

1. 引言:

提到图像处理，人们通常想到的工具是 MATLAB。诚然，MATLAB 提供了一个强大的图像处理工具箱。但是，对于简单的图像处理任务而言，采用一种高级的语言将起到事半功倍的效果。Python 无疑就是实现这一功能的理想选择。Python 的面向对象、弱数据类型等等特性都使得用它来进行简单的图像处理的时候非常的简洁方便。

2. 简介:

PythonWare 公司提供了免费的图像处理工具包 PIL(Python Image Library), 该软件包提供了基本的图像处理功能，如：改变图像大小，旋转图像，图像格式转换，色场空间转换，图像增强，直方图处理，插值和滤波等等。虽然在这个软件包上要实现类似 MATLAB 中的复杂的图像处理算法并不太适合，但是 Python 的快速开发能力以及面向对象等等诸多特点使得它非常适合用来进行原型开发。

在 PIL 中，任何一副图像都是用一个 Image 对象表示，而这个类由和它同名的模块导出，因此，要加载一副图像，最简单的形式是这样的：

```
import Image

img = Image.open("dip.jpg")
```

注意：第一行的 Image 是模块名；第二行的 img 是一个 Image 对象； Image 类是在 Image 模块中定义的。关于 Image 模块和 Image 类，切记不要混淆了。现在，我们就可以对 img 进行各种操作了，所有对 img 的操作最终都会反映到到 dip.jpg 图像上。

PIL 提供了丰富的功能模块：Image, ImageDraw, ImageEnhance, ImageFile 等等。最常用到的模块是 Image, ImageDraw, ImageEnhance 这三个模块。下面我对此分别做一介绍。

3. Image 模块:

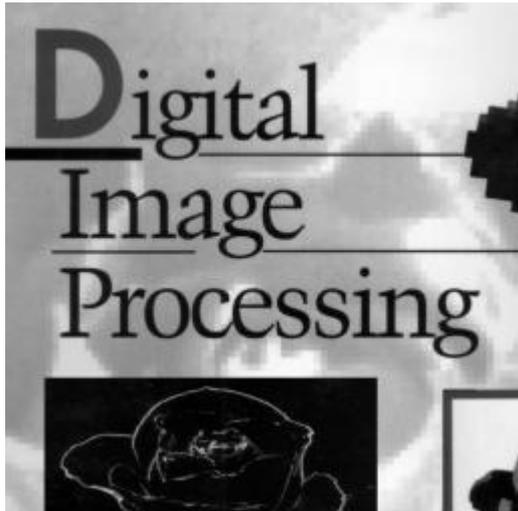
Image 模块是 PIL 最基本的模块，其中导出了 Image 类，一个 Image 类实例对象就对应了一副图像。www.linuxidc.com 同时，Image 模块还提供了很多有用的函数。

(1) 打开一副图像文件:

```
import Image

img = Image.open("dip.jpg")
```

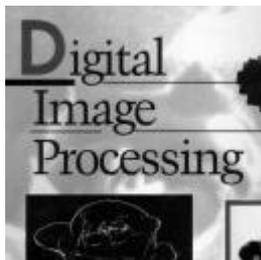
这将返回一个 Image 类实例对象，后面的所有的操作都是在 img 上完成的。在这里，我们读入的图像是：



(2) 调整图像大小:

```
import Image  
  
img = Image.open("img.jpg")  
  
new_img = img.resize((128, 128), Image.BILINEAR)  
  
new_img.save("new_img.jpg")
```

原来的图像大小是 256x256, 现在, 保存的 new_img.jpg 的大小是 128x128:



就是这么简单, 需要说明的是 Image.BILINEAR 指定采用双线性法对像素点插值。

(3) 旋转图像:

现在我们把刚才调整过大小的图像旋转 45 度:

```
import Image  
  
img = Image.open("img.jpg")  
  
new_img = img.resize((128, 128), Image.BILINEAR)
```

```
rot_img = new_img.rotate(45)
```

```
rot_img.save("rot_img.jpg")
```

于是我们保存到 rot_img.jpg 的图像看起来像下面这样：



(4) 格式转换：

假设我们要把上面生成的 rot_img.jpg 转换成 bmp 图像，要做到这一点这太简单了：只需要在上面的代码后面添加下面这样一行即可：

```
rot_img.save("con_img.bmp")
```

如果不指定保存格式，PIL 将自动根据文件名后缀完成格式之间的转换，是不是很简单呢？

(5) 直方图统计：

Image 类实例的 histogram() 方法能够对直方图数据进行统计，并将结果做为一个列表(list)返回。比如，我们对上面的旋转后生成的图像进行直方图统计：

```
import Image

img = Image.open("img.jpg")

new_img = img.resize((128,128), Image.BILINEAR)

rot_img = new_img.rotate(45)

print rot_img.histogram()
```

运行之后将打印出所有 256 个灰度级像素点个数的统计值：

```
[2819, 22, 82, 119, 186, 204, 212, 218, 223, 200, 151, 103, 129, 74, 80, 83, 110, 70, 59, 64,
59, 58, 35, 45, 42, 38, 32, 39, 33, 19, 24, 26, 32, 17, 33, 24, 34, 19, 18, 15, 11, 23, 16, 15,
21, 13, 20, 22, 27, 10, 29, 26, 18, 16, 28, 18, 26, 37, 36, 25, 28, 36, 28, 31, 22, 20, 15, 13,
15, 18, 12, 15, 21, 21, 12, 18, 17, 12, 11, 18, 16, 14, 21, 20, 18, 19, 15, 20, 22, 16, 22, 15,
23, 26, 16, 8, 13, 19, 30, 16, 15, 11, 22, 12, 14, 8, 10, 14, 13, 8, 12, 22, 11, 13, 18, 16, 21,
21, 14, 14, 11, 14, 15, 9, 23, 19, 15, 17, 9, 10, 11, 12, 14, 16, 9, 17, 15, 20, 14, 18, 18, 32,
```

34, 55, 54, 51, 72, 78, 83, 99, 118, 171, 138, 177, 191, 158, 159, 123, 106, 136, 121, 121, 148, 137, 118, 145, 150, 150, 133, 98, 111, 118, 111, 104, 129, 124, 104, 144, 126, 118, 133, 124, 108, 87, 87, 83, 85, 75, 76, 75, 62, 84, 46, 61, 54, 54, 63, 45, 54, 66, 46, 52, 51, 49, 51, 52, 62, 50, 67, 72, 53, 53, 83, 54, 39, 57, 31, 53, 31, 38, 38, 42, 31, 29, 38, 39, 39, 26, 43, 36, 45, 68, 57, 60, 42, 39, 41, 38, 46, 44, 40, 47, 57, 45, 59, 53, 59, 81, 78, 75, 95, 46, 62, 1, 0, 0]

小结：以上介绍了 Image 模块最基本的功能，作为对 PIL 库初步的认识已经足够了，值得说明的是，这里提到只是 Image 一部分功能而已，要对整个 Image 模块的功能有一个全面的了解和掌握，请参见 PIL-handbook.pdf。

4. ImageDraw 模块：

ImageDraw 模块提供了基本的图形能力，这里的图形能力指的主要是图形的绘制能力。PIL 库提供了比较丰富的图形绘制函数，可以绘制直线、弧线、矩形、多边形、椭圆、扇形等等。ImageDraw 实现了一个 Draw 类，所有的图形绘制功能都是在 Draw 类实例的方法中实现的。实例化一个 Draw 类实例很简单：

```
import Image, ImageDraw

img = Image.open("img.jpg")

draw = ImageDraw.Draw(img)
```

首先要导入 ImageDraw 模块。然后，因为绘图操作是在图像上进行的，因此实例化 Draw 类的时候要把 Image 对象 img 通过参数传递给 Draw 类的构造函数。现在，你就可以调用 draw 的各种方法在 img 上绘制图形了。

(1) 绘制直线：

```
import Image, ImageDraw

img = Image.open("img.jpg")

draw = ImageDraw.Draw(img)

width,height = img.size

draw.line(((0, 0), (width-1, height-1)), fill=255)

draw.line(((0, height-1), (width-1, 0)), fill=255)

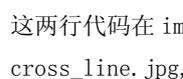
img.save("cross_line.jpg")
```

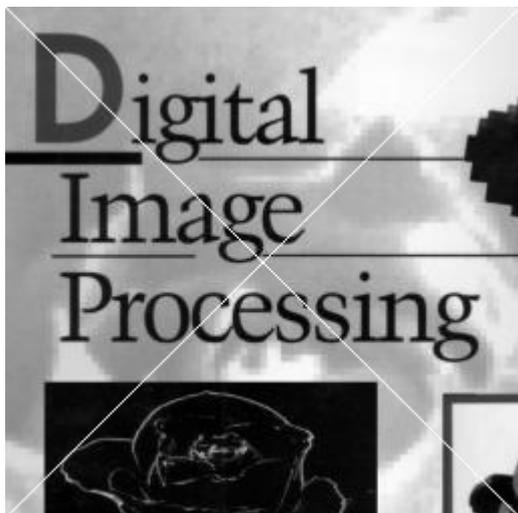
解释一下上面这段代码：

前面三行这里就不解释了。width,height = img.size 是得到 img 的大小，得到这两个属性的主要目的是要在下面的两行代码中使用：

```
draw.line(((0, 0), (width-1, height-1)), fill=255)
```

```
draw.line(((0, height-1), (width-1, 0)), fill=255)
```

这两行代码在图像的两个对角线方向绘制了两条直线。最后，我们把绘制了两条对角线的图像保存为cross_line.jpg, 最后得到的效果如下面所示：



(2) 绘制圆：

```
import Image, ImageDraw
```

```
img = Image.open("img.jpg")
```

```
width, height = img.size
```

```
draw = ImageDraw.Draw(img)
```

```
draw.arc((0, 0, width-1, height-1), 0, 360, fill=255)
```

```
img.save("circle.jpg")
```

这几行代码和上面绘制对角线的代码相比，只更改了一行，即：

```
draw.arc((0, 0, width-1, height-1), 0, 360, fill=255)
```

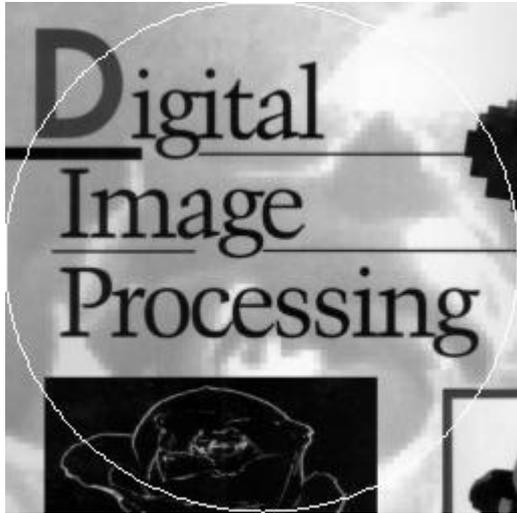
说明：

(0, 0, width-1, height-1) 指定了所画弧线的界限；

0, 360 是所画弧线的起始角度和终止角度；

fill=255 指定了所画线的颜色，注意：PIL 的文档上说这里应该用 outline=255, 但是我发现实际只能用 fill=255 来指定弧线的颜色。

绘制圆后的图像：



小结：有关图形绘制的操作都是类似的，因此这里只给出一个简略的介绍。详细规范请参见 PIL-handbook.pdf。

5. ImageEnhance 模块：

这个模块提供了一个常用的图像增强工具箱。可以用来进行色彩增强、亮度增强、对比度增强、图像尖锐化等等增强操作。所有操作都有相同形式的接口——通过相应类的 enhance 方法实现：色彩增强通过 Color 类的 enhance 方法实现；亮度增强通过 Brightness 类的 enhance 方法实现；对比度增强通过 Contrast 类的 enhance 方法实现；尖锐化通过 Sharpness 类的 enhance 方法实现。所有的操作都需要向类的构造函数传递一个 Image 对象作为参数，这个参数定义了增强作用的对象。同时所有的操作都返回一个新的 Image 对象。如果传给 enhance 方法的参数是 1.0，则不对原图像做任何改变，直接返回原图像的一个拷贝。下面我们通过几个简单的例子进行说明：

(1) 亮度增强：

```
import Image, ImageEnhance

img = Image.open("img.jpg")

brightness = ImageEnhance.Brightness(img)

bright_img = brightness.enhance(2.0)

bright_img.save("bright.jpg")
```

说明：

```
brightness = ImageEnhance.Brightness(img)
```

这一行把 `img` 传给 `Brightness` 类，得到一个 `Brightness` 类实例；

```
bright_img = brightness.enhance(2.0)
```

这一行调用 `brightness` 实例的 `enhance` 方法，传入的参数指定将亮度增强 2 倍；

我们最后得到 `bright.jpg` 图像看起来像这样：



右边的的图像是增强之前的图像(原图像)，注意两者的亮度差比是很大的。

(2) 图像尖锐化：

```
import Image, ImageEnhance
```

```
img = Image.open("img.jpg")
```

```
sharpness = ImageEnhance.Sharpness(img)
```

```
sharp_img = sharpness.enhance(7.0)
```

```
sharp_img.save("bright.jpg")
```

这段代码和上面的完全类似，因此这里不做过多的说明。我们来看一下增强前后的效果对比：



右边的的图像是增强之前的原图像，注意两者的尖锐化程度是很不一样的。

(3) 对比度增强:

```
import Image, ImageEnhance  
  
img = Image.open("img.jpg")  
  
contrast = ImageEnhance.Contrast(img)  
  
contrast_img = contrast.enhance(2.0)  
  
contrast_img.save("contrast.jpg")
```

同上，我们只看增强前后的效果对比:



很明显，增强之后的图像（左边）比原来的图像（右边）对比度提高了。

小结：用 ImageEnhance 来进行常用的图像增强是有效的，并且很简单。当然，对于精细复杂的图像增强操作而言这里提供的功能不够强大，但是在进行简单的图像增强操作的时候，一种简单易行的解决方案无疑是很吸引人的。

6. 总结：

在批处理或者简单的图像处理任务中，采用 python 和 PIL (Python Image Library) 的组合来完成图像处理任务是一个很不错的选择。设想有一个需要对某个文件夹下的所有图像将对比度提高 2 倍的任务。用 python 来做将是十分简单的。当然，我也不得不承认 python 在图像处理方面的功能还比较弱，显然还不适合用来进行滤波、特征提取等等一些更为复杂的应用。我个人的观点是，当你要实现这些“高级”的算法的时候，好吧，把它交给 MATLAB 去完成。但是，如果你面对的只是一个通常的不要求很复杂算法的图像处理任务，那么，python 应该才是你的最佳搭档。