CHAPTER 4

Basic Operations on Images

In the last chapter, you started with image processing using Pillow. You also used Tkinter for displaying images. In this chapter, you will learn various arithmetic and logical operations on images. You will explore Image, ImageChops, and ImageOps modules in Pillow for implementing arithmetic and logical operations. You will also learn how to use the slide bar in Tkinter to dynamically change the input to the Pillow methods.

Image Module

In the last chapter, you used the `open()` and `show()` functions in the Image module of Pillow. In this chapter, you will explore the module in detail. You will study and implement all the methods of this module that are used for basic operations on images.

Splitting and Merging Image Channels

In the grayscale images, there is only one image channel. This means that the grayscale image is made of only a single, two-dimensional matrix that lists the grayscale intensity of the corresponding pixels (values range from 0 to 255). For RGB images, there are three image channels—Red, Green, and Blue. You can see the channel intensities separately by splitting the images into constituent channels using the `split()` method. You can also merge separate channels into a single image with the `merge()` method. Python code that demonstrates the `split()` and `merge()` methods is shown in Listing 4-1.

Listing 4-1. prog01.py

```
from PIL import Image, ImageTk
import tkinter as tk

im = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

root = tk.Tk()
root.title("RED Channel Demo")

r, g, b = im.split()

photo1 = ImageTk.PhotoImage(r)
l1 = tk.Label(root, image=photo1)
```


A. Pajankar, *Raspberry Pi Image Processing Programming*,
DOI 10.1007/978-1-4842-2731-2_4

```
l1.pack()
l1.photo = photo1

photo2 = ImageTk.PhotoImage(Image.merge("RGB", (r, g, b)))
l2 = tk.Label(root, image=photo2)
l2.pack()
l2.photo = photo2

root.mainloop()
```

The code in Listing 4-1 splits the image into separate channels. Then, it displays the Red channel of the image. It also displays the original image by merging previously split channels. The single channel is a matrix of intensity values. So, it is displayed as a grayscale image on the screen. The colors manifest themselves when the code combines all the channels into a single image.

The output window is shown in Figure 4-1.

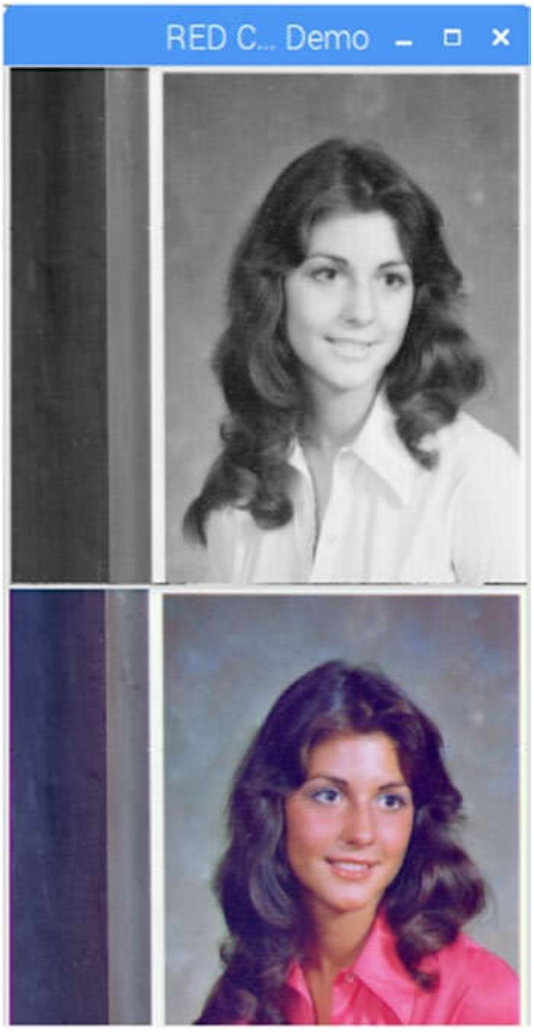

Figure 4-1. *Red channel and the original image*

Image Mode Conversion

You can change the mode of an image using the `convert()` method, as shown in Listing 4-2.

Listing 4-2. prog02.py

```
from PIL import Image, ImageTk
import tkinter as tk

im1 = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")
res1 = im1.convert("L")

root = tk.Tk()
root.title("Colorspace Conversion Demo")

photo = ImageTk.PhotoImage(res1)
l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

root.mainloop()
```

The code in Listing 4-2 changes the mode of the image to `L`. The `convert()` method supports all possible conversions between the `RGB`, `CMYK`, and `L` modes.

Image Blending

You can blend two images using the `blend()` method. It takes three arguments—the two images to be blended and value of alpha. The mathematical formula it uses for blending is as follows:

```
output = image1 * (1.0 - alpha) + image2 * alpha
```

Now, you will write a program that can change the value of alpha dynamically so that you can experience the blending effect. For that, you will use the scale widget in Tkinter.

The program in Listing 4-3 demonstrates this process.

Listing 4-3. prog03.py

```
from PIL import Image, ImageTk
import tkinter as tk

def show_value_1(alpha):
    print('Alpha: ', alpha)

    img = Image.blend(im1, im2, float(alpha))
    photo = ImageTk.PhotoImage(img)
```

```
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Blending Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")
im2 = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

photo = ImageTk.PhotoImage(im1)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Alpha", from_=0, to=1,
      resolution=0.01, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()
```

The code in Listing 4-3 creates a scale widget using the `tk.Scale()` statement. The statement has more than 79 characters, so in order to conform to the PEP8 standard, I wrote it to fit in two lines, each consisting less than 79 characters. The parameters passed to `tk.Scale()` are as follows:

- The Tkinter window variable name
- `label`: The label to be associated with `scale`
- `from_ and to`: The range of values
- `resolution`: The resolution of the scale bar
- `command`: The function to execute when the value of the scale changes
- `orient`: The orientation of the scale

When you change the slider with the mouse pointer, it calls the custom `show_value_1()` function. You are printing the current value of the track bar position to the console for debugging purposes. The `img = Image.blend(im1, im2, float(alpha))` statement creates a blended image. This code

```
photo = ImageTk.PhotoImage(img)
l['image'] = photo
l.photo = photo
```

updates the image in the Tkinter window. The `show_value_1()` function updates every time you change the slider position and the new image is computed. This makes the program interesting and interactive, as you can change the value of alpha to see the transition effect.

The output is shown in Figure 4-2.

Figure 4-2. *Image blending tool*

You will use the same basic code skeleton to look at many of the methods in Pillow.

Resizing an Image

You can resize an image using the `resize()` function, as shown in Listing 4-4.

Listing 4-4. prog04.py

```
from PIL import Image, ImageTk
import tkinter as tk

def show_value_1(size):
    print('Resize: ', size, ' : ', size)

    img = im.resize((int(size), int(size)))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.attributes('-fullscreen', True)
im = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

photo = ImageTk.PhotoImage(im)
```

```
l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Resize", from_=128,
      to=512, resolution=1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()
```

This example varies the image size from (128, 128) to (512, 512). The `resize()` command takes the new size tuple as an argument. The code also invokes the Tkinter window in full-screen mode with the `root.attributes()` function call. To close this window, you have to press Alt+F4 from the keyboard.

Run the code and take a look at the output.

Rotating an Image

You can use `rotate()` method, which takes the angle of rotation as an argument. The code in Listing 4-5 demonstrates this idea.

Listing 4-5. prog05.py

```
from PIL import Image, ImageTk
import tkinter as tk

def show_value_1(angle):
    print('Angle: ', angle)

    img = im.rotate(float(angle))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title("Rotation Demo")
im = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

photo = ImageTk.PhotoImage(im)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Angle", from_=0, to=90,
      resolution=1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()
```

Run this code to experience the rotation effect on an image.

You can also transpose the images using the `transpose()` function. It takes one of `PIL.Image.FLIP_LEFT_RIGHT`, `PIL.Image.FLIP_TOP_BOTTOM`, `PIL.Image.ROTATE_90`, `PIL.Image.ROTATE_180`, `PIL.Image.ROTATE_270`, or `PIL.Image.TRANSPOSE` as an argument. The code example in Listing 4-6 shows rotation at 180 degrees.

Listing 4-6. prog06.py

```
from PIL import Image, ImageTk
import tkinter as tk

root = tk.Tk()
root.title("Transpose Demo")
im = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

out = im.transpose(Image.ROTATE_180)

photo = ImageTk.PhotoImage(out)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

root.mainloop()
```

Also, if you change the argument to `transpose()` as follows:

```
out = im.transpose(Image.FLIP_TOP_BOTTOM)
```

It will flip the image vertically.

Crop and Paste Operations

You can crop a part of an image using the `crop()` method. It takes an argument of four-tuples specifying the coordinates of the box to be cropped from the image. Pillow also has a `paste()` method to paste a rectangular image to another image. The `paste()` method takes the image to be pasted and the coordinates as arguments. The program in Listing 4-7 demonstrates how you can rotate a face using a clever combination of `crop()`, `rotate()`, and `paste()`.

Listing 4-7. prog08.py

```
from PIL import Image

im = Image.open("/home/pi/DIP/Dataset/4.1.03.tiff")
face_box = (100, 100, 150, 150)
face = im.crop(face_box)
rotated_face = face.transpose(Image.ROTATE_180)
im.paste(rotated_face, face_box)
im.show()
```

The output of the code in Listing 4-7 is shown in Figure 4-3.

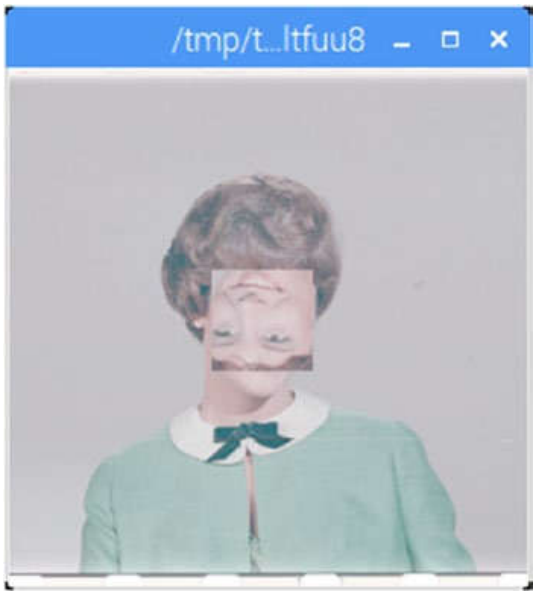

Figure 4-3. *Crop and paste demo*

Copying and Saving Images to a File

You can use the `copy()` method to copy an entire pillow image to another Python variable. You can save a Pillow image to a file using the `save()` method. A demonstration is shown in Listing 4-8.

Listing 4-8. prog09.py

```
from PIL import Image

im = Image.open("/home/pi/DIP/Dataset/4.1.03.tiff")

im1 = im.copy()
im1.save("test.tiff")
```

The code in Listing 4-8 opens an image from a given location, copies it into the `im1` variable, and saves it to the current location as `test.tiff`.

Knowing the Value of a Particular Pixel

You can determine the value of a particular pixel using `getpixel()`. It is usually a tuple that represents the channel intensities. With RGB images, you get the Red, Green, and Blue intensities. It is used as follows:

```
print(im.getpixel((100,100)))
```

ImageChops Module

This module contains many basic arithmetic and logical operations that you can use on your images. Let's look at them quickly one by one.

You can add two images using the `add()` method. The following is the sample code for adding images:

```
im3 = ImageChops.add(im1, im2)
```

The `add_module()` method adds two images without clipping the result:

```
im3 = ImageChops.add_modulo(im1, im2)
```

The `darker()` method compares two images, pixel-by-pixel, and returns the darker pixels.

```
im3 = ImageChops.darker(im1, im2)
```

The `difference()` method returns the difference of the absolute values of two images:

```
im3 = ImageChops.difference(im1, im2)
```

It uses the following mathematical formula for calculating the difference:

```
image3 = abs(image1 - image2)
```

You can invert an image as follows:

```
im2 = ImageChops.invert(im1)
```

Just like with `darker()`, you can use the `lighter()` method to return the set of lighter pixels:

```
im3 = ImageChops.lighter(im1, im2)
```

`logical_and()` and `logical_or()` are the logical operations on images. These are explained with the help of black-and-white images. The following are example uses:

```
im1 = Image.open("/home/pi/DIP/Dataset/5.1.13.tiff")
im2 = Image.open("/home/pi/DIP/Dataset/5.1.13.tiff")
im2 = im2.transpose(Image.ROTATE_90)

im3 = ImageChops.logical_and(im1.convert("1"), im2.convert("1"))

im3 = ImageChops.logical_or(im1.convert("1"), im2.convert("1"))
```

These examples convert the grayscale images to black-and-white images first and then perform the logical operations on them. The result is shown in Figure 4-4.

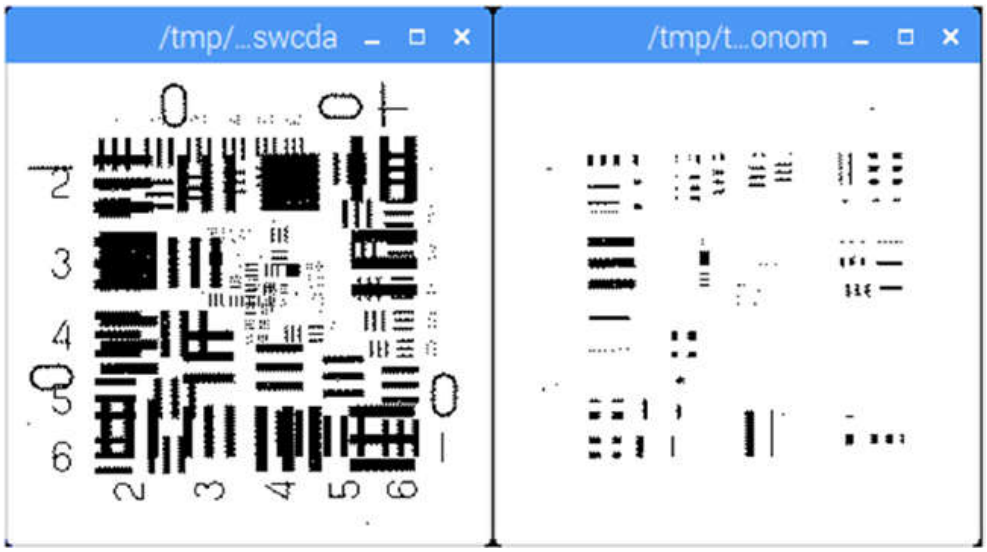

Figure 4-4. *Logical operations on images*

You can superimpose an image on another using the `multiply()` method:

```
im3 = ImageChops.multiply(im1, im2)
```

The `screen()` method superimposes inverted images on top of each other.

```
im3 = ImageChops.screen(im1, im2)
```

You can subtract one image from another using the `subtract()` method as follows:

```
im3 = ImageChops.subtract(im1, im2)
```

You can subtract without clipping the result as follows:

```
im3 = ImageChops.subtract_modulo(im1, im2)
```

ImageOps Module

This module has many predefined and useful operations. You can automatically adjust the contrast of an image as follows:

```
im2 = ImageOps.autocontrast(im1)
```

You can crop the borders of an image equally from all sides as follows:

```
im2 = ImageOps.crop(im1, 50)
```

The first argument of the `ImageOps.crop()` method is the image and the second argument is the width of the cropped border in pixels.

You can also expand the border of an image. Expanded borders will be filled with black pixels equally on all the sides.

```
im2 = ImageOps.expand(im1, 50)
```

You can flip an image vertically as follows:

```
im2 = ImageOps.flip(im1)
```

You can also flip it horizontally as follows:

```
im2 = ImageOps.mirror(im1)
```

You can reduce the number of bits of all the color channels using the `posterize()` method. It takes the image and the number of bits to keep for every channel as arguments. The following is an example:

```
im2 = ImageOps.posterize(im1, 3)
```

This example keeps only three bits per channel. The result is shown in Figure 4-5.

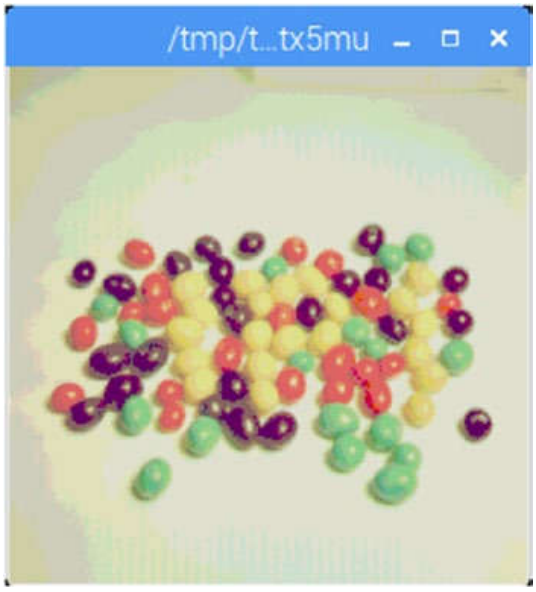

Figure 4-5. *Posterizing an image*

The `solarize()` method inverts all the pixels above a particular grayscale threshold.

```
im2 = ImageOps.solarize(im1, 100)
```

The result is shown in Figure 4-6.

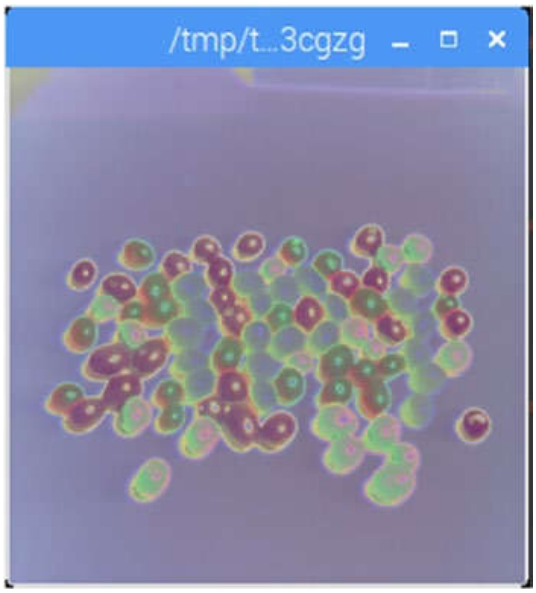

Figure 4-6. *Solarize operation*

EXERCISE

Complete the following coding exercises to gain a better understanding of Pillow.

1. Write the code to show the Green and Blue channels of an image.
2. Write the code for image conversion between the `CMYK` and `L` modes.

Conclusion

This chapter explored Image, ImageChops, and ImageOps in detail. In the next chapter, you will explore a few more Pillow modules for advanced operations on images like filtering, enhancements, histograms, and quantization.