图像处理

Pillow 是 Python Imaging Library 的简称，是 Python 语言中最为常用的图像处理库。Pillow 库提供了对 Python3 的支持，为 Python3 解释器提供了图像处理的功能。通过使用 Pillow 库，可以方便地使用 Python 程序对图片进行处理，例如常见的尺寸、格式、色彩、旋转等处理。

Pillow 库的安装

Pillow 库是 Python 开发者最为常见的图像处理库，它提供了广泛的文件格式支持、强大的图像处理能力，主要包括图像存储、图像显示、格式转换以及基本的图像处理操作等。安装 Pillow 库的方法与安装 Python 其他第三方库的方法相同，也可以到 Python 官方网站下载 Pillow 库的压缩包。

1.pip 安装 pillow，执行如下命令：

```
pip install pillow
```

图象处理基本知识

图像的 RGB 色彩模式

RGB 三个颜色通道的变化和叠加得到各种颜色，其中

- R 红色，取值范围，0-255
- G 绿色，取值范围，0-255
- B 蓝色，取值范围，0-255

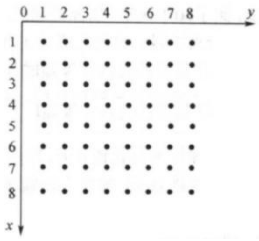
比如，我们常见的黄色就是由红色和绿色叠加而来。

红色的 RGB 表示（255,0,0）　　绿色的 RGB 表示（0,255,0）

蓝色的 RGB 表示（0,0,255）　　黄色的 RGB 表示（255,255,0）

像素阵列

数字图像可以看成一个整数阵列，阵列中的元素称为像素(Pixel),见下图的数字阵列



每个点代表 1 个像素(Pixel)，一个点包含 RGB 三种颜色。也就是 1 个像素包含 3 个字节的信息：（R,G,B）。假如这个像素是红色，则信息是：（255,0,0）。那么，理论上我们只要操作每个点上的这三个数字，就能实现任何的图形。一幅图像上的所有像素点的信息就完全可以采用矩阵来表示，通过矩阵的运算实现更加复杂的操作。

Image 模块

打开和新建

在 Pillow 库中，通过使用 Image 模块，可以从文件中加载图像，或者处理其他图像，或者从 scratch 中创建图像。在对图像进行处理时，首先需要打开要处理的图片。在 Image 模块中使用函数 open()打开一副图片，执行后返回 Image 类的实例。当文件不存在时，会引发 IOError 错误。使用函数 open()语法格式如下所示。

```
open(fp,mode)
```

(1) fp:指打开文件的路径。

(2) mode：可选参数，表示打开文件的方式，通常使用默认值 r。

在 Image 模块中，可以使用函数 new()新建图像。具体语法格式如下所示：

```
new(mode,size,color=0)
```

(1) mode:图片模式，具体取值如下表

(2) size：表示图片尺寸，是使用宽和高两个元素构成的元组

(3) color：默认颜色（黑色）

Pillow 库支持的常用图片模式信息

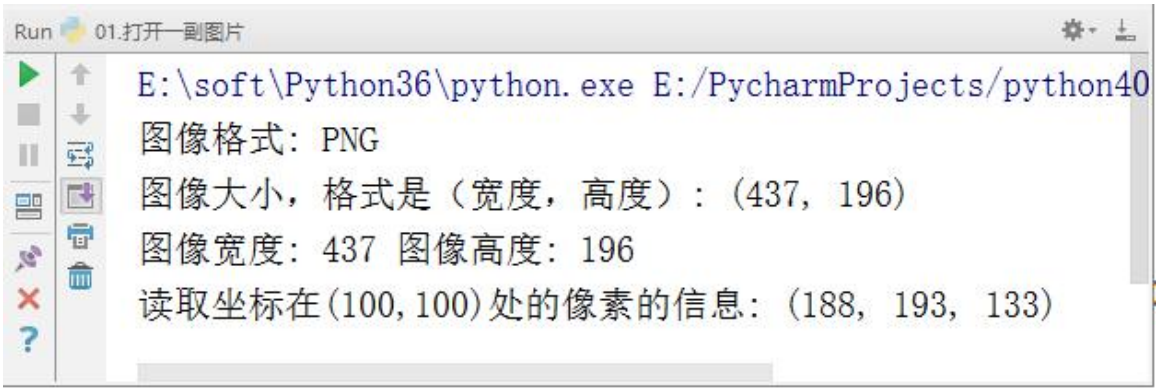
mode(模式)	bands（通道）	说明
“1”	1	数字 1，表示黑白二值图像，每个像素用 0 或者 1 共 1 位二进制代码表示
“L”	1	灰度图，每个像素用 8 位二进制代码表示
“P”	1	索引图，每个像素用 8 位二进制代码表示
“RGB”	3	24 位真彩图，每个像素用 3 个字节的二进制代码表示
“RGBA”	4	“RGB”+透明通道表示，每个像素用 4 字节的二进制代码表示
“CMYK”	4	印刷模式图像，每个像素用 4 字节的二进制代码表示
“YCbCr”	3	彩色视频颜色隔离模式，每个像素用 3 个字节的二进制代码表示
“LAB”	3	lab 颜色空间，每个像素用 3 字节的二进制代码表示
“HSV”	3	每个像素用 3 字节的二进制代码表示
“I”	1	使用整数形式表示像素，每个像素用 4 字节的二进制代码表示
“F”	1	使用浮点数形式表示像素，每个像素用 4 字节的二进制代码表示

【示例】使用 Image 打开一副图片

```
#导入 Image 模块
from PIL import Image
#打开图片
im=Image.open('bjsxt.png')
```

```
#显示图片
im.show()
#查看图片的信息
print('图像格式:',im.format)
print('图像大小，格式是（宽度，高度）:',im.size)
print('图像宽度:',im.width,'图像高度:',im.height)
print('读取坐标在(100,100)处的像素的信息:',im.getpixel((100,100)))
```

运行结果如下图：



混合

（1） 透明度混合处理

在 Pillow 库的 Image 模块中，可以使用函数 blend()实现透明度混合处理。具体语法格式如下所示：

```
blend(im1,im2,alpha)
```

其中 im1、im2 指参与混合的图片 1 和图片 2，alpha 指混合透明度，取值是 0-1。

通过使用函数 blend()，可以将 im1 和 im2 这两幅图片（尺寸相同）以一定的透明度进行混合。具体混合过程如下：

```
(im1*(1-alpha)+im2*alpha)
```

当混合透明度为 0 时，显示 im1 原图。当混合透明度 alpha 取值为 1 时，显示 im2 原图片。

【示例】透明度混合图片

```
from PIL import Image
img1=Image.open('bjsxt.jpg').convert(mode="RGB")
img2=Image.new("RGB",img1.size,"red")
# #混合两幅图
Image.blend(img1,img2,alpha=0.5).show()
```

执行结果如下图：

（2） 遮罩混合处理

在 Pillow 库中 Image 模块中，可以使用函数 composite()实现遮罩混合处理。具体语法格式如下所示：

```
composite(im1,im2,mask)
```

其中 im1 和 im2 表示混合处理的图片 1 和图片 2.mask 也是一个图像，mode 可以为“1”，“L”, or “RGBA”，并且大小要和 im1、im2 一样。

函数 composite()的功能是使用 mask 来混合图片 im1 和 im2，并且要求 mask、im1 和 im2 三幅图片的尺寸相同。下面的实例代码演示了使用 Image 模块实现图片遮罩混合处理的过程。

【示例】遮罩混合图片

```
from PIL import Image
img1=Image.open('img1.jpg')
img2=Image.open('img2.jpg')
img2=img2.resize(img1.size)
r,g,b=img2.split()
Image.composite(img2,img1,b).show()
```

执行结果如下图：



复制和缩放

（1） 复制图像

在 Pillow 库的 Image 模块中，可以使用函数 Image.copy()复制指定的图片，这可以用于在处理或粘贴时需要持有源图片。

（2） 缩放像素

在 Pillow 库的 Image 模块中，可以使用函数 eval()实现像素缩放处理，能够使用函数 fun()计算输入图片的每个像素并返回。使用函数 eval()语法格式如下：

```
eval(image,fun)
```

其中 image 表示输入的图片，fun 表示给输入图片的每个像素应用此函数，fun()函数只允许接收一个整型参数。如果一个图片含有多个通道，则每个通道都会应用这个函数。

【示例】缩放指定的图片，实现图像每个像素值×2

```
from PIL import Image
img=Image.open('img2.jpg')
Image.eval(img,lambda i:i*2).show()
```

（3） 缩放图像

在 Pillow 库的 Image 模块中，可以使用函数 thumbnail()原生地缩放指定的图像 。具体语法格式如下：

```
Image.thmbnail(size,resample=3)
```

【示例】缩放成指定的大小

```
from PIL import Image
img=Image.open('img2.jpg')
imgb=img.copy()
#缩放为指定大小(220,168)
imgb.thumbnail((220,168))
imgb.show()
```

粘贴和裁剪

（1） 粘贴

在 Pillow 库的 Image 模块中，函数 paste()的功能是粘贴源图像或像素至该图像中。具体语法格式如下：

```
Image.paste(im,box=None,mask=None)
```

其中 im 是源图或像素值；box 是粘贴的区域；mask 是遮罩。参数 box 可以分为以下 3 中情况。

1.（x1,y1）:将源图像左上角对齐（x1,y1）点，其余超出被粘贴图像的区域被抛弃。

2.（x1,y1,x2,y2）:源图像与此区域必须一致。

3.None：源图像与被粘贴的图像大小必须一致。

（2） 裁剪图像

在 Pillow 库的 Image 模块中，函数 crop()的功能是剪切图片中 box 所指定的区域，具体语法如下：

```
Image.crop(box=None)
```

参数 box 是一个四元组，分别定义了剪切区域的左、上、右、下 4 个坐标。

【示例】对指定图片剪切和粘贴操作

```
from PIL import Image
img=Image.open('bjsxt.png')
#复制图片
imgb=img.copy()
imgc=img.copy()
#剪切图片
region=imgb.crop((5,5,120,120))
#粘贴图片
imgc.paste(region,(30,30))
imgc.show()
```

图像旋转

在 Pillow 库的 Image 模块中，函数 rotate()的功能返回此图像的副本，围绕其中心逆时针旋转给定的度数。具体语法格式如下：

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

【示例】函数 rotate()实现图像旋转

```
from PIL import Image
img=Image.open('bjsxt.png')
img.rotate(90).show()
```

格式转换

（1） covert()

在 Pillow 库的 Image 模块中，函数 convert()的功能是返回模式转换后的图像实例。具体转换的语法格式如下：

```
Image.convert(mode=None,matrix=None,dither=None,palette=0,colors=256)
```

其中 mode：转换文件的模式，默契支持的模式有“L”、“RGB”“CMYK”；matrix：转使用的矩阵；dither：取值为 None 切转为黑白图时非 0（1-255）像素均为白，也可以设置此参数为 FLOYDSTEINBERG。

（2） transpose()

在 Pillow 库的 Image 模块中，函数 transpose()函数功能是实现图像格式的转换。具体语法格式如下：

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

转换图像后，返回转换后的图像，“method”的取值有以下几个。

1. PIL.Image.FLIP_LEFT_RIGHT：左右镜像

2. PIL.Image.FLIP_TOP_BOTTOM ：上下镜像
3. PIL.Image.ROTATE_90：旋转 90
4. PIL.Image.ROTATE_180：旋转 180
5. PIL.Image.ROTATE_270：旋转 270
6. PIL.Image.TRANSPOSE :颠倒顺序

【示例】对指定图片进行转换操作

```
from PIL import Image
#打开指定的图片
img1=Image.open('bjsxt.png')
img2=img1.copy()
#convert()
img_convert=img2.convert('CMYK')
# img_convert.show()
#transpose()
img_transpose=img2.transpose(Image.ROTATE_90)
img_transpose.show()
```

分离和合并

（1） 分离

在 Pillow 库的 Image 模块中，使用函数 split()可以将图片分割为多个通道列表。使用函数 split()的语法格式如下所示：

```
Image.split()
```

（2） 合并

在 Pillow 库的 Image 模块中，使用函数 merge()可以将一个通道的图像合并到更多通道图像中。使用函数 merge()的语法格式如下所示：

```
Image.merge(mode,bands)
```

其中 mode 指输出图像的模式，bands 波段通道，一个序列包含单个带图通道。

【示例】对指定图片进行合并和分离操作

```
from PIL import Image
img1=Image.open('blend1.jpg')
img2=Image.open('blend2.jpg')
img2=img2.resize(img1.size)
r1,g1,b1= img1.split()
r2,g2,b2= img2.split()
```

```
tmp=[r1,g2,b1]
img = Image.merge("RGB",tmp)
img.show()
```

滤镜

在 Pillow 库中的 Image 模块中，使用函数 filter()可以对指定的图片使用滤镜效果，在 Pillow 库中可以用的滤镜保存在 ImageFilter 模块中。使用函数 filter()的语法格式如下所示：

```
Image.filter(filter)
```

通过函数 filter()，可以使用给定的滤镜过虑指定的图像，参数“filter”表示滤镜内核。

【示例】对指定图片实现滤镜模糊操作

```
from PIL import Image,ImageFilter
#使用函数 filter()实现滤镜效果
img=Image.open('bjsxt.png')
b=img.filter(ImageFilter.GaussianBlur)
b.show()
```

其他内置函数

在 Pillow 库的 Image 模块中，还有很多其他重要的内置函数和属性。

常用的属性：

1. Image.format：源图像格式
2. Image.mode:图像模式字符串
3. Image.size:图像尺寸

在 Pillow 库的 Image 模块中，其他常用的内置函数如下所示：

1. Image.getbands()：获取图像每个通道的名称列表，例如 RGB 图像返回[‘R’,’G’,’B’]。
2. Image.getextrema():获取图像最大、最小像素的值。
3. Image.getpixel(xy):获取像素点值。
4. Image.histogram(mask=None,extrema=None):获取图像直方图，返回像素计数的列表。
5. Image.point(function):使用函数修改图像的每个像素。
6. Image.putalpha(alpha):添加或替换图像的 alpha 层。
7. Image.save(fp,format=None,**params):保存图片。
8. Image.show(title=None,command=None):显示图片。
9. Image.transform(size,method,data=None,resample=0,fill=1):变换图像。
10. Image.verify():校验文件是否损坏。
11. Image.close():关闭文件。

ImageFilter 模块

内置模块 ImageFilter 实现了滤镜功能，可以用来创建图像特效，或以此效果作为媒介实现进一步处理。

在模块 ImageFilter 中，提供了一些预定义的滤镜和自定义滤镜函数。其中最为常用的预定义滤镜如下所示：

BLUE:模糊

CONTOUR:轮廓

DETAIL:详情

EDGE_ENHANCE：边缘增强

EDGE_ENHANCE_MORE:边缘更多增强

EMBOSS:浮雕

FIND_EDGES:寻找边缘

SHARPEN:锐化

SMOOTH:平滑

在模块 ImageFilter 中，常用的自定义滤镜函数如下所示：

函数名	功能
ImageFilter.GaussianBlur（radius = 2 ）	高斯模糊
ImageFilter.UnsharpMask（radius = 2，percent = 150，threshold = 3 ）	不清晰的掩模滤镜
ImageFilter.MinFilter（size = 3 ）	最小值滤波
ImageFilter.MedianFilter（size = 3 ）	中值滤波
ImageFilter.ModeFilter（size = 3 ）	模式滤波

【示例】使用 ImageFilter 对指定图片实现滤镜特效

```
from PIL import Image,ImageFilter
#打开图片
imga=Image.open('img2.jpg')
w,h=imga.size
#创建图像区域
img_output=Image.new('RGB',(2*w,h))
#将创建的部分粘贴图片
img_output.paste(imga,(0,0))
#创建列表存储滤镜
fltrs=[]
fltrs.append(ImageFilter.EDGE_ENHANCE)#边缘强化滤镜
```

```
fltrs.append(ImageFilter.FIND_EDGES) #查找边缘滤镜
fltrs.append(ImageFilter.GaussianBlur)#高斯模糊滤镜
for fltr in fltrs:
    r=imga.filter(fltr)
    img_output.paste(r,(w,0))
    img_output.show()
```

ImageChops 模块

在 Pillow 库的内置模块 ImageChops 中包含了多个用于实现图片合成的函数。这些合成功能是通过计算通道中像素值的方式来实现的。其主要用于制作特效、合成图片等操作。常用的内置函数如下所示：

（1）相加函数 add()，功能是对两张图片进行算术加法运算。具体语法如下所示：

```
ImageChops.add（image1，image2，scale = 1.0，offset = 0 ）
```

在合成后图像中的每个像素值，是两幅图像对应像素值依据下面的公式进行计算得到的。

```
out = ((image1 + image2) / scale + offset)
```

【示例】使用 add 图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
#对两张图片进行算术加法运算
ImageChops.add(imga,imgb,1,0).show()
```

（2）减法函数 subtract()，功能是对两张图片进行算术减法运算。具体语法如下所示：

```
ImageChops.subtract（image1，image2，scale = 1.0，offset = 0 ）
```

在合成后图像中的每个像素值，是两幅图像对应像素值依据下面的公式进行得到的。

```
out = ((image1 - image2) / scale + offset)
```

【示例】使用 subtract()图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
```

```
#对两张图片进行减法运算
ImageChops.subtract(imga,imgb,1,0).show()
```

（3）变暗函数 darker()，功能是比较两个图片的像素，取两张图片中对应像素的较小值，所以合成时两幅图像中对应位置的暗部分得到保留，而去除亮部分。具体语法如下所示：

```
ImageChops.darker（image1，image2 ）
```

像素的计算公式如下所示：

```
out = min(image1, image2)
```

【示例】使用 darker()图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
#使用变暗函数 darker()
ImageChops.darker(imga,imgb).show()
```

（4）变亮函数 lighter()，与变暗函数 darker()相反，功能是比较两个图片（逐像素比较），返回一幅新的图片，这幅新的图片是将两张图片中较亮的部分叠加得到的。也就是说，在某一点上，两张图中哪个的值大（亮）则取之。具体语法如下所示：

```
ImageChops.lighter（image1，image2 ）
```

函数 lighter()与函数 darker()的功能相反，计算后得到的图像是两幅图像对应位置的亮部分。像素的计算公式如下所示：

```
out = max(image1, image2)
```

【示例】使用 lighter()图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
#使用变亮函数 lighter()
ImageChops.lighter(imga,imgb).show()
```

（5）叠加函数 multiply(),功能是将两张图片互相叠加。如果用纯黑色与某图片进行叠加操作，就会得到一幅纯黑色的图片。如果用纯白色与图片作叠加，则图片不受影响。具体语法

如下所示：

```
ImageChops.multiply（image1，image2 ）
```

合成的图像的效果类似两张图片在透明的描图纸上叠放在一起观看的效果。其对应像素的计算公式如下所示：

```
out = image1 * image2 / MAX
```

【示例】使用 multiply()图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
#将两张图片相互叠加
ImageChops.multiply(imga,imgb).show()
```

（6）屏幕函数 screen()，功能是先反色后叠加，实现合成图像的效果，就像将两张幻灯片用两台投影机同时投影到一个屏幕上的效果。具体语法如下所示：

```
ImageChops.screen（image1，image2 ）
```

其对应像素的计算公式如下所示：

```
out = MAX - ((MAX - image1) * (MAX - image2) / MAX)
```

【示例】使用 screen()图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
#实现反色后叠加
# ImageChops.screen(imga,imgb).show()
```

（7）反色函数 invert()，类似于集合操作中的求补集，最大值为 Max，每个像素做减法，取出反色。在反色时将用 255 减去一幅图像的每个像素值，从而得到原来图像的反相。也就是说，其表现为“底片”性质的图像。具体语法如下所示：

```
ImageChops.invert（image）
```

其对应像素的计算公式如下所示：

```
out = MAX - image
```

【示例】使用 invert()图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
#使用反色函数 invert()
ImageChops.invert(imga).show()
```

（8）比较函数 difference()，可以逐像素做减法操作，计算出绝对值。函数 difference()能够两幅图像的对应像素值相减后的图像，对应像素值相同的，则为黑色。函数 difference()通常用来找出图像之间的差异。具体语法如下所示：

```
ImageChops.difference（image1，image2 ）
```

其对应像素的计算公式如下所示：

```
out = abs(image1 - image2)
```

【示例】使用 difference 图片合成

```
from PIL import Image
from PIL import ImageChops
#打开图片
imga=Image.open('blend1.jpg')
imgb=Image.open('blend2.jpg')
#使用比较函数 difference()
ImageChops.difference(imga,imgb).show()
```

ImageEnhance 模块

内置的 ImageEnhance 模块中包含了多个用于增强图像效果的函数，主要用来调整图像的色彩、对比度、亮度和清晰度等，感觉上和调整电视机的显示参数一样。

在模块 ImageEnhance 中，所有的图片增强对象都实现一个通用的接口。这个接口只包含如下一个方法。

方法 enhance()会返回一个增强的 Image 对象，参数 factor 是一个大于 0 的浮点数，1 表示返回原始图片。

当在 Python 程序中使用模块 ImageEnhance 增强图像效果时，需要首先创建对应的增强调整器，然后调用调整器输出函数，根据指定的增强系数（小于 1 表示减弱，大于 1 表示增强，等于 1 表示原图不变）进行调整，最后输出调整后的图像。

在模块 ImageEnhance 中，常用的内置函数如下所示：

（1）ImageEnhance.Color（image ）：功能是调整图像色彩平衡，相当于彩色电视机的

色彩调整，实现了上边提到的接口的 enhance 方法。

（2）ImageEnhance.Contrast（image ）：功能是调整图像对比度，相当于彩色电视机的对比度调整。

（3）ImageEnhance.Brightness（image ）：功能是调整图像亮度。

（4）ImageEnhance.Sharpness（image ）：功能是调整图像清晰度，用于锐化/钝化图片。

锐化操作的 factor 是 0~2 之间的一个浮点数。当 factor=0 时，返回一个模糊的图片对象；当 factor=2 时，返回一个锐化的图片对象；当 factor=1 时，返回原始图片对象。

【示例】使用 ImageEnhance 实现图像色彩平衡

```
from PIL import Image
from PIL import ImageChops,ImageEnhance
#打开图片
imga=Image.open('blend1.jpg')
w,h=imga.size
#创建图像区域
img_output=Image.new('RGB',(2*w,h))
#将创建的部分粘贴图片
img_output.paste(imga,(0,0))
#调整图像色彩平衡
nhc=ImageEnhance.Color(imga)
for ratio in [0.6,1.8]:#减弱和增强两个系数
    b=nhc.enhance(ratio)#增强处理
    img_output.paste(b,(w,0))#粘贴修改后的图像
    img_output.show()
```

执行结果：

【示例】使用 ImageEnhance 实现图像亮度

```
from PIL import Image
from PIL import ImageChops,ImageEnhance
#打开图片
imga=Image.open('blend1.jpg')
w,h=imga.size
#创建图像区域
img_output=Image.new('RGB',(2*w,h))
#将创建的部分粘贴图片
img_output.paste(imga,(0,0))
#调整图像的亮度
nhb=ImageEnhance.Brightness(imga)
for ratio in [0.6,1.8]:#减弱和增强两个系数
    b=nhb.enhance(ratio)#增强处理
    img_output.paste(b,(w,0))#粘贴修改后的图像
    img_output.show()
```

【示例】使用图像点运算实现图像整体变暗、变亮

```
from PIL import Image
#打开图片
imga=Image.open('blend1.jpg')
w,h=imga.size
#创建图像区域
img_output=Image.new('RGB',(3*w,h))
#将创建的部分粘贴图片
img_output.paste(imga,(0,0))
imgb=imga.point(lambda i:i*1.3)
img_output.paste(imgb,(w,0))
imgc=imga.point(lambda i:i*0.4)
img_output.paste(imgc,(2*w,0))
img_output.show()
```

执行结果如下图：

ImageDraw 模块

ImageDraw 模块实现了绘图功能。可以通过创建图片的方式来绘制 2D 图像；还可以在原有的图片上进行绘图，已达到修饰图片或对图片进行注释的目的。

在 ImageDraw 模块绘图时需要首先创建一个 ImageDraw.Draw 对象，并且提供指向文件的参数。然后引用创建的 Draw 对象方法进行绘图。最后保存或直接输出绘制的图像。

```
    drawObject=ImageDraw.Draw(black)
```

（1）绘制直线

```
drawObject.line（[x1,y1,x2,y2]，fill = None，width = 0，joint = None ）
```

表示以（x1,y1）为起始点，以（x2,y2）为终止点画一条直线。[x1,y1,x2,y2]也可以写为(x1,y1,x2,y2)、[(x1,y1),(x2,y2)]等；fill 用于设置指定线条颜色；width 设置线条的宽度；joint 表示一系列线之间的联合类型。它可以是“曲线”。

（2）绘制圆弧

```
drawObject.arc（[x1,y1,x2,y2],start,end,fill = None,width = 0 ）
```

在左上角坐标为(x1,y1)，右下角坐标为（x2,y2）的矩形区域内，满圆 O 内，以 start 为起始角度，以 end 为终止角度，截取圆 O 的一部分圆弧并画出来。如果[x1,y1,x2,y2]区域不是正方形，则在该区域内的最大椭圆中根据角度截取片段。参数 fill 和 width 与 line 方法相同。

（3）绘制椭圆

```
drawObject.ellipse（[x1,y1,x2,y2],fill = None，outline = None，width = 0 ）
```

用法同 arc 类似，用于画圆（或者椭圆）。outline 表示只规定圆的颜色。

（4）绘制弦

```
drawObject.chord（[x1,y1,x2,y2],start，end，fill = None，outline = None，width = 0 ）
```

用法同 arc 类似，用于画圆中从 start 到 end 的弦。fill 表示弦与圆弧之间空间用指定颜色填满，设置为 outline 表示只规定弦线的颜色。

（5）绘制扇形

```
drawObject.pieslice（[x1,y1,x2,y2],start，end，fill = None，outline = None，width = 0 ）
```

用法同 elipse 类似，用于画起止角度间的扇形区域。fill 表示将扇形区域用指定颜色填满，设置为 outline 表示只用指定颜色描出区域轮廓。

（6）绘制多边形

```
drawObject.polygon（[x1,y1,x2,y2,....]，fill = None，outline = None ）
```

根据坐标画多边形，Python 会根据第一个参量中的（x,y）坐标对，连接出整个图形。fill 表示将多边形区域用指定颜色填满，outline 只用于设置指定颜色描出区域轮廓。

（7）绘制矩形

```
drawObject.rectangle（[x1,y1,x2,y2]，fill = None，outline = None，width = 0 ）
```

在指定的区域内画一个矩形，（x1,y1）表示矩形左上角的坐标，（x2,y2）表示矩形右下角的坐标。fill 用于将矩形区域颜色填满，outline 用于描出区域轮廓。

（8）绘制文字

```
drawObject.text（position，text，fill = None，font = None，anchor = None，spacing = 0，
align =“left”，direction = None，features = None，language = None ）
```

在图像内添加文字。其中参数 position 是一个二元组，用于指定文字左上角的坐标；text 表示要写入的文字内容；fill 表示文本的颜色；font 必须为 ImageFont 中指定的 font 类型；spacing 表示行之间的像素数；align 表示位置“left”，“center”或“right”；direction 表

示文字的方向。它可以是'rtl'（从右到左），'ltr'（从左到右）或'ttb'（从上到下）。

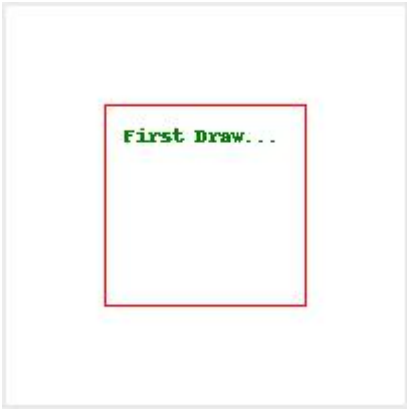
（9）绘制点

```
drawObject.point（xy，fill = None ）
```

给定坐标处绘制点（单个像素）。

【示例】创建图片的方式来绘制

```
from PIL import Image,ImageDraw
a=Image.new('RGB',(200,200),'white')
#新建一幅白色背景的图像
drw=ImageDraw.Draw(a)
drw.rectangle((50,50,150,150),outline='red')
drw.text((60,60),'First Draw...',fill='green')
a.show()
```



【示例】在原图片上绘制

```
from PIL import Image, ImageDraw
img = Image.open("lena.jpg")
draw = ImageDraw.Draw(img)
width, height = img.size
draw.arc( (0, 0, width-1, height-1), 0, 360, fill='blue')
img.save("circle.jpg")
```

ImageFont 模块

ImageFont 的功能是实现对字体和字型的处理。比较常用的内置函数如下所示：

（1）load()：从指定的文件中加载一种字体，该函数返回对应的字体对象。如果该函数运行失败，那么将产生 IOError 异常。语法格式如下：

```
ImageFont.load（文件名）
```

（2）load_path()：和函数 load()一样，但是如果没有指定当前路径，就会从文件 sys.path 开始查找指定的字体文件。语法格式如下：

```
ImageFont.load_path（文件名）
```

（3）truetype()：有两种定义格式。第 1 种格式的功能是加载一个 TrueType 或者 OpenType 字体文件，并且创建一个字体对象。在 Windows 系统中，如果指定的文件不存在，加载器就会顺便看看 Windows 的字体目录下它是否存在。语法格式如下：

```
ImageFont.truetype（file,size）
```

第 2 种格式的功能是，加载一个 TrueType 或者 OpenType 字体文件，并且创建一个字体对象。通常的编码方式是“unic”（Unicode）、“symb”（MicrosoftSymbol）、“ADOB”（Adobe Standard）、“ADBE”(Adobe Expert)和“armn”（Apple Roman）。语法格式如下：

```
ImageFont.truetype（file,size,encoding=value）
```

（4）load_default()：功能是加载一种默认的字体。

```
ImageFont.load_default（）
```

（5）getsize()：返回给定文本的宽度和高度，返回值是一个二元组。具体语法格式如下：

```
ImageFont.getsize（）
```

【示例】在原图片上绘制

```
from PIL import Image,ImageDraw,ImageFont
im = Image.open("bjsxt.png")
draw = ImageDraw.Draw(im)
ft=ImageFont.truetype('SIMYOU.TTF',16)
draw.text((30,30),u'图像处理库 PIL',font=ft,fill='red')
ft=ImageFont.truetype('C:\\Windows\\Fonts\\SIMLI.TTF',20)
draw.text((30,80),u'图像处理库 PIL',font=ft,fill='blue')
ft=ImageFont.truetype('C:\\Windows\\Fonts\\STXINGKA.TTF',30)
draw.text((30,130),u'图像处理库 PIL',font=ft,fill='green')
im.show()
```

操作示例

【示例】绘制十字

```
from PIL import Image, ImageDraw
im = Image.open("img1.jpg")
draw = ImageDraw.Draw(im)
draw.line((0, 0) + im.size, fill=128,width=5)
draw.line((0, im.size[1], im.size[0], 0), fill=128,width=5)
im.show()
```

执行结果：

【示例】绘制验证码

```
from PIL import Image, ImageFilter, ImageFont, ImageDraw
import random
width=100
height=100
#最后一个参数是背景颜色，像素默认值
im = Image.new("RGB",(width,height),(255,255,255))
draw = ImageDraw.Draw(im)

#获取颜色
def get_color1():
    return (random.randint(200, 255), random.randint(200, 255), random.randint(200, 255))

# 获取一个字母或数字
def get_char():
    return chr(random.randint(65,90))

#填充每个像素
```

```
for x in range(width):
    for y in range(height):
        draw.point((x,y),fill=get_color1())

font = ImageFont.truetype('simsun.ttc', 36)
for i in range(4):
    draw.text((10+i*20,50),get_char(),font=font,fill=(255,0,0))

#干扰线
for i in range(2):
    draw.line(((10,10),(80,80)),fill=(0,255,0),width=3)
im.show()
```

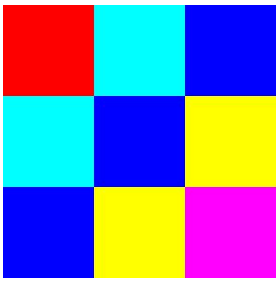
执行结果如下图：

【示例】控制像素生成九宫格

```
from PIL import Image, ImageFilter, ImageFont, ImageDraw
width=300;height=300
x,y=0,0
im = Image.new("RGB",(width,height),(255,255,255))      #最后一个参数是背景颜色，像素默认值
draw = ImageDraw.Draw(im)
def get_color1():
    a = (x//100)+(y//100)
    if a == 0:
        return (255,0,0)
    elif a == 1:
        return (0,255,255)
    elif a ==2:
        return (0,0,255)
    elif a==3:
        return (255,255,0)
    elif a==4:
```

```
            return (255,0,255)
        else:
            return (0,0,0)
#填充每个像素
for x in range(width):
    for y in range(height):
        draw.point((x,y),fill=get_color1())
im.show()
```

执行结果如下：



【示例】将图片中的黄色变换成红色

```
from PIL import Image, ImageFilter, ImageFont, ImageDraw
im = Image.open("bjsxt.png")
draw = ImageDraw.Draw(im)
def get_color(oldColor):
    '''
    如果是黄色(255,255,0)，则换算成红色,把绿色通道置为 0
    可以点击：windows 中的画图软件调色板观察黄色的区间。
    :return:
    '''
    print(oldColor)
    if oldColor[0]>60 and oldColor[1]>60:
        return (oldColor[0],0,oldColor[2])
    else:
        return oldColor
for x in range(im.width):
    for y in range(im.height):
        draw.point((x,y),fill=get_color(im.getpixel((x,y))))
im.show()
```

执行结果如下：

【示例】读取图片保存到大的一维数组

```
#将图片转换为大的一维数组保存到文件，读取文件内容将结果保存成图片
from PIL import Image
import numpy as np
import os
import pickle
#读取图片的路径
image_dir='./images/'
#保存图片的路径
result_dir='./result/'
#存放数组的文件
array_file='./arr.bin'

#读取图片，将图片保存到大的一维数组中
def image_to_array():
    filenames=os.listdir(image_dir)
    image_arr=np.array([])
    for fileName in filenames:
        img=Image.open(image_dir+fileName)
        #将图片按三色提取
        r,g,b=img.split()
        #r g b  转换为一维数组
        r_arr=np.array(r).reshape(62500)
        g_arr=np.array(g).reshape(-1)
        b_arr=np.array(b).reshape(-1)
        #将三色拼接
        arr=np.concatenate((r_arr,g_arr,b_arr))
        #将 8 张图片的 arr 拼接到一个大的一维数组中
        image_arr=np.concatenate((arr,image_arr))
```

【示例】将大的一维数组保存到文件

```
    #一维数组保存到文件
```

```
        f=open(array_file,'wb')
        pickle.dump(image_arr,f)
        f.close()
```

【示例】读取文件内容合并成图片

```
#读取文件内容，保存图片
def file_to_image():
    with open(array_file,'rb') as f:
        images=pickle.load(f)
        #将一维数组转换为 8,3,250,250
        image_arr=images.reshape((8,3,250,250))
    #循环遍历每一张图片对应的大的数组恢复成图片
        for i in range(8):
            r=Image.fromarray(image_arr[i][0]).convert('L')
            g=Image.fromarray(image_arr[i][1]).convert('L')
            b=Image.fromarray(image_arr[i][2]).convert('L')
            image=Image.merge('RGB',(r,g,b))
            #将图片保存
            image.save(result_dir+str(i)+'.jpg')
```