耳机检测

3.安装流程

下面开始介绍 ImageAI 的具体用法，安装过程比较简单，直接调用 pip 工具即可。

- pip install imageai --upgrade
- pip install imageai-2.0.2-py3-none-any.whl

Processing c:\users\xxx\downloads\imageai-2.0.2-py3-none-any.whl

```
(base) C:\Users\xiuzhang>activate tensorflow
(tensorflow) C:\Users\xiuzhang>cd Downloads

(tensorflow) C:\Users>pip install imageai-2.0.2-py3-none-any.whl
Processing c:\users\xiuzhang\downloads\imageai-2.0.2-
```

```
py3-none-any.whl
Installing collected packages: imageai
Successfully installed imageai-2.0.2
```

本地安装下载链接如下：

- https://github.com/OlafenwaMoses/ImageAI/releases/download/2.0.2/imageai-2.0.2-py3-none-any.whl

依赖扩展包：

- Python 3.5.1
- Tensorflow 1.4.0
- Numpy 1.13.1
- SciPy 0.19.1
- OpenCV
- pillow
- Matplotlib
- h5py
- Keras 2.x

4.TinyYOLOv3 模型对象检测案例

(1) 案例实现

对象检测是计算机视觉领域中的一种技术，它处理识别和跟踪图像和视频中存在的对象。目标检测有多种应用，如人脸检测、车辆检测、行人计数、自动驾

驶汽车、安全系统等。ImageAI 提供了非常方便和强大的方法来对图像进行对象检测并从图像中提取每个对象。ImageAI 包含几乎所有最先进的深度学习算法的 Python 实现，如 RetinaNet、YOLOv3 和 TinyYOLOv3。

ImageAI 使用对象检测、视频检测和对象跟踪 API，无需访问网络即可调用。ImageAI 使用预先训练的模型并且可以轻松定制。ImageAI 中的 ObjectDetectionImageAI 库的类包含使用预训练模型对任何图像或图像集执行对象检测的函数。借助 ImageAI，可以检测和识别 80 种不同的常见日常物品。

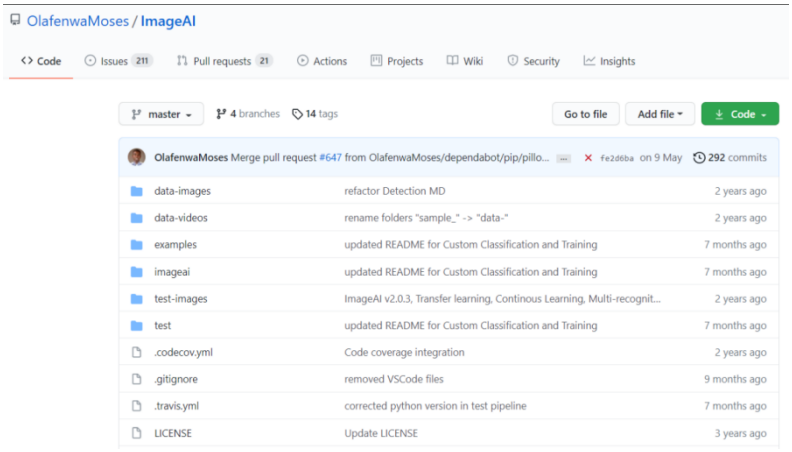


图 45-43 开源代码

下面开始实现一个简单的对象检测案例。

第一步，成功安装 ImageAI 后，下载包含将用于对象检测的分类模型的 TinyYOLOv3 模型文件。

- https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo-tiny.h5

第二步，创建如下图所示的文件夹。

- Object-detection：根文件夹
- models：存储预先训练的模型
- input：存储要执行对象检测的图像文件
- output：存储检测到对象的图像文件

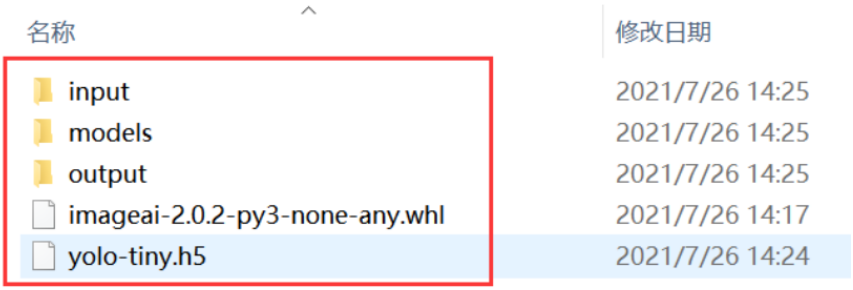


图 45-44 开源代码框架

```
|   test_detector.py
|
├──input
|       test_car.png
|
├──models
|       yolo-tiny.h5
|
└──output
```

第三步，创建 test_detector.py 文件，从 ImageAI 库导入 ObjectDetection 类。

- from imageai.Detection import ObjectDetection

第四步，实例化 ObjectDetection 并指定相关的路径文件。

- model_path = "./models/yolo-tiny.h5"
- input_path = "./input/test_car-02.png"
- output_path = "./output/pre_car-02.png"
- detector = ObjectDetection()

第五步，利用实例化 ObjectDetection 类，调用其中的函数实现不同的功能。该类包含以下功能调用预先训练模式：

- setModelTypeAsRetinaNet()
- setModelTypeAsYOLOv3()
- setModelTypeAsTinyYOLOv3()

第六步，使用预训练 TinyYOLOv3 模型，利用 setModelTypeAsTinyYOLOv3()函数加载模型。

- detector.setModelTypeAsTinyYOLOv3()

第七步，设置模型路径并实现图像对象检测。这里需要 detectObjectsFromImage 使用 detector 创建的对象，并输出结果。

- detector.setModelPath(model_path)
- detector.loadModel()
- detection = detector.detectObjectsFromImage(input_image=input_path, output_image_path=output_path)

完整代码如下所示：

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 26 14:26:27 2021
@author: xiuzhang
"""

from imageai.Detection import ObjectDetection

#实例化
detector = ObjectDetection()

#路径定义
model_path = "./models/yolo-tiny.h5"
input_path = "./input/test_car.png"
output_path = "./output/pre_car.png"

#预训练模式
detector.setModelTypeAsTinyYOLOv3()

#设置预训练模型路径
```

```
detector.setModelPath(model_path)


#加载模型
detector.loadModel()


#创建对象
detection                                                    =
detector.detectObjectsFromImage(input_image=input_pa
th, output_image_path=output_path)


#检测结果
for eachItem in detection:
    print(eachItem["name"]          ,          "          :          ",
eachItem["percentage_probability"])
```

输入图像如图 45-45 所示，位于 input 文件夹中。

图 45-45 输入图像

输出结果如下图所示，可以看到不同列类别的预测概率及对应轮廓。

```
car   :   53.41089963912964
car   :   55.577123165130615
car   :   60.30995845794678
car   :   62.34543323516846
car   :   72.32768535614014
car   :   78.66987586021423
car   :   83.33227038383484
car   :   87.24269270896912
car   :   89.72907662391663
```

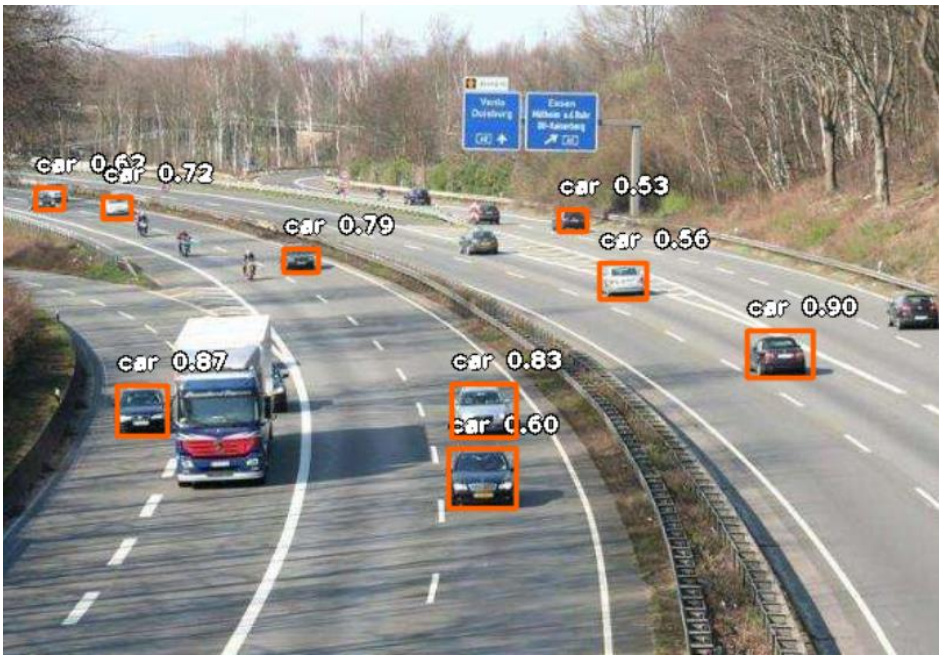


图 45-46 输出图像

同时，还可以预测其他车辆及行人，其预测准确率和结果分别如下图所示。

```
Use tf.where in 2.0, which has the same broadcast rule as np.where
car   :  54.752784967422485
car   :  59.75421667098999
car   :  63.04115056991577
car   :  74.26922917366028
car   :  91.21516942977905
car   :  97.27715849876404
car   :  97.50525951385498
person   :  53.488123416900635
person   :  56.76842927932739
person   :  72.32001423835754
```

图 45-47 预测准确率

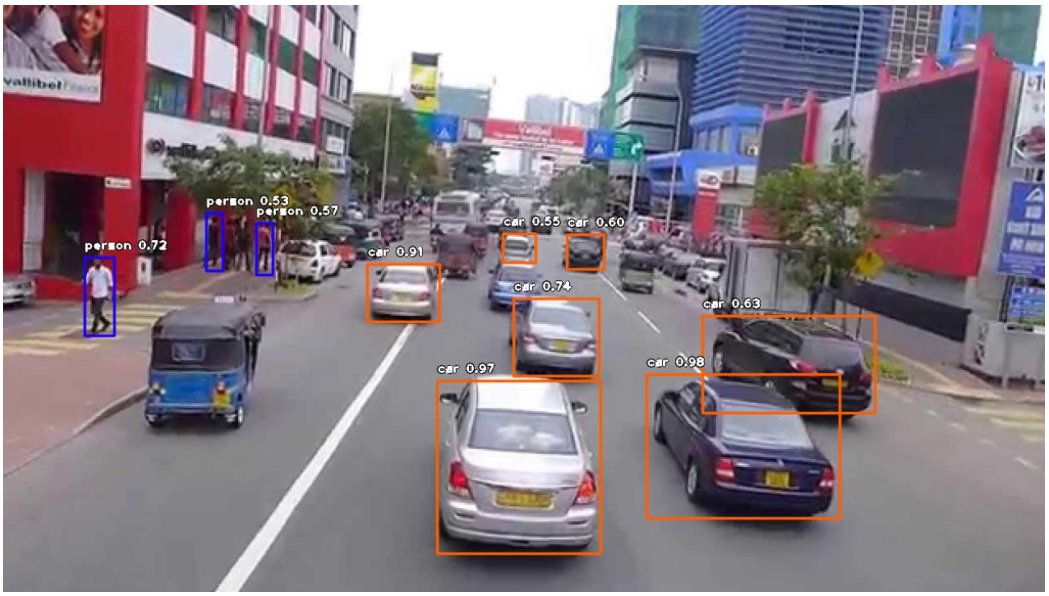


图 45-48 预测输出结果

(2) 学习建议

个人认为 Python 提供了良好的扩展包供大家学习，如果您对 CV 感兴趣，可以先阅读作者前面的图像处理系列文章。后面再进行目标检测的实验，而目标检测可以尝试先了解最简单调包的实现，后面再一步步告诉大家如果构建 YOLO 或 SSD 模型。

此外，Python 开源库提供了强大的源代码，大家可以去学习及修改对应的功能。比如：

- setModelTypeAsTinyYOLOv3 函数原型

```
om_objects_dict = {}
t_values = [person, bicycle, car, motorcycle, airplane,
          bus, train, truck, boat, traffic_light, fire_hydrant, stop_sign,
          parking_meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra,
          giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard,
          sports_ball, kite, baseball_bat, baseball_glove, skateboard, surfboard, tennis_racket,
          bottle, wine_glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange,
          broccoli, carrot, hot_dog, pizza, donot, cake, chair, couch, potted_plant, bed,
          dining_table, toilet, tv, laptop, mouse, remote, keyboard, cell_phone, microwave,
          oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy_bear, hair_dryer,
          toothbrush]
al_labels = ["person", "bicycle", "car", "motorcycle", "airplane",
          "bus", "train", "truck", "boat", "traffic light", "fire hydrant", "stop sign",
          "parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear",
          "giraffe", "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboar
          "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", "tennis r
          "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich
          "broccoli", "carrot", "hot dog", "pizza", "donot", "cake", "chair", "couch", "potted plant",
          "dining table", "toilet", "tv", "laptop", "mouse", "remote", "keyboard", "cell phone", "micro
          "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors", "teddy bear",
          "toothbrush"]

input_value, actual_label in zip(input_values, actual_labels):
if(input_value == True):
    custom_objects_dict[actual_label] = "valid"
else:
```

图 45-49 函数原型

```
import cv2
import numpy as np

from .colors import label_color


def draw_box(image, box, color, thickness=2):
    """ Draws a box on an image with a given color.

    # Arguments
        image     : The image to draw on.
        box       : A list of 4 elements (x1, y1, x2, y2).
        color     : The color of the box.
        thickness : The thickness of the lines to draw a box with.
    """
    b = np.array(box).astype(int)
    cv2.rectangle(image, (b[0], b[1]), (b[2], b[3]), color, thickness, cv2.LINE_AA)


def draw_caption(image, box, caption):
    """ Draws a caption above the box in an image.

    # Arguments
        image   : The image to draw on.
        box     : A list of 4 elements (x1, y1, x2, y2).
        caption : String containing the text to draw.
    """
    b = np.array(box).astype(int)
    #绘制标题
    cv2.putText(image, caption, (b[0], b[1] - 10), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 0), 3)
    cv2.putText(image, caption, (b[0], b[1] - 10), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 2)
```

图 45-50 函数原型

此外，该扩展包可以检测多种类型的对象，从上图源代码中可以看到，也可以看到绘制方块及 Text 的过程。

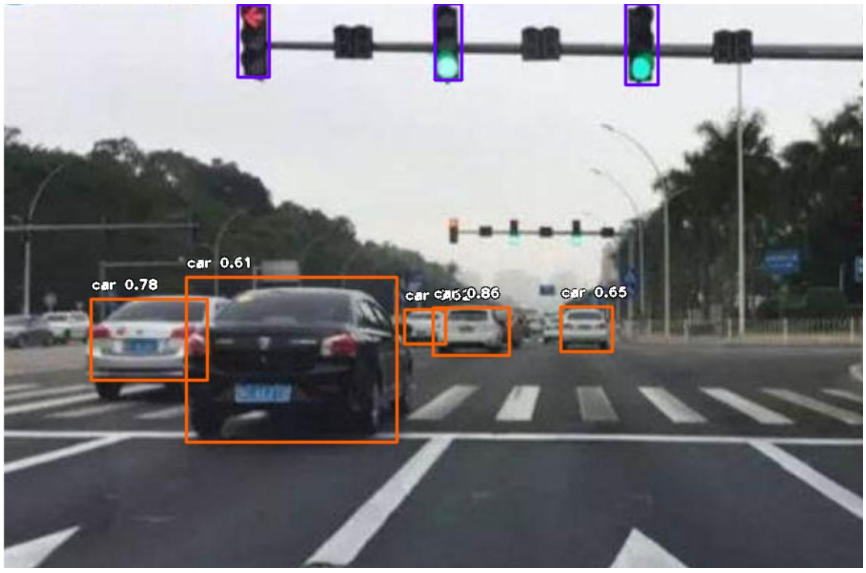


图 45-51 预测效果图

后续作者会想办法尝试自定义数据集进行目标识别，比如农产品、飞行器、自然灾害等。

- https://github.com/OlafenwaMoses/ImageAI/blob/master/imageai/Classification/CUSTOMCLASSIFICATION.md

最后推荐刘兄的系列文章，也真心不错。即：《深度学习和目标检测系列教程 2-300：小试牛刀，使用 ImageAI 进行对象检测》。

```
from imageai.Classification.Custom import CustomImageClassification
import os

execution_path = os.getcwd()

prediction = CustomImageClassification()
```

```
prediction.setModelTypeAsResNet50()
prediction.setModelPath(os.path.join(execution_path,
"idenprof_resnet_ex-056_acc-0.993062.h5"))
prediction.setJsonPath(os.path.join(execution_path,
"idenprof.json"))
prediction.loadModel(num_objects=10)

predictions, probabilities =
prediction.classifyImage(os.path.join(execution_path,
"4.jpg"), result_count=5)

for eachPrediction, eachProbability in zip(predictions,
probabilities):
    print(eachPrediction + " : " + eachProbability)
```

5.总结

写到这里，这篇目标检测入门文章就介绍完毕，文章内容主要包括：

- 一.目标检测入门普及
  - 1.什么是目标检测?
  - 2.目标检测的核心问题是什么?
  - 3.目标检测算法常用分类

- 4.目标检测常见算法
- 5.目标检测应用
- 6.目标检测未来挑战
- 7.目标检测原理简述

■ 二.ImageAI 简介

■ 三.安装流程

■ 四.TinyYOLOv3 模型对象检测案例

- 1.案例实现
- 2.学习建议

希望大家喜欢这篇文章，并能结合本章知识点，围绕自己的研究领域或工程项目进行深入的学习，实现所需的图像识别的研究或论文。

大学之道在明明德，在亲民，在止于至善。这周又回答了很多博友的问题，有大一学生的困惑，有论文的咨询，也有老乡和考博的疑问，还有无数博友奋斗路上的相互勉励。虽然自己早已忙成狗，但总忍不住去解答别人的问题。最后那一句感谢和祝福，永远是我最大的满足。虽然会花费我一些时间，但也挺好的，无所谓了，跟着心走。不负遇见，感恩同行。莫愁前路无知己，继续加油。晚安娜和珞。

参考文献：

[1] yegeli 老师. 目标检测（Object Detection）. https://blog.csdn.net/yegeli/article/details/109861867.

[2] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, et al. Object Detection in 20 Years: A Survey, Computer Vision and Pattern Recognition. https://arxiv.org/abs/1905.05055v2.

[3] Object Detection in 20 Years: A Survey 综述：目标检测的二十年 - 牛戈老师

[4] SIGAI 老师. 目标检测最新进展总结与展望. https://zhuanlan.zhihu.com/p/46595846.

[5] clover_my. 论文笔记-2019-Object Detection in 20 Years: A Survey. https://blog.csdn.net/clover_my/article/details/92794719.

[6] https://github.com/OlafenwaMoses/ImageAI.

[7] https://imageai-cn.readthedocs.io/zh_CN/latest.

[8] https://github.com/OlafenwaMoses/ImageAI/blob/master/imageai/Classification/CUSTOMCLASSIFICATION.md.

[9] https://github.com/eastmountyxz/ImageProcessing-Python.

[10] 刘润森. 深度学习和目标检测系列教程 2-300：小试牛刀，使用 ImageAI 进行对象检测. https://maoli.blog.csdn.net/article/details/118367859.

第 46 篇 Keras 构建 CNN 识别阿拉伯手写文字图像

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前文介绍了目标检测基础知识，并通过 ImageAI 实现对象检测的经典案例。这篇文章将详细讲解 Keras 深度学习构建 CNN 模型识别阿拉伯手写文字图像，一篇非常经典的图像分类文章。

1.数据集描述

阿拉伯字母共有共有 28 个，如图 46-1 所示。

图 46-1 阿拉伯文字

众所周知，手写识别数字是非常经典的图像分类数据集（MNIST），这里则使用 kaggle 的另一种数据集。该数据集是由 60 名参与者书写的 16800 个字符组成。数据集地址如下：

- https://www.kaggle.com/mloey1/ahcd1

网站显示的效果图如下所示。

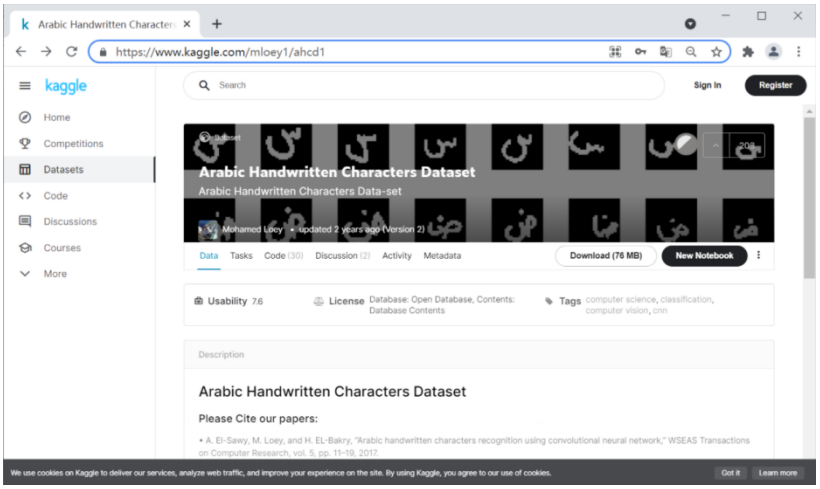


图 46-2 数据集网站

整个数据集在两种形式上写下每个字符（从“alef”到“yeh”）共十次，数据集包含文件如图 46-3 所示。

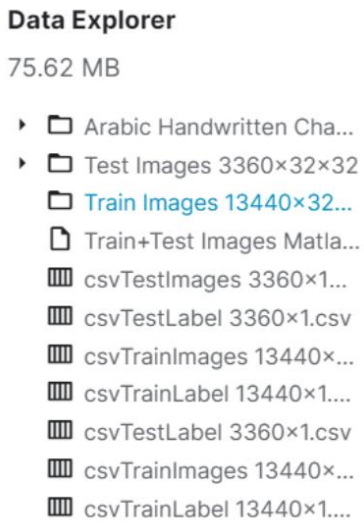


图 46-3 数据集呈现

解压后如下图所示。

图 46-4 数据集

其中，训练集共有 13440 个字符图像，28 个类别，每章图像 32x32 大小，如下图所示。

- Train Images 13440x32x32

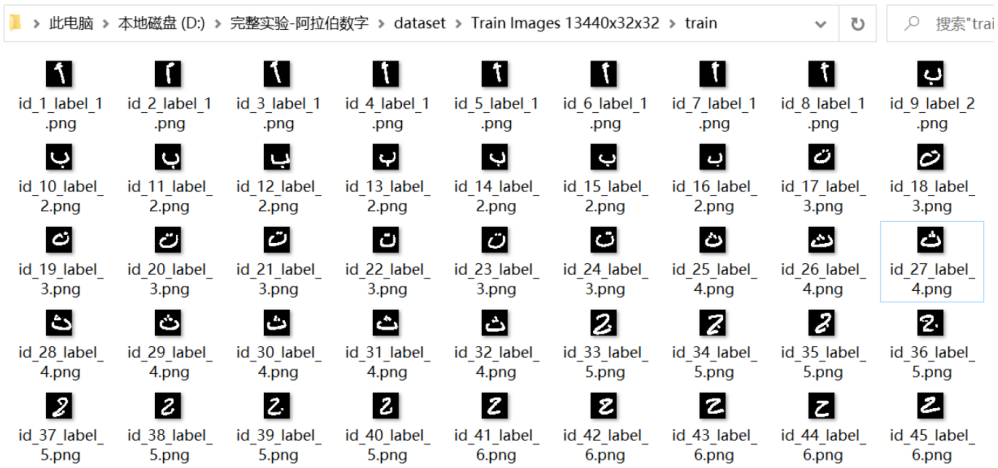


图 46-5 训练集

训练集共有 3360 个字符图像，28 个类别，每章图像 32x32 大小，如图所示。

- Test Images 3360x32x32

图 46-6 训练集

同时，数据集中包含 CSV 文件，对应图像像素的分布情况，比如测试集的数据如下图所示。

- csvTestImages 3360x1024.csv

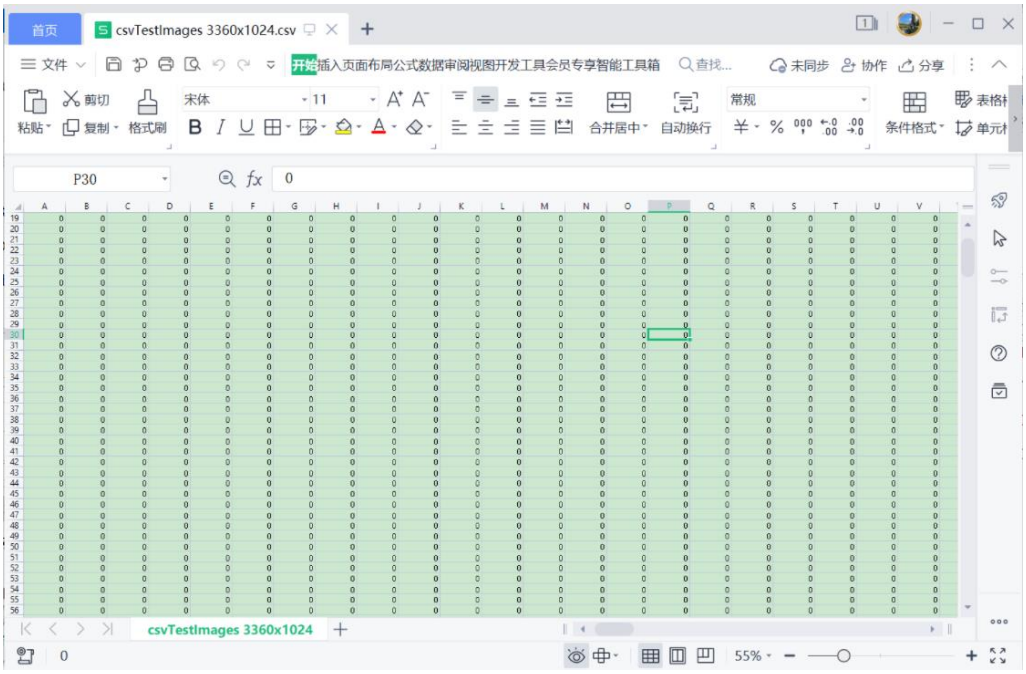


图 46-7 测试集数据分布情况

对应类别如下图所示，从 1 到 28，因此在数据处理时应转换为 0 到 27 下标。

- csvTrainLabel 13440x1.csv

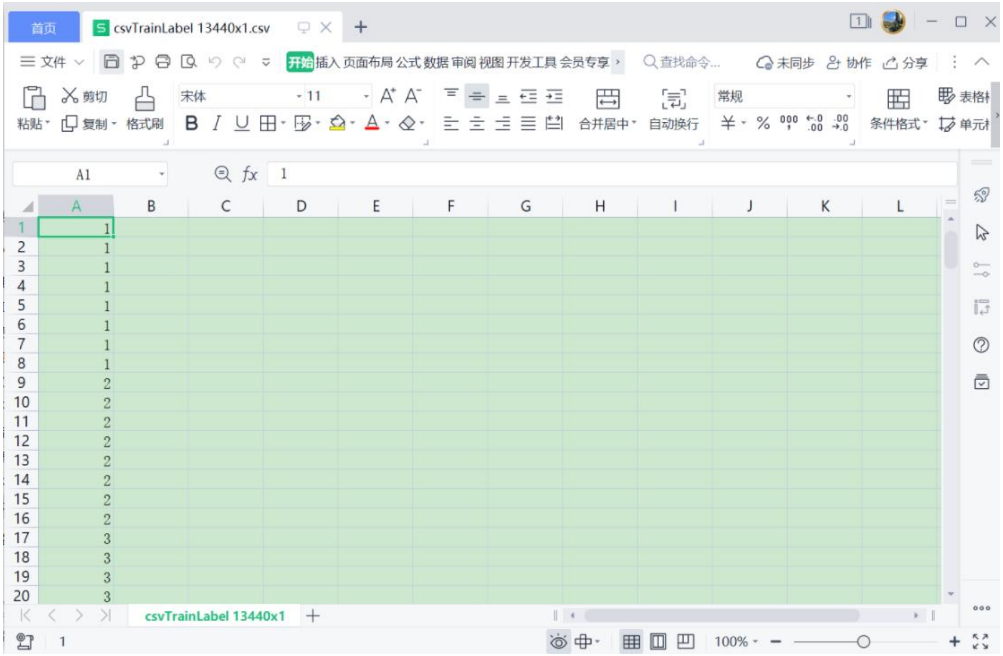


图 46-8 数据集对应类标

2.数据读取

首先，我们需要读取数据，下面代码是读取的过程。

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jul  7 18:54:36 2021
@author: xiuzhang CSDN
"""
import numpy as np
import pandas as pd
from IPython.display import display
import csv
from PIL import Image
from scipy.ndimage import rotate


#----------------------------------------------------------------------------
#                              第一步 读取数据
#----------------------------------------------------------------------------
```

```
------------------------------------------

#训练数据 images 和 labels

letters_training_images_file_path =

"dataset/csvTrainImages 13440x1024.csv"

letters_training_labels_file_path = "dataset/csvTrainLabel

13440x1.csv"

#测试数据 images 和 labels

letters_testing_images_file_path =

"dataset/csvTestImages 3360x1024.csv"

letters_testing_labels_file_path = "dataset/csvTestLabel

3360x1.csv"


#加载数据

training_letters_images =

pd.read_csv(letters_training_images_file_path,

header=None)

training_letters_labels =

pd.read_csv(letters_training_labels_file_path,

header=None)

testing_letters_images =

pd.read_csv(letters_testing_images_file_path,
```

```
header=None)
testing_letters_labels =
pd.read_csv(letters_testing_labels_file_path,
header=None)
print("%d 个 32x32 像素的训练阿拉伯字母图像" %
training_letters_images.shape[0])
print("%d 个 32x32 像素的测试阿拉伯字母图像" %
testing_letters_images.shape[0])
print(training_letters_images.head())
print(np.unique(training_letters_labels))
```

输出结果如图 46-9 所示。

```
13440个32x32像素的训练阿拉伯字母图像。
3360个32x32像素的测试阿拉伯字母图像。
   0     1     2     3     4     5     ...  1018  1019  1020  1021
1022  1023
0     0     0     0     0     0     0  ...     0     0     0     0
0     0
1     0     0     0     0     0     0  ...     0     0     0     0
0     0
2     0     0     0     0     0     0  ...     0     0     0     0
0     0
3     0     0     0     0     0     0  ...     0     0     0     0
0     0
4     0     0     0     0     0     0  ...     0     0     0     0
0     0

[5 rows x 1024 columns]
```

图 46-9 输出结果

输出类别共有 28 类。

- [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]

3.数据预处理

主要包括两个核心步骤：数值特征转换图像、图像标准化。

（1）数值特征转换图像

接着尝试将 CSV 文件的数值特征转换为一张图像，图像像素为 32x32。需要注意，由于原始数据集被反射，因此使用 np.flip 翻转图像，并通过 rotate 旋转从而获得更好的图像。

```
#------------------------------------------------
------------------------------------------------
#                          第二步 数值转换为图像特征
#------------------------------------------------
------------------------------------------------
#原始数据集被反射使用 np.flip 翻转它 通过 rotate 旋转从而获得更好的图像
def                convert_values_to_image(image_values, display=False):
    #转换成 32x32
    image_array = np.asarray(image_values)
    image_array                                                =
```

```
image_array.reshape(32,32).astype('uint8')
    #翻转+旋转
    image_array = np.flip(image_array, 0)
    image_array = rotate(image_array, -90)
    #图像显示
    new_image = Image.fromarray(image_array)
    if display == True:
        new_image.show()
    return new_image


convert_values_to_image(training_letters_images.loc[0],
True)
```

输出结果是一张字母图像。

图 46-10 输出图像

（2）图像标准化处理

图像标准化是将图像中的每个像素点除以 255，从而缩放并标准化到[0, 1]范围。同时，数据类型进行相应转换。

```
#----------------------------------------
```

```
------------------------------------------
#                第三步 图像标准化处理
#------------------------------------------
------------------------------------------
training_letters_images_scaled                    =
training_letters_images.values.astype('float32')/255
training_letters_labels                           =
training_letters_labels.values.astype('int32')
testing_letters_images_scaled                     =
testing_letters_images.values.astype('float32')/255
testing_letters_labels                            =
testing_letters_labels.values.astype('int32')
print("Training images of letters after scaling")
print(training_letters_images_scaled.shape)
print(training_letters_images_scaled[0:5])
```

输出结果如下所示：

```
Training images of letters after scaling
(13440, 1024)
[[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）输出 One-hot 编码转换

由于本文是一个多分类问题，共包括 28 个阿拉伯字母，因此需要将将类别（0 到 27）进行 One-hot 转换，导入如下扩展包。

- from keras.utils import to_categorical

to_categorical 是将类别向量转换为二进制（只有 0 和 1）的矩阵类型表示,其表现为将原有的类别向量转换为 One-hot 编码的形式。举一个简单的代码示例如下：

```
from keras.utils.np_utils import *
#类别向量定义
b = [0,1,2,3,4,5,6,7,8]
#调用 to_categorical 将 b 按照 9 个类别来进行转换
b = to_categorical(b, 9)
print(b)

"""
执行结果如下：
[[1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

```
 [0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]]
"""
```

从以上代码运行可以看出，将原来类别向量中的每个值都转换为矩阵里的一个行向量，从左到右依次是 0 到 8 个类别。比如 2 表示为[0. 0. 1. 0. 0. 0. 0. 0. 0.]，只有第 3 个为 1，作为有效位，其余全部为 0。one_hot encoding（独热编码）又称为一位有效位编码，上边代码例子中其实就是将类别向量转换为独热编码的类别矩阵。即如下转换：

```
      0   1   2   3   4   5   6   7   8
0=> [1. 0. 0. 0. 0. 0. 0. 0. 0.]
1=> [0. 1. 0. 0. 0. 0. 0. 0. 0.]
2=> [0. 0. 1. 0. 0. 0. 0. 0. 0.]
3=> [0. 0. 0. 1. 0. 0. 0. 0. 0.]
4=> [0. 0. 0. 0. 1. 0. 0. 0. 0.]
5=> [0. 0. 0. 0. 0. 1. 0. 0. 0.]
6=> [0. 0. 0. 0. 0. 0. 1. 0. 0.]
7=> [0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

```
8=> [0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

该部分的预处理代码如下：

```
#------------------------------------------
------------------------------------
#                          第四步 输出 One-hot 编码转换
#------------------------------------------
------------------------------------
import keras
from keras.utils import to_categorical
number_of_classes = 28
training_letters_labels_encoded                    =
to_categorical(training_letters_labels-1,


num_classes=number_of_classes)
testing_letters_labels_encoded                     =
to_categorical(testing_letters_labels-1,


num_classes=number_of_classes)
print(training_letters_labels)
print(training_letters_labels_encoded)
```

```
print(training_letters_images_scaled.shape)

# (13440, 1024)
```

输出结果如下图所示：

```
Using TensorFlow backend.
[[ 1]
 [ 1]
 [ 1]
 ...
 [28]
 [28]
 [28]]
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
(13440, 1024)
```

图 46-11 输出结果

（4）形状修改

深度学习中输入的形状非常重要，本文将输入图像修改为 32x32x1，在 Keras 中接收一个四维数组作为输入，对应形状为：

- 图像样本总数
- 行
- 列
- 通道

代码如下：

```
#----------------------------------------------------

--------------------
```

```
#                          第五步 形状修改
#------------------------------------------------
--------------------
#输入形状 32x32x1
training_letters_images_scaled                    =
training_letters_images_scaled.reshape([-1, 32, 32, 1])
testing_letters_images_scaled                     =
testing_letters_images_scaled.reshape([-1, 32, 32, 1])
print(training_letters_images_scaled.shape,
      training_letters_labels_encoded.shape,
      testing_letters_images_scaled.shape,
      testing_letters_labels_encoded.shape)
```

本文图像是 32x32 的灰度图，输出结果如下所示：

- (13440, 32, 32, 1) (13440, 28) (3360, 32, 32, 1) (3360, 28)

4.CNN 模型搭建

（1）卷积神经网络概念

卷积神经网络的英文是 Convolutional Neural Network，简称 CNN。它通常应用于图像识别和语音识等领域，并能给出更优秀的结果，也可以应用于视频分析、机器翻译、自然语言处理、药物发现等领域。著名的阿尔法狗让计算机看懂围棋就是基于卷积神经网络的。

神经网络是由很多神经层组成，每一层神经层中存在很多神经元，这些神经元是识别事物的关键，当输入是图片时，其实就是一堆数字。

首先，卷积是什么意思呢?

卷积是指不在对每个像素做处理，而是对图片区域进行处理，这种做法加强了图片的连续性，看到的是一个图形而不是一个点，也加深了神经网络对图片的理解。

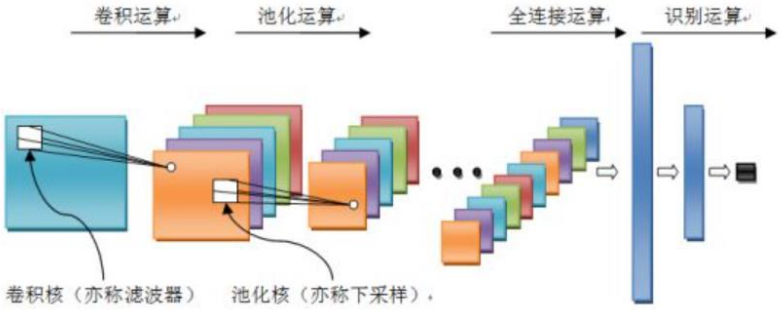


图 46-12 卷积神经网络

卷积神经网络批量过滤器，持续不断在图片上滚动搜集信息，每一次搜索都是一小块信息，整理这一小块信息之后得到边缘信息。比如第一次得出眼睛鼻子轮廓等，再经过一次过滤，将脸部信息总结出来，再将这些信息放到全神经网络中进行训练，反复扫描最终得出的分类结果。如图 46-13 所示，猫的一张照片需要转换为数学的形式，这里采用长宽高存储，其中黑白照片的高度为 1，彩色照片的高度为 3(RGB)。

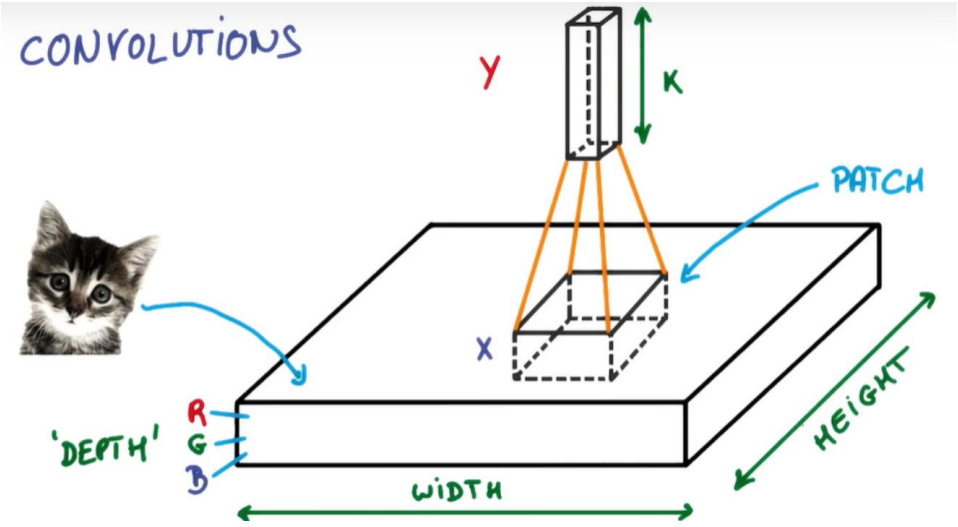


图 46-13 卷积处理

过滤器搜集这些信息，将得到一个更小的图片，再经过压缩增高信息嵌入到普通神经层上，最终得到分类的结果，这个过程即是卷积。Convnets 是一种在空间上共享参数的神经网络，如下图所示，它将一张 RGB 图片进行压缩增高，得到一个很长的结果。

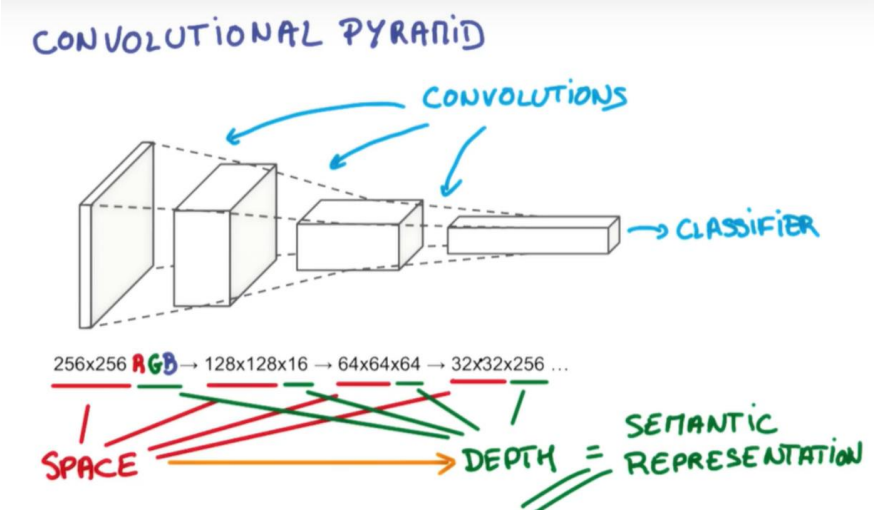


图 46-14 CNN 模型结构

一个卷积网络是组成深度网络的基础，我们将使用数层卷积而不是数层的矩阵相乘。如上图所示，让它形成金字塔形状，金字塔底是一个非常大而浅的图片，

仅包括红绿蓝，通过卷积操作逐渐挤压空间的维度，同时不断增加深度，使深度信息基本上可以表示出复杂的语义。同时，你可以在金字塔的顶端实现一个分类器，所有空间信息都被压缩成一个标识，只有把图片映射到不同类的信息保留，这就是 CNN 的总体思想。

上图的具体流程如下：

- 首先，这是有一张彩色图片，它包括 RGB 三原色分量，图像的长和宽为 256*256，三个层面分别对应红（R）、绿（G）、蓝（B）三个图层，也可以看作像素点的厚度。
- 其次，CNN 将图片的长度和宽度进行压缩，变成 128x128x16 的方块，压缩的方法是把图片的长度和宽度压小，从而增高厚度。
- 再次，继续压缩至 64x64x64，直至 32x32x256，此时它变成了一个很厚的长条方块，我们这里称之为分类器 Classifier。该分类器能够将我们的分类结果进行预测，MNIST 手写体数据集预测结果是 10 个数字，比如[0,0,0,1,0,0,0,0,0,0]表示预测的结果是数字 3，Classifier 在这里就相当于这 10 个序列。
- 最后，CNN 通过不断压缩图片的长度和宽度，增加厚度，最终会变成了一个很厚的分类器，从而进行分类预测。

近几年神经网络飞速发展，其中一个很重要的原因就是 CNN 卷积神经网络的提出，这也是计算机视觉处理的飞跃提升。关于 TensorFlow 中的 CNN，Google 公司也出了一个非常精彩的视频教程，也推荐大家去学习。

（2）模型设计

该部分采用 Keras 搭建四层 CNN 网络，是本文的核心过程，具体过程如下。

```
#-----------------------------------------------
-----------------------------------------------
#                          第六步  CNN 模型设计
#-----------------------------------------------
-----------------------------------------------
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D,
GlobalAveragePooling2D, BatchNormalization, Dropout,
Dense
#定义模型
def create_model(optimizer='adam',
kernel_initializer='he_normal', activation='relu'):
    #第一个卷积层
    model = Sequential()
    model.add(Conv2D(filters=16, kernel_size=3,
padding='same', input_shape=(32, 32, 1),
kernel_initializer=kernel_initializer, activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
```

```
    model.add(Dropout(0.2))

    #第二个卷积层
    model.add(Conv2D(filters=32, kernel_size=3,
padding='same', kernel_initializer=kernel_initializer,
activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
    model.add(Dropout(0.2))

    #第三个卷积层
    model.add(Conv2D(filters=64, kernel_size=3,
padding='same', kernel_initializer=kernel_initializer,
activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
    model.add(Dropout(0.2))

    #第四个卷积层
    model.add(Conv2D(filters=128, kernel_size=3,
padding='same', kernel_initializer=kernel_initializer,
```

```
activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
    model.add(Dropout(0.2))
    model.add(GlobalAveragePooling2D())

    #全连接层输出 28 类结果
    model.add(Dense(28, activation='softmax'))

    #损失函数定义
    model.compile(loss='categorical_crossentropy',
metrics=['accuracy'], optimizer=optimizer)
    return model

#创建模型
model = create_model(optimizer='Adam',
kernel_initializer='uniform', activation='relu')
model.summary()
```

模型包括四个卷积神经网络，通过全连接层实现分类。每个卷积层内容如下：

- 输入层包括 16 个特征图，大小为 3×3 和一个 relu 激活函数；
- 接着是批量标准化层，主要用于解决特征分布在训练和测试数据中的变

化，BN 层添加在激活函数前，对输入激活函数的输入进行归一化，解决数据发送偏移和增大的影响；

- 池化层主要对输入进行下采样，从而减小参数的学习次数和训练时间；
- 使用 dropout 正则化，设置参数为 0.2，即被配置为随机排除层中 20% 的神经元以减少过度拟合。

总共是 16、32、64、128 个元素的隐藏层，最后通过输出层（类别 28），并使用 softmax 激活函数，利用交叉熵作为损失函数。整个模型的结构输出如下所示：

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        160
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 28)                3612
=================================================================
Total params: 101,724
Trainable params: 101,244
Non-trainable params: 480
_________________________________________________________________
```

图 46-15 CNN 模型结构

（3）模型绘制

在 Keras 中，我们可以调用 keras.utils.vis_utils 模块绘制模型，该模块提供了 graphviz 绘制功能。

```
#----------------------------------------------------------------------
#                          第七步  模型绘制
#----------------------------------------------------------------------
from keras.utils.vis_utils import plot_model
from IPython.display import Image as IPythonImage

plot_model(model, to_file="model.png", show_shapes=True)
display(IPythonImage('model.png'))
```

输出结果如图 46-16 所示：

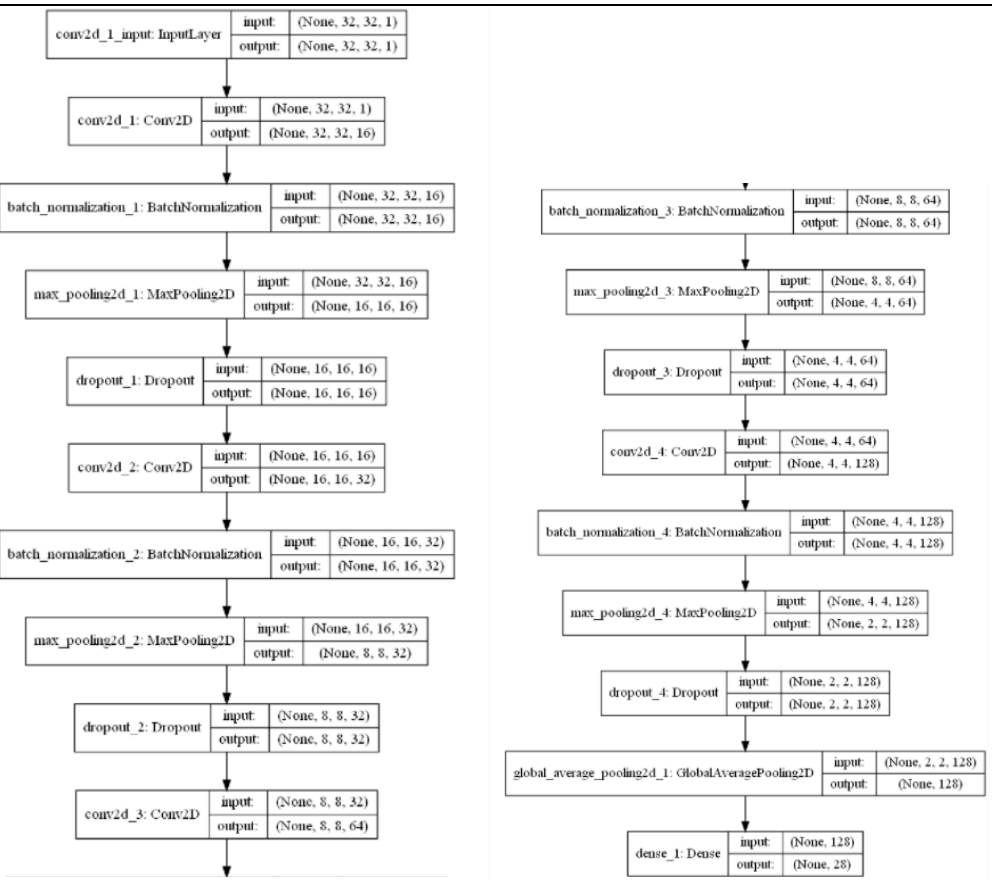


图 46-16 graphviz 绘制模型结构

5.实验评估

（1）模型训练

接着是模型训练，这里使用检查点来保存模型权重，同时 batch 大小设置为 20，epochs 设置为 15。

```
#-----------------------------------------------------------
-----------------------------------
#                                第八步  模型训练
#-----------------------------------------------------------
```

```
_________________________________

from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath='weights.hdf5',

                               verbose=1,

                               save_best_only=True)

history = model.fit(training_letters_images_scaled,

                    training_letters_labels_encoded,


validation_data=(testing_letters_images_scaled,


testing_letters_labels_encoded),

                    epochs=15,

                    batch_size=20,

                    verbose=1,

                    callbacks=[checkpointer])

print(history)
```

训练过程如下图所示：

```
Train on 13440 samples, validate on 3360 samples
Epoch 1/15
13440/13440 [==============================] - 21s 2ms/step - loss: 1.3694 - accuracy: 0.5759 - val_loss:

Epoch 00001: val_loss improved from inf to 0.86641, saving model to weights.hdf5
Epoch 2/15
13440/13440 [==============================] - 20s 1ms/step - loss: 0.4939 - accuracy: 0.8394 - val_loss:

Epoch 00002: val_loss improved from 0.86641 to 0.28324, saving model to weights.hdf5
Epoch 3/15
13440/13440 [==============================] - 20s 1ms/step - loss: 0.3559 - accuracy: 0.8861 - val_loss:

Epoch 00003: val_loss did not improve from 0.28324
Epoch 4/15
13440/13440 [==============================] - 20s 1ms/step - loss: 0.2913 - accuracy: 0.9059 - val_loss:

Epoch 00004: val_loss improved from 0.28324 to 0.22108, saving model to weights.hdf5
Epoch 5/15
13440/13440 [==============================] - 21s 2ms/step - loss: 0.2557 - accuracy: 0.9183 - val_loss:

Epoch 00005: val_loss improved from 0.22108 to 0.17819, saving model to weights.hdf5
Epoch 6/15
13440/13440 [==============================] - 18s 1ms/step - loss: 0.2287 - accuracy: 0.9237 - val_loss:

Epoch 00006: val_loss improved from 0.17819 to 0.16022, saving model to weights.hdf5
Epoch 7/15
13440/13440 [==============================] - 21s 2ms/step - loss: 0.2162 - accuracy: 0.9295 - val_loss:
```

图 46-17 训练过程

（2）绘制误差和准确率曲线

该部分核心代码如下：

```
#----------------------------------------------------------
----------------------------------
#                                  第九步 绘制图形
#----------------------------------------------------------
----------------------------------
import matplotlib.pyplot as plt


def plot_loss_accuracy(history):
```

```
    # Loss
    plt.figure(figsize=[8,6])
    plt.plot(history.history['loss'],'r',linewidth=3.0)
    plt.plot(history.history['val_loss'],'b',linewidth=3.0)
    plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
    plt.xlabel('Epochs ',fontsize=16)
    plt.ylabel('Loss',fontsize=16)
    plt.title('Loss Curves',fontsize=16)

    # Accuracy
    plt.figure(figsize=[8,6])
    plt.plot(history.history['accuracy'],'r',linewidth=3.0)

plt.plot(history.history['val_accuracy'],'b',linewidth=3.0)
    plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
    plt.xlabel('Epochs ',fontsize=16)
    plt.ylabel('Accuracy',fontsize=16)
    plt.title('Accuracy Curves',fontsize=16)
```

```
plot_loss_accuracy(history)
```

绘制如下图所示结果。

图 46-18 Loss 曲线

（3）模型评估

注意，由于上一个步骤已经训练好了模型，这里我们增加一个判断，防止重复训练，直接调用 weights.hdf5 模型进行预测。

```
#----------------------------------------------------------------
----------------------------------------
#                                第八步  模型训练+输出结果
#----------------------------------------------------------------
----------------------------------------
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt


#绘制图形
```

```
def plot_loss_accuracy(history):
    # Loss
    plt.figure(figsize=[8,6])
    plt.plot(history.history['loss'],'r',linewidth=3.0)
    plt.plot(history.history['val_loss'],'b',linewidth=3.0)
    plt.legend(['Training loss', 'Validation
Loss'],fontsize=18)
    plt.xlabel('Epochs ',fontsize=16)
    plt.ylabel('Loss',fontsize=16)
    plt.title('Loss Curves',fontsize=16)

    # Accuracy
    plt.figure(figsize=[8,6])
    plt.plot(history.history['accuracy'],'r',linewidth=3.0)

plt.plot(history.history['val_accuracy'],'b',linewidth=3.0)
    plt.legend(['Training Accuracy', 'Validation
Accuracy'],fontsize=18)
    plt.xlabel('Epochs ',fontsize=16)
    plt.ylabel('Accuracy',fontsize=16)
    plt.title('Accuracy Curves',fontsize=16)
```

```
#混淆矩阵
def get_predicted_classes(model, data, labels=None):
    image_predictions = model.predict(data)
    predicted_classes = np.argmax(image_predictions,
axis=1)
    true_classes = np.argmax(labels, axis=1)
    return predicted_classes, true_classes,
image_predictions

def get_classification_report(y_true, y_pred):
    print(classification_report(y_true, y_pred, digits=4)) #
小数点 4 位

checkpointer = ModelCheckpoint(filepath='weights.hdf5',
                               verbose=1,
                               save_best_only=True)
flag = "test"
if flag=="train":
    history = model.fit(training_letters_images_scaled,
                        training_letters_labels_encoded,
```

```
validation_data=(testing_letters_images_scaled,

testing_letters_labels_encoded),
                              epochs=15,
                              batch_size=20,
                              verbose=1,
                              callbacks=[checkpointer])
    print(history)
    plot_loss_accuracy(history)
else:
    #加载具有最佳验证损失的模型
    model.load_weights('weights.hdf5')
    metrics                                                    =
model.evaluate(testing_letters_images_scaled,

testing_letters_labels_encoded,
                                    verbose=1)
    print("Test Accuracy: {}".format(metrics[1]))
    print("Test Loss: {}".format(metrics[0]))
```

```
    y_pred, y_true, image_predictions =
get_predicted_classes(model,

testing_letters_images_scaled,

testing_letters_labels_encoded)
    get_classification_report(y_true, y_pred)
```

最终输出结果如下所示：

```
3360/3360 [==============================] - 2s
466us/step
Test Accuracy: 0.9639880657196045
Test Loss: 0.13888128581900328
              precision    recall  f1-score   support

           0       0.9756    1.0000    0.9877       120
           1       0.9675    0.9917    0.9794       120
           2       0.9187    0.9417    0.9300       120
           3       0.9504    0.9583    0.9544       120
           4       0.9600    1.0000    0.9796       120
           5       0.9590    0.9750    0.9669       120
           6       0.9587    0.9667    0.9627       120
```

7	0.9435	0.9750	0.9590	120
8	0.9134	0.9667	0.9393	120
9	0.9370	0.9917	0.9636	120
10	0.9817	0.8917	0.9345	120
11	0.9008	0.9833	0.9402	120
12	0.9832	0.9750	0.9791	120
13	0.9818	0.9000	0.9391	120
14	0.9912	0.9333	0.9614	120
15	0.9375	1.0000	0.9677	120
16	0.9912	0.9417	0.9658	120
17	0.9912	0.9417	0.9658	120
18	0.9914	0.9583	0.9746	120
19	0.9286	0.9750	0.9512	120
20	0.9739	0.9333	0.9532	120
21	0.9600	1.0000	0.9796	120
22	1.0000	1.0000	1.0000	120
23	0.9916	0.9833	0.9874	120
24	0.9737	0.9250	0.9487	120
25	0.9832	0.9750	0.9791	120
26	0.9741	0.9417	0.9576	120
27	1.0000	0.9667	0.9831	120

accuracy			0.9640	3360
macro avg	0.9650	0.9640	0.9640	3360
weighted avg	0.9650	0.9640	0.9640	3360

（4）绘制预测图片

最后，绘制测试数据集的图片，包括真实类别和预测类别。

```
#------------------------------------------
------------------------------------
#                          第九步 绘制测试图像
#------------------------------------------
------------------------------------
fig = plt.figure(0, figsize=(14,14))
indices = np.random.randint(0,
testing_letters_labels.shape[0], size=49)
y_pred =
np.argmax(model.predict(training_letters_images_scaled
), axis=1)

for i, idx in enumerate(indices):
    plt.subplot(7,7,i+1)
```

```
    image_array                                                =
training_letters_images_scaled[idx][:,:,0]
    image_array = np.flip(image_array, 0)
    image_array = rotate(image_array, -90)


    plt.imshow(image_array, cmap='gray')
    plt.title("Pred: {} - Label: {}".format(y_pred[idx],
                (training_letters_labels[idx] -1)))
    plt.xticks([])
    plt.yticks([])
plt.show()
plt.savefig("resutl.png")
```

输出结果如下图所示：

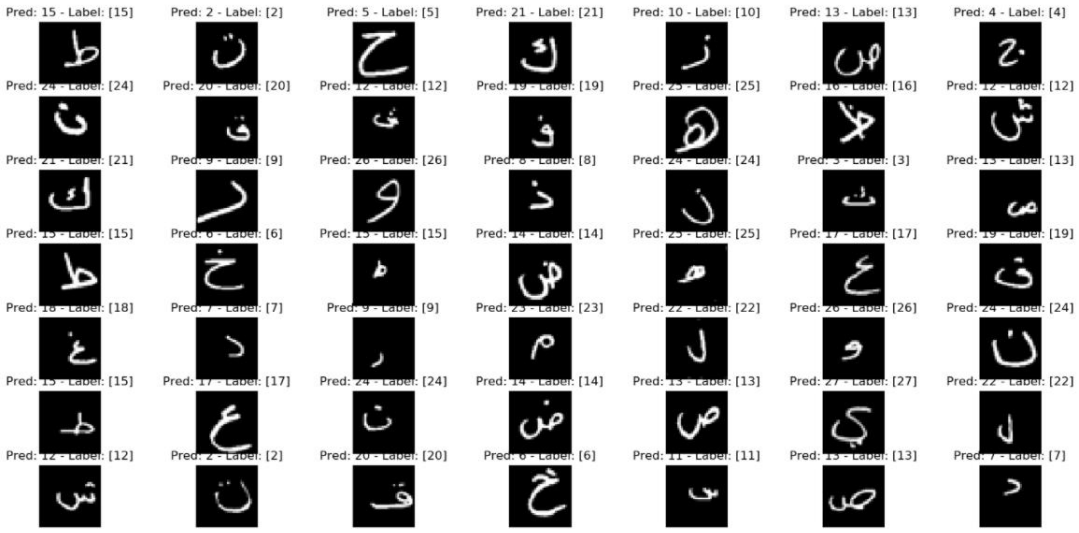


图 46-19 预测结果

6.完整代码

最终的完整代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jul  7 18:54:36 2021
@author: xiuzhang CSDN
"""
import numpy as np
import pandas as pd
from IPython.display import display
import csv
from PIL import Image
from scipy.ndimage import rotate


#----------------------------------------------------------------------
#                          第一步  读取数据
#----------------------------------------------------------------------
#训练数据 images 和 labels
```

```
letters_training_images_file_path =
"dataset/csvTrainImages 13440x1024.csv"
letters_training_labels_file_path = "dataset/csvTrainLabel
13440x1.csv"
#测试数据 images 和 labels
letters_testing_images_file_path =
"dataset/csvTestImages 3360x1024.csv"
letters_testing_labels_file_path = "dataset/csvTestLabel
3360x1.csv"

#加载数据
training_letters_images =
pd.read_csv(letters_training_images_file_path,
header=None)
training_letters_labels =
pd.read_csv(letters_training_labels_file_path,
header=None)
testing_letters_images =
pd.read_csv(letters_testing_images_file_path,
header=None)
testing_letters_labels =
```

```
pd.read_csv(letters_testing_labels_file_path,
header=None)
print("%d 个 32x32 像素的训练阿拉伯字母图像" %
training_letters_images.shape[0])
print("%d 个 32x32 像素的测试阿拉伯字母图像" %
testing_letters_images.shape[0])
print(training_letters_images.head())
print(np.unique(training_letters_labels))


#----------------------------------------
----------------------------------
#                    第二步 数值转换为图像特征
#----------------------------------------
----------------------------------
#原始数据集被反射使用 np.flip 翻转它 通过 rotate 旋转从而获得更
好的图像
def              convert_values_to_image(image_values,
display=False):
    #转换成 32x32
    image_array = np.asarray(image_values)
    image_array                                    =
```

```
image_array.reshape(32,32).astype('uint8')
    #翻转+旋转
    image_array = np.flip(image_array, 0)
    image_array = rotate(image_array, -90)
    #图像显示
    new_image = Image.fromarray(image_array)
    if display == True:
        new_image.show()
    return new_image

convert_values_to_image(training_letters_images.loc[0],
True)

#-----------------------------------------
----------------------------------
#                    第三步 图像标准化处理
#-----------------------------------------
----------------------------------
training_letters_images_scaled                    =
training_letters_images.values.astype('float32')/255
training_letters_labels                           =
```

```
training_letters_labels.values.astype('int32')
testing_letters_images_scaled =
testing_letters_images.values.astype('float32')/255
testing_letters_labels =
testing_letters_labels.values.astype('int32')
print("Training images of letters after scaling")
print(training_letters_images_scaled.shape)
print(training_letters_images_scaled[0:5])

#----------------------------------------
--------------------------------
#                第四步 输出 One-hot 编码转换
#----------------------------------------
--------------------------------
import keras
from keras.utils import to_categorical
number_of_classes = 28
training_letters_labels_encoded =
to_categorical(training_letters_labels-1,

num_classes=number_of_classes)
```

```
testing_letters_labels_encoded                    =
to_categorical(testing_letters_labels-1,


num_classes=number_of_classes)
print(training_letters_labels)
print(training_letters_labels_encoded)
print(training_letters_images_scaled.shape)
# (13440, 1024)


#-----------------------------------------
----------------------------------
#                          第五步  形状修改
#-----------------------------------------
----------------------------------
#输入形状  32x32x1
training_letters_images_scaled                    =
training_letters_images_scaled.reshape([-1, 32, 32, 1])
testing_letters_images_scaled                     =
testing_letters_images_scaled.reshape([-1, 32, 32, 1])
print(training_letters_images_scaled.shape,
      training_letters_labels_encoded.shape,
```

```
        testing_letters_images_scaled.shape,
        testing_letters_labels_encoded.shape)
# (13440, 32, 32, 1) (13440, 28) (3360, 32, 32, 1) (3360, 28)


#-----------------------------------------------
---------------------------------
#                          第六步 CNN 模型设计
#-----------------------------------------------
---------------------------------
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D,
GlobalAveragePooling2D, BatchNormalization, Dropout,
Dense


#定义模型
def create_model(optimizer='adam',
kernel_initializer='he_normal', activation='relu'):
    #第一个卷积层
    model = Sequential()
    model.add(Conv2D(filters=16, kernel_size=3,
padding='same', input_shape=(32, 32, 1),
```

```
kernel_initializer=kernel_initializer, activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
    model.add(Dropout(0.2))

    #第二个卷积层
    model.add(Conv2D(filters=32, kernel_size=3,
padding='same', kernel_initializer=kernel_initializer,
activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
    model.add(Dropout(0.2))

    #第三个卷积层
    model.add(Conv2D(filters=64, kernel_size=3,
padding='same', kernel_initializer=kernel_initializer,
activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
    model.add(Dropout(0.2))
```

```
    #第四个卷积层
    model.add(Conv2D(filters=128, kernel_size=3,
padding='same', kernel_initializer=kernel_initializer,
activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2))
    model.add(Dropout(0.2))
    model.add(GlobalAveragePooling2D())

    #全连接层输出 28 类结果
    model.add(Dense(28, activation='softmax'))

    #损失函数定义
    model.compile(loss='categorical_crossentropy',
metrics=['accuracy'], optimizer=optimizer)
    return model

#创建模型
model = create_model(optimizer='Adam',
kernel_initializer='uniform', activation='relu')
model.summary()
```

```
#------------------------------------------
----------------------------------
#                          第七步 模型绘制
#------------------------------------------
----------------------------------
from keras.utils.vis_utils import plot_model
from IPython.display import Image as IPythonImage

plot_model(model,                    to_file="model.png",
show_shapes=True)
display(IPythonImage('model.png'))

#------------------------------------------
----------------------------------
#                          第八步 模型训练+输出结果
#------------------------------------------
----------------------------------
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

```
#绘制图形
def plot_loss_accuracy(history):
    # Loss
    plt.figure(figsize=[8,6])
    plt.plot(history.history['loss'],'r',linewidth=3.0)
    plt.plot(history.history['val_loss'],'b',linewidth=3.0)
    plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
    plt.xlabel('Epochs ',fontsize=16)
    plt.ylabel('Loss',fontsize=16)
    plt.title('Loss Curves',fontsize=16)

    # Accuracy
    plt.figure(figsize=[8,6])
    plt.plot(history.history['accuracy'],'r',linewidth=3.0)

plt.plot(history.history['val_accuracy'],'b',linewidth=3.0)
    plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
    plt.xlabel('Epochs ',fontsize=16)
```

```
    plt.ylabel('Accuracy',fontsize=16)
    plt.title('Accuracy Curves',fontsize=16)


#混淆矩阵
def get_predicted_classes(model, data, labels=None):
    image_predictions = model.predict(data)
    predicted_classes = np.argmax(image_predictions,
axis=1)
    true_classes = np.argmax(labels, axis=1)
    return predicted_classes, true_classes,
image_predictions


def get_classification_report(y_true, y_pred):
    print(classification_report(y_true, y_pred, digits=4)) #
小数点 4 位


checkpointer = ModelCheckpoint(filepath='weights.hdf5',
                               verbose=1,
                               save_best_only=True)
flag = "test"
if flag=="train":
```

```
    history = model.fit(training_letters_images_scaled,
                        training_letters_labels_encoded,

validation_data=(testing_letters_images_scaled,

testing_letters_labels_encoded),
                        epochs=15,
                        batch_size=20,
                        verbose=1,
                        callbacks=[checkpointer])
    print(history)
    plot_loss_accuracy(history)
else:
    #加载具有最佳验证损失的模型
    model.load_weights('weights.hdf5')
    metrics                                                    =
model.evaluate(testing_letters_images_scaled,

testing_letters_labels_encoded,
                                verbose=1)
    print("Test Accuracy: {}".format(metrics[1]))
```

```
    print("Test Loss: {}".format(metrics[0]))

    y_pred, y_true, image_predictions = get_predicted_classes(model,

testing_letters_images_scaled,

testing_letters_labels_encoded)
    get_classification_report(y_true, y_pred)

    #----------------------------------------
----------------------------------
    #                          第九步 绘制测试图像
    #----------------------------------------
----------------------------------
    fig = plt.figure(0, figsize=(14,14))
    indices = np.random.randint(0,
testing_letters_labels.shape[0], size=49)
    y_pred =
np.argmax(model.predict(training_letters_images_scaled
), axis=1)
```

```
    for i, idx in enumerate(indices):
        plt.subplot(7,7,i+1)

        image_array                                          =
training_letters_images_scaled[idx][:,:,0]
        image_array = np.flip(image_array, 0)
        image_array = rotate(image_array, -90)

        plt.imshow(image_array, cmap='gray')
        plt.title("Pred: {} - Label: {}".format(y_pred[idx],
                  (training_letters_labels[idx] -1)))
        plt.xticks([])
        plt.yticks([])
    plt.show()
    plt.savefig("resutl.png")
```

7.总结

写到这里，这篇文章就介绍结束了，希望对您有所帮助。当然还可以增加热度图以及不同算法的对比，后面尝试分享更前沿的图像分类算法。

- 一.数据集描述

- 二.数据读取
- 三.数据预处理
  - 1.数值特征转换图像
  - 2.图像标准化处理
  - 3.输出 One-hot 编码转换
  - 4.形状修改
- 四.CNN 模型搭建
  - 1.卷积神经网络概念
  - 2.模型设计
  - 3.模型绘制
- 五.实验评估
  - 1.模型训练
  - 2.绘制误差和准确率曲线
  - 3.模型评估
  - 4.绘制预测图片
- 六.完整代码

此外，作者给出热力图预测结果，其代码如下：

```
import seaborn as sns
from sklearn import metrics


Labname = [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]
print(y_pre_test)
y_pre_test = [num+1 for num in y_pre_test]
print(np.argmax(testing_letters_labels,axis=1))
confm = metrics.confusion_matrix(testing_letters_labels,
                                 y_pre_test)
plt.figure(figsize=(10,10))
sns.heatmap(confm.T, square=True, annot=True,
            fmt='d', cbar=True, linewidths=.6,
            cmap="YlGnBu")
plt.xlabel('True label',size = 12)
plt.ylabel('Predicted label', size = 12)
plt.xticks(np.arange(28)+0.5, Labname, size = 10)
plt.yticks(np.arange(28)+0.5, Labname, size = 10)
plt.savefig('headmap.png')
plt.show()
```

输出结果如下图所示。

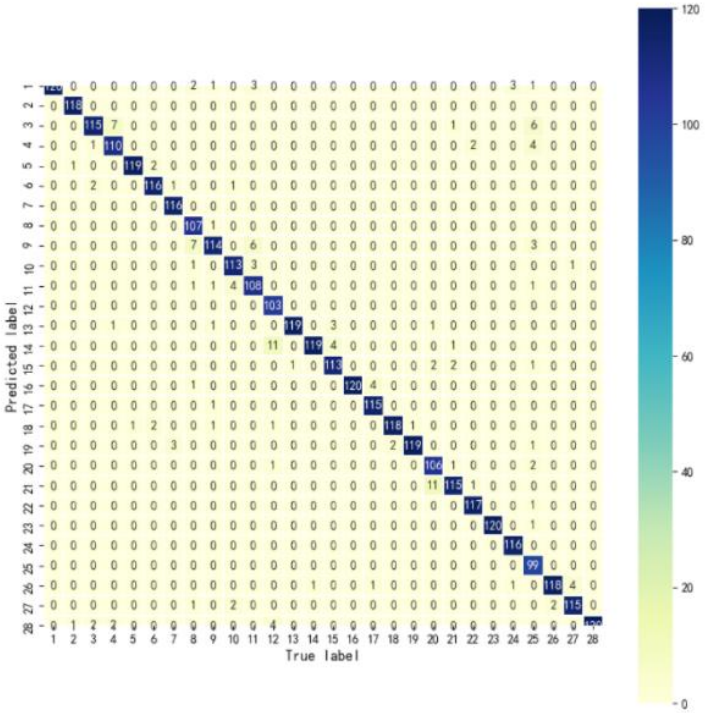


图 46-21 热力图绘制

希望大家喜欢这篇文章，并能结合本章知识点，围绕自己的研究领域或工程项目进行深入的学习，实现所需的图像识别的研究或论文。

参考文献：

[1] https://graphviz.gitlab.io/_pages/Download/windows/graphviz-2.38.msi

[2] https://github.com/eastmountyxz/AI-for-Keras

[3] https://github.com/eastmountyxz/AI-for-TensorFlow

[4] Eastmount. [Python 图像识别] 四十七.Keras 深度学习构建 CNN 识别阿拉伯手写文字图像. https://blog.csdn.net/Eastmount/article/details/120593269.

[5] 刘润森. 教你使用 TensorFlow2 对阿拉伯语手写字符数据集进行识别.

https://maoli.blog.csdn.net/article/details/117688738.

第 47 篇 Pytorch 构建 Faster-RCNN 检测小麦图像

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

前文介绍了 Keras 深度学习构建 CNN 模型识别阿拉伯手写文字图像，这是非常经典的图像分类文章。这篇文章将详细讲解 Pytorch 构建 Faster-RCNN 模型实现小麦目标检测，主要参考 kaggle 大佬和刘兄的模型。本文提供的案例将为读者提供深入的理解，希望您喜欢。

1.Pytorch 安装

Pytorch 安装需要在官网选择对应的环境，接着按自动生成的安装命令执行。

- 官网：https://pytorch.org/

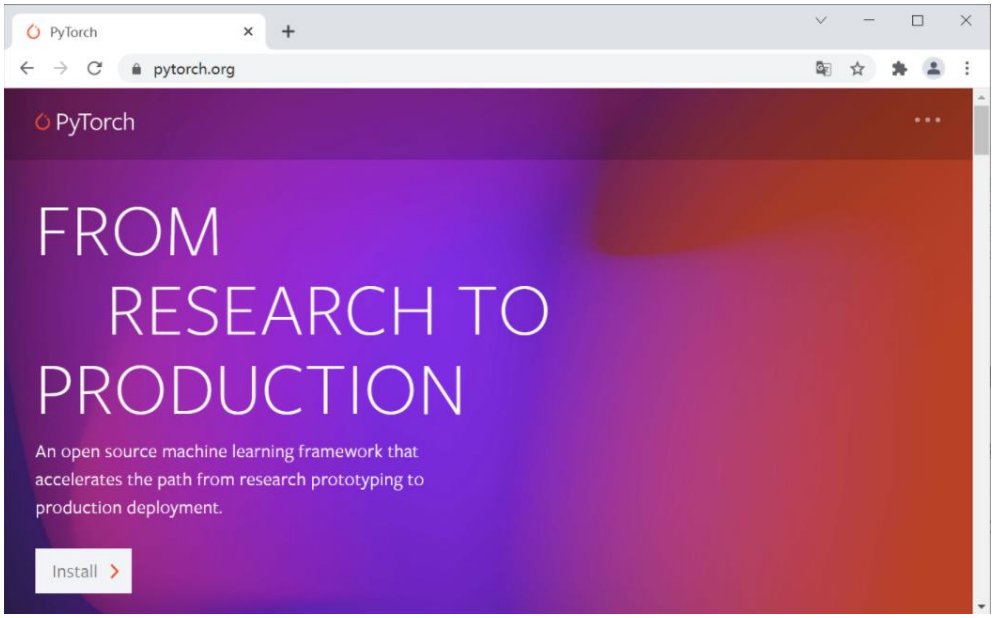


图 47-1 Pytorch 官网

选择与自己相匹配的版本，这里显示是我安装的选择。

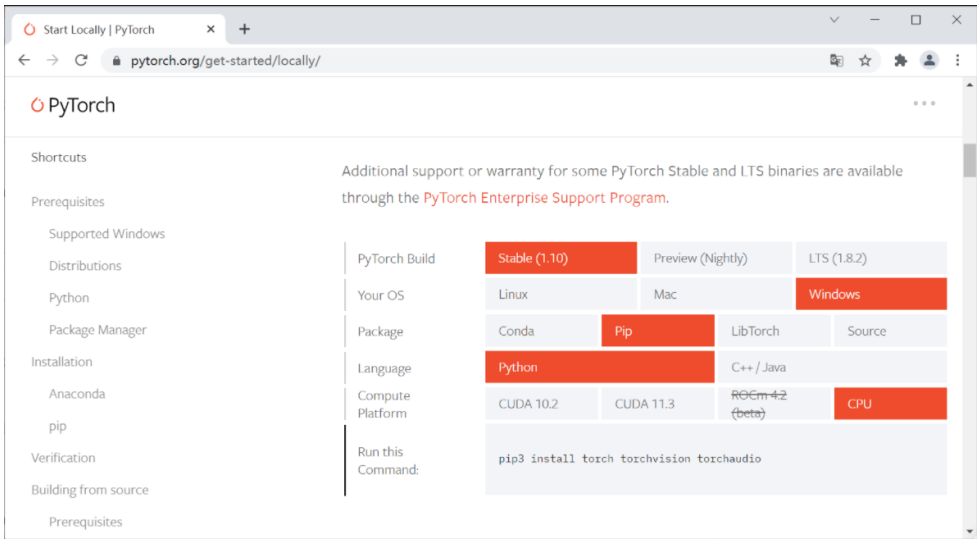


图 47-2 Pytorch 下载选项

安装代码：

- pip3 install torch torchvision torchaudio
- conda install pytorch torchvision torchaudio cpuonly -c pytorch

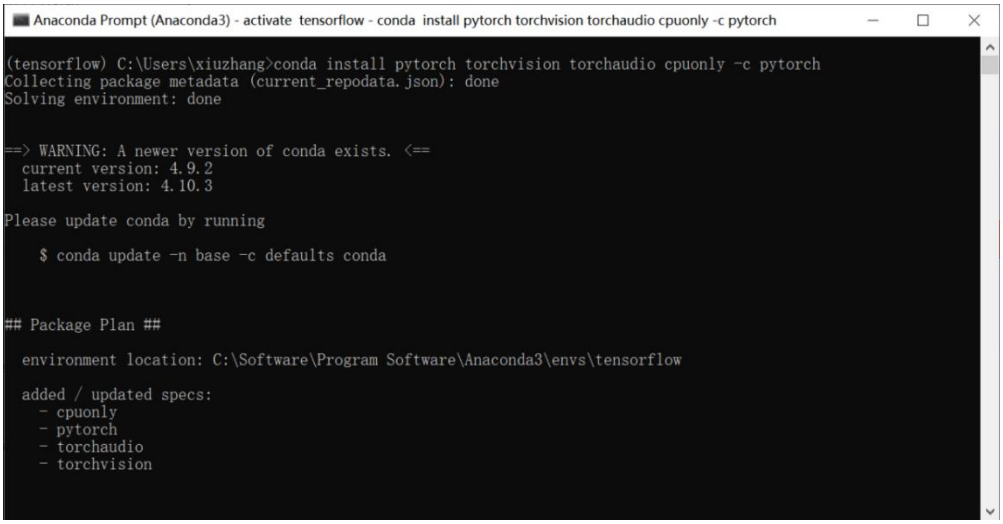


图 47-3 安装过程

同时安装扩展包 albumentations。

- pip install albumentations

2.数据集描述

（1）Kaggle 赛题

数据集是来自 Kaggle 的——全球小麦检测数据，题目是“您能使用图像分析帮助识别小麦吗？”

- https://www.kaggle.com/c/global-wheat-detection

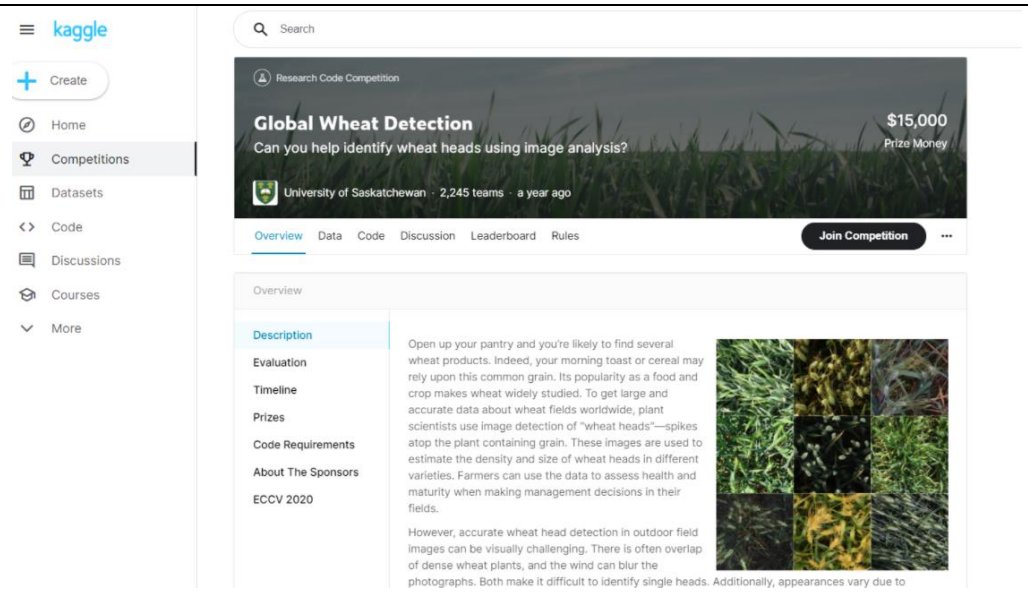


图 47-4 官网数据集介绍

题目介绍如下：

打开您的食品储藏室，您很可能会找到几种小麦产品。事实上，您的早餐吐司或麦片可能依赖于这种常见的谷物。它作为一种流行的食物和作物，让小麦得到了广泛的研究。为了获得有关全球麦田的大量准确数据，植物科学家使用“小麦头”的图像，检测含有谷物的植物顶部尖峰。这些图像用于估计不同品种小麦的密度和大小。农民在他们的田地做出管理决策时，可以使用这些数据来评估其健康和成熟度。

然而，在室外田间图像中准确检测麦头在视觉上具有挑战性。密密麻麻的小麦植株经常重叠，风会模糊照片。两者都使识别单个头部变得困难。此外，外观因成熟度、颜色、基因类型和头部方向而异。最后，由于小麦在世界范围内种植，因此必须考虑不同的品种、种植密度、模式和田间条件。小麦开发模型需要在不同的生长环境之间进行概括。当前的检测方法涉及一级和二级检测器（Yolo-v3 和 Faster-

RCNN），但即使使用大型数据集进行训练，对训练区域的偏差仍然存在。

在全球小麦头数据集是由来自七个国家的九个研究机构主导，包括东京大学等。此后，许多机构都加入了他们追求准确检测小麦头部的行列，包括全球食品安全研究所、DigitAg、Kubota 和 Hiphen。在本次比赛中，您将从小麦植物的室外图像中检测小麦头，包括来自全球的小麦数据集。使用全球数据，您将专注于通用解决方案来估计小麦头的数量和大小。为了更好地衡量未知基因型、环境和观察条件的性能，训练数据集涵盖多个区域。您将使用来自欧洲（法国、英国、瑞士）和北美（加拿大）的 3,000 多张图像。测试数据包括来自澳大利亚、日本和中国的约 1,000 张图像。

小麦是全球的主食，这就是为什么这种竞争必须考虑到不同的生长条件。为小麦表型开发的模型需要能够在环境之间进行概括。如果成功，研究人员可以准确估计不同品种小麦头的密度和大小。通过改进的检测，农民可以更好地评估他们的作物，最终将谷物、烤面包和其他喜爱的菜肴带到您的餐桌上。有关数据采集和过程的更多详细信息，请访问：

- https://arxiv.org/abs/2005.02162

（2）数据集介绍

① 我们应该期望数据格式是什么？

数据是麦田的图像，每个识别的麦头都有边界框，并非所有图像都包含小麦头/边界框。这些图像被记录在世界各地的许多地方。

- CSV 数据很简单，图像 ID 与给定图像的文件名相匹配，并且包含图像的宽度和高度以及边界框（见下文）。train.csv 每个边界框都有一行，并非所有图像都有边界框。大多数测试集图像是隐藏的，包含一小部分测试图像供您在编写代码时使用。

② 我们在预测什么?

正在尝试预测图像中每个小麦头周围的边界框。如果没有小麦头，则必须预测没有边界框。

③ 数据集包含四个文件

- train.csv – 训练数据
- sample_submission.csv – 格式正确的示例提交文件
- train.zip – 训练图像
- test.zip – 测试图像

数据集如图 47-5 所示。

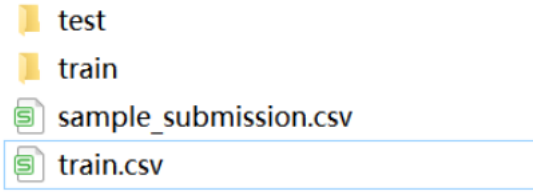


图 47-5 数据集

文件夹中包含小麦图像，名称是其 ID，如图 47-6 所示。

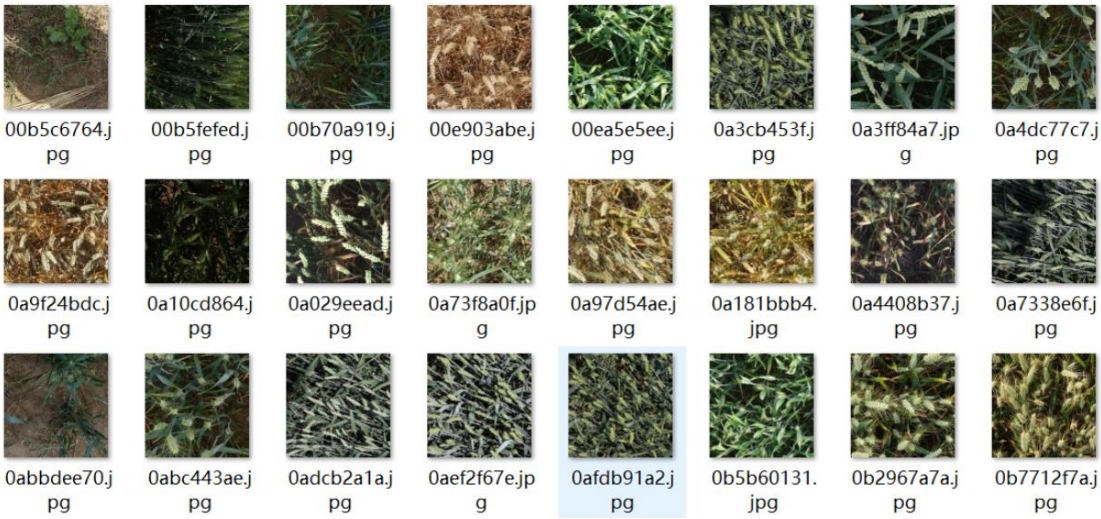


图 47-5 小麦示例图片

train.csv 中对应五列结果，分别是：

- image_id - 唯一的图像 ID
- width - 图像的宽度
- height - 图像的高度
- bbox - 一个边界框，格式为 [xmin, ymin, width, height]样式的列表
- source - 图像对应的类别

	A	B	C	D	E
1	image_id	width	height	bbox	source
2	b6ab77fd7	1024	1024	[834.0, 222.0, 56.0, 36.0]	usask_1
3	b6ab77fd7	1024	1024	[226.0, 548.0, 130.0, 58.0]	usask_1
4	b6ab77fd7	1024	1024	[377.0, 504.0, 74.0, 160.0]	usask_1
5	b6ab77fd7	1024	1024	[834.0, 95.0, 109.0, 107.0]	usask_1
6	b6ab77fd7	1024	1024	[26.0, 144.0, 124.0, 117.0]	usask_1
7	b6ab77fd7	1024	1024	[569.0, 382.0, 119.0, 111.0]	usask_1
8	b6ab77fd7	1024	1024	[52.0, 602.0, 82.0, 45.0]	usask_1
9	b6ab77fd7	1024	1024	[627.0, 302.0, 122.0, 75.0]	usask_1
10	b6ab77fd7	1024	1024	[412.0, 367.0, 68.0, 82.0]	usask_1
11	b6ab77fd7	1024	1024	[953.0, 220.0, 56.0, 103.0]	usask_1
12	b6ab77fd7	1024	1024	[30.0, 70.0, 126.0, 133.0]	usask_1
13	b6ab77fd7	1024	1024	[35.0, 541.0, 46.0, 46.0]	usask_1
14	b6ab77fd7	1024	1024	[103.0, 60.0, 117.0, 83.0]	usask_1
15	b6ab77fd7	1024	1024	[417.0, 4.0, 110.0, 91.0]	usask_1
16	b6ab77fd7	1024	1024	[764.0, 299.0, 119.0, 93.0]	usask_1
17	b6ab77fd7	1024	1024	[539.0, 58.0, 58.0, 130.0]	usask_1
18	b6ab77fd7	1024	1024	[139.0, 274.0, 121.0, 76.0]	usask_1
19	b6ab77fd7	1024	1024	[461.0, 634.0, 118.0, 64.0]	usask_1
20	b6ab77fd7	1024	1024	[215.0, 634.0, 113.0, 75.0]	usask_1

图 47-6 数据集示例

训练集中各小麦类型分布如下图所示：

图 47-7 小麦类型数据集分布情况

整个小麦预测的大致流程如下图所示：

Supplementary Materials

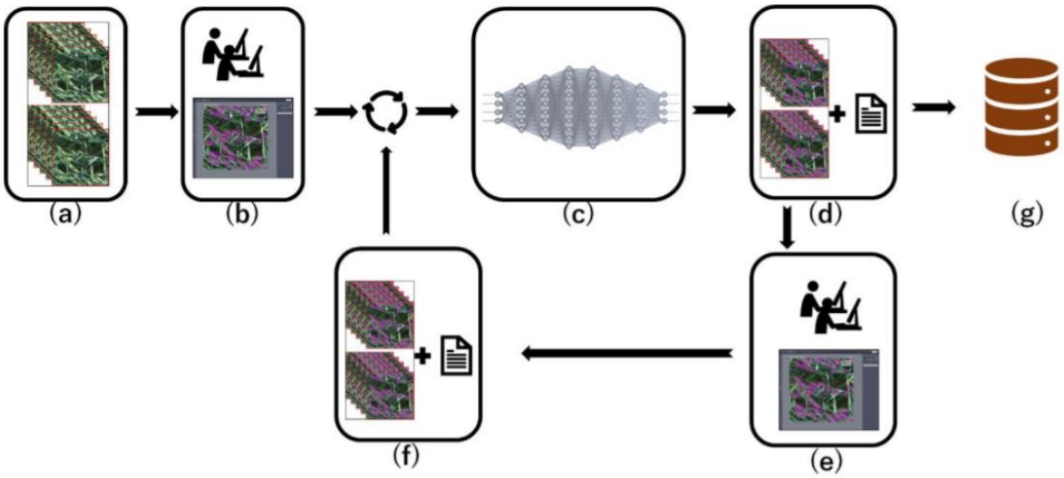


图 47-8 小麦预测流程

模型评估参数如下所示，推荐大家阅读 kaggle 官网介绍。

这种竞争是根据不同交集的平均精度在联合 (IoU) 阈值上进行评估的。一组预测边界框和真实边界框的 IoU 计算如下：

$$IoU(A,B) = \frac{A \cap B}{A \cup B}.$$

该指标扫描一系列 IoU 阈值，在每个点计算平均精度值。阈值范围从 0.5 到 0.75，步长为 0.05。换句话说，在阈值为 0.5 时，如果预测对象与真实对象联合的交集大于 0.5，则该对象被视为"命中"。

在每个阈值 t，根据将预测对象与所有真实对象进行比较而产生的真阳性 (TP)、假阴性 (FN) 和假阳性 (FP) 的数量计算精度值：

$$\frac{TP(t)}{TP(t) + FP(t) + FN(t)}.$$

当单个预测对象与 IoU 高于阈值的地面实况对象匹配时，就计算为真阳性。误报表示预测对象没有关联的地面实况对象。假阴性表示真实对象没有关联的预测对象。

重要提示：如果给定图像根本没有地面实况对象，任何数量的预测（误报）都会导致图像得分为零，并包含在平均平均精度中。

单个图像的平均精度计算为每个 IoU 阈值下上述精度值的平均值：

$$\frac{1}{|thresholds|} \sum_t \frac{TP(t)}{TP(t) + FP(t) + FN(t)}.$$

图 47-9 评价流程

提交格式需要以空格分隔的一组边界框。例如：

- ce4833752, 0.5 0 0 100 100：表示图像 ce4833752 有一个边界框，aconfidence 为 0.5，在 x 为 0 且 y 为 0，awidth 和 height 为 100。

该文件应包含标题并具有以下格式，您提交的每一行都应包含给定图像的所有边界框。

```
image_id,PredictionString
ce4833752,1.0 0 0 50 50
adcfa13da,1.0 0 0 50 50
6ca7b2650,
1da9078c1,0.3 0 0 50 50 0.5 10 10 30 30
7640b4963,0.5 0 0 50 50
```

3.代码实现

下面我们参考 Kaggle Peter 老师的代码，来复现 Faster-RCNN 模型。

- https://www.kaggle.com/pestipeti/pytorch-starter-fasterrcnn-train

模型的框架如下图所示，相信大家都比较熟悉，也推荐大家使用并深入了解背后的原理。

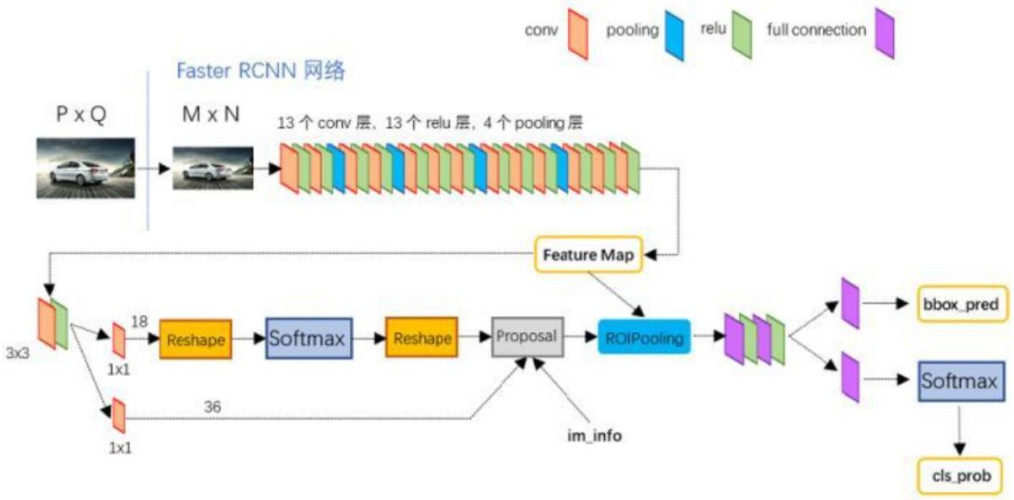


图 47-10 Faster-RCNN 网络框架

（1）读取小麦数据

读取小麦数据集的代码如下：

```
# -*- coding: utf-8 -*-
"""

Created on Fri Oct 29 13:42:38 2021

@author: xiuzhang
"""

import os

import re

import cv2

import pandas as pd

import numpy as np

from PIL import Image
```

```
import albumentations as A
from matplotlib import pyplot as plt
from albumentations.pytorch.transforms import ToTensorV2

import torch
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator
from torch.utils.data import DataLoader, Dataset
from torch.utils.data.sampler import SequentialSampler

from dataset import WheatDataset

#-----------------------------------------------------------------
#第一步 函数定义
#-----------------------------------------------------------------
```

```
--------------------------------

#提取 box 的四个坐标

def expand_bbox(x):

    r = np.array(re.findall("([0-9]+[.]?[0-9]*)", x))

    if len(r) == 0:

        r = [-1, -1, -1, -1]

    return r


#训练图像增强 Albumentations

def get_train_transform():

    return A.Compose([

        A.Flip(0.5),

        ToTensorV2(p=1.0)

    ], bbox_params={'format': 'pascal_voc', 'label_fields':

['labels']})


#验证图像增强

def get_valid_transform():

    return A.Compose([

        ToTensorV2(p=1.0)

    ], bbox_params={'format': 'pascal_voc', 'label_fields':
```

```
['labels']})


def collate_fn(batch):
    return tuple(zip(*batch))


#-----------------------------------------------------
---------------------------------
#第二步 定义变量并读取数据
#-----------------------------------------------------
---------------------------------
DIR_INPUT = 'data'
DIR_TRAIN = f'{DIR_INPUT}/train'
DIR_TEST = f'{DIR_INPUT}/test'
train_df = pd.read_csv(f'{DIR_INPUT}/train.csv')
print(train_df.shape)


train_df['x'] = -1
train_df['y'] = -1
train_df['w'] = -1
train_df['h'] = -1
```

```
#读取 box 四个坐标
train_df[['x', 'y', 'w', 'h']] = np.stack(train_df['bbox'].apply(lambda x: expand_bbox(x)))
train_df.drop(columns=['bbox'], inplace=True)
train_df['x'] = train_df['x'].astype(np.float)
train_df['y'] = train_df['y'].astype(np.float)
train_df['w'] = train_df['w'].astype(np.float)
train_df['h'] = train_df['h'].astype(np.float)


#获取图像 id
image_ids = train_df['image_id'].unique()
valid_ids = image_ids[-665:]
train_ids = image_ids[:-665]
valid_df = train_df[train_df['image_id'].isin(valid_ids)]
train_df = train_df[train_df['image_id'].isin(train_ids)]
print(valid_df.shape, train_df.shape)
print(train_df.head())
```

显示结果如下图所示，分别获取图像 id 和数据，并划分为 train（训练）和 valid（验证）。

```
Reloaded modules: dataset
(147793, 5)
(25006, 8) (122787, 8)
    image_id  width  height   source      x      y      w      h
0  b6ab77fd7   1024    1024  usask_1  834.0  222.0   56.0   36.0
1  b6ab77fd7   1024    1024  usask_1  226.0  548.0  130.0   58.0
2  b6ab77fd7   1024    1024  usask_1  377.0  504.0   74.0  160.0
3  b6ab77fd7   1024    1024  usask_1  834.0   95.0  109.0  107.0
4  b6ab77fd7   1024    1024  usask_1   26.0  144.0  124.0  117.0
```

图 47-11 实验结果

其中，dataset.py 文件代码如下：

- 获取图像 id
- 获取图像像素值并归一化处理
- 获取图像对应的边界（x | y | w | h）

```
# -*- coding: utf-8 -*-

"""

Created on Fri Oct 29 13:42:38 2021

@author: xiuzhang

"""

import numpy as np

import cv2

import torch

from torch.utils.data import Dataset




class WheatDataset(Dataset):
```

```
    def __init__(self, dataframe, image_dir,
transforms=None):
        super().__init__()

        self.image_ids = dataframe['image_id'].unique()
        self.df = dataframe
        self.image_dir = image_dir
        self.transforms = transforms

    def __getitem__(self, index: int):
        image_id = self.image_ids[index]
        records = self.df[self.df['image_id'] == image_id]

        image =
cv2.imread(f'{self.image_dir}/{image_id}.jpg',
cv2.IMREAD_COLOR)
        image = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB).astype(np.float32)
        image /= 255.0
```

```
        boxes = records[['x', 'y', 'w', 'h']].values
        boxes[:, 2] = boxes[:, 0] + boxes[:, 2]
        boxes[:, 3] = boxes[:, 1] + boxes[:, 3]

        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] -
boxes[:, 0])
        area = torch.as_tensor(area, dtype=torch.float32)

        # there is only one class
        labels = torch.ones((records.shape[0],),
dtype=torch.int64)

        # suppose all instances are not crowd
        iscrowd = torch.zeros((records.shape[0],),
dtype=torch.int64)

        target = {}
        target['boxes'] = boxes
        target['labels'] = labels
        # target['masks'] = None
        target['image_id'] = torch.tensor([index])
```

```
            target['area'] = area
            target['iscrowd'] = iscrowd

            if self.transforms:
                sample = {
                    'image': image,
                    'bboxes': target['boxes'],
                    'labels': labels
                }
                sample = self.transforms(**sample)
                image = sample['image']

                target['boxes']                                    =
torch.stack(tuple(map(torch.tensor,
zip(*sample['bboxes'])))).permute(1, 0)

            return image, target, image_id

    def __len__(self) -> int:
            return self.image_ids.shape[0]
```

（2）可视化展示

接下来我们对小麦图像进行简单的可视化操作，代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 29 13:42:38 2021
@author: xiuzhang
"""
import os
import re
import cv2
import pandas as pd
import numpy as np
from PIL import Image
import albumentations as A
from matplotlib import pyplot as plt
from albumentations.pytorch.transforms import ToTensorV2

import torch
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
```

```
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator
from torch.utils.data import DataLoader, Dataset
from torch.utils.data.sampler import SequentialSampler

from dataset import WheatDataset

#-----------------------------------------------------------------
#第一步 函数定义
#-----------------------------------------------------------------
#提取 box 的四个坐标
def expand_bbox(x):
    r = np.array(re.findall("([0-9]+[.]?[0-9]*)", x))
    if len(r) == 0:
        r = [-1, -1, -1, -1]
    return r

#训练图像增强 Albumentations
```

```
def get_train_transform():
    return A.Compose([
        A.Flip(0.5),
        ToTensorV2(p=1.0)
    ], bbox_params={'format': 'pascal_voc', 'label_fields':
['labels']})


#验证图像增强
def get_valid_transform():
    return A.Compose([
        ToTensorV2(p=1.0)
    ], bbox_params={'format': 'pascal_voc', 'label_fields':
['labels']})


def collate_fn(batch):
    return tuple(zip(*batch))


#-----------------------------------------
--------------------------------
#第二步 定义变量并读取数据
#-----------------------------------------
```

```
--------------------------------

DIR_INPUT = 'data'

DIR_TRAIN = f'{DIR_INPUT}/train'

DIR_TEST = f'{DIR_INPUT}/test'

train_df = pd.read_csv(f'{DIR_INPUT}/train.csv')

print(train_df.shape)


train_df['x'] = -1

train_df['y'] = -1

train_df['w'] = -1

train_df['h'] = -1


#读取 box 四个坐标

train_df[['x', 'y', 'w', 'h']] =

np.stack(train_df['bbox'].apply(lambda x:

expand_bbox(x)))

train_df.drop(columns=['bbox'], inplace=True)

train_df['x'] = train_df['x'].astype(np.float)

train_df['y'] = train_df['y'].astype(np.float)

train_df['w'] = train_df['w'].astype(np.float)

train_df['h'] = train_df['h'].astype(np.float)
```

```
#获取图像 id
image_ids = train_df['image_id'].unique()
valid_ids = image_ids[-665:]
train_ids = image_ids[:-665]
valid_df = train_df[train_df['image_id'].isin(valid_ids)]
train_df = train_df[train_df['image_id'].isin(train_ids)]
print(valid_df.shape, train_df.shape)
print(train_df.head())

#-----------------------------------------------
--------------------------------
#第三步 加载数据
#-----------------------------------------------
--------------------------------
train_dataset = WheatDataset(train_df, DIR_TRAIN,
get_train_transform())
valid_dataset = WheatDataset(valid_df, DIR_TRAIN,
get_valid_transform())

train_data_loader = DataLoader(
```

```
    train_dataset,
    batch_size=2,
    shuffle=False,
    num_workers=0,
    collate_fn=collate_fn
)

valid_data_loader = DataLoader(
    valid_dataset,
    batch_size=2,
    shuffle=False,
    num_workers=0,
    collate_fn=collate_fn
)

#-----------------------------------------
--------------------------------
#第四步 数据可视化
#-----------------------------------------
--------------------------------
#提取训练数据和类别
```

```
device = torch.device('cuda') if torch.cuda.is_available()
else torch.device('cpu')
images, targets, image_ids = next(iter(train_data_loader))
images = list(image.to(device) for image in images)
targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
boxes = targets[0]['boxes'].cpu().numpy().astype(np.int32)
sample = images[0].permute(1, 2, 0).cpu().numpy()

fig, ax = plt.subplots(1, 1, figsize=(10, 8))

#绘制小麦目标识别 box
for box in boxes:
    cv2.rectangle(sample,
                  (box[0], box[1]),
                  (box[2], box[3]),
                  (255, 0, 0), 3)

    ax.text(box[0],
            box[1] - 2,
            '{:s}'.format('wheat'),
            bbox=dict(facecolor='blue', alpha=0.5),
```

```
                fontsize=12,
                color='white')


ax.set_axis_off()
ax.imshow(sample)
plt.show()
```

输出结果如图 47-12 所示，按照 train.csv 定义好的边界我们绘制了 wheat 红色框，将小麦标记。最终的测试集，我们希望能自动预测小麦的边界，从而有效识别小麦区域和数量。

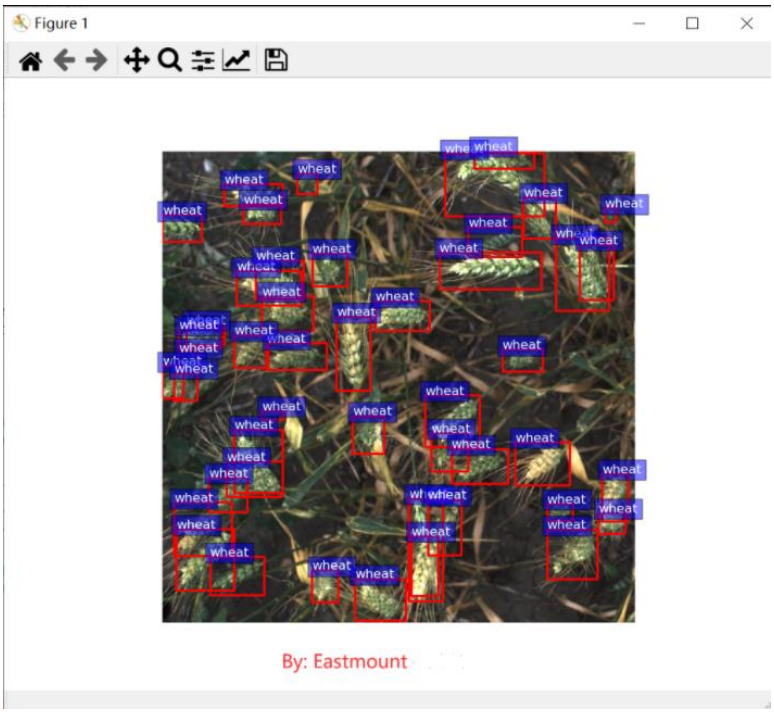


图 47-12 小麦识别结果

可能存在的警告如下：

- Clipping input data to the valid range for imshow with RGB

data ([0…1] for floats or [0…255] for integers).

（3）构建 Faster-RCNN 模型

接下来构造 Faster-RCNN 模型，这是目标检测的经典模型。模型如下：

model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

模型构建的核心代码如下：

```
#-----------------------------------------
--------------------------------
#第五步  模型构建
#-----------------------------------------
--------------------------------
num_classes = 2   #1 class (wheat) + background
lr_scheduler = None
num_epochs = 1
itr = 1


class Averager:
    def __init__(self):
        self.current_total = 0.0
        self.iterations = 0.0
```

```
    def send(self, value):
        self.current_total += value
        self.iterations += 1

    @property
    def value(self):
        if self.iterations == 0:
            return 0
        else:
            return 1.0 * self.current_total / self.iterations

    def reset(self):
        self.current_total = 0.0
        self.iterations = 0.0

#load a model pre-trained on COCO
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(pr
etrained=True)

#获取分类器输入特征数量
```

```
in_features =
model.roi_heads.box_predictor.cls_score.in_features

#replace the pre-trained head with a new one
model.roi_heads.box_predictor =
FastRCNNPredictor(in_features, num_classes)

#参数设置
model.to(device)
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
momentum=0.9, weight_decay=0.0005)
#lr_scheduler =
torch.optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.1)

loss_hist = Averager()
print("Start training....")

# 迭代训练
for epoch in range(num_epochs):
```

```
    loss_hist.reset()

    for images, targets, image_ids in train_data_loader:

        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]
        for t in targets:
            t['boxes'] = t['boxes'].float()

        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())
        loss_value = losses.item()
        loss_hist.send(loss_value)
        print("loss is :",loss_value)

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()
        if itr % 50 == 0:
            print(f"Iteration    #{itr}/{len(train_data_loader)}
```

```
loss: {loss_value}")
            itr += 1


    #更新学习率
    if lr_scheduler is not None:
        lr_scheduler.step()
    print(f"Epoch #{epoch} loss: {loss_hist.value}")


torch.save(model.state_dict(),
'fasterrcnn_resnet50_fpn.pth')
print("Next Test....")
```

运行过程如下图所示：

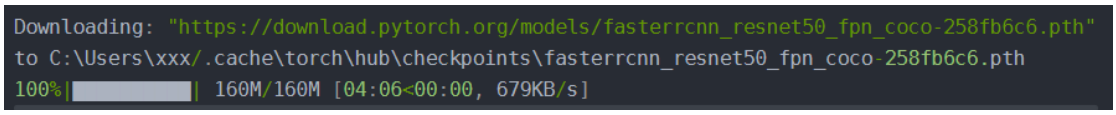


图 47-13 运行结果

（4）模型预测

增加模型预测的最终完整代码如下：

- 第一步：函数定义
- 第二步：定义变量并读取数据
- 第三步：加载数据
- 第四步：数据可视化

- 第五步：Faster-RCNN 模型构建
- 第六步：模型测试

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 29 13:42:38 2021
@author: xiuzhang
"""
import os
import re
import cv2
import pandas as pd
import numpy as np
from PIL import Image
import albumentations as A
from matplotlib import pyplot as plt
from albumentations.pytorch.transforms import ToTensorV2

import torch
import torchvision
from torchvision.models.detection.faster_rcnn import
```

```
FastRCNNPredictor

from torchvision.models.detection import FasterRCNN

from torchvision.models.detection.rpn import AnchorGenerator

from torch.utils.data import DataLoader, Dataset

from torch.utils.data.sampler import SequentialSampler


from dataset import WheatDataset


#-----------------------------------------------------------------

#第一步 函数定义

#-----------------------------------------------------------------

#提取 box 的四个坐标

def expand_bbox(x):

    r = np.array(re.findall("([0-9]+[.]?[0-9]*)", x))

    if len(r) == 0:

        r = [-1, -1, -1, -1]

    return r
```

```
#训练图像增强 Albumentations
def get_train_transform():
    return A.Compose([
        A.Flip(0.5),
        ToTensorV2(p=1.0)
    ], bbox_params={'format': 'pascal_voc', 'label_fields':
['labels']})

#验证图像增强
def get_valid_transform():
    return A.Compose([
        ToTensorV2(p=1.0)
    ], bbox_params={'format': 'pascal_voc', 'label_fields':
['labels']})

def collate_fn(batch):
    return tuple(zip(*batch))

#-------------------------------------------
-------------------------------
#第二步 定义变量并读取数据
```

```
#----------------------------------------------------------------------
DIR_INPUT = 'data'
DIR_TRAIN = f'{DIR_INPUT}/train'
DIR_TEST = f'{DIR_INPUT}/test'
train_df = pd.read_csv(f'{DIR_INPUT}/train.csv')
print(train_df.shape)


train_df['x'] = -1
train_df['y'] = -1
train_df['w'] = -1
train_df['h'] = -1


#读取 box 四个坐标
train_df[['x', 'y', 'w', 'h']] =
np.stack(train_df['bbox'].apply(lambda x:
expand_bbox(x)))
train_df.drop(columns=['bbox'], inplace=True)
train_df['x'] = train_df['x'].astype(np.float)
train_df['y'] = train_df['y'].astype(np.float)
train_df['w'] = train_df['w'].astype(np.float)
```

```
train_df['h'] = train_df['h'].astype(np.float)


#获取图像 id
image_ids = train_df['image_id'].unique()
valid_ids = image_ids[-665:]
train_ids = image_ids[:-665]
valid_df = train_df[train_df['image_id'].isin(valid_ids)]
train_df = train_df[train_df['image_id'].isin(train_ids)]
print(valid_df.shape, train_df.shape)
print(train_df.head())


#-----------------------------------------
---------------------------------
#第三步 加载数据
#-----------------------------------------
---------------------------------
train_dataset = WheatDataset(train_df, DIR_TRAIN,
get_train_transform())
valid_dataset = WheatDataset(valid_df, DIR_TRAIN,
get_valid_transform())
```

```
train_data_loader = DataLoader(
    train_dataset,
    batch_size=2,
    shuffle=False,
    num_workers=0,
    collate_fn=collate_fn
)

valid_data_loader = DataLoader(
    valid_dataset,
    batch_size=2,
    shuffle=False,
    num_workers=0,
    collate_fn=collate_fn
)

#----------------------------------------
--------------------------------
#第四步 数据可视化
#----------------------------------------
--------------------------------
```

```
#提取训练数据和类别
device = torch.device('cuda') if torch.cuda.is_available()
else torch.device('cpu')
images, targets, image_ids = next(iter(train_data_loader))
images = list(image.to(device) for image in images)
targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
boxes = targets[0]['boxes'].cpu().numpy().astype(np.int32)
sample = images[0].permute(1, 2, 0).cpu().numpy()

fig, ax = plt.subplots(1, 1, figsize=(10, 8))

#绘制小麦目标识别 box
for box in boxes:
    cv2.rectangle(sample,
                  (box[0], box[1]),
                  (box[2], box[3]),
                  (255, 0, 0), 3)

    ax.text(box[0],
            box[1] - 2,
            '{:s}'.format('wheat'),
```

```
                bbox=dict(facecolor='blue', alpha=0.5),
                fontsize=12,
                color='white')

ax.set_axis_off()
ax.imshow(sample)
plt.show()

#----------------------------------------
--------------------------------
#第五步 模型构建
#----------------------------------------
--------------------------------
num_classes = 2   #1 class (wheat) + background
lr_scheduler = None
num_epochs = 1
itr = 1

class Averager:
    def __init__(self):
        self.current_total = 0.0
```

```
        self.iterations = 0.0

    def send(self, value):
        self.current_total += value
        self.iterations += 1

    @property
    def value(self):
        if self.iterations == 0:
            return 0
        else:
            return 1.0 * self.current_total / self.iterations

    def reset(self):
        self.current_total = 0.0
        self.iterations = 0.0

#load a model pre-trained on COCO
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(pr
etrained=True)
```

```
#获取分类器输入特征数量
in_features =
model.roi_heads.box_predictor.cls_score.in_features

#replace the pre-trained head with a new one
model.roi_heads.box_predictor =
FastRCNNPredictor(in_features, num_classes)

#参数设置
model.to(device)
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
momentum=0.9, weight_decay=0.0005)
#lr_scheduler =
torch.optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.1)

loss_hist = Averager()
print("Start training....")
```

```
# 迭代训练
for epoch in range(num_epochs):
    loss_hist.reset()

    for images, targets, image_ids in train_data_loader:

        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]
        for t in targets:
            t['boxes'] = t['boxes'].float()

        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())
        loss_value = losses.item()
        loss_hist.send(loss_value)
        #print("loss is :",loss_value)

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()
```

```
            if itr % 50 == 0:
                print(f"Iteration    #{itr}/{len(train_data_loader)}
loss: {loss_value}")
            itr += 1


        #更新学习率
        if lr_scheduler is not None:
            lr_scheduler.step()
        print(f"Epoch #{epoch} loss: {loss_hist.value}")


torch.save(model.state_dict(),
'fasterrcnn_resnet50_fpn.pth')
print("Next Test....")


#-----------------------------------------
--------------------------------
#第六步  模型测试
#-----------------------------------------
--------------------------------
images, targets, image_ids = next(iter(valid_data_loader))
images = list(img.to(device) for img in images)
```

```
targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
boxes = targets[0]['boxes'].cpu().numpy().astype(np.int32)
sample = images[0].permute(1, 2, 0).cpu().numpy()

model.eval()
cpu_device = torch.device("cpu")

outputs = model(images)
outputs = [{k: v.to(cpu_device) for k, v in t.items()} for t in
outputs]
fig, ax = plt.subplots(1, 1, figsize=(16, 8))
for box in boxes:
    cv2.rectangle(sample,
                  (box[0], box[1]),
                  (box[2], box[3]),
                  (220, 0, 0), 3)
ax.set_axis_off()
ax.imshow(sample)
plt.show()
```

模型运行结果如下图所示，可以看到迭代的 loss，推荐大家用好的目标检

测环境实验。

```
Start training....
Iteration #50/1354 loss: 0.9862774014472961
Iteration #100/1354 loss: 0.7824723124504089
Iteration #150/1354 loss: 1.0995712280273438
Iteration #200/1354 loss: 0.947432816028595
Iteration #250/1354 loss: 0.8192988634109497
Iteration #300/1354 loss: 1.0232043266296387
Iteration #350/1354 loss: 1.128899335861206
Iteration #400/1354 loss: 0.9684562683105469
Iteration #450/1354 loss: 1.015764832496643
Iteration #500/1354 loss: 0.8706054091453552
Iteration #550/1354 loss: 0.9101477265357971
Iteration #600/1354 loss: 0.8566261529922485
Iteration #650/1354 loss: 0.38778477907180786
Iteration #700/1354 loss: 0.6073868870735168
Iteration #750/1354 loss: 0.9497542977333069
Iteration #800/1354 loss: 0.5617823004722595
Iteration #850/1354 loss: 0.8001148700714111
Iteration #900/1354 loss: 0.7002604007720947
Iteration #950/1354 loss: 0.7241110801696777
Iteration #1000/1354 loss: 0.7318949699401855
Iteration #1050/1354 loss: 0.6532547473907471
Iteration #1100/1354 loss: 0.8252599239349365
Iteration #1150/1354 loss: 0.8113700151443481
Iteration #1200/1354 loss: 0.7127389907836914
Iteration #1250/1354 loss: 1.2049918174743652
Iteration #1300/1354 loss: 0.7468252182006836
Iteration #1350/1354 loss: 0.8966559767723083
Epoch #0 loss: 0.8672925769578407
Next Test....
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

图 47-14 误差降低结果

同时对测试集或验证集的图像进行识别，如图 47-15 所示。

图 47-15 小麦检测结果

4.总结

写到这里，这篇文章就介绍结束了，希望对您有所帮助。详细的对比实验和算法评估还请读者自行完成，后续作者的文章也会深入介绍。

- 一.Pytorch 安装
- 二.数据集描述
  - 1.Kaggle 赛题
  - 2.数据集介绍
- 三.代码实现
  - 1.读取小麦数据
  - 2.可视化展示
  - 3.构建 Faster-RCNN 模型
  - 4.模型预测

希望大家喜欢这篇文章，并能结合本章知识点，围绕自己的研究领域或工程项目进行深入的学习，实现所需的图像识别的研究或论文。

参考文献：

[1] https://github.com/eastmountyxz/ImageProcessing-Python

[2] https://www.kaggle.com/c/global-wheat-detection

[3] https://arxiv.org/abs/2005.02162

[4] https://www.kaggle.com/pestipeti/pytorch-starter-fasterrcnn-train

[5] https://blog.csdn.net/Eastmount/article/details/121018973

[6] https://blog.csdn.net/qq_39071739/article/details/108193935

[7] https://maoli.blog.csdn.net/article/details/118575041

第 48 篇 GAN 入门知识详解及手写数字图像生成

该系列文章主要讲解 Python OpenCV 图像处理和图像识别知识，前期主要讲解图像处理基础知识、OpenCV 基础用法、常用图像绘制方法、图像几何变换等，中期讲解图像处理的各种运算，包括图像点运算、形态学处理、图像锐化、图像增强、图像平滑等，后期研究图像识别、图像分割、图像分类、图像特效处理以及图像处理相关应用。希望文章对您有所帮助，如果有不足之处，还请海涵。

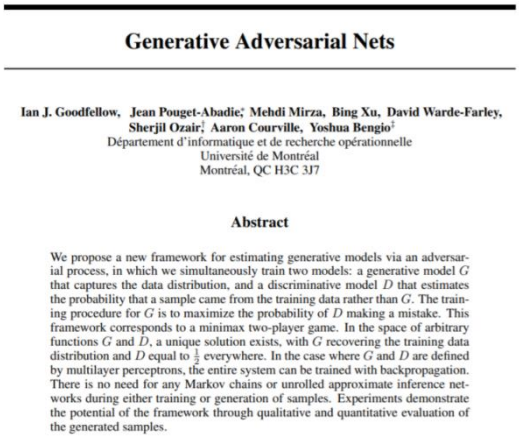
前文介绍了 Pytorch 构建 Faster-RCNN 模型实现小麦目标检测人脸检测的应用案例。这篇文章将详细讲解生成对抗网络 GAN 的基础知识，包括什么是 GAN、常用算法（CGAN、DCGAN、infoGAN、WGAN）、发展历程、预备知识，并通过 Keras 搭建最简答的手写数字图片生成案例。本文主要学习小象学院老师的视频，并结合论文介绍，希望对您有所帮助！不服 GAN，让我们开始吧，这也是整本书的最后一篇文章，且看且珍惜，更希望大家下来不断实践。

1.GAN 简介

（1）GAN 背景知识

Ian Goodfellow 因提出了生成对抗网络（GANs，Generative Adversarial Networks）而闻名， GAN 最早由 Ian Goodfellow 于 2014 年提出，以其优越的性能，在不到两年时间里，迅速成为一大研究热点。他也被誉为“GANs 之父”，甚至被推举为人工智能领域的顶级专家。

- GAN 原文地址如下：https://arxiv.org/abs/1406.2661



Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie†, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair‡, Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

图 48-1 GAN 论文

实验运行结果如下图所示，生成了对应的图像。

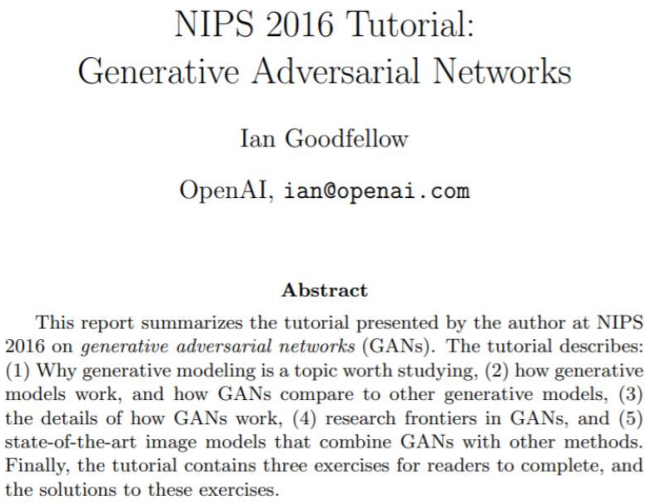
Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

图 48-2 GAN 生成图像

或许，你对这个名字还有些陌生，但如果你对深度学习有过了解，你就会知道他。最畅销的这本《深度学习》作者正是 Ian Goodfellow 大佬。在 2016

年，Ian Goodfellow 大佬又通过 50 多页的论文详细介绍了 GAN，这篇文章也推荐大家去学习。

- 文章地址：https://arxiv.org/pdf/1701.00160.pdf



NIPS 2016 Tutorial:
Generative Adversarial Networks

Ian Goodfellow

OpenAI, ian@openai.com

Abstract

This report summarizes the tutorial presented by the author at NIPS 2016 on *generative adversarial networks* (GANs). The tutorial describes: (1) Why generative modeling is a topic worth studying, (2) how generative models work, and how GANs compare to other generative models, (3) the details of how GANs work, (4) research frontiers in GANs, and (5) state-of-the-art image models that combine GANs with other methods. Finally, the tutorial contains three exercises for readers to complete, and the solutions to these exercises.

图 48-3 GAN 论文详解

Yann LeCun 称 GAN 为“过去十年机器学习界最有趣的 idea”。GAN 在 github 上的火热程度如下图所示，呈指数增涨，出现各种变形。当然，其中也存在很多比较水的文章，推荐大家尽量学习比较经典的模型。

- https://github.com/hindupuravinash/the-gan-zoo

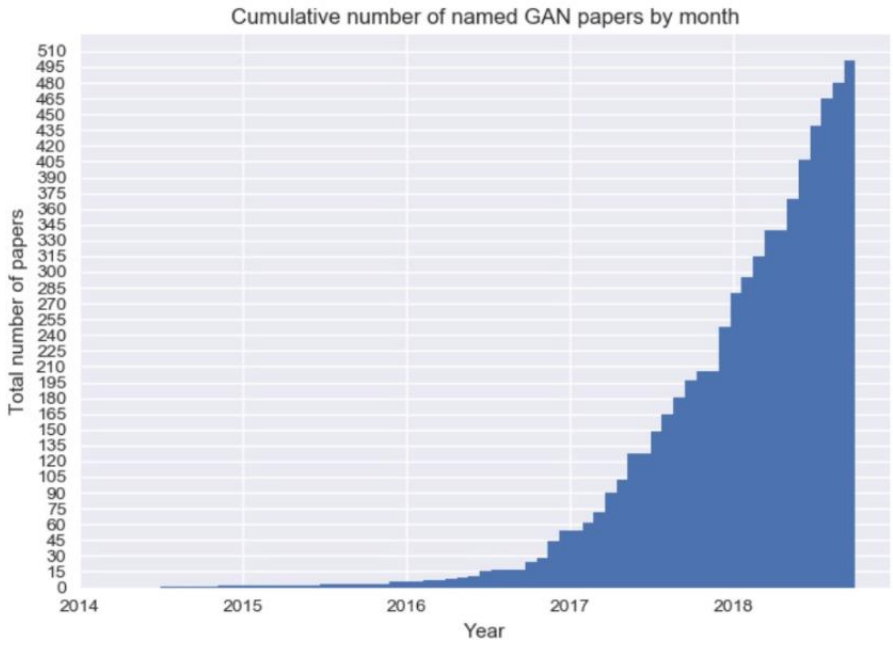


图 48-4 GAN 研究趋势

（2）GAN 原理解析

① 首先，什么是 GAN?

GANs（Generativeadversarial networks，对抗式生成网络）可以把这三个单词拆分理解。

- Generative：生成式模型
- Adversarial：采取对抗的策略
- Networks：网络（不一定是深度学习）

正如 shunliz 老师总结：

GANs 是一类生成模型，从字面意思不难猜到它会涉及两个“对手”，即：

- 一个称为 Generator（生成者）
- 一个称为 Discriminator（判别者）

Goodfellow 最初 arxiv 上挂出的 GAN tutorial 文章中将它们分别比喻

为伪造者（Generator）和警察（Discriminator）。伪造者总想着制造出能够以假乱真的钞票，而警察则试图用更先进的技术甄别真假。两者在博弈过程中不断升级自己的技术。

从博弈论的角度来看，如果是零和博弈（zero-sum game），两者最终会达到纳什均衡（Nash equilibrium），即存在一组策略(g, d)，如果Generator 不选择策略 g，那么对于 Discriminator 来说，总存在一种策略使得 Generator 输得更惨；同样地，将 Generator 换成 Discriminator 也成立。

如果 GANs 定义的 lossfunction 满足零和博弈，并且有足够多的样本，双方都有充足的学习能力情况，在这种情况下，Generator 和 Discriminator 的最优策略即为纳什均衡点，也即：Generator 产生的都是“真钞”（材料、工艺技术与真钞一样，只是没有得到授权），Discriminator 会把任何一张钞票以 1/2 的概率判定为真钞。

② 那么，GAN 究竟能做什么呢?

如图 48-5 所示，这是一张非常有意思的图，最左边是真实的图，我们希望去预测视频后几帧的模样，中间这张图是用 MSE 做的，最右边的图是生成对抗网络做的。通过细节分析，我们可以看到中间这张图的耳朵和眼睛都是模糊的，而 GAN 生成的效果明显更好。

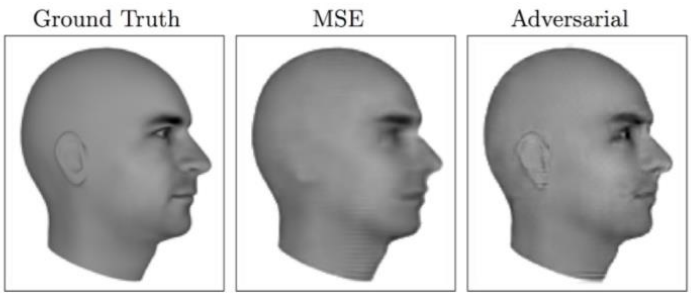


Figure 3: Lotter *et al.* (2015) provide an excellent illustration of the importance of being able to model multi-modal data. In this example, a model is trained to predict the next frame in a video sequence. The video depicts a computer rendering of a moving 3D model of a person's head. The image on the left shows an example of an actual frame of video, which the model would ideally predict. The image in the center shows what happens when the model is trained using mean squared error between the actual next frame and the model's predicted next frame. The model is forced to choose a single answer for what the next frame will look like. Because there are many possible futures, corresponding to slightly different positions of the head, the single answer that the model chooses corresponds to an average over many slightly different images. This causes the ears to practically vanish and the eyes to become blurry. Using an additional GAN loss, the image on the right is able to understand that there are many possible outputs, each of which is sharp and recognizable as a realistic, detailed image.

图 48-5 GAN 生成图像示例

接着我们在看一个超分辨率的实例。首先给出一张超分辨率的图，最左边的图像是原始高分辨率图像（original），然后要对其进行下采样，得到低分辨率图像，接着采用不同的方法对低分辨率图像进行恢复，具体工作如下：

- **bicubic**：第二张图是 bicubic 方法恢复的图像。经过压缩再拉伸还原图像，通过插值运算实现，但其图像会变得模糊。
- **SRResNet**：第三张图像是通过 SRResNet 实现的恢复，比如先压缩图像再用 MSE 和神经网络学习和真实值的差别，再进行恢复。（SRResNet is a neural network trained with mean squared error）
- **SRGAN**：第四张图是通过 SRGAN 实现的，其恢复效果更优。SRGAN 是在 GAN 基础上的改进，它能够理解有多个正确的答案，而不是在许多答案中给出一个最佳输出。

Figure 4: Ledig *et al.* (2016) demonstrate excellent single-image superresolution results that show the benefit of using a generative model trained to generate realistic samples from a multimodal distribution. The leftmost image is an original high-resolution image. It is then downsampled to make a low-resolution image, and different methods are used to attempt to recover the high-resolution image. The bicubic method is simply an interpolation method that does not use the statistics of the training set at all. SRResNet is a neural network trained with mean squared error. SRGAN is a GAN-based neural network that improves over SRGAN because it is able to understand that there are multiple correct answers, rather than averaging over many answers to impose a single best output.

图 48-6 图像恢复对比

我们注意观察图像头部雕饰的细节，发现 GAN 恢复的轮廓更清晰。该实验显示了使用经过训练的生成模型从多模态分布生成真实样本的优势。

图 48-7 GAN 图像恢复效果最佳

在这里，我们也科普下超分辨率——SRCNN。它最早是在论文《Learning a Deep Convolutional Network for Image Super-Resolution》中提出，这篇文章的四位作者分别为董超、Chen Change Loy、何凯明和汤晓欧，也都是妥妥的大神。从 CV 角度来看，这篇论文是真的厉害。

现假设要解决一个问题：能不能解决超分辨率，从一个低分辨率的图像恢复成一个高分辨率的图像，那怎么做呢？他们通过增加两个卷积层的网络就解决了一个实际问题，并且这篇文章发了一个顶会。

- 文章地址：https://link.springer.com/chapter/10.1007/978-3-319-10593-2_13

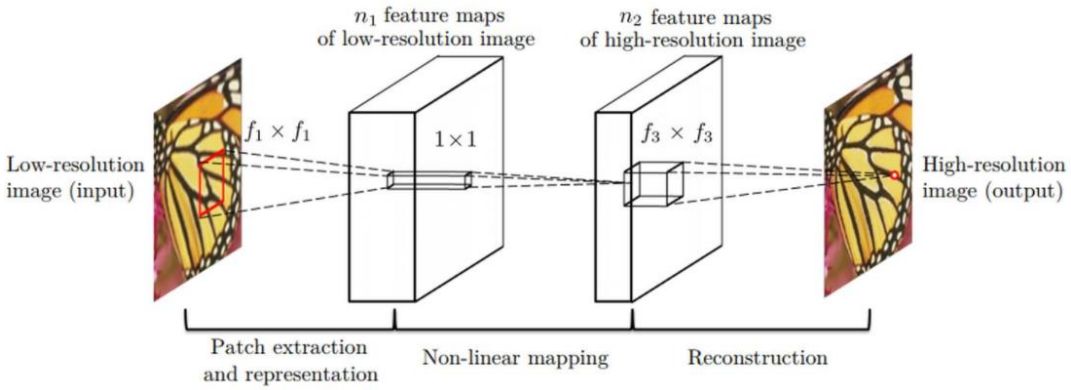


图 48-8 SRCNN 模型框架

更详细的介绍参考知乎 oneTaken 大佬的分享。下面简单进行总结：

“这是第一篇将端到端的深度学习训练来进行超分的论文，整篇论文的的过程现在看起来还是比较简单的，先将低分辨率图片双三次插值上采样到高分辨率图片，然后再使用两层卷积来进行特征映射，最后使用 MSE 来作为重建损失函数进行训练。从现在来看很多东西还是比较粗糙的，但这篇论文也成为很多超分论文的 baseline。”

整篇论文的创新点有：

- 使用了一个卷积神经网络来进行超分，端到端的学习低分辨率与超分辨率之间的映射。
- 将提出的神经网络模型与传统的稀疏编码方法之间建立联系，这种联系

还指导用来设计神经网络模型。

- 实验结果表明深度学习方法可以用于超分中，可以获得较好的质量和较快的速度。

整个的模型架构非常的简单，先是对于输入图片进行双三次插值采样到高分辨空间，然后使用一层卷积进行特征提取，再用 ReLU 进行非线性映射，最后使用一个卷积来进行重建，使用 MSE 来作为重建损失。中间一个插曲是将传统用于超分的稀疏编码算法进行了延伸，可以看作是一种具有不同非线性映射的卷积神经网络模型。

（3）GAN 经典案例

GNN 究竟能做什么呢？下面来看看一些比较有趣的 GAN 案例。

首先是一个视频，这篇文章中介绍了 Zhu 等人开发了交互式（interactive）生成对抗网络（iGAN），用户可以绘制图像的粗略草图，就使用 GAN 生成相似的真实图像。在这个例子中，用户潦草地画了几条绿线，就把它变成一块草地，用户再花了一条黑色的三角形，就创建了一个山包。

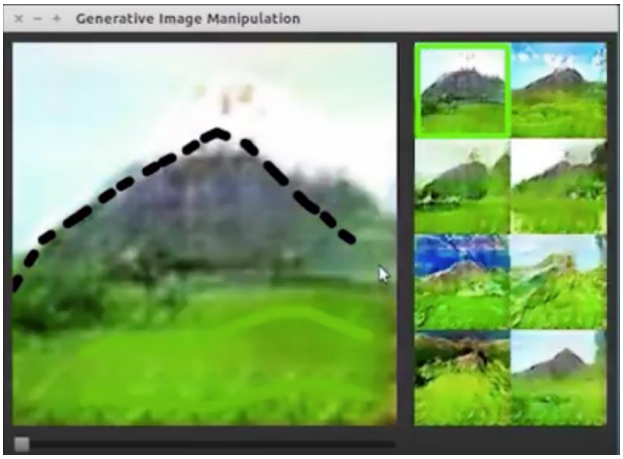


图 48-9 GAN 案例

另一个比较经典的案例是左侧输入的皮包简图最终生成接近真实包的图像，或者将卫星照片转换成地图，将阈值车辆图像转换为现实中逼真的图像。

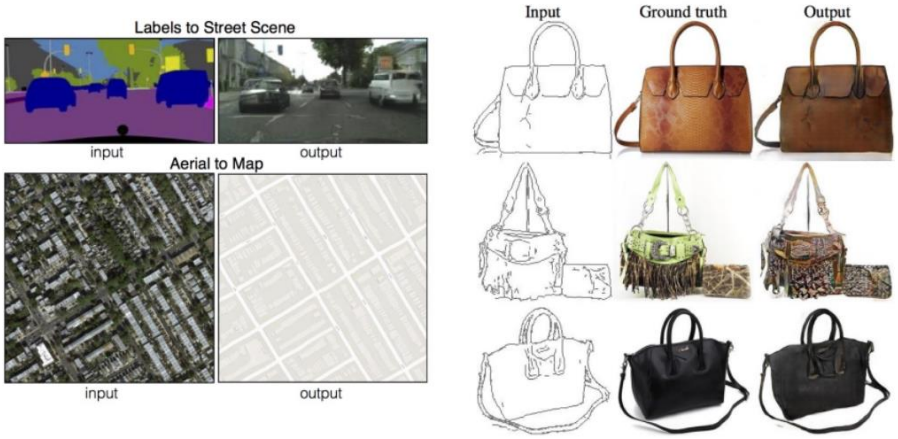


图 48-10 GAN 生成图像案例

再比如通过 GAN 去预测视频中下一帧动画会发生什么，比如右下角给了一张火车的静态图片，会生成一段火车跑动的动态视频。

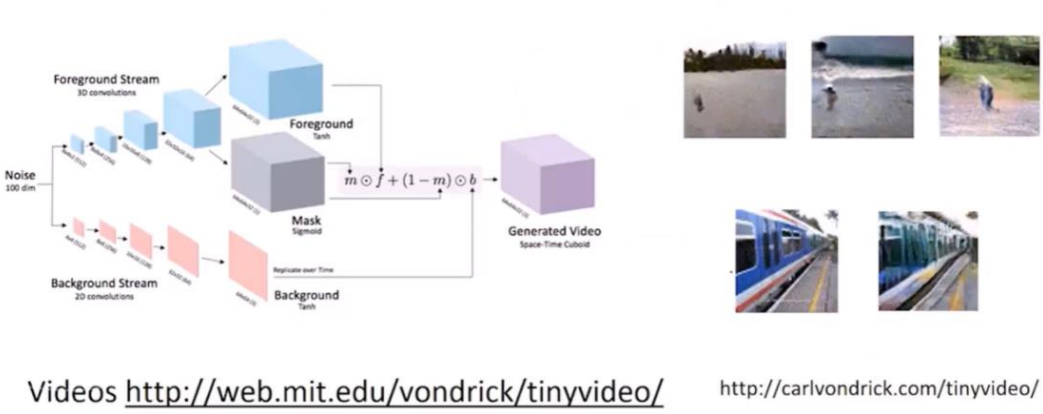


图 48-11 GAN 生成动画案例

Wu 等在 NIPS 2016 中通过 GAN 实现了用噪声去生成一张 3D 椅子模型。

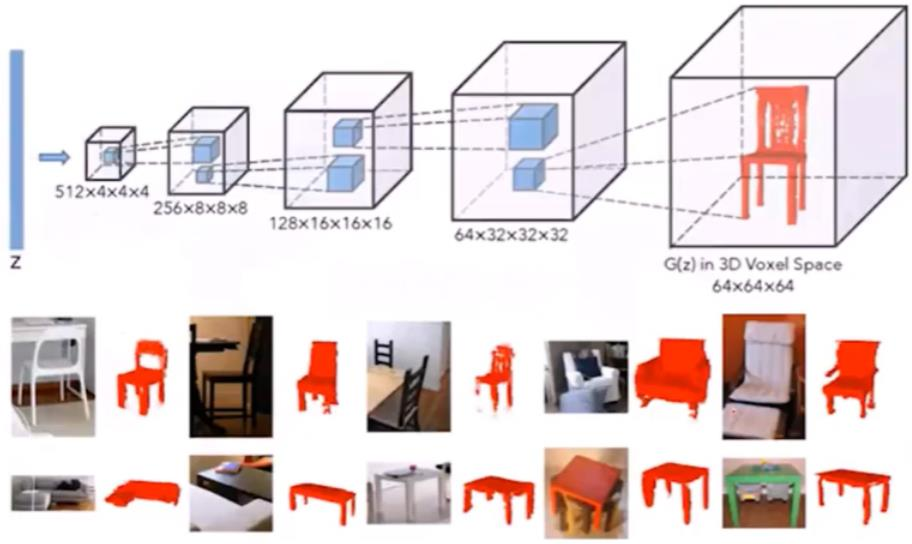


图 48-12 GAN 生成 3D 椅子案例

下图是 starGAN。左侧输入的是一张人脸，然后 GAN 会生成对应的喜怒哀乐表情，这篇文章的创新不是说 GAN 能做这件事，而是提出一个方案，所有的核心功能都在一起，只训练一个生成器，即不是生成多对多的生成器，而只训练一个生成器就能实现这些功能。

starGAN 转移从 RaFD 数据集中学到的知识，在 CelebA 数据集上的多域图像转换结果。第一和第六列显示输入图像，其余列是由 starGAN 生成的图像。请注意，这些图像是由一个单一的生成器网络生成的，而愤怒、快乐和恐惧等面部表情标签都来自 RaFD，而不是 CelebA。

- 论文地址：http://cn.arxiv.org/pdf/1711.09020.pdf

StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation

Yunjey Choi[1,2] Minje Choi[1,2] Munyoung Kim[2,3] Jung-Woo Ha[2] Sunghun Kim[2,4] Jaegul Choo[1,2]
[1] Korea University [2] Clova AI Research, NAVER Corp.
[3] The College of New Jersey [4] Hong Kong University of Science & Technology

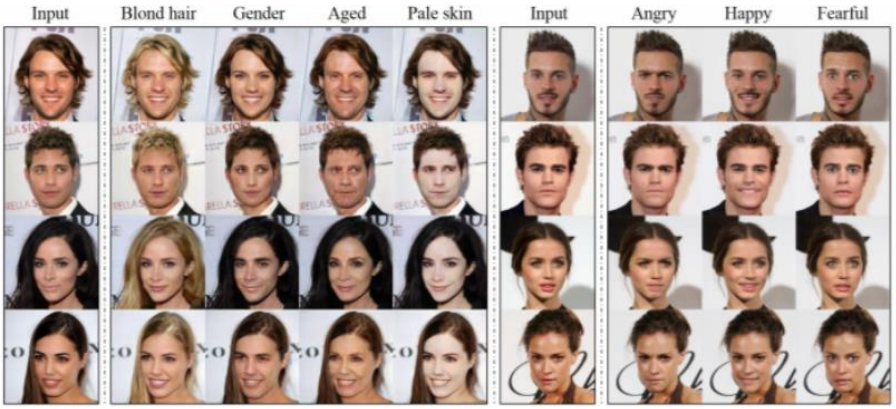


Figure 1. Multi-domain image-to-image translation results on the CelebA dataset via transferring knowledge learned from the RaFD dataset. The first and sixth columns show input images while the remaining columns are images generated by StarGAN. Note that the images are generated by a single generator network, and facial expression labels such as angry, happy, and fearful are from RaFD, not CelebA.

图 48-13 starGAN 论文

2.GAN 预备知识

为什么要讲预备知识呢?

通过学习神经网络的基础知识，能进一步加深我们对 GAN 的理解。当然，看到这篇文章的读者可能很多已经对深度学习有过了解或者是大佬级别，这里也照顾下初学者，普及下 GAN 相关基础知识。这里推荐初学者去阅读作者该系列文章，介绍了很多基础原理。

（1）什么是神经网络

首先，深度学习就是模拟人的脑神经（生物神经网络），比如下图左上方①中的神经元，可以认为是神经网络的接收端，它有很多的树突接收信号，对应 Neuron 的公式如下：

$$z = a_1 w_1 + ... + a_k w_k + ... + a_K w_K + b$$

其中，a 表示信号（树突接收），w 表示对应的权重，它们会进行加权求和组合且包含一个偏置 b。通过激活函数判断能否给下一个神经元传递信号。

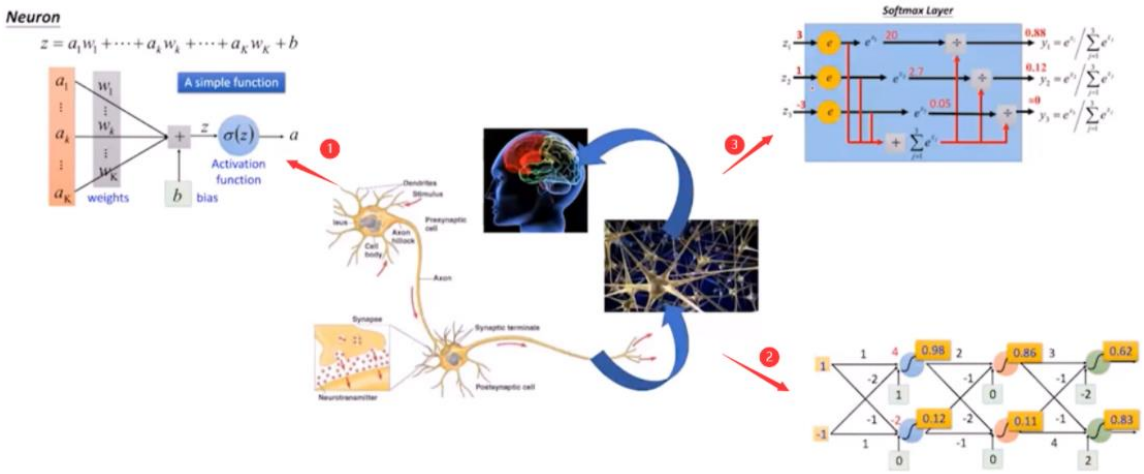


图 48-14 神经元传递信息

有了这个神经元之后，我们需要构建网络，如右下方②所示。经过一层、两层、三层神经网络，我们最后会有一个判断，如右上方③所示，经过 Softmax 函数判断，决策这幅图像是什么，比如猫或狗。

其次，深度学习有哪些知识点呢？深度学习的网络设计如下所示：

- 神经网络常见层

全连接层、激活层、BN 层、Dropout 层、卷积层、池化层、循环层、Embedding 层、Merege 层等

- 网络配置

损失函数、优化器、激活函数、性能评估、初始化方法、正则项等

- 网络训练流程

预训练模型、训练流程、数据预处理（归一化、Embedding）、数据增强（图片翻转旋转曝光生成海量样本）等

图 48-15 神经网络模型

此外，由于深度学习的可解释性非常差，很多时候不知道它为什么正确。NLP 会议上也经常讨论这个可解释性到底重不重要。个人认为，如果用传统的方法效果能达到 80%，而深度学习如果提升非常大，比如 10%，个人感觉工业界还是会用的，因为能提升性能并解决问题。除非比如风控任务，美团检测异常刷单情况，此时需要准确的确认是否刷单。

（2）全连接层

隐藏层的输入和输出都有关联，即全连接层的每一个结点都与上一层的所有结点相连，用来把前边提取到的特征综合起来。由于其全相连的特性，一般全连接层的参数也是最多的。

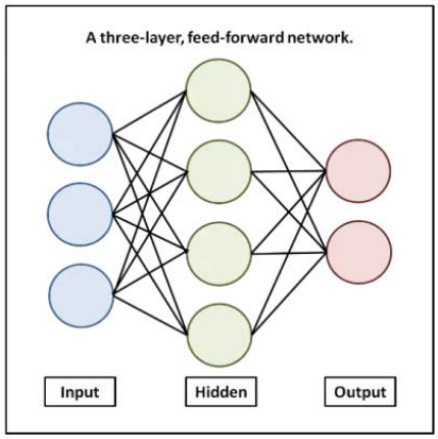


图 48-16 全连接层

全连接层包括神经元的计算公式、维度（神经元个数）、激活函数、权值初始化方法（w、b）、正则项。

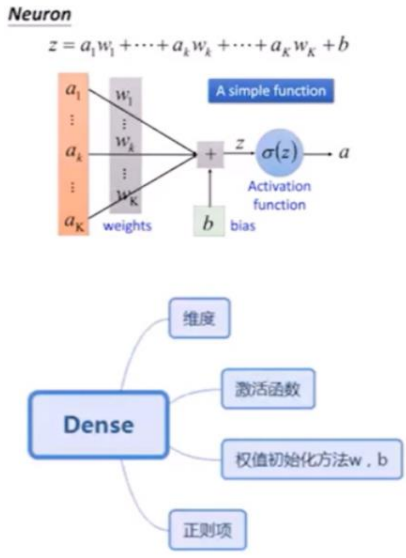


图 48-17 全连接层内容

（3）激活函数

激活函数（activation function）会让某一部分神经元先激活，然后把激活的信息传递给后面一层的神经系统中。比如，某些神经元看到猫的图片，它会对猫的眼睛特别感兴趣，那当神经元看到猫的眼睛时，它就被激励了，它的数值

就会被提高。

激活函数相当于一个过滤器或激励器，它把特有的信息或特征激活，常见的激活函数包括 softplus、sigmoid、relu、softmax、elu、tanh 等。

- 对于隐藏层，我们可以使用 relu、tanh、softplus 等非线性关系；
- 对于分类问题，我们可以使用 sigmoid（值越小越接近于 0，值越大越接近于 1）、softmax 函数，对每个类求概率，最后以最大的概率作为结果；
- 对于回归问题，可以使用线性函数（linear function）来实验。

激活函数可以参考作者前面的文章，具体如下图所示。

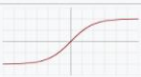
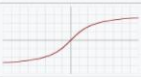
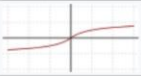
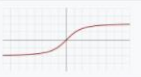
Name	Plot	Equation	Derivative (with respect to x)
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x\rvert}$	$f'(x) = \frac{1}{(1+\lvert x\rvert)^2}$
Inverse square root unit (ISRU)[9]		$f(x) = \frac{x}{\sqrt{1+\alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3$

图 48-18 激活函数

常用的激活函数包括：

- Sigmoid
- Tanh

- ReLU
- Leaky ReLU

对应的曲线如图 48-19 所示。

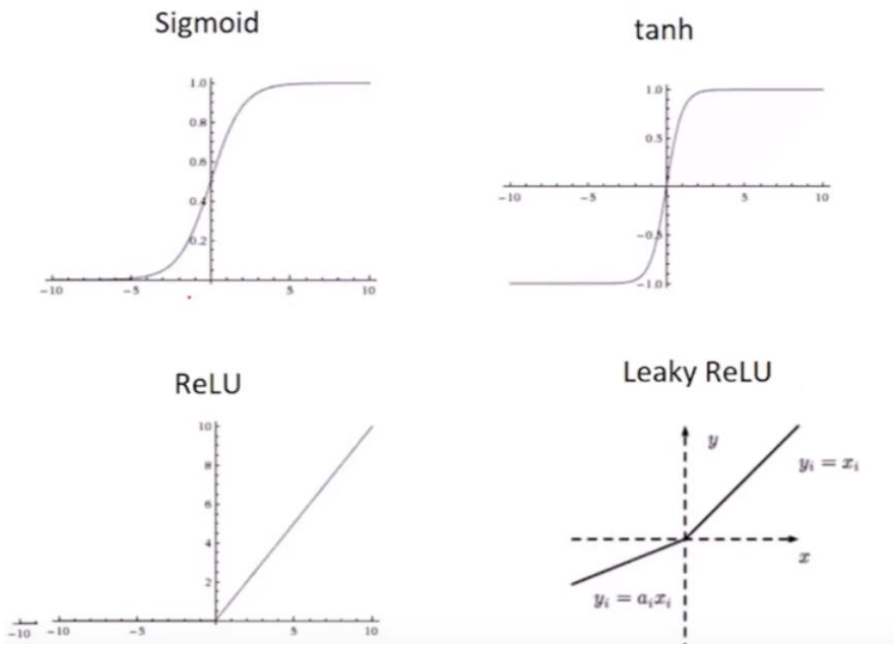


图 48-19 激活函数曲线

（4）反向传播

BP 神经网络是非常经典的网络，这里通过知乎 EdisonGzq 大佬的两张图来解释神经网络的反向传播。对于一个神经元而言，就是计算最后的误差传回来对每个权重的影响，即计算每层反向传递的梯度变化。

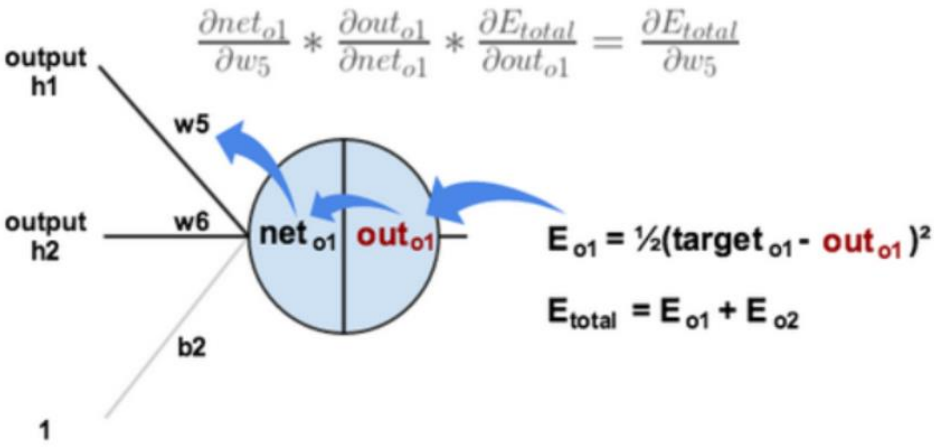


图 48-20 反向传播

对于多个神经元而言，它是两条线的输出反向传递，如下图所示 Eo1 和 Eo2。

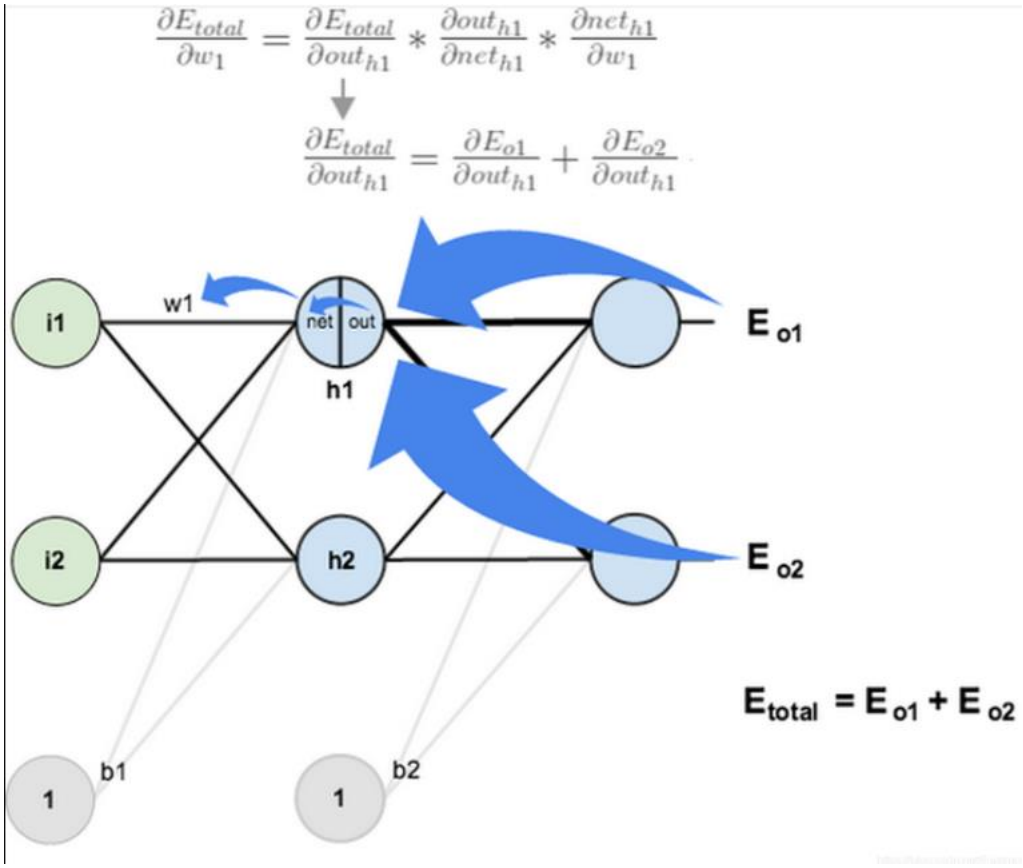


图 48-21 反向传播计算过程

（5）优化器选择

存在梯度变化后，会有一个迭代的方案，这种方案会有很多选择。优化器有很多种，但大体分两类：

- 一种优化器是跟着梯度走，每次只观察自己的梯度，它不带重量
- 一种优化器是带重量的

class tf.train.Optimizer 是优化器（optimizers）类的基类。优化器有很多不同的种类，最基本的一种是 GradientsDescentOptimizer，它也是机器学习中最重要或最基础的线性优化。七种常见的优化器包括：

- class tf.train.GradientDescentOptimizer
- class tf.train.AdagradOptimizer
- class tf.train.AdadeltaOptimizer
- class tf.train.MomentumOptimizer
- class tf.train.AdamOptimizer
- class tf.train.FtrlOptimizer
- class tf.train.RMSPropOptimizer

下面简单介绍其中四个常用的优化器：（推荐 优化器总结 ）

- GradientDescentOptimizer

梯度下降 GD 取决于传进数据的 size，比如只传进去全部数据的十分之一，Gradient Descent Optimizer 就变成了 SGD，它只考虑一部分的数据，一部分一部分的学习，其优势是能更快地学习到去往全局最小量（Global minimum）的路径。

- MomentumOptimizer

它是基于学习效率的改变，它不仅仅考虑这一步的学习效率，还加载了上一步的学习效率趋势，然后上一步加这一步的 learning_rate，它会比 GradientDescentOptimizer 更快到达全局最小量。

- AdamOptimizer

Adam 名字来源于自适应矩估计（Adaptive Moment Estimation），也是梯度下降算法的一种变形，但是每次迭代参数的学习率都有一定的范围，不会因为梯度很大而导致学习率（步长）也变得很大，参数的值相对比较稳定。Adam 算法利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。

- RMSPropOptimizer

Google 用它来优化阿尔法狗的学习效率。RMSProp 算法修改了 AdaGrad 的梯度积累为指数加权的移动平均，使得其在非凸设定下效果更好。

各种优化器用的是不同的优化算法（如 Mmentum、SGD、Adam 等），本质上都是梯度下降算法的拓展。下图通过可视化对各种优化器进行了对比分析，机器学习从目标学习到最优的过程，有不同的学习路径，由于 Momentum 考虑了上一步的学习（learning_rate），走的路径会很长；GradientDescent 的学习时间会非常慢。

如果您是初学者，建议使用 GradientDescentOptimizer 即可，如果您有一定的基础，可以考虑下 MomentumOptimizer、AdamOptimizer 两个常用的优化器，高阶的话，可以尝试学习 RMSPropOptimizer 优化器。总之，您最好结合具体的研究问题，选择适当的优化器。

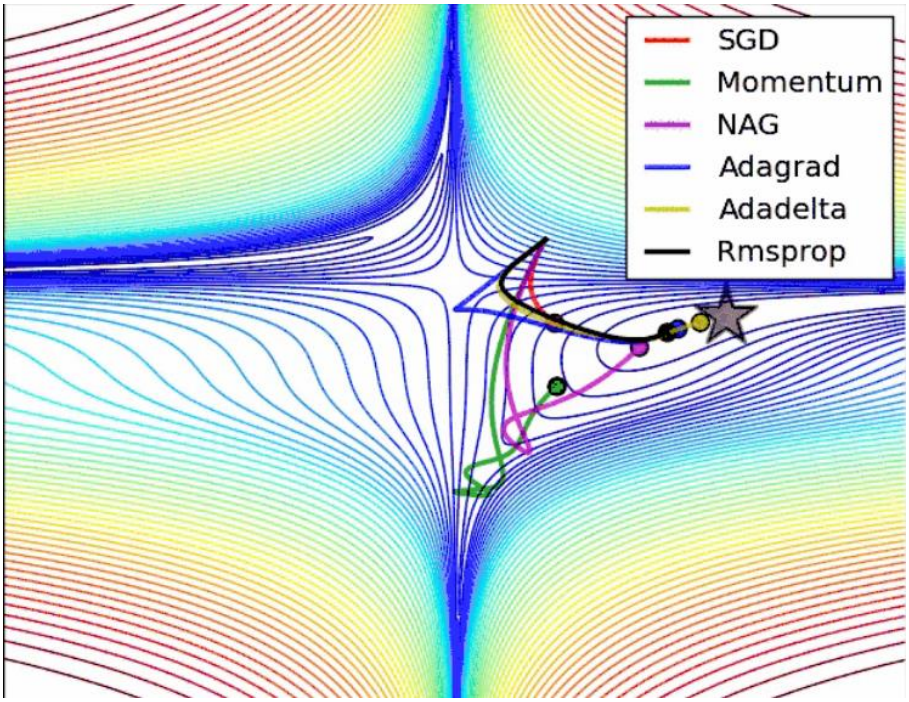


图 48-22 优化器性能比较

（6）卷积层

为什么会提出卷积层呢？因为全连接层存在一个核心痛点：

- 图片参数太多，比如 1000*1000 的图片，加一个隐藏层，隐藏层节点同输入维数，全连接的参数是 10^12，根本训练不过来这么多参数。

解决方法：利器一是局部感知

提出了一个卷积核的概念，局部感知信息。

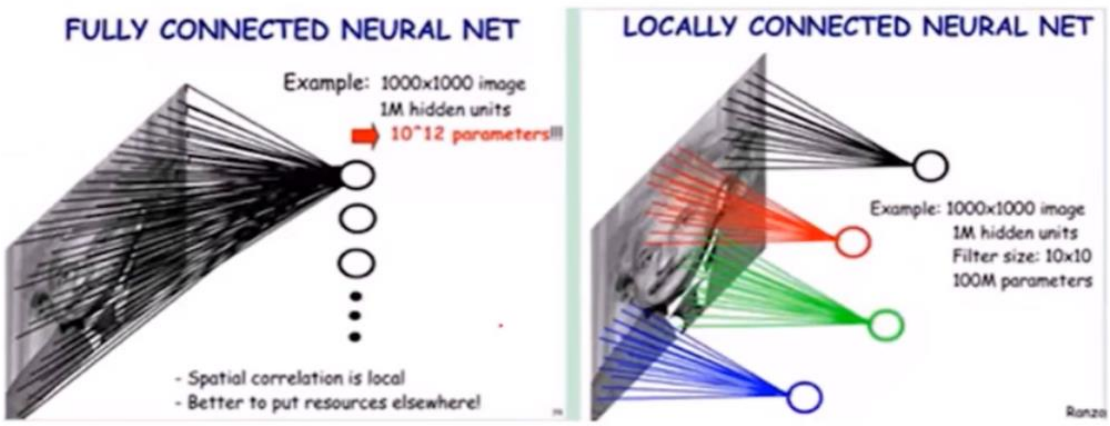


图 48-23 局部感知

解决方法：利器二是参数共享

从图像的左上角按照 3x3 扫描至右下角，获得如右图所示的结果，通过卷积共享减少了参数个数。注意，这里的卷积核是如下：

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

当前扫描的区域为如下，最终计算结果为 2。

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

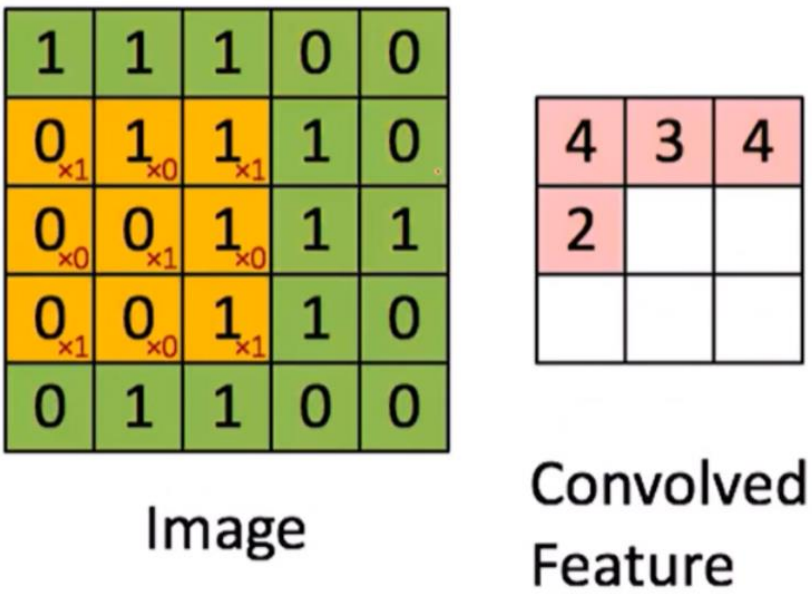


图 48-24 卷积计算

卷积层的核心知识点如下：

- 卷积核数目

- 卷积核大小：如上面 3x3 卷积核
- 卷积核数目
- 卷积核步长：上面的步长是 1，同样可以调格
- 激活函数
- Padding：比如上图需要输出 5x5 的结果图，我们需要对其外圆补零
- 是否使用偏置
- 学习率
- 初始化

卷积层的核心结构如下图所示。

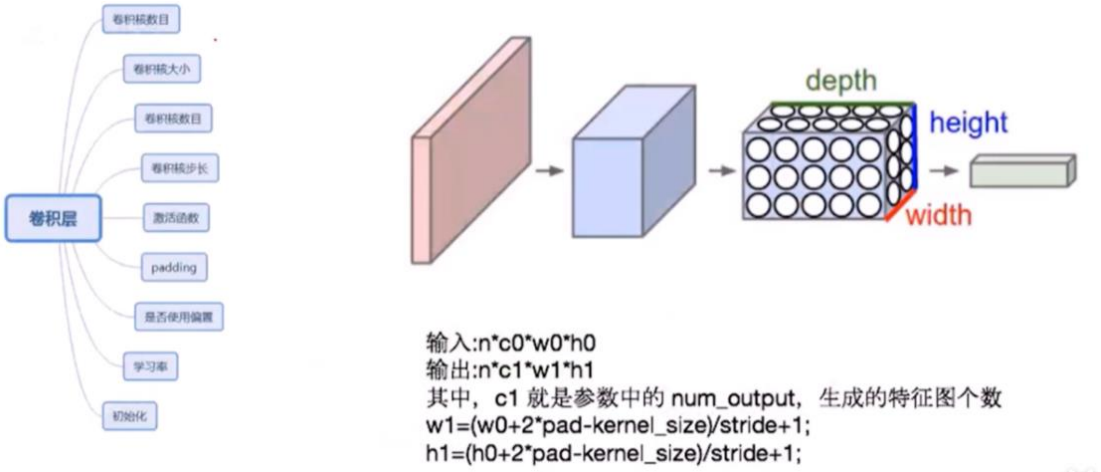


图 48-25 卷积结构

图 48-26 展示了五层卷积层，每层输出的内容。它从最初简单的图形学习到后续的复杂图形。

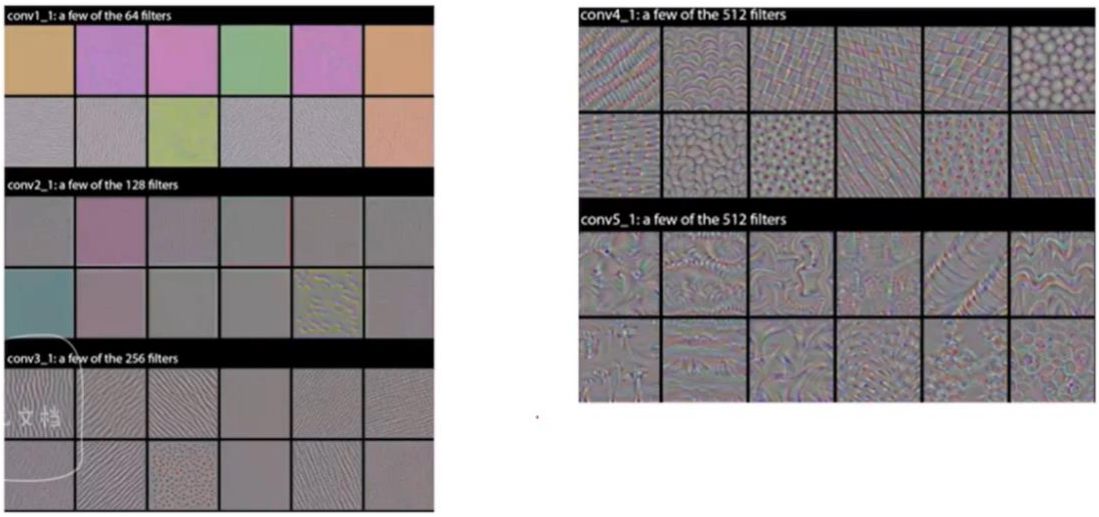


图 48-26 卷积结果

（7）池化层

池化层主要解决的问题是：

- 使特征图变小，简化网络；特征压缩，提取主要特征

常用池化层包括：

- 最大池化：比如从左上角红色区域中选择最大的 6，接着是 8、3、4
- 平均池化：选择平均值

基本结构如图 48-26 所示。

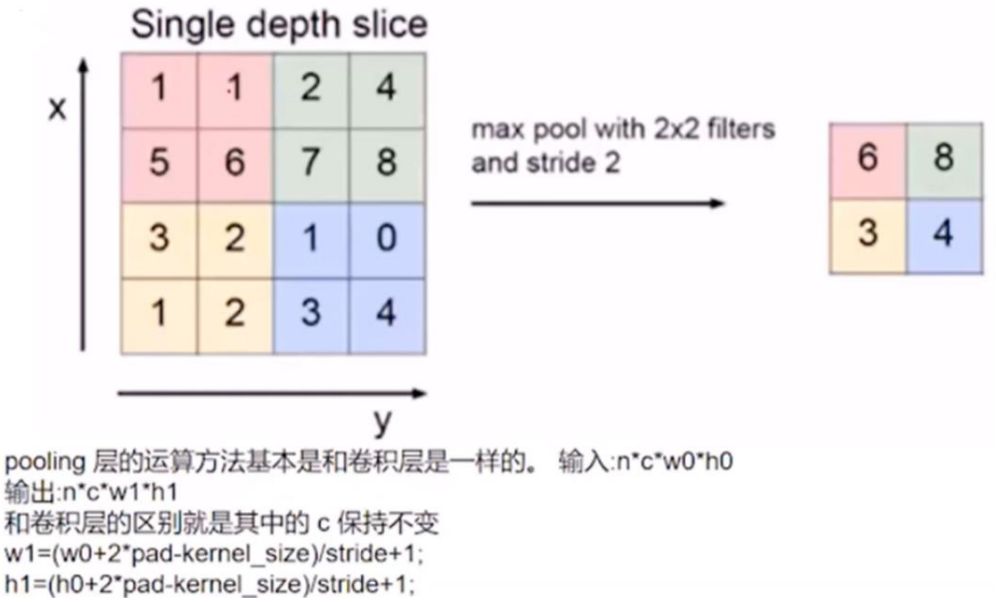


图 48-27 池化层结构

基本知识点如下图所示：

图 48-28 池化层知识点

（8）图像问题基本思路

此时，我们通过介绍的全连接层、卷积层、池化层，就能解决实际的问题。如下图所示：

- 输入层

如 NLP 句子、句对，图像的像素矩阵，语音的音频信息

■ 表示层

DNN：全连接+非线性（特征非线性融合）

CNN：Conv1d、Conv2d、Pooling

RNN：LSTM、GRU（选择记忆性）

■ 应用层

分类、回归、序列预测、匹配

对应的问题结构用图 48-29 描述。

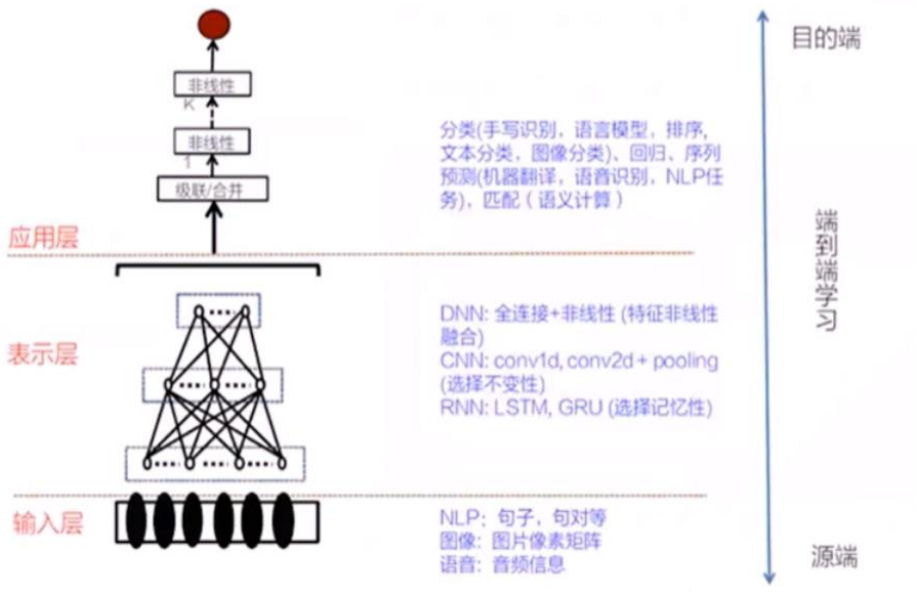


图 48-29 图像处理基本流程

此外，可以将图像问题基本思路简化为下图的模型。

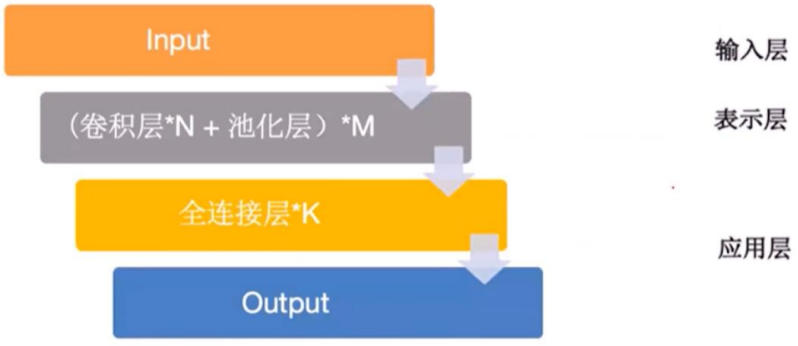


图 48-30 图像问题基本思路

至此，预备知识介绍完毕！接下来我们进入 GAN 网络实战分析。

3.GAN 模型解析

GANs（Generativeadversarial networks）对抗式生成网络。

- Generative：生成式模型
- Adversarial：采取对抗的策略
- Networks：网络

首先，我们先说说 GAN 要做什么呢?

- 最开始在图(a)中我们生成绿线，即生成样本的概率分布，黑色的散点是真实样本的概率分布，这条蓝线是一个判决器，判断什么时候应该是真的或假的。
- 我们第一件要做的事是把判决器判断准，如图(b)中蓝线，假设在 0.5 的位置下降，之前的认为是真实样本，之后的认为是假的样本。
- 当它固定完成后，在图(c)中，生成器想办法去和真实数据作拟合，想办法去误导判决器。
- 最终输出图(d)，如果你真实的样本和生成的样本完全一致，分布完全一致，判决器就傻了，无法继续判断。

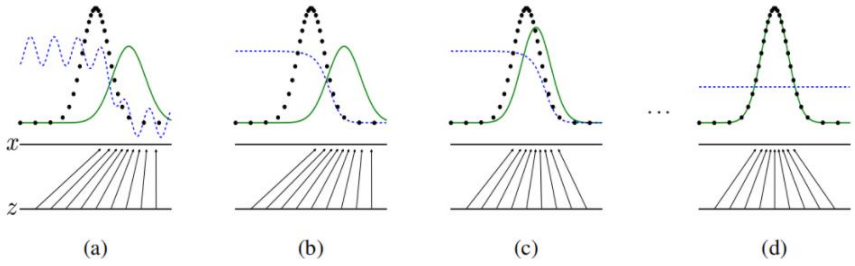


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

图 48-31 GAN 模型解析

GAN 的整体思路是一个生成器，一个判别器，并且 GoodFellow 论文证明了 GAN 全局最小点的充分必要条件是：生成器的概率分布和真实值的概率分布是一致的时候。

$$GlobalOptimalityof p_g = p_{data}$$

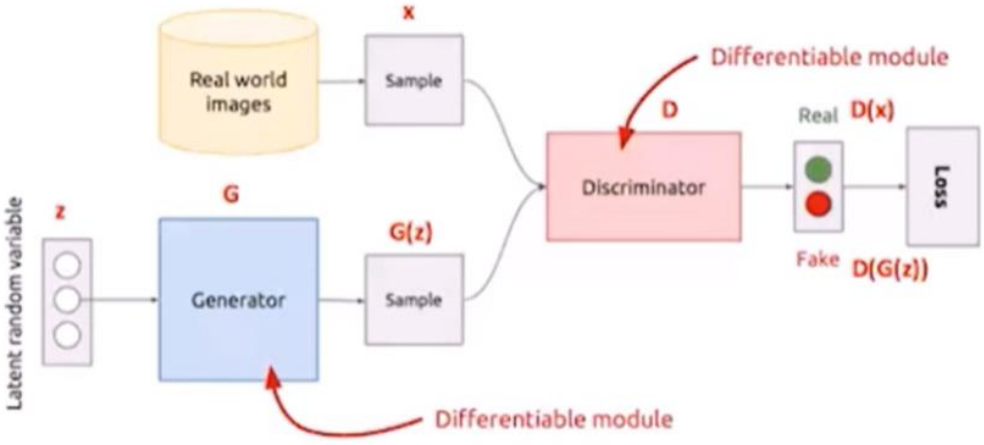


图 48-31 GAN 模型

其次，GAN 还需要分析哪些问题呢?

- 目标函数如何设定?
- 如何生成图片?
- G 生成器和 D 判决器应该如何设置?
- 如何进行训练?

(1) 目标函数

该目标函数如下所示，其中：

- max()式子是第一步，表示把生成器 G 固定，让判别器尽量区分真实样本和假样本，即希望生成器不动的情况下，判别器能将真实的样本和生成的样本区分开。
- min()式子是第二步，即整个式子。判别器 D 固定，通过调整生成器，希望判别器出现失误，尽可能不要让它区分开。

这也是一个博弈的过程。

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

整个公式的具体含义如下：

- 式子由两项构成，x 表示真实图片，z 表示输入 G 网络的噪声，而 G(z)表示 G 网络生成的图片。
- D(x)表示 D 网络判断真实图片是否真实的概率（因为 x 就是真实的，所以对于 D 来说，这个值越接近 1 越好）。
- D(G(z))是 D 网络判断 G 生成的图片是否真实的概率。
- G 的目的：G 应该希望自己生成的的图片越接近真实越好。

- D 的目的：D 的能力越强，D(x)应该越大，D(G(x))应该越小，这时 V(D,G)会变大，因此式子对于 D 来说是求最大（max_D）。
- trick：为了前期加快训练，生成器的训练可以把 log(1−D(G(z)))换成 −log(D(G(z)))损失函数。

接着我们回到大神的原论文，看看其算法（Algorithm 1）流程。

- 最外层是一个 for 循环，接着是 k 次 for 循环，中间迭代的是判决器。
- k 次 for 循环结束之后，再迭代生成器。
- 最后结束循环。

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

① 判别器

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

② 生成器

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

By: Eastmount CSDN

图 48-33 GAN 算法流程

(2) GAN 图片生成

接着我们介绍训练方案，通过 GAN 生成图片。

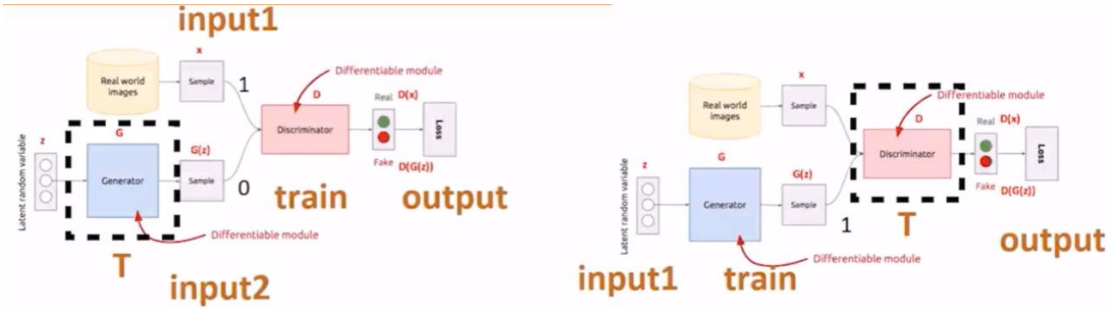


图 48-33 GAN 生成图片

- 第一步（左图）：希望判决器尽可能地分开真实数据和我生成的数据。那么，怎么实现呢？我的真实数据就是 input1（Real World images），我生成的数据是 input2（Generator）。input1 的正常输出是 1，input2 的正常输出是 0，对于一个判决器（Discriminator）而言，我希望它判决好，首先把生成器固定住（虚线 T），然后生成一批样本和真实数据混合给判决器去判断。此时，经过训练的判决器变强，即固定生成器且训练判决器。
- 第二步（右图）：固定住判决器（虚线 T），我想办法去混淆它，刚才经过训练的判决器很厉害，此时我们想办法调整生成器，从而混淆判别器，即通过固定判决器并调整生成器，使得最后的输出 output 让生成的数据也输出 1（第一步为 0）。

GAN 的核心就是这些，再简单总结下，即：

- 步骤 1 是在生成器固定的时候，我让它产生一批样本，然后让判决器正确区分真实样本和生成样本。（生成器标签 0、真实样本标签 1）。
- 步骤 2 是固定判决器，通过调整生成器去尽可能的瞒混判决器，所以实际上此时训练的是生成器。（生成器的标签需要让判决器识别为 1，即真

实样本）。

其伪代码如下：

```
for 迭代 in range(迭代总数):
    for batch in range(batch_size):
        新 batch = input1 的 batch + input2 的 batch (batch 加倍)
        for 轮数 in range(判别器中轮数):
            步骤一 训练 D
        步骤二 训练 G
```

4.生成手写数字案例

接下来我们通过手写数字图像生成代码来加深读者的印象。这是一个比较经典的共有数据集，包括图像分类各种案例较多，这里我们主要是生成手写数字图像，如图 48-34 所示。

图 48-34 手写数字图像

首先，我们看看生成器是如何生成一个图像（从噪音生成）？

核心代码如下，它首先要随机生成一个噪音（noise），所有生成的图片都是靠噪音实现的。Keras 参考代码下载地址为：

- https://github.com/jacobgil/keras-dcgan/blob/master/dcgan.py

(1) 生成器 G

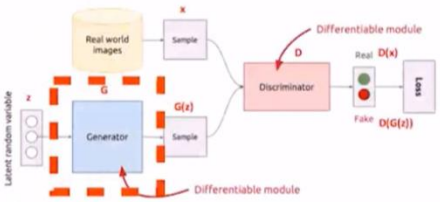
生成器总共包括：

- 全连接层：输入 100 维，输出 1024 维
- 全连接层：128x7x7 表示图片 128 通道，大小 7x7
- BatchNormalization：如果不加它 DCGAN 程序会奔溃
- UpSampling2D：对卷积结果进行上采样从而将特征图放大 14x14
- Conv2D：卷积操作像素尺度不变（same）
- UpSampling2D：生成 28x28
- Conv2D：卷积操作
- Activation：激活函数 tanh

其核心代码实现如图 48-35 所示。

```
noise = np.random.uniform(-1, 1, (batch_size, latent_size))

def generator_model():
    model = Sequential()
    model.add(Dense(input_dim=100, output_dim=1024))
    model.add(Activation('relu'))
    model.add(Dense(128*7*7))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Reshape((7, 7, 128), input_shape=(128*7*7,)))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(64, (5, 5), padding='same'))
    model.add(Activation('relu'))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(1, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    return model
```



https://github.com/jacobgil/keras-dcgan/blob/master/dcgan.py

图 48-35 生成器代码

(2) 判别器 D

判别器就是做一个二分类的问题，要么真要么假。

- Conv2D：卷积层
- MaxPooling2D：池化层
- Conv2D：卷积层
- MaxPooling2D：池化层
- Flatten：拉直一维
- Dense：全连接层
- Activation：sigmoid 二分类

```
def discriminator_model():
    model = Sequential()
    model.add(Conv2D(64, (5, 5),  padding='same',input_shape=(28, 28, 1)))
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (5, 5)))
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation('tanh'))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    return model
```

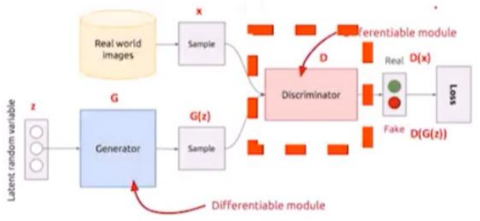


图 48-36 判别器代码

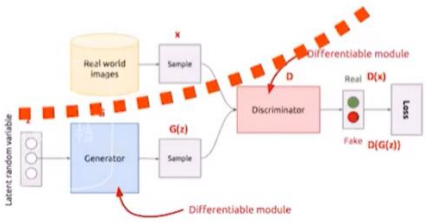
(3) 辅助函数

如何把 D 固定去调整 G 的函数 generator_containing_discriminator。

- model.add(g)：加载生成器 G
- d.trainable=False：判决器 D 固定

combine_images 函数实现合并图像的操作。

```
def generator_containing_discriminator(g, d):
    model = Sequential()
    model.add(g)
    d.trainable = False
    model.add(d)
    return model
```




```
def combine_images(generated_images):
    num = generated_images.shape[0]
    width = int(math.sqrt(num))
    height = int(math.ceil(float(num)/width))
    shape = generated_images.shape[1:3]
    image = np.zeros((height*shape[0], width*shape[1]),
            dtype=generated_images.dtype)
    for index, img in enumerate(generated_images):
        i = int(index/width)
        j = index % width
        image[i*shape[0]:(i+1)*shape[0], j*shape[1]:(j+1)*shape[1]] = \
        img[:, :, 0]
    return image
```

图 48-37 辅助函数代码

(4) GAN 图片生成训练

GAN 核心流程包括：

- load_data：载入图片

- d = discriminator_model：定义判别器 D
- g = generator_model：定义生成器 G
- generator_containing_discriminator：固定 D 调整 G
- SGD、compile：定义参数、学习率
- for epoch in range、for index in rangeBATCH
- X = np.concatenate：图像数据和生成数据混合
- y = [1] x BATCH_SIZE + [0] x BTCH_SIZE：输出 label
- d_loss = d.train_on_batch(X,y)：训练 D 判别器（步骤一）
- d.trainable = False：固定 D
- g_loss = d_on_g.train_on_batch(noise, [1]xBATCH_SIZE)：训练 G 生成器（步骤二），混淆
- d.trainable = True：打开 D 重复操作
- 保存参数和模型

图片生成训练的核心代码如下图所示。

```
def train(BATCH_SIZE):
    (X_train, y_train), (X_test, y_test) = mnist.load_data()
    X_train = (X_train.astype(np.float32) - 127.5)/127.5
    X_train = X_train[:, :, :, None]
    X_test = X_test[:, :, :, None]
    # X_train = X_train.reshape((X_train.shape, 1) + X_train.shape[1:])
    d = discriminator_model()
    g = generator_model()
    d_on_g = generator_containing_discriminator(g, d)
    d_optim = SGD(lr=0.0005, momentum=0.9, nesterov=True)
    g_optim = SGD(lr=0.0005, momentum=0.9, nesterov=True)
    g.compile(loss='binary_crossentropy', optimizer="SGD")
    d_on_g.compile(loss='binary_crossentropy', optimizer=g_optim)
    d.trainable = True
    d.compile(loss='binary_crossentropy', optimizer=d_optim)
```

```
for epoch in range(100):
    print("Epoch is", epoch)
    print("Number of batches", int(X_train.shape[0]/BATCH_SIZE))
    for index in range(int(X_train.shape[0]/BATCH_SIZE)):
        noise = np.random.uniform(-1, 1, size=(BATCH_SIZE, 100))
        image_batch = X_train[index*BATCH_SIZE:(index+1)*BATCH_SIZE]
        generated_images = g.predict(noise, verbose=0)
        if index % 20 == 0:
            image = combine_images(generated_images)
            image = image*127.5+127.5
            Image.fromarray(image.astype(np.uint8)).save(
              str(epoch)+"_"+str(index)+".png")
        X = np.concatenate((image_batch, generated_images))
        y = [1] * BATCH_SIZE + [0] * BATCH_SIZE
        d_loss = d.train_on_batch(X, y)
        print("batch %d d_loss : %f" % (index, d_loss))
        noise = np.random.uniform(-1, 1, (BATCH_SIZE, 100))
        d.trainable = False
        g_loss = d_on_g.train_on_batch(noise, [1] * BATCH_SIZE)
        d.trainable = True
        print("batch %d g_loss : %f" % (index, g_loss))
        if index % 10 == 9:
            g.save_weights('generator', True)
            d.save_weights('discriminator', True)
```

图 48-38 GAN 图片生成训练代码

(5) 生成图片

模型训练好之后，我们想办法用 GAN 生成图片。

- g = generator_model：定义生成器模型
- g.load_weights：载入训练好的生成器（generator）
- noise：随机产生噪声
- 然后用 G 生成一幅图像，该图像就能欺骗判别器 D

最终完整代码如下，该代码能实现一个简单的 GAN 生成图片应用。

```
# -*- coding: utf-8 -*-
"""
Created on 2021-03-19
@author: xiuzhang Eastmount CSDN
参考：https://github.com/jacobgil/keras-dcgan
"""
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers.core import Activation
from keras.layers.normalization import
BatchNormalization
from keras.layers.convolutional import UpSampling2D
```

```
from keras.layers.convolutional import Conv2D,
MaxPooling2D
from keras.layers.core import Flatten
from keras.optimizers import SGD
from keras.datasets import mnist
import tensorflow as tf
import numpy as np
from PIL import Image
import argparse
import math
import os


## GPU 处理 读者如果是 CPU 注释该部分代码即可
## 指定每个 GPU 进程中使用显存的上限 0.9 表示可以使用 GPU
90%的资源进行训练
os.environ["CUDA_DEVICES_ORDER"] = "PCI_BUS_IS"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
gpu_options =
tf.GPUOptions(per_process_gpu_memory_fraction=0.8)
sess =
tf.Session(config=tf.ConfigProto(gpu_options=gpu_option
```

```
s))


#----------------------------------------
--------------------
#生成器
def generator_model():
    model = Sequential()
    model.add(Dense(input_dim=100, output_dim=1024))
    model.add(Activation('tanh'))
    model.add(Dense(128*7*7))          #7x7 128 通道
    model.add(BatchNormalization())
    model.add(Activation('tanh'))
    model.add(Reshape((7,                7,                128),
input_shape=(128*7*7,)))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(64, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(1, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    return model
```

```
#-----------------------------------------
-------------------
#判别器
def discriminator_model():
    model = Sequential()
    model.add(
            Conv2D(64, (5, 5),
            padding='same',
            input_shape=(28, 28, 1))
            )
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (5, 5)))
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation('tanh'))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
```

```
    return model


#-----------------------------------------
--------------------
#辅助函数 固定 D 调整 G
def generator_containing_discriminator(g, d):
    model = Sequential()
    model.add(g)
    d.trainable = False
    model.add(d)
    return model


#辅助函数 合并图像
def combine_images(generated_images):
    num = generated_images.shape[0]
    width = int(math.sqrt(num))
    height = int(math.ceil(float(num)/width))
    shape = generated_images.shape[1:3]
    image = np.zeros((height*shape[0], width*shape[1]),
                     dtype=generated_images.dtype)
    for index, img in enumerate(generated_images):
```

```
            i = int(index/width)
            j = index % width
            image[i*shape[0]:(i+1)*shape[0],
j*shape[1]:(j+1)*shape[1]] = \
                img[:, :, 0]
    return image


#------------------------------------------
--------------------
#训练
def train(BATCH_SIZE):
    (X_train, y_train), (X_test, y_test) = mnist.load_data()
    X_train = (X_train.astype(np.float32) - 127.5)/127.5
    X_train = X_train[:, :, :, None]
    X_test = X_test[:, :, :, None]
    #X_train = X_train.reshape((X_train.shape, 1) +
X_train.shape[1:])
    d = discriminator_model()
    g = generator_model()
    d_on_g = generator_containing_discriminator(g, d)
    d_optim = SGD(lr=0.0005, momentum=0.9,
```

```
nesterov=True)
    g_optim = SGD(lr=0.0005, momentum=0.9,
nesterov=True)
    g.compile(loss='binary_crossentropy',
optimizer="SGD")
    d_on_g.compile(loss='binary_crossentropy',
optimizer=g_optim)
    d.trainable = True
    d.compile(loss='binary_crossentropy',
optimizer=d_optim)
    for epoch in range(100):
        print("Epoch is", epoch)
        print("Number of batches",
int(X_train.shape[0]/BATCH_SIZE))
        for index in
range(int(X_train.shape[0]/BATCH_SIZE)):
            noise = np.random.uniform(-1, 1,
size=(BATCH_SIZE, 100))
            image_batch =
X_train[index*BATCH_SIZE:(index+1)*BATCH_SIZE]
            generated_images = g.predict(noise,
```

```
verbose=0)
                if index % 20 == 0:
                    image                                    =
combine_images(generated_images)
                    image = image*127.5+127.5

Image.fromarray(image.astype(np.uint8)).save(
                        str(epoch)+"_"+str(index)+".png")
                X       =       np.concatenate((image_batch,
generated_images))
                y = [1] * BATCH_SIZE + [0] * BATCH_SIZE
                d_loss = d.train_on_batch(X, y)
                print("batch %d d_loss : %f" % (index, d_loss))
                noise       =       np.random.uniform(-1,       1,
(BATCH_SIZE, 100))
                d.trainable = False
                g_loss  =  d_on_g.train_on_batch(noise,  [1]  *
BATCH_SIZE)
                d.trainable = True
                print("batch %d g_loss : %f" % (index, g_loss))
                if index % 10 == 9:
```

```
                g.save_weights('generator', True)
                d.save_weights('discriminator', True)


#--------------------------------------------------
--------------------
#GAN 图片生成
def generate(BATCH_SIZE, nice=False):
    g = generator_model()
    g.compile(loss='binary_crossentropy',
optimizer="SGD")
    g.load_weights('generator')
    if nice:
        d = discriminator_model()
        d.compile(loss='binary_crossentropy',
optimizer="SGD")
        d.load_weights('discriminator')
        noise = np.random.uniform(-1, 1,
(BATCH_SIZE*20, 100))
        generated_images = g.predict(noise, verbose=1)
        d_pret = d.predict(generated_images, verbose=1)
        index = np.arange(0, BATCH_SIZE*20)
```

```
            index.resize((BATCH_SIZE*20, 1))
            pre_with_index = list(np.append(d_pret, index,
axis=1))
            pre_with_index.sort(key=lambda x: x[0],
reverse=True)
            nice_images = np.zeros((BATCH_SIZE,) +
generated_images.shape[1:3], dtype=np.float32)
            nice_images = nice_images[:, :, :, None]
            for i in range(BATCH_SIZE):
                idx = int(pre_with_index[i][1])
                nice_images[i, :, :, 0] =
generated_images[idx, :, :, 0]
            image = combine_images(nice_images)
        else:
            noise = np.random.uniform(-1, 1, (BATCH_SIZE,
100))
            generated_images = g.predict(noise, verbose=1)
            image = combine_images(generated_images)
        image = image*127.5+127.5
        Image.fromarray(image.astype(np.uint8)).save(
            "generated_image.png")
```

```
#参数设置
def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--mode", type=str)
    parser.add_argument("--batch_size",          type=int,
default=128)
    parser.add_argument("--nice",                dest="nice",
action="store_true")
    parser.set_defaults(nice=False)
    args = parser.parse_args()
    return args

if __name__ == "__main__":
    """
    args = get_args()
    if args.mode == "train":
        train(BATCH_SIZE=args.batch_size)
    elif args.mode == "generate":
        generate(BATCH_SIZE=args.batch_size,
nice=args.nice)
```

```
"""
mode = "train"
if mode == "train":
    train(BATCH_SIZE=128)
elif mode == "generate":
    generate(BATCH_SIZE=128)
```

代码执行参数：

```
Training:
python dcgan.py --mode train --batch_size <batch_size>
python dcgan.py --mode train --path ~/images --batch_size 128

Image generation:
python dcgan.py --mode generate --batch_size <batch_size>
python dcgan.py --mode generate --batch_size <batch_size> --nice : top 5% im
python dcgan.py --mode generate --batch_size 128
```

图 48-39 GAN 图片生成执行参数

训练过程，首先手写数字 MNIST 图片数据集可以下载存储至该位置，也可以运行代码在线下载。

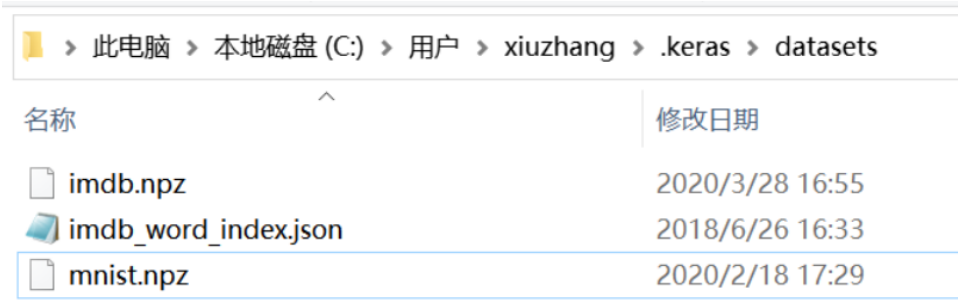


图 48-40 数据集下载

运行过程如下所示：

```
Epoch is 0
```

```
Number of batches 468
batch 0 d_loss : 0.648902
batch 0 g_loss : 0.672132
batch 1 d_loss : 0.649307
....
batch 466 g_loss : 1.305099
batch 467 d_loss : 0.375284
batch 467 g_loss : 1.298173


Epoch is 1
Number of batches 468
batch 0 d_loss : 0.461435
batch 0 g_loss : 1.231795
batch 1 d_loss : 0.412679
....
```

运行过程中会生成很多图像，随着训练次数增加图像会越来越清晰。

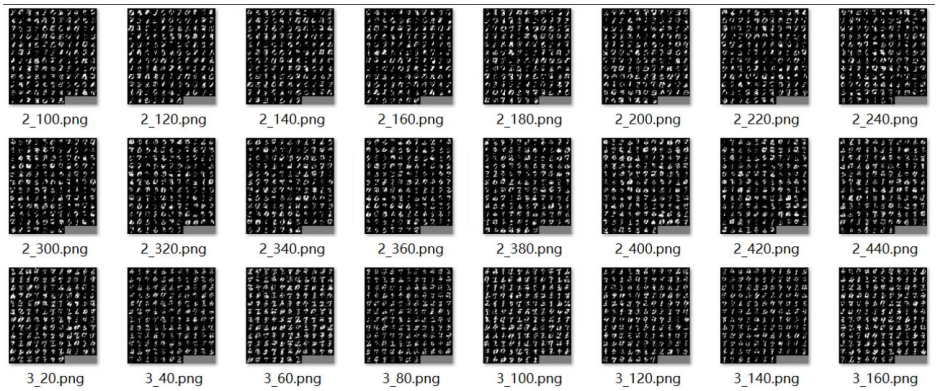


图 48-41 数据集训练

然后参数设置为“generate”，利用 GAN 最终生成图像，如下图所示。

图 48-42 GAN 最终生成图像

5.其他常见 GAN 网络

(1) CGAN

首先，GAN 如何输出指定类的图像呢?

CGAN 出场。这里简单介绍下 GAN 和 CGAN 的区别：GAN 只能判断生成的东西是真的或假的，如果想指定生成图像如 1、2、3 呢? GAN 会先生成

100 张图像，然后从中去挑选出 1、2、3，这确实不方便。

在 2014 年提出 GAN 时，CGAN 也被提出来了。CGAN 除了生成以外，还要把条件带出去，即带着我们要生成一个什么样的图条件去混淆，如下右图：

- 噪声 z 向量+条件 c 向量去生成。

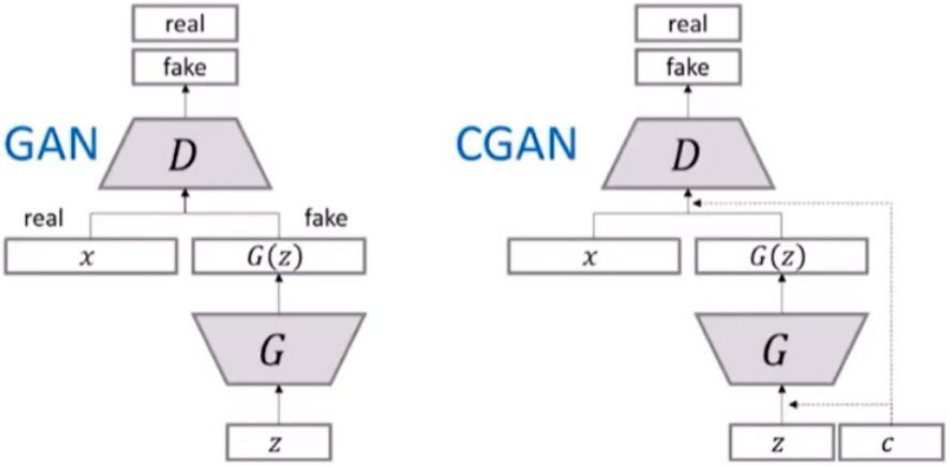
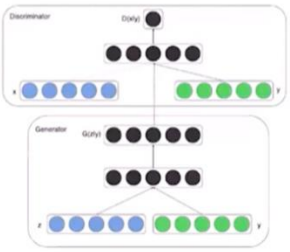


图 48-43 CGAN 结构

所以整套流程大体不变，接着我们看看公式，它在 D(x|y)和 G(z|y)中增加了 y。其中，y 不一定是指定类的输出，可以是一些条件。

CGAN Conditional Generative Adversarial Nets

Fig 1 illustrates the structure of a simple conditional adversarial net.



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]. \quad (2)$$

y不一定是指定类的输出，可以是一些条件
这是GAN生成太自由的解决方案

图 48-44 CGAN 结构

(2) DCGAN

DCGAN （ Deep Convolutional Generative Adversarial

Networks）是卷积神经网络和对抗神经网络结合起来的一篇经典论文，核心要素是：

- 在不改变 GAN 原理的情况下提出一些有助于增强稳定性的 tricks。

注意，这一点很重要。因为 GAN 训练时并没有想象的稳定，生成器最后经常产生无意义的输出或奔溃，但是 DCGAN 按照 tricks 能生成较好的图像。

- 论文地址：https://arxiv.org/pdf/1511.06434.pdf

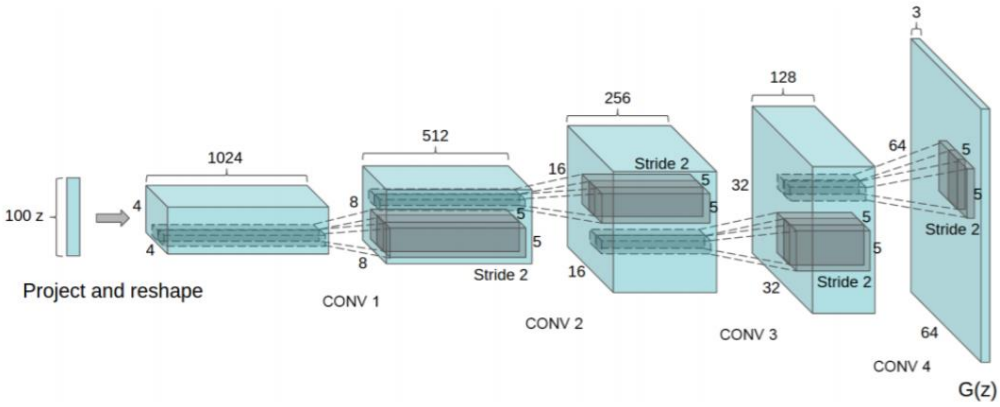


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

图 48-45 DCGAN 结构

DCGAN 论文使用的 tricks 包括：

- 所有 pooling 都用 strided convolutions 代替，pooling 的下采样是损失信息的，strided convolutions 可以让模型自己学习损失的信息
- 生成器 G 和判别器 D 都要用 BN 层（解决过拟合）
- 把全连接层去掉，用全卷积层代替

- 生成器除了输出层，激活函数统一使用 ReLU，输出层用 Tanh
- 判别器所有层的激活函数统一都是 LeakyReLU

DCGAN 应用案例如图 48-46 所示。

Figure 2: Generated bedrooms after one training pass through the dataset. Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate.

图 48-46 DCGAN 生成图片

(3) ACGAN

ACGAN（既能生成图像又能进行分类）来自论文“Conditional Image Synthesis with Auxiliary Classifier GANs”，该判别器不仅要判断是真（real）或假（fake），还要判断其属于哪一类。

- 论文地址：https://arxiv.org/pdf/1610.09585.pdf

ACGAN 模型结果如图 48-47 所示。

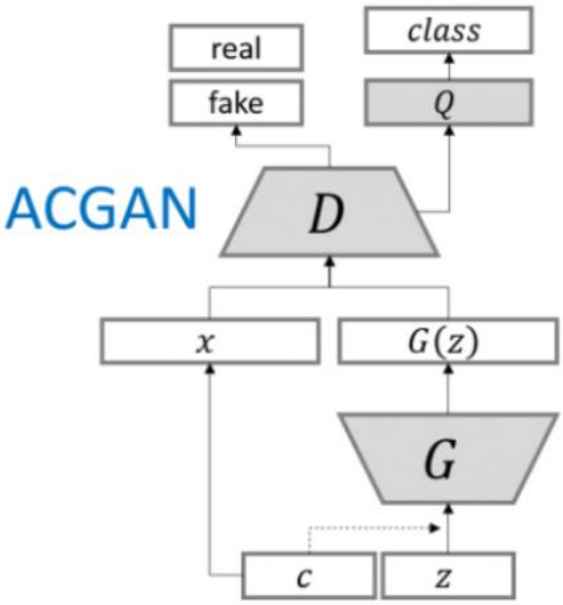


图 48-47 ACGAN 模型结构

(4) infoGAN

InfoGAN : Interpretable Representation Learning by Information Maximizing Generative Adversarial Networks。这个号称是 OpenAI 在 2016 年的五大突破之一。核心内容包括:

- D 网络的输入只有 x，不加 c
- Q 网络和 D 网络共享同一个网络，只是到最后一层独立输出
- G(z)的输出和条件 c 区别大

原文下载地址如下:

- https://arxiv.org/abs/1606.03657

InfoGAN 结构如下图所示。

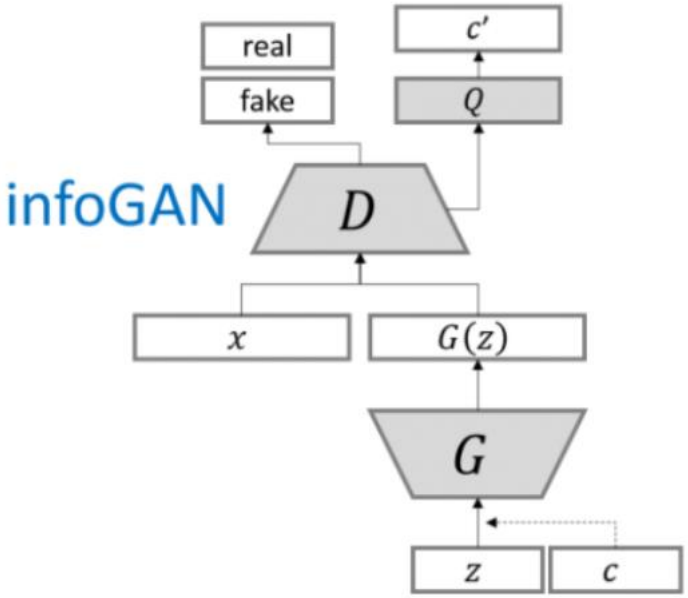


图 48-48 InfoGAN 模型结构

其理论如下：

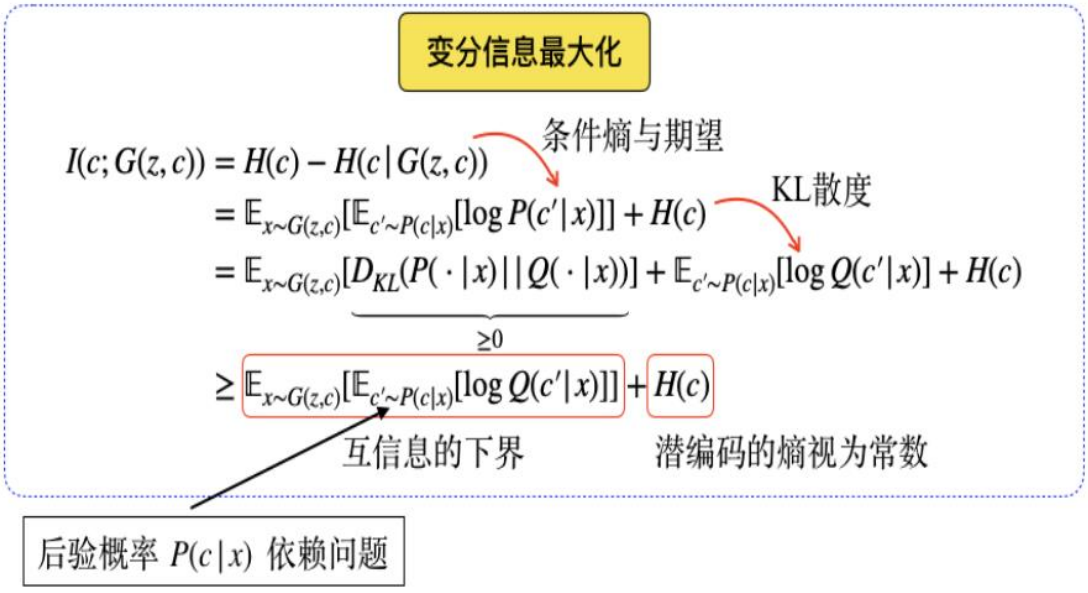
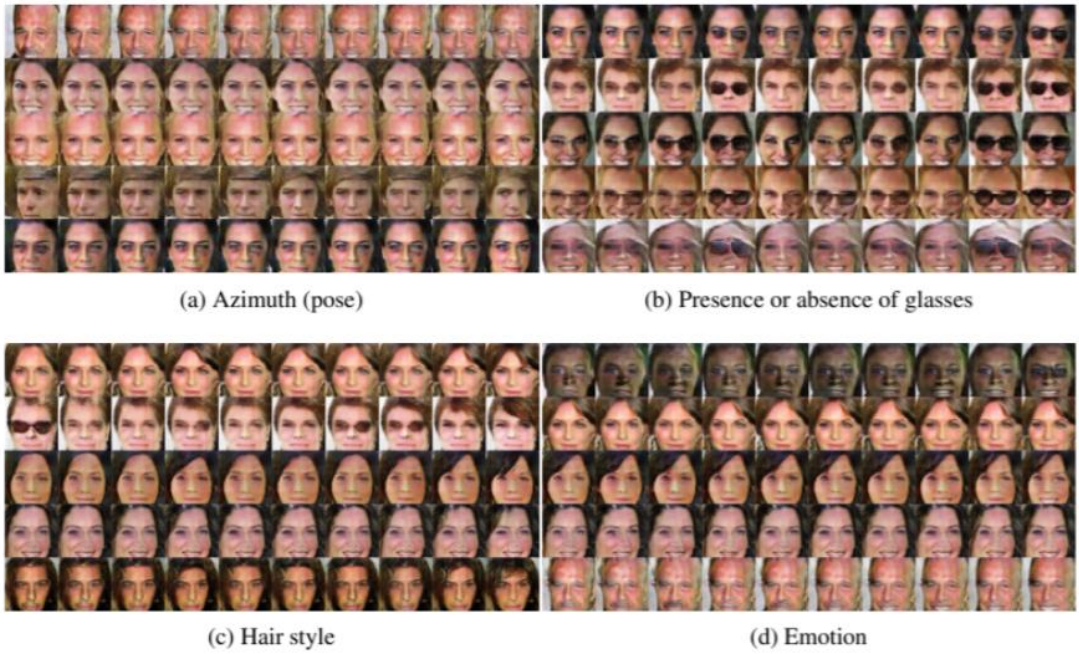


图 48-49 InfoGAN 理论知识

整个网络的训练在原目标函数的基础上，增加互信息下界 L(G,Q)，因此 InfoGAN 的目标函数最终表示为：

$$\min_{G,Q} \max_{D} V_{InfoGAN}(D,G) = V(D,G) - \lambda L_I(G,Q)$$

实验结果如图 48-50 所示。



(a) Azimuth (pose)　(b) Presence or absence of glasses

(c) Hair style　(d) Emotion

Figure 6: **Manipulating latent codes on CelebA:** (a) shows that a categorical code can capture the azimuth of face by discretizing this variation of continuous nature; in (b) a subset of the categorical code is devoted to signal the presence of glasses; (c) shows variation in hair style, roughly ordered from less hair to more hair; (d) shows change in emotion, roughly ordered from stern to happy.

图 48-50 InfoGAN 实验结果

(5) LAPGAN

下面介绍一个比较有趣的网络拉普拉斯 GAN。我们的目标是如何通过噪音生成一张图片，噪声本身生成图片比较困难，不可控量太多，所以我们逐层生成（生成从右往左看）。

- 首先用噪声去生成一个小的图片，分辨率极低，我们对其拉伸。
- 拉伸之后，想办法通过之前训练好的 GAN 网络生成一个它的残差。
- 残差和拉伸图相加就生成一张更大的图片，以此类推，拉普拉斯生成一

张大图。

那么，如何训练呢？对原来这个大图的鸟进行压缩，再生成一张图去判别，依次逐层训练即可。

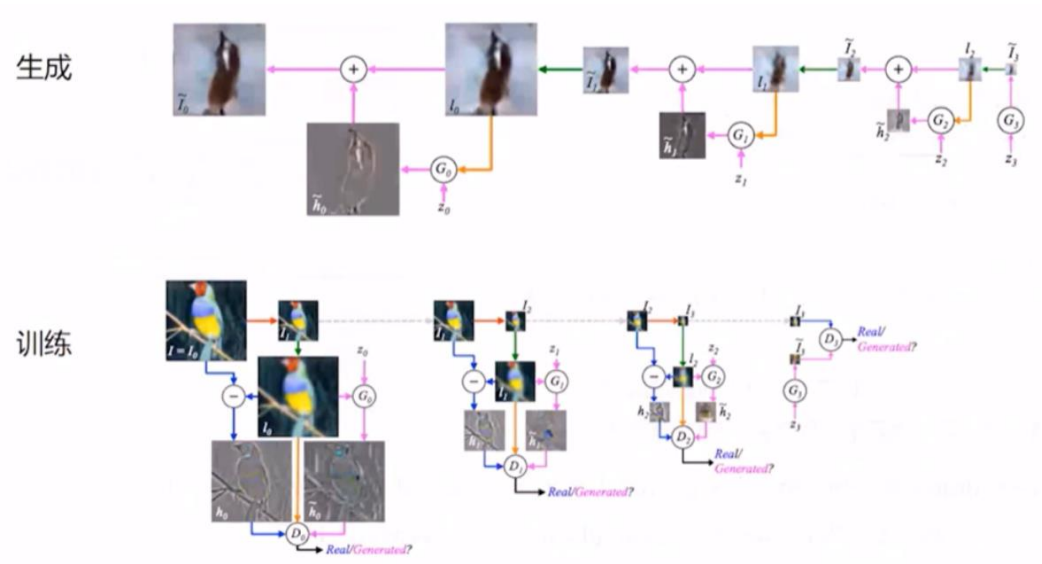


图 48-51 LAPGAN 模型结构

(6) EBGAN

再来看一个 EBGAN（Energy-based GAN），它抛弃了之前说的对和错的概念。它增加了一个叫能量的东西，经过自动编码器 Enc（中间提取特征）和 Dec 解码器（输出），它希望生成一个跟真实图片的能量尽可能小，跟假的图片能量更大。

- 《Energy-based Generative Adversarial Network》Junbo Zhao, arXiv:1609.03126v2

其结果如下图所示。

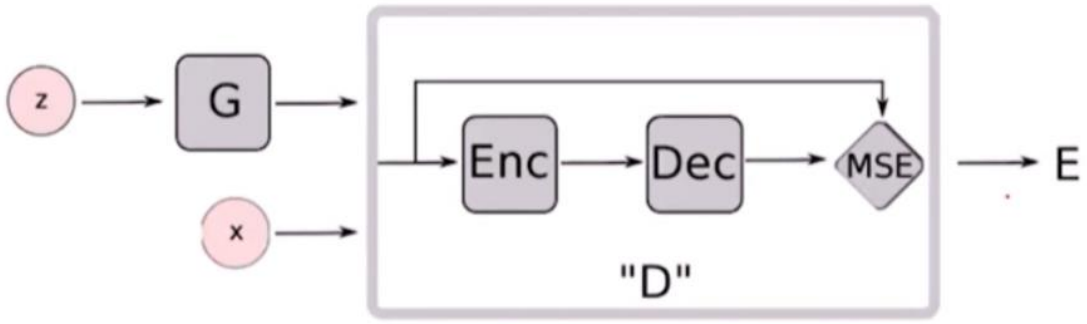


Figure 1: EBGAN architecture.

图 48-51 EBGAN 模型结构

其生成器和判别器的损失函数计算公式如下（分段函数）：

$$\mathcal{L}_D(x,z) = D(x) + [m - D(G(z))]^+$$
$$\mathcal{L}_G(z) = D(G(z))$$

下图展示了 GAN、EBGAN、EBGAN-PT 模型生成的图像。

Figure 4: Generation from the grid search on MNIST. Left(a): Best GAN model; Middle(b): Best EBGAN model. Right(c): Best EBGAN-PT model.

$$f_{PT}(S) = \frac{1}{N(N-1)} \sum_i \sum_{j \neq i} \left(\frac{S_i^\top S_j}{\|S_i\| \|S_j\|} \right)^2.$$

解决多样性问题

图 48-51 EBGAN 生成图片

6.GAN 改进策略

你以为解决了所有问题了吗？其实并没有。如下图所示误差，我们无法判断 GAN 训练的好坏。

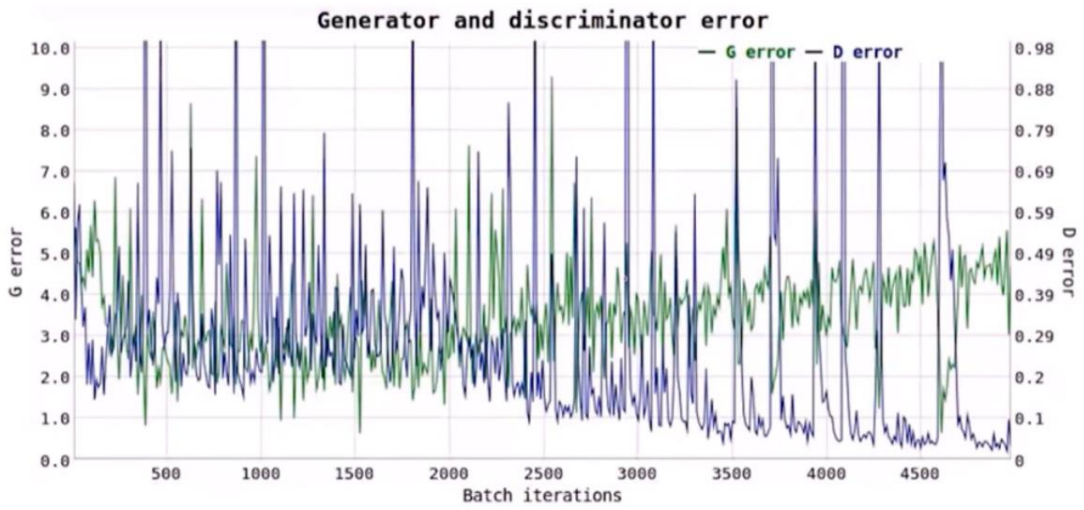


图 48-52 GAN 生成图片评价

GAN 需要重视三个知识点，现在很多工作也是解决这三个问题。

- 稳定（训练不崩）
- 多样性（各种样本）
- 清晰度（质量好）

同时，我们还需要讨论如下问题：

- G、D 迭代的方式能达到全局最优解吗？大部分情况是局部最优解。
- 不一定收敛，学习率不能高，G、D 要共同成长，不能某个成长过快
  - 判别器训练得太好，生成器梯度消失，生成器 loss 降不下去
  - 判别器训练得不好，生成器梯度不准，四处乱跑
- 奔溃的问题。通俗说 G 找到 D 的漏洞，每次都生成一样的骗 D
- 无需预先建模，模型过于自由不可控

为什么 GAN 存在这些问题，这是因为 GAN 原论文将 GAN 目标转换成了 KL 散度的问题，KL 散度就是存在这些坑。

$$
\begin{aligned}
C(G) &= \max_D V(G,D) \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right]
\end{aligned}
$$

Theorem 1. The global minimum of the virtual training criterion C(G) is achieved if and only if $p_g = p_{data}$. At that point, C(G) achieves the value log4.

$$
C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right)
$$

Generative Adversarial Nets

图 48-53 KL 散度问题

最终导致偏向于生成“稳妥”的样本，如下图所示，目标 target 是均匀分布的，但最终生成偏稳妥的样本。

- “生成器没能生成真实的样本”惩罚小
- “生成器生成不真实的样本”惩罚大

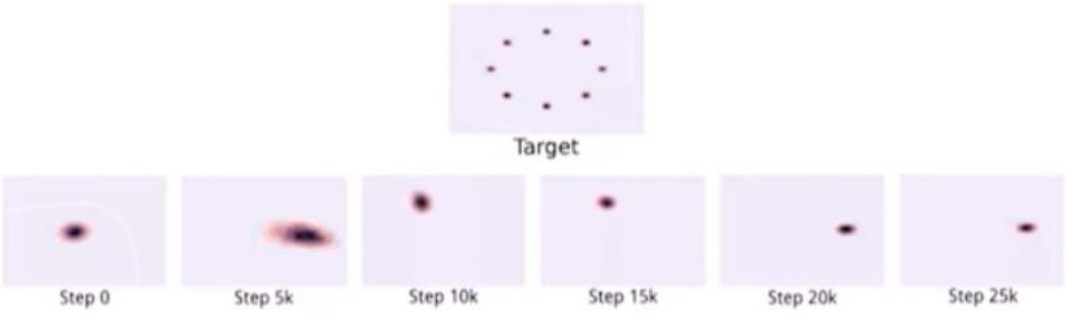


图 48-54 目标生成问题

那么，有没有解决方法呢?

WGAN（Wasserstein GAN）在 2017 年被提出，也算是 GAN 中里程碑式的论文，它从原理上解决了 GAN 的问题。具体思路为:

- 判别器最后一层去掉 sigmoid

- 生成器和判别器的 loss 不取 log
- 每次更新判别器的参数之后把它们的绝对值截断到不超过一个固定的常数 c
- 不要用基于动量的优化算法（包括 Momentum 和 Adam），推荐使用 RMSProp、SGD
- 用 Wasserstein 距离代替 KL 散度，训练网络稳定性大大增强，不用拘泥 DCGAN 的那些策略（tricks）

GAN $\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$.

WGAN $\max_{w\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z)]$

后续接着改进，提出了 WGAN-GP（WGAN with gradient penalty），不截断，只对梯度增加惩罚项生成质量更高的图像。它一度被称为“state of the art”。

WGAN_GP $L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2]}_{\text{Our gradient penalty}}$

接下来，做 GAN 的就会出来反驳“谁说 GAN 就不如 WGAN，我们加上 Gradient Penalty，大家效果都差不多”。

- https://arxiv.org/pdf/1705.07215.pdf

该篇论文的算法流程如下：

Algorithm 1 DRAGAN (Deep Regret Analytic Generative Adversarial Networks)

Initial weights (ϕ_0, θ_0)
for number of training iterations **do**
  Get a mini-batch of examples $\{x^1, x^2, ..., x^m\}$ from training data.
  Get a mini-batch of samples $\{z^1, z^2, ..., z^m\}$ from $\mathcal{N}_k(0, 1)$.
  Generate $\{x^1 + \delta^1, x^2 + \delta^2, ..., x^m + \delta^m\}$ where each δ^i is a small pixel-wise noise vector.
  Update the discriminator by descending along:

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \left[-\log D_\theta(x^i) - \log(1 - D_\theta(G_\phi(z^i))) + \lambda \cdot (||\nabla_{\mathbf{x}} D_\theta(x^i + \delta^i)||_2 - 1)^2 \right]$$

  Update the generator by descending along:

$$\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} \log \left[1 - D_\theta(G_\phi(z^i)) \right]$$

end for

效果如下图所示：

图 48-55 对比效果图

此外，《Google Brain: Are GANs Created Equal? A Large-Scale Study》这篇论文详细对比了各 GAN 模型点心 LOSS 优化变种。

- https://arxiv.org/pdf/1711.10337.pdf
- https://arxiv.org/pdf/1706.08500.pdf

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda \mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha)\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x} - 1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda \mathbb{E}_{\hat{x}\sim p_d + \mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|\|x - \mathrm{AE}(x)\|\|_1] - k_t \mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$

图 48-55 Loss 对比

这篇文章比较的结论为：

- 特定的数据集说特定的事情，没有哪一种碾压其他。好的算法还得看成本，时间短的效果某家强，但训练时间长了，反倒会变差。根据评价标准的不同，场景的不同，效果差的算法也可以逆袭。工业界更看重稳定性，比如 WGAN。

接下来，我们引用知乎苏剑林老师的回答进一步理解 GAN 改进策略。

首先，从理论完备的角度来看，原始的 GAN（SGAN）就是一个完整的 GAN 框架，只不过它可能存在梯度消失的风险。而论文比较的是 “大家都能稳定训练到收敛的情况下，谁的效果更好” 的问题，这答案是显然易见的：不管是 SGAN 还是 WGAN，大家都是理论完备的，只是从不同角度看待概率分布的问题而已，所以效果差不多是正常的。

甚至可以说，SGAN 的理论更完备一些（因为 WGAN 需要 L 约束，而目前 L 约束的各种加法都有各自的缺点），所以通常来说 SGAN 的效果还比 WGAN 效果好一些。那么 WGAN 它们的贡献是什么呢？WGAN 的特点就是基本上都能 “稳定训练到收敛”，而 SGAN 相对而言崩溃的概率更大。所以，如果在“大家都能稳定训练到收敛”的前提下比较效果，那对于 WGAN 这些模型本来就很不公平的，因为它们都是致力于怎么才能“稳定训练到收敛”，而这篇论文直接将它作为大前提，直接抹杀了 WGAN 所作的贡献了。

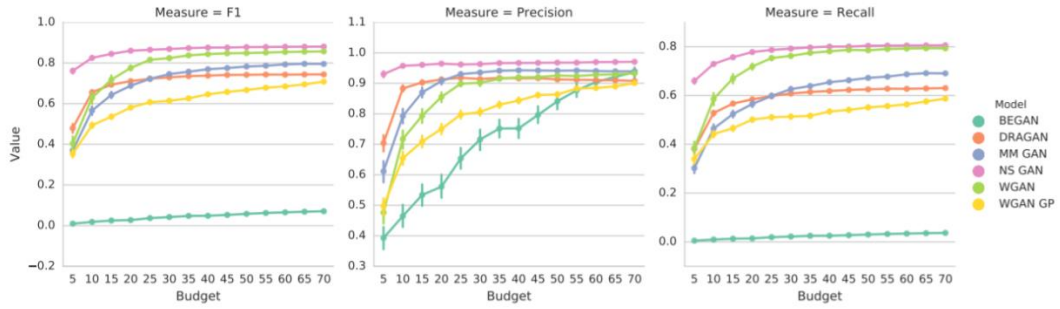


Figure 5: How does F_1 score vary with computational budget? The plot shows the distribution of the maximum F_1 score achievable for a fixed budget with a 95% confidence interval. For each budget, we estimate the mean and confidence interval (of the mean) using 5000 bootstrap resamples out of 100 runs. When optimizing for F_1 score, both NS GAN and WGAN enjoy high precision and recall. The underwhelming performance of BEGAN and VAE on this particular data set merits further investigation.

	MNIST	FASHION	CIFAR	CELEBA
MM GAN	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN	7.8 ± 0.6*	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8*
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	24.5 ± 2.1	55.8 ± 0.9	30.0 ± 1.0
DRAGAN	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	22.9 ± 0.9	71.4 ± 1.6	38.9 ± 0.9
VAE	23.8 ± 0.6	58.7 ± 1.2	155.7 ± 11.6	85.7 ± 3.8

图 48-56 各 GAN 模型对比效果

7.总结

写到这里，这篇文章就介绍结束了，希望对您有所帮助。首先非常感谢小象学院美图老师的介绍，文章虽然很冗余，但还是能学到知识，尤其是想学 GAN 的同学，这算一个非常不错的普及。当然，后续随着作者深入，会分享更简洁的介绍和案例。

个人感觉 GAN 有一部分很大的应用是在做强化学习，同时在推荐领域、对抗样本、安全领域均有应用，希望随着作者深入能分享更多的实战性 GAN 论文。比如如果图片被修改，GAN 能不能第一次时间反馈出来或优化判决器。最

后给出各类 GAN 模型对比图。

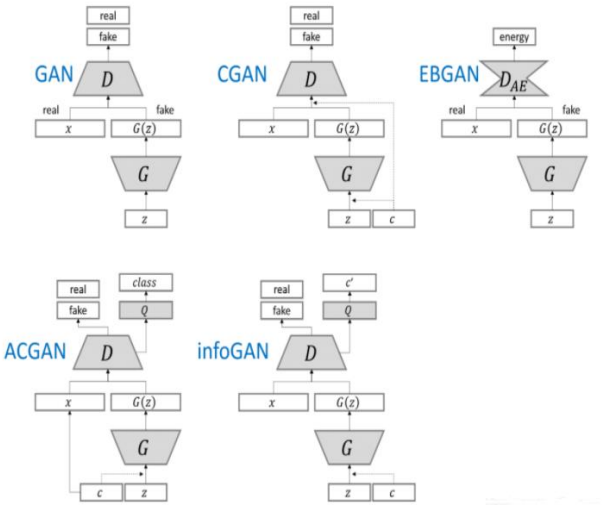


图 48-56 各 GAN 模型

这也是整本电子书的最后一篇文章，感谢华为云，感谢大家的阅读，未来也期待分享更多电子书与大家一起进步。且看且珍惜，也希望大家下来不断实践。感恩前行！最后，请大家记住一个名叫 Eastmount 的技术分享者，很高兴认识您。

参考文献：

[1] https://arxiv.org/abs/1406.2661

[2] https://www.bilibili.com/video/BV1ht411c79k

[3] https://www.cntofu.com/book/85/dl/gan/gan.md

[4] https://github.com/hindupuravinash/the-gan-zoo

[5] https://arxiv.org/pdf/1701.00160.pdf

[6] https://link.springer.com/chapter/10.1007/978-3-319-10593-2_13

[7] https://zhuanlan.zhihu.com/p/76520991

[8] http://cn.arxiv.org/pdf/1711.09020.pdf

[9] https://www.sohu.com/a/121189842_465975

[10] https://www.jianshu.com/p/88bb976ccbd9

[11] https://zhuanlan.zhihu.com/p/23270674

[12] ttps://blog.csdn.net/weixin_40170902/article/details/80092628

[13] https://www.jiqizhixin.com/articles/2016-11-21-4

[14] https://github.com/jacobgil/keras-dcgan/blob/master/dcgan.py

[15] https://arxiv.org/abs/1511.06434

[16] https://arxiv.org/pdf/1511.06434.pdf

[17] https://blog.csdn.net/weixin_41697507/article/details/87900133

[18] https://zhuanlan.zhihu.com/p/91592775

[19] https://liuxiaofei.com.cn/blog/acgan 与 cgan 的区别/

[20] https://arxiv.org/abs/1606.03657

[21] https://blog.csdn.net/sdnuwjw/article/details/83614977

[22]《 Energy-based Generative Adversarial Network 》Junbo Zhao, arXiv:1609.03126v2

[23] https://www.jiqizhixin.com/articles/2017-03-27-4

[24] https://zhuanlan.zhihu.com/p/25071913

[25] https://arxiv.org/pdf/1705.07215.pdf

[26] https://arxiv.org/pdf/1706.08500.pdf

[27] https://arxiv.org/pdf/1711.10337.pdf

[28] https://www.zhihu.com/question/263383926

HUAWEI

华为云开发者社区

更多精彩内容，扫码关注交流讨论

华为云开发者社区

华为云开发者社区微信公众号