# 第 5 章

CHAPTER 5

# 图 像 运 算

图像运算大致分为像素运算、算术运算、邻域运算、几何变换、空间变换和二值图像打包解包。

## 5.1 像素运算

### 5.1.1 获取像素

impixel()函数用于获取图像中特定像素的值。

impixel()函数允许传入 3 个参数,第 1 个参数是图像,第 2 个参数是 $x$ 坐标,第 3 个参数是 $y$ 坐标,此时将返回该图像在指定坐标处的像素的值。

获取图像 input.png 中(1,2)坐标处的像素的值,代码如下:

```
>> impixel(input_png, 1, 2)
ans =

    68  68  68
    39  39  39
    25  25  25
```

此外,impixel()函数允许传入 4 个参数,第 1 个参数是图像,第 2 个参数是索引,第 3 个参数是 $x$ 坐标,第 4 个参数是 $y$ 坐标,此时将返回该图像在指定坐标处的像素的值。

获取图像 input_gif.gif 中(1,2)坐标处的像素的值,代码如下:

```
>> impixel(input_gif_ind, input_gif_map, 1, 2)
```

此外,impixel()函数允许传入 5 个参数,第 1 个参数是 xData,第 2 个参数是 yData,第 3 个参数是图像,第 4 个参数是 $x$ 坐标,第 5 个参数是 $y$ 坐标,此时将返回该图像在指定坐标处的像素的值。

获取图像 input.png 中(1,2)坐标处的像素的值,xData 为 1:10,yData 为 2:20,代码如下:

```
>> impixel(1:10, 2:20, input_png, 1, 2)
ans =

    69  69  69
    40  40  40
    26  26  26
```

此外，impixel()函数允许传入 6 个参数，第 1 个参数是 xData，第 2 个参数是 yData，第 3 个参数是图像，第 4 个参数是索引，第 5 个参数是 $x$ 坐标，第 6 个参数是 $y$ 坐标，此时将返回该图像在指定坐标处的像素的值。

获取图像 input_gif.gif 中(1,2)坐标处的像素的值，xData 为 1:10，yData 为 2:20，代码如下：

```
>> impixel(1:10, 2:20, input_gif_ind, input_gif_map, 1, 2)
```

## 5.1.2　根据像素生成二值图像

roicolor()函数用于根据像素生成二值图像。roicolor()函数至少需要传入两个参数，第 1 个参数是图像，第 2 个参数是颜色，此时将图像中等于指定颜色的像素设为 1 并将其他颜色设为 0，返回这个新的二值图像。

将图像指定为 input_png 且将颜色指定为 150，生成二值图像的代码如下：

```
>> roicolor(input_png, 150);
```

根据像素生成二值图像的结果如图 5-1 所示。

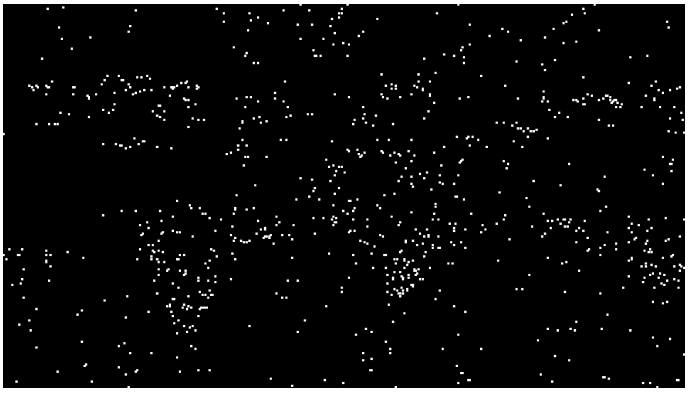

**图 5-1　根据像素生成二值图像**

此外，roicolor()函数还允许传入 3 个参数，第 1 个参数是图像，第 2 个参数是颜色范围的下界，第 3 个参数是颜色范围的上界，此时将图像中在颜色范围中的像素设为 1 并将其他颜色设为 0，返回这个新的二值图像。

将图像指定为 input_png 且将颜色范围的下界和上界分别指定为 100 和 150，生成二值图像的代码如下：

```
>> roicolor(input_png, 100, 150);
```

根据像素生成二值图像的结果如图 5-2 所示。

**图 5-2　根据像素生成二值图像 颜色下界和上界分别为 100 和 150**

## 5.1.3　量化生成图像

imquantize()函数用于量化生成图像。imquantize()函数至少需要传入两个参数，第 1 个参数是图像，第 2 个参数是多个阈值。如果指定 $n$ 个阈值，则图像按阈值量化得到的像素为 $1,2,\cdots,n+1$。

将图像指定为 input_png 且将多个阈值指定为[3,6,9]，量化生成图像的代码如下：

```
>> imquantize(input_png, [3, 6, 9]);
```

此外，imquantize()函数允许追加传入第 3 个参数，这个参数被认为是多个量化得到的像素值。如果指定 $n$ 个阈值，则必须指定 $n+1$ 个量化得到的像素值。

将图像指定为 input_png，将多个阈值指定为[3,6,9]且将多个量化得到的像素值指定为[10,50,100,150]，量化生成图像的代码如下：

```
>> imquantize(input_png, [3, 6, 9], [10, 50, 100, 150]);
```

## 5.1.4　像素排序

OpenCV 库可以用于像素排序。OpenCV 库支持的像素排序方式如表 5-1 所示。

**表 5-1　OpenCV 库支持的像素排序方式**

| 像素排序方式 | 含　义 |
| --- | --- |
| SORT_EVERY_ROW | 按行独立排序 |
| SORT_EVERY_COLUMN | 按列独立排序；<br>与按行独立排序互斥 |

续表

| 像素排序方式 | 含　义 |
| --- | --- |
| SORT_ASCENDING | 升序排序 |
| SORT_DESCENDING | 降序排序；<br>与升序排序互斥 |

**注意**：OpenCV 库在像素排序时，需要同时传入两种排序方式。

使用 OpenCV 库进行像素排序的代码如下：

```
#第 5 章/sort.py
import sys
import cv2

def sort(image_path, output_path, flags = cv2.SORT_EVERY_COLUMN):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.sort(image, flags)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    sort(sys.argv[1], sys.argv[2], eval(sys.argv[3]))
```

**1. 按行独立排序且按升序排序**

将图像指定为 input.png，将输出图像指定为 output_png.png，按行独立排序且按升序排序的代码如下：

```
>> python("sort.py", "input.png", "output_png.png", "\"cv2.SORT_EVERY_ROW + cv2.SORT_
ASCENDING\"")
```

按行独立排序且按升序排序的结果如图 5-3 所示。

**图 5-3　按行独立排序且按升序排序**

**2. 按行独立排序且按降序排序**

将图像指定为 input.png,将输出图像指定为 output_png.png,按行独立排序且按降序排序的代码如下:

```
>> python("sort.py", "input.png", "output_png.png", "\"cv2.SORT_EVERY_ROW + cv2.SORT_
DESCENDING\"")
```

按行独立排序且按降序排序的结果如图 5-4 所示。

**图 5-4 按行独立排序且按降序排序**

**3. 按列独立排序且按升序排序**

将图像指定为 input.png,将输出图像指定为 output_png.png,按列独立排序且按升序排序的代码如下:

```
>> python("sort.py", "input.png", "output_png.png",
"\"cv2.SORT_EVERY_COLUMN + cv2.SORT_ASCENDING\"")
```

按列独立排序且按升序排序的结果如图 5-5 所示。

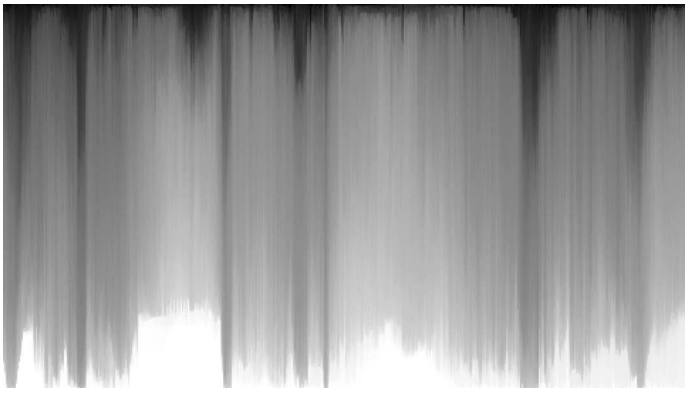

**图 5-5 按列独立排序且按升序排序**

**4. 按列独立排序且按降序排序**

将图像指定为 input.png，将输出图像指定为 output_png.png，按列独立排序且按降序排序的代码如下：

```
>> python("sort.py", "input.png", "output_png.png",
"\"cv2.SORT_EVERY_COLUMN + cv2.SORT_DESCENDING\"")
```

按列独立排序且按降序排序的结果如图 5-6 所示。

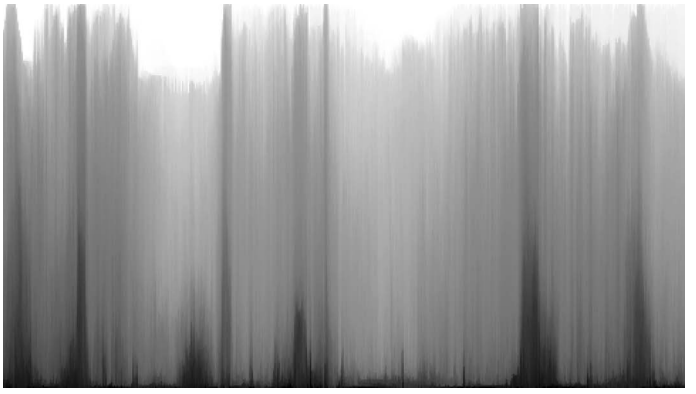

**图 5-6　按列独立排序且按降序排序**

## 5.1.5　固定阈值法

**1. 用于图像二值化的阈值**

graythresh()函数用于自动计算灰度图像转换为二值图像的阈值。graythresh()函数允许传入 1 个参数进行调用，这个参数代表灰度图像。

自动计算图像 input_png 转换为二值图像的阈值，代码如下：

```
>> graythresh(input_png)
ans = 0.5020
```

此外，graythresh()函数允许追加传入第 2 个参数进行调用，这个参数代表自动计算阈值的方式。graythresh()函数支持的自动计算阈值的方式如表 5-2 所示。

**表 5-2　graythresh()函数支持的自动计算阈值的方式**

| 自动计算阈值的方式 | 含　义 | 自动计算阈值的方式 | 含　义 |
|---|---|---|---|
| Otsu | 大津法 | mean | 均值 |
| concavity | 凹性 | MinError | 最小错误 |
| intermodes | 中间帧模式 | minimum | 最小值 |
| intermeans | 中间帧均值 | moments | 几何矩阈值 |
| MaxEntropy | 最大熵 | percentile | 百分比法 |
| MaxLikelihood | 最大似然估计 | | |

将自动计算阈值的方式指定为 concavity，自动计算图像 input_png 转换为二值图像的阈值，代码如下：

```
>> graythresh(input_png, "concavity")
ans = 0.4431
```

此外，graythresh()函数允许追加传入第 3 个参数进行调用，这个参数代表自动计算阈值的选项。graythresh()函数支持的自动计算阈值的选项如表 5-3 所示。

**表 5-3 graythresh()函数支持的自动计算阈值的选项**

| 自动计算阈值的方式 | 自动计算阈值的选项 | 自动计算阈值的方式 | 自动计算阈值的选项 |
|---|---|---|---|
| Otsu | — | mean | — |
| concavity | — | MinError | — |
| intermodes | — | minimum | — |
| intermeans | — | moments | — |
| MaxEntropy | — | percentile | 假定占比在[0,1]的像素为背景 |
| MaxLikelihood | — | | |

将自动计算阈值的方式指定为 percentile，假定占比 0.5(50%)的像素为背景，自动计算图像 input_png 转换为二值图像的阈值，代码如下：

```
>> graythresh(input_png, "percentile", 0.5)
ans = 0.4431
```

**2. 使用大津法计算阈值**

otsuthresh()函数用于使用大津法计算阈值进行图像二值化。otsuthresh()函数允许传入 1 个参数，这个参数是图像的直方图。

指定直方图 nn，使用大津法计算阈值进行图像二值化，代码如下：

```
>> otsuthresh(nn)
```

**3. 将灰度图像转换为二值图像**

grayslice()函数用于将灰度图像转换为二值图像。grayslice()函数允许传入 1 个参数进行调用，这个参数代表灰度图像。

将图像 input_pgm 转换为二值图像，代码如下：

```
>> grayslice(input_pgm);
```

此外，grayslice()函数追加允许传入第 2 个参数进行调用，这个参数代表多个阈值。

将多个阈值指定为[0 0.5 1]，将图像 input_pgm 转换为二值图像，代码如下：

```
>> grayslice(input_pgm, [0 0.5 1]);
```

**4. 通过固定阈值法进行像素运算**

im2bw()函数用于将彩色图像或灰度图像转换为二值图像。im2bw()函数允许传入 1 个参数进行调用，这个参数代表 RGB 颜色空间下的图像或灰度图像。

将图像 input_png 转换为二值图像，代码如下：

```
>> im2bw(input_png);
```

转换为二值图像的结果如图 5-7 所示。

**图 5-7　转换为二值图像**

将图像 input_pgm 转换为二值图像，代码如下：

```
>> im2bw(input_pgm);
```

此外，im2bw()函数允许传入两个参数进行调用，第 1 个参数代表索引图像，第 2 个参数代表颜色图。

将颜色图指定为 jet，将图像 input_gif 转换为二值图像，代码如下：

```
>> im2bw(input_gif, jet);
```

im2bw()函数可以追加传入 1 个参数进行调用，这个参数代表阈值，用于区分转换后的颜色是 0 或 1。

将阈值指定为 0.3，将图像 input_png 转换为二值图像，代码如下：

```
>> im2bw(input_png, 0.3);
```

转换为二值图像的结果如图 5-8 所示。

**图 5-8　转换为二值图像 阈值为 0.3**

此外，im2bw()函数可以追加传入1个参数进行调用，这个参数代表graythresh()函数自动计算阈值的方式。

将自动计算阈值的方式指定为concavity，将图像input_png转换为二值图像，代码如下：

```
>> im2bw(input_png, "concavity");
```

转换为二值图像的结果如图5-9所示。

**图5-9　转换为二值图像 自动计算阈值**

OpenCV库可以通过固定阈值法进行像素运算。OpenCV库支持的固定阈值类型如表5-4所示。

**表5-4　OpenCV库支持的固定阈值类型**

| 固定阈值类型 | 含　义 |
|---|---|
| THRESH_BINARY | 用于图像二值化的阈值；<br>如果像素值大于此阈值，则将像素转换为最大值，否则将像素转换为最小值 |
| THRESH_BINARY_INV | 用于图像反向二值化的阈值；<br>如果像素值大于此阈值，则将像素转换为最小值，否则将像素转换为最大值 |
| THRESH_TRUNC | 用于图像截断的阈值；<br>如果像素值大于此阈值，则将像素转换为最大值，否则像素不变 |
| THRESH_TOZERO | 用于图像截断并取0的阈值；<br>如果像素值大于此阈值，则像素不变，否则将像素转换为0 |
| THRESH_TOZERO_INV | 用于图像反向截断并取0的阈值；<br>如果像素值大于此阈值，则将像素转换为0，否则像素不变 |
| THRESH_MASK | — |
| THRESH_OTSU | 使用大津法计算阈值 |
| THRESH_TRIANGLE | 使用三角法计算阈值 |

使用 OpenCV 库通过固定阈值法进行像素运算的代码如下：

```
#第 5 章/threshold.py
import sys
import cv2
import numpy as np

def threshold(image_path, output_path, thresh,
              maxval = 255, type = cv2.THRESH_BINARY):
    image = cv2.imread(image_path)
    if type == cv2.THRESH_OTSU or type == cv2.THRESH_TRIANGLE:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    thr, ret = cv2.threshold(image, thresh = thresh,
                         maxval = maxval, type = type)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    threshold(sys.argv[1], sys.argv[2],
              float(sys.argv[3]), 255,
              eval(sys.argv[4]))
```

将图像指定为 input.png，将输出图像指定为 output.png，将阈值指定为 100，将最大值指定为 255，并将固定阈值类型指定为 cv2.THRESH_BINARY，通过固定阈值法进行像素运算的代码如下：

```
>> python("threshold.py", "input.png", "output.png", "100",
"cv2.THRESH_BINARY")
```

固定阈值法的结果如图 5-10 所示。

**图 5-10　固定阈值类型为 cv2.THRESH_BINARY**

将图像指定为 input.png，将输出图像指定为 output.png，将阈值指定为 100，将最大值指定为 255，并将固定阈值类型指定为 cv2.THRESH_BINARY_INV，通过固定阈值法进行像素运算的代码如下：

```
>> python("threshold.py", "input.png", "output.png", "100",
"cv2.THRESH_BINARY_INV")
```

固定阈值法的结果如图 5-11 所示。

**图 5-11　固定阈值类型为 cv2.THRESH_BINARY_INV**

将图像指定为 input.png，将输出图像指定为 output.png，将阈值指定为 100，将最大值指定为 255，并将固定阈值类型指定为 cv2.THRESH_TRUNC，通过固定阈值法进行像素运算的代码如下：

```
>> python("threshold.py", "input.png", "output.png", "100",
"cv2.THRESH_TRUNC")
```

固定阈值法的结果如图 5-12 所示。

**图 5-12　固定阈值类型为 cv2.THRESH_TRUNC**

将图像指定为 input.png，将输出图像指定为 output.png，将阈值指定为 100，将最大值指定为 255，并将固定阈值类型指定为 cv2.THRESH_TOZERO，通过固定阈值法进行像素运算的代码如下：

```
>> python("threshold.py", "input.png", "output.png", "100",
"cv2.THRESH_TOZERO")
```

固定阈值法的结果如图 5-13 所示。

**图 5-13　固定阈值类型为 cv2. THRESH_TOZERO**

将图像指定为 input. png，将输出图像指定为 output. png，将阈值指定为 100，将最大值指定为 255，并将固定阈值类型指定为 cv2. THRESH_TOZERO_INV，通过固定阈值法进行像素运算的代码如下：

```
>> python("threshold.py", "input.png", "output.png", "100",
"cv2.THRESH_TOZERO_INV")
```

固定阈值法的结果如图 5-14 所示。

**图 5-14　固定阈值类型为 cv2. THRESH_TOZERO_INV**

将图像指定为 input. png，将输出图像指定为 output. png，将阈值指定为 100，将最大值指定为 255，并将固定阈值类型指定为 cv2. THRESH_OTSU，通过固定阈值法进行像素运算的代码如下：

```
>> python("threshold.py", "input.png", "output.png", "100",
"cv2.THRESH_OTSU")
```

固定阈值法的结果如图 5-15 所示。

图 5-15　固定阈值类型为 cv2. THRESH_OTSU

将图像指定为 input. png，将输出图像指定为 output. png，将阈值指定为 100，将最大值指定为 255，并将固定阈值类型指定为 cv2. THRESH_TRIANGLE，通过固定阈值法进行像素运算的代码如下：

```
>> python("threshold.py", "input.png", "output.png", "100",
"cv2.THRESH_TRIANGLE")
```

固定阈值法的结果如图 5-16 所示。

图 5-16　固定阈值类型为 cv2. THRESH_TRIANGLE

GraphicsMagick 可以基于强度阈值将图像转换为二值图像，如果像素的强度低于阈值，则设为黑色，其余像素设为白色。将图像指定为 input. png，将输出图像指定为 output. png，将强度阈值指定为 40%，将图像转换为二值图像的代码如下：

```
>> system("gm convert input.png -threshold 40% output.png")
```

### 5.1.6　基于通道阈值更改像素

GraphicsMagick 可以基于通道阈值更改像素，如果像素的 R 通道、G 通道或 B 通道的

像素值低于对应阈值，则将对应通道的像素值设为 0，其余通道的像素值不变。将图像指定为 input. png，将输出图像指定为 output. png，将 R 通道阈值指定为 40%，将 G 通道阈值指定为 50%，将 B 通道阈值指定为 60%，基于通道阈值更改像素的代码如下：

```
>> system("gm convert input.png -black-threshold 40%,50%,60% output.png")
```

基于通道阈值更改像素的结果如图 5-17 所示。

**图 5-17　基于通道阈值更改像素**

## 5.1.7　像素扩散

GraphicsMagick 可以用于像素扩散。将图像指定为 input. png，将输出图像指定为 output. png，将像素扩散数指定为 5，颜色替换的代码如下：

```
>> system("gm convert input.png -spread 5 output.png")
```

像素扩散的结果如图 5-18 所示。

**图 5-18　像素扩散**

## 5.2 算术运算

### 5.2.1 绝对差值

imabsdiff()函数用于计算两幅图像之间的绝对差值。如果两幅图像的尺寸和类型相同,则返回的结果代表两幅图像之间的绝对差值。imabsdiff()函数至少需要传入两个参数,这两个参数是图像。

返回图像 input_png 和 input_png2 的绝对差值,代码如下:

```
>> imabsdiff(input_png, input_png2);
```

绝对差值如图 5-19 所示。

**图 5-19 绝对差值**

此外,imabsdiff()函数允许追加传入第 3 个参数,即图像类型。图像类型可以是 double 或 logical。

返回图像 input_png 和 input_png2 的绝对差值,类型为 double,代码如下:

```
>> imabsdiff(input_png, input_png2, "double")
```

OpenCV 库可以用于计算图像间的绝对差值。使用 OpenCV 库计算图像间的绝对差值的代码如下:

```
#第 5 章/absdiff.py
import sys
import cv2

def absdiff(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    diff = cv2.absdiff(image1, image2)
    cv2.imwrite(output_path, diff)
```

```
if __name__ == "__main__":
    absdiff(sys.argv[1], sys.argv[2], sys.argv[3])
```

将两幅图像指定为 input.png 和 input2.png，将输出图像指定为 output_png.png，计算图像间的绝对差值的代码如下：

```
>> python("absdiff.py", "input.png", "input2.png", "output_png.png")
```

## 5.2.2　图像加法

### 1. 直接相加

imadd()函数用于将两幅图像相加。如果两幅图像的尺寸和类型相同，则返回的结果代表将两幅图像相加后的结果。imadd()函数至少需要传入两个参数，这两个参数是图像。

将图像 input_png 和 input_png2 相加，代码如下：

```
>> imadd(input_png, input_png2)
```

图像加法的结果如图 5-20 所示。

**图 5-20　图像加法**

此外，imadd()函数允许追加传入第 3 个参数，这个参数是图像类型。图像类型可以是 double 或 logical。

将图像 input_png 和 input_png2 相加，类型为 double，代码如下：

```
>> imadd(input_png, input_png2, "double")
```

OpenCV 库可以用于将两幅图像相加。使用 OpenCV 库将两幅图像相加的代码如下：

```
#第 5 章/image_add.py
import sys
import cv2

def image_add(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
```

```
    ret = cv2.add(image1, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    image_add(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input.jpg，将输出图像指定为 output_png.png，将两幅图像相加的代码如下：

```
>> python("image_add.py", "input.png", "input.jpg", "output_png.png")
```

**2. 按比例相加**

OpenCV 库可以用于将两幅图像按比例相加。使用 OpenCV 库将两幅图像按比例相加的代码如下：

```
#第 5 章/scale_add.py
import sys
import cv2

def scale_add(image1_path, image2_path, scale, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.scaleAdd(image1, scale, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    scale_add(sys.argv[1], sys.argv[2],
              float(sys.argv[3]), sys.argv[4])
```

将图像指定为 input.png 和 input2.png，将比例指定为 2，将输出图像指定为 output_png.png，将两幅图像按比例相加的代码如下：

```
>> python("scale_add.py", "input.png", "input2.png", "2",
"output_png.png")
```

图像按比例相加的结果如图 5-21 所示。

**图 5-21　图像按比例相加**

### 5.2.3　图像减法

imsubtract()函数用于将两个图像相减。如果两幅图像的尺寸和类型相同,则返回的结果代表将两幅图像相减后的结果。imsubtract()函数至少需要传入两个参数,这两个参数是图像。

将图像 input_png 和 input_png2 相减,代码如下:

```
>> imsubtract(input_png, input_png2)
```

图像减法的结果如图 5-22 所示。

**图 5-22　图像减法**

此外,imsubtract()函数允许追加传入第 3 个参数,这个参数是图像类型。图像类型可以是 double 或 logical。

将图像 input_png 和 input_png2 相减,类型为 double,代码如下:

```
>> imsubtract(input_png, input_png2, "double")
```

OpenCV 库可以用于将两幅图像相减。使用 OpenCV 库将两幅图像相减的代码如下:

```
#第 5 章/image_subtract.py
import sys
import cv2

def image_subtract(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.subtract(image1, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    image_subtract(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input.jpg,将输出图像指定为 output_png.png,将两幅图

像相减的代码如下：

```
>> python("image_subtract.py", "input.png", "input.jpg",
"output_png.png")
```

### 5.2.4 图像乘法

immultiply()函数用于对两幅图像按元素进行相乘。如果两幅图像的尺寸和类型相同，则返回的结果代表将两幅图像按元素相乘的结果。immultiply()函数至少需要传入两个参数，这两个参数是图像。

将图像 input_png 和 input_png2 按元素相乘，代码如下：

```
>> immultiply(input_png, input_png2)
```

图像乘法的结果如图 5-23 所示。

**图 5-23 图像乘法**

此外，immultiply()函数允许追加传入第 3 个参数，这个参数是图像类型。图像类型可以是 double 或 logical。

将图像 input_png 和 input_png2 按元素相乘，类型为 double，代码如下：

```
>> immultiply(input_png, input_png2, "double")
```

OpenCV 库可以用于将两幅图像按元素相乘。使用 OpenCV 库将两幅图像按元素相乘的代码如下：

```
#第 5 章/image_mul.py
import sys
import cv2

def image_mul(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.multiply(image1, image2)
    cv2.imwrite(output_path, ret)
```

```
if __name__ == "__main__":
    image_mul(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input.jpg，将输出图像指定为 output_png.png，将两幅图像按元素相乘的代码如下：

```
>> python("image_mul.py", "input.png", "input.jpg", "output_png.png")
```

OpenCV 库可以用于将图像和自身的转置相乘。使用 OpenCV 库将图像和自身的转置相乘的代码如下：

```
#第 5 章/image_mul_transposed.py
import sys
import cv2

def image_mul_transposed(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.mulTransposed(image, True)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    image_mul_transposed(sys.argv[1], sys.argv[2])
```

将图像指定为 input.png，将输出图像指定为 output_png.png，将图像和自身转置相乘的代码如下：

```
>> python("image_mul_transposed.py", "input.png", "output_png.png")
```

OpenCV 库可以用于将图像的转置和自身相乘。使用 OpenCV 库将图像的转置和自身相乘的代码如下：

```
#第 5 章/image_transposed_mul.py
import sys
import cv2

def image_transposed_mul(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.mulTransposed(image, False)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    image_transposed_mul(sys.argv[1], sys.argv[2])
```

将图像指定为 input.png，将输出图像指定为 output_png.png，将图像的转置和自身相乘的代码如下：

```
>> python("image_mul_transposed.py", "input.png", "output_png.png")
```

### 5.2.5 图像除法

imdivide()函数用于对两幅图像按元素相除。如果两幅图像的尺寸和类型相同,则返回的结果代表将两幅图像按元素相除的结果。imdivide()函数至少需要传入两个参数,这两个参数是图像。

将图像 input_png 和 input_png2 按元素相除,代码如下:

```
>> imdivide(input_png, input_png2)
```

图像除法的结果如图 5-24 所示。

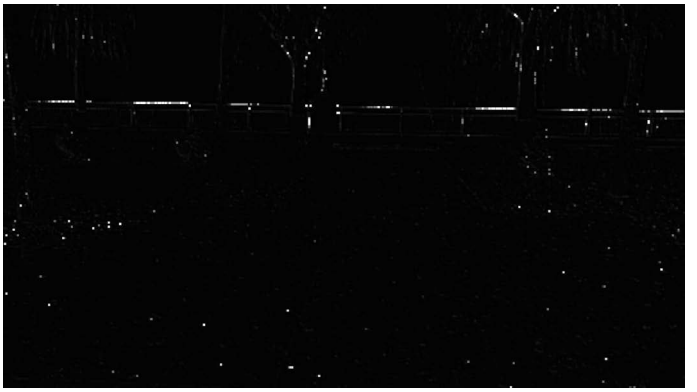

**图 5-24 图像除法**

此外,imdivide()函数允许追加传入第 3 个参数,这个参数是图像类型。图像类型可以是 double 或 logical。

将图像 input_png 和 input_png2 按元素相除,类型为 double,代码如下:

```
>> imdivide(input_png, input_png2, "double")
```

OpenCV 库可以用于将两幅图像按元素相除。使用 OpenCV 库将两幅图像按元素相除的代码如下:

```
#第 5 章/image_divide.py
import sys
import cv2

def image_divide(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.divide(image1, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    image_divide(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input.jpg，将输出图像指定为 output_png.png，将两幅图像按元素相除的代码如下：

```
>> python("image_divide.py", "input.png", "input.jpg", "output_png.png")
```

## 5.2.6　图像幂运算

OpenCV 库可以用于对图像进行幂运算。使用 OpenCV 库对图像进行幂运算的代码如下：

```
#第 5 章/pow.py
import sys
import cv2

def pow(image_path, output_path, power):
    image = cv2.imread(image_path)
    ret = cv2.pow(image, power)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    pow(sys.argv[1], sys.argv[2], float(sys.argv[3]))
```

将图像指定为 input.png，将输出图像指定为 output_png.png，将幂次指定为 3，对图像进行幂运算的代码如下：

```
>> python("pow.py", "input.png", "output_png.png", "3")
```

图像幂运算的结果如图 5-25 所示。

**图 5-25　图像幂运算**

## 5.2.7　图像开方运算

OpenCV 库可以用于对图像进行开方运算。使用 OpenCV 库对图像进行开方运算的代码如下：

```
#第 5 章/sqrt.py
import sys
import cv2

def sqrt(image_path, output_path):
    image = cv2.imread(image_path).astype(float)
    ret = cv2.sqrt(image)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    sqrt(sys.argv[1], sys.argv[2])
```

将图像指定为 input.png,将输出图像指定为 output_png.png,对图像进行开方运算的代码如下:

```
>> python("sqrt.py", "input.png", "output_png.png")
```

图像开方运算的结果如图 5-26 所示。

**图 5-26　图像开方运算**

## 5.2.8　图像指数运算

OpenCV 库可以用于对图像进行指数运算。使用 OpenCV 库对图像进行指数运算的代码如下:

```
#第 5 章/exp.py
import sys
import cv2

def exp(image_path, output_path):
    image = cv2.imread(image_path).astype(float)
    ret = cv2.exp(image)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    exp(sys.argv[1], sys.argv[2])
```

将图像指定为 input.png，将输出图像指定为 output_png.png，对图像进行指数运算的代码如下：

```
>> python("exp.py", "input.png", "output_png.png")
```

图像指数运算的结果如图 5-27 所示。

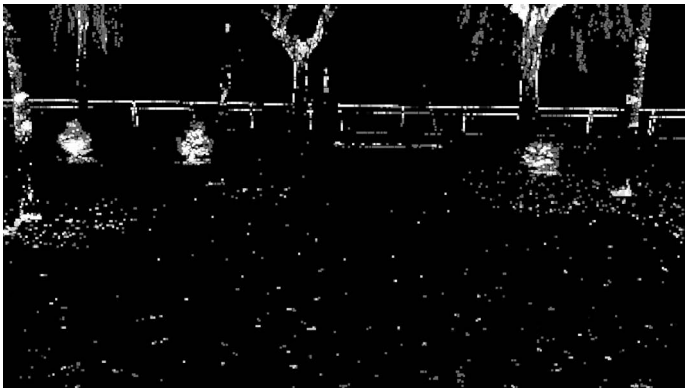

**图 5-27　图像指数运算**

## 5.2.9　图像对数运算

OpenCV 库可以用于对图像进行求自然对数。使用 OpenCV 库对图像进行求自然对数的代码如下：

```
#第 5 章/log.py
import sys
import cv2

def log(image_path, output_path):
    image = cv2.imread(image_path).astype(float)
    ret = cv2.log(image)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    log(sys.argv[1], sys.argv[2])
```

将图像指定为 input.png，将输出图像指定为 output_png.png，对图像进行求自然对数的代码如下：

```
>> python("log.py", "input.png", "output_png.png")
```

## 5.2.10　图像求逆

OpenCV 库可以用于求逆。如果图像的逆不存在，则求伪逆。使用 OpenCV 库对图像进行求逆的代码如下：

```
#第 5 章/invert.py
import sys
import cv2
import numpy as np

def invert(image_path, output_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    w, h = image.shape
    if w <= h:
        image = image[:w, :w]
    else:
        image = image[:h, :h]
    image = image.astype(np.float64)
    is_trueinv, ret = cv2.invert(image)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    invert(sys.argv[1], sys.argv[2])
```

将图像指定为 input.png，将输出图像指定为 output.png，对图像进行求逆的代码如下：

```
>> python("invert.py", "input.png", "output.png")
```

### 5.2.11 图像转置

OpenCV 库可以用于转置。使用 OpenCV 库对图像进行转置的代码如下：

```
#第 5 章/transpose.py
import sys
import cv2
import numpy as np

def transpose(image_path, output_path):
    image = cv2.imread(image_path)
    ret = cv2.transpose(image)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    transpose(sys.argv[1], sys.argv[2])
```

将图像指定为 input.png，将输出图像指定为 output.png，对图像进行转置的代码如下：

```
>> python("transpose.py", "input.png", "output.png")
```

图像转置的结果如图 5-28 所示。

图 5-28　图像转置

## 5.2.12　图像按位与运算

OpenCV 库可以用于对两幅图像进行按位与运算。使用 OpenCV 库对两幅图像进行按位与运算的代码如下：

```
#第 5 章/bitwise_and.py
import sys
import cv2

def bitwise_and(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.bitwise_and(image1, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    bitwise_and(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input2.png，将输出图像指定为 output_png.png，对两幅图像进行按位与运算的代码如下：

```
>> python("bitwise_and.py", "input.png", "input2.png", "output_png.png")
```

图像按位与运算的结果如图 5-29 所示。

图 5-29　图像按位与运算

## 5.2.13　图像按位或运算

OpenCV 库可以用于对两幅图像进行按位或运算。使用 OpenCV 库对两幅图像进行按位或运算的代码如下：

```
#第 5 章/bitwise_or.py
import sys
import cv2

def bitwise_or(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.bitwise_or(image1, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    bitwise_or(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input2.png，将输出图像指定为 output_png.png，对两幅图像进行按位或运算的代码如下：

```
>> python("bitwise_or.py", "input.png", "input2.png", "output_png.png")
```

图像按位或运算的结果如图 5-30 所示。

图 5-30　图像按位或运算

## 5.2.14　图像按位非运算

OpenCV 库可以用于对两幅图像进行按位非运算。使用 OpenCV 库对两幅图像进行按位非运算的代码如下：

```
#第 5 章/bitwise_not.py
import sys
import cv2

def bitwise_not(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.bitwise_not(image1, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    bitwise_not(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input2.png，将输出图像指定为 output_png.png，对两幅图像进行按位非运算的代码如下：

```
>> python("bitwise_not.py", "input.png", "input2.png", "output_png.png")
```

图像按位非运算的结果如图 5-31 所示。

**图 5-31　图像按位非运算**

## 5.2.15　图像按位异或运算

OpenCV 库可以用于对两幅图像进行按位异或运算。使用 OpenCV 库对两幅图像进行按位异或运算的代码如下：

```
#第 5 章/bitwise_xor.py
import sys
```

```
import cv2

def bitwise_xor(image1_path, image2_path, output_path):
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)
    ret = cv2.bitwise_xor(image1, image2)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    bitwise_xor(sys.argv[1], sys.argv[2], sys.argv[3])
```

将图像指定为 input.png 和 input2.png，将输出图像指定为 output_png.png，对两幅图像进行按位异或运算的代码如下：

```
>> python("bitwise_xor.py", "input.png", "input2.png", "output_png.png")
```

图像按位异或运算的结果如图 5-32 所示。

**图 5-32　图像按位异或运算**

## 5.2.16　图像加权组合

imlincomb()函数用于计算一幅或多幅图像加权组合。imlincomb()函数至少需要传入两个参数，第 1 个参数是权重，第 2 个参数是图像。

将权重指定为 0.5，将输入图像指定为 input_png，计算图像加权组合，代码如下：

```
>> imlincomb(0.5, input_png)
```

图像加权组合的结果如图 5-33 所示。

此外，imlincomb()函数允许追加传入多组权重和图像。

将权重指定为 0.5，将输入图像指定为 input_png；将权重指定为 0.6，将输入图像指定为 input_jpg，计算图像加权组合，代码如下：

```
>> imlincomb(0.5, input_png, 0.6, input_jpg)
```

图 5-33　图像加权组合

此外，imlincomb()函数允许追加传入偏移量，此时最终的输出结果会加上偏移量。

将权重指定为 0.5，将输入图像指定为 input_png；将权重指定为 0.6，将输入图像指定为 input_jpg；将偏移量指定为 0.1，计算图像的加权组合，代码如下：

```
>> imlincomb(0.5, input_png, 0.6, input_jpg, 0.1)
```

## 5.2.17　图像线性变换

imapplymatrix()函数用于将线性变换矩阵应用到图像上，通常用于颜色空间转换。imapplymatrix()函数至少需要传入两个参数，第 1 个参数是变换矩阵，第 2 个参数是图像。

将变换矩阵指定为 ones(3,3)，将输入图像指定为 input_png，将线性变换矩阵应用到图像上，代码如下：

```
>> imapplymatrix(ones(3,3), input_png)
```

将线性变换矩阵应用到图像上的结果如图 5-34 所示。

图 5-34　将线性变换矩阵应用到图像上

此外，imapplymatrix()函数允许追加传入第 3 个参数，这个参数是常数矩阵。

将变换矩阵指定为 ones(3,3)，将输入图像指定为 input_png，常数矩阵是 ones(size([])(1),1)，将线性变换矩阵应用到图像上，代码如下：

```
>> imapplymatrix(ones(3,3), input_png, ones(size([])(1), 1))
```

此外，imapplymatrix()函数允许指定输出的数据类型，这里的数据类型可以是 Octave 中的任意输出类型。

将变换矩阵指定为 ones(3,3)，将输入图像指定为 input_png，将输出的数据类型指定为 uint8，将线性变换矩阵应用到图像上，代码如下：

```
>> imapplymatrix(ones(3,3), input_png, "uint8")
```

## 5.3 邻域运算

nlfilter()函数用于对图像以邻域方式进行处理。nlfilter()函数需要传入 3 个参数，第 1 个参数是图像，第 2 个参数是邻域尺寸，第 3 个参数是运算函数。

将邻域尺寸指定为[2 3]，将运算函数指定为@(x) sum (x(:))，对图像 input_png 以邻域方式进行处理，代码如下：

```
>> nlfilter(input_png, [2 3], @(x) sum (x(:)))
```

此外，nlfilter()函数允许追加传入其他参数进行调用，这些参数是自定义函数所需要的参数。

先设计一个需要传入 1 个参数的函数 add_diag，代码如下：

```
>> function ret = add_diag(x, param1)
>>     ret = x + diag(param1);
>> endfunction
```

将邻域尺寸指定为[2 3]，将自定义函数指定为 add_diag，add_diag 函数需要追加传入的参数为[1,2;3,4]，对图像 input_png 以邻域方式进行处理，代码如下：

```
>> nlfilter(input_png, [2 3], "add_diag", [1,2;3,4])
```

colfilt()函数用于对图像在列方向以邻域方式进行处理。colfilt()函数允许传入 4 个参数进行调用，第 1 个参数是图像，第 2 个参数是分块尺寸，第 3 个参数是分块类型，第 4 个参数是自定义函数。

将分块尺寸指定为[5 5]，将分块类型指定为 distinct，并将自定义函数指定为 sample_function，对图像 input_png 在列方向以邻域方式进行处理，代码如下：

```
>> colfilt(input_png, [5 5], "distinct", sample_function)
```

此外，colfilt()函数允许追加传入 1 个参数进行调用，这个参数是子尺寸，用于在图像分

块之前先分成小块。

将分块尺寸指定为[5 5],将子尺寸指定为[10 10],将分块类型指定为 distinct,将自定义函数指定为 sample_function,对图像 input_png 在列方向以邻域方式进行处理,代码如下:

```
>> colfilt(input_png, [5 5], [10 10], "distinct", sample_function)
```

此外,colfilt()函数允许追加传入 indexed 参数进行调用,此时需要传入索引图像。

指定 indexed 参数,将分块尺寸指定为[5 5],将子尺寸指定为[10 10],将分块类型指定为 distinct,并将自定义函数指定为 sample_function,对图像 input_gif 在列方向以邻域方式进行处理,代码如下:

```
>> colfilt(input_gif, "indexed", [5 5], [10 10], "distinct",
sample_function)
```

此外,colfilt()函数允许追加传入其他参数进行调用,这些参数是自定义函数所需要的参数。

先设计一个需要传入 1 个参数的函数 add_diag,代码如下:

```
>> function ret = add_diag(x, param1)
>>     ret = x + diag(param1);
>> endfunction
```

再指定 indexed 参数,将分块尺寸指定为[5 5],将子尺寸指定为[10 10],将分块类型指定为 distinct,并将自定义函数指定为 add_diag,add_diag 函数需要追加传入的参数为[1,2;3,4],对图像 input_gif 在列方向以邻域方式进行处理,代码如下:

```
>> colfilt(input_gif, "indexed", [5 5], [10 10], "distinct",
sample_function, [1,2;3,4])
```

## 5.4　几何变换

### 5.4.1　高斯金字塔

impyramid()函数用于创建高斯金字塔。impyramid()函数需要传入两个参数,第 1 个参数是图像,第 2 个参数是高斯金字塔的方向。

将图像指定为 input_png,将高斯金字塔的方向指定为扩大(上一层金字塔),创建高斯金字塔,代码如下:

```
>> impyramid(input_png, "expand")
```

OpenCV 库可以用于创建高斯金字塔,将高斯金字塔的方向指定为扩大。使用 OpenCV 库创建高斯金字塔的代码如下:

```
#第 5 章/pyr_up.py
import sys
import cv2

def pyr_up(image_path, output_path):
    image = cv2.imread(image_path)
    ret = cv2.pyrUp(image)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    pyr_up(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png,将输出图像指定为 output.png,创建高斯金字塔的代码如下:

```
>> python("pyr_up.py", "input.png", "output.png")
```

将输入图像指定为 input_png,将高斯金字塔的方向指定为缩小(下一层金字塔),创建高斯金字塔,代码如下:

```
>> impyramid(input_png, "reduce")
```

OpenCV 库可以用于创建高斯金字塔,将高斯金字塔的方向指定为缩小。使用 OpenCV 库创建高斯金字塔的代码如下:

```
#第 5 章/pyr_down.py
import sys
import cv2

def pyr_down(image_path, output_path):
    image = cv2.imread(image_path)
    ret = cv2.pyrDown(image)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    pyr_down(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png,将输出图像指定为 output.png,创建高斯金字塔的代码如下:

```
>> python("pyr_down.py", "input.png", "output.png")
```

## 5.4.2 镜像复制

**1. 将图像的下半部分以镜像方式复制到图像的上半部分**

OpenCV 库可以用于镜像复制,将图像的下半部分以镜像方式复制到图像的上半部分形成对称。使用 OpenCV 库将图像的下半部分以镜像方式复制到图像的上半部分的代码如下:

```
#第 5 章/symm_bottom_to_top.py
import sys
import cv2

def symm_bottom_to_top(image_path, output_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    w, h = image.shape
    if w <= h:
        image = image[:w, :w]
    else:
        image = image[:h, :h]
    ret = cv2.completeSymm(image, True)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    symm_bottom_to_top(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png，将输出图像指定为 output.png，将图像的下半部分以镜像方式复制到图像的上半部分的代码如下：

```
>> python("symm_bottom_to_top.py", "input.png", "output.png")
```

将图像的下半部分以镜像方式复制到图像的上半部分的结果如图 5-35 所示。

**图 5-35　将图像的下半部分以镜像方式复制到图像的上半部分**

**2. 将图像的上半部分以镜像方式复制到图像的下半部分**

OpenCV 库可以用于镜像复制，将图像的上半部分以镜像方式复制到图像的下半部分。使用 OpenCV 库将图像的上半部分以镜像方式复制到图像的下半部分的代码如下：

```
#第 5 章/symm_top_to_bottom.py
import sys
```

```
import cv2

def symm_top_to_bottom(image_path, output_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    w, h = image.shape
    if w <= h:
        image = image[:w, :w]
    else:
        image = image[:h, :h]
    ret = cv2.completeSymm(image, False)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    symm_top_to_bottom(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png,将输出图像指定为 output.png,将图像的上半部分以镜像方式复制到图像的下半部分的代码如下:

```
>> python("symm_top_to_bottom.py", "input.png", "output.png")
```

将图像的上半部分以镜像方式复制到图像的下半部分的结果如图 5-36 所示。

**图 5-36　将图像的上半部分以镜像方式复制到图像的下半部分**

## 5.4.3　镜像翻转

**1. 垂直镜像翻转**

OpenCV 库可以用于镜像翻转,将图像垂直镜像翻转。使用 OpenCV 库将图像垂直镜像翻转的代码如下:

```
#第 5 章/flip_vertical.py
import sys
import cv2

def flip_vertical(image_path, output_path):
    image = cv2.imread(image_path)
    ret = cv2.flip(image, 0)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    flip_vertical(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png,将输出图像指定为 output.png,将图像垂直镜像翻转的代码如下:

```
>> python("flip_vertical.py", "input.png", "output.png")
```

垂直镜像翻转的结果如图 5-37 所示。

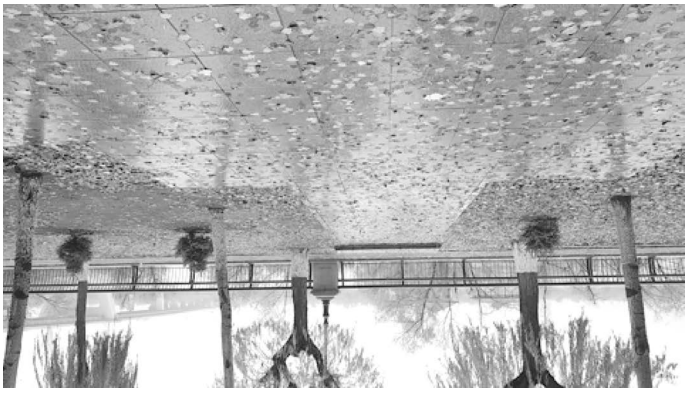

**图 5-37　垂直镜像翻转**

GraphicsMagick 可以垂直镜像翻转图像。将输入图像指定为 input.png,将输出图像指定为 output.png,垂直镜像翻转图像的代码如下:

```
>> system("gm convert input.png -flip output.png")
```

**2. 水平镜像翻转**

OpenCV 库可以用于镜像翻转,将图像水平镜像翻转。使用 OpenCV 库将图像水平镜像翻转的代码如下:

```
#第 5 章/flip_horizontal.py
import sys
import cv2

def flip_horizontal(image_path, output_path):
    image = cv2.imread(image_path)
    ret = cv2.flip(image, 1)
```

```
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    flip_horizontal(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png,将输出图像指定为 output.png,将图像水平镜像翻转的代码如下:

```
>> python("flip_horizontal.py", "input.png", "output.png")
```

水平镜像翻转的结果如图 5-38 所示。

**图 5-38　水平镜像翻转**

GraphicsMagick 可以水平镜像翻转图像。将输入图像指定为 input.png,将输出图像指定为 output.png,水平镜像翻转图像的代码如下:

```
>> system("gm convert input.png -flop output.png")
```

**3. 同时垂直和水平镜像翻转**

OpenCV 库可以用于镜像翻转,将图像同时垂直和水平镜像翻转。使用 OpenCV 库将图像同时垂直和水平镜像翻转的代码如下:

```
#第 5 章/flip_vertical_horizontal.py
import sys
import cv2

def flip_vertical_horizontal(image_path, output_path):
    image = cv2.imread(image_path)
    ret = cv2.flip(image, -1)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    flip_vertical_horizontal(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png,将输出图像指定为 output.png,将图像同时垂直和水平镜像翻转的代码如下:

```
>> python("flip_vertical_horizontal.py", "input.png", "output.png")
```

同时垂直和水平镜像翻转的结果如图 5-39 所示。

**图 5-39　同时垂直和水平镜像翻转**

GraphicsMagick 可以同时垂直和水平镜像翻转图像。将输入图像指定为 input.png，将输出图像指定为 output.png，同时垂直和水平镜像翻转图像的代码如下：

```
>> system("gm convert input.png -flip -flop output.png")
```

OpenCV 库可以用于镜像翻转，将图像按某一维度镜像翻转。使用 OpenCV 库将图像按某一维度镜像翻转的代码如下：

```
#第 5 章/flip_nd.py
import sys
import cv2

def flip_nd(image_path, output_path, axis = 0):
    image = cv2.imread(image_path)
    ret = cv2.flipND(image, axis)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    flip_nd(sys.argv[1], sys.argv[2], int(sys.argv[3]))
```

将输入图像指定为 input.png，将输出图像指定为 output.png，将维度指定为 1，将图像按此维度镜像翻转的代码如下：

```
>> python("flip_vertical_horizontal.py", "input.png", "output.png", "1")
```

### 5.4.4　图像复制

OpenCV 库可以用于图像复制，按水平方向和垂直方向复制。使用 OpenCV 库复制图像的代码如下：

```
#第 5 章/repeat.py
import sys
import cv2

def repeat(image_path, output_path, ny = 1, nx = 1):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.repeat(gray, ny = ny, nx = nx)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    repeat(sys.argv[1], sys.argv[2],
           int(sys.argv[3]), int(sys.argv[4]))
```

将输入图像指定为 input.png,将输出图像指定为 output.png,指定按水平方向复制 1 次,指定按垂直方向复制 2 次,图像复制的代码如下:

```
>> python("repeat.py", "input.png", "output.png", "2", "1")
```

图像复制的结果如图 5-40 所示。

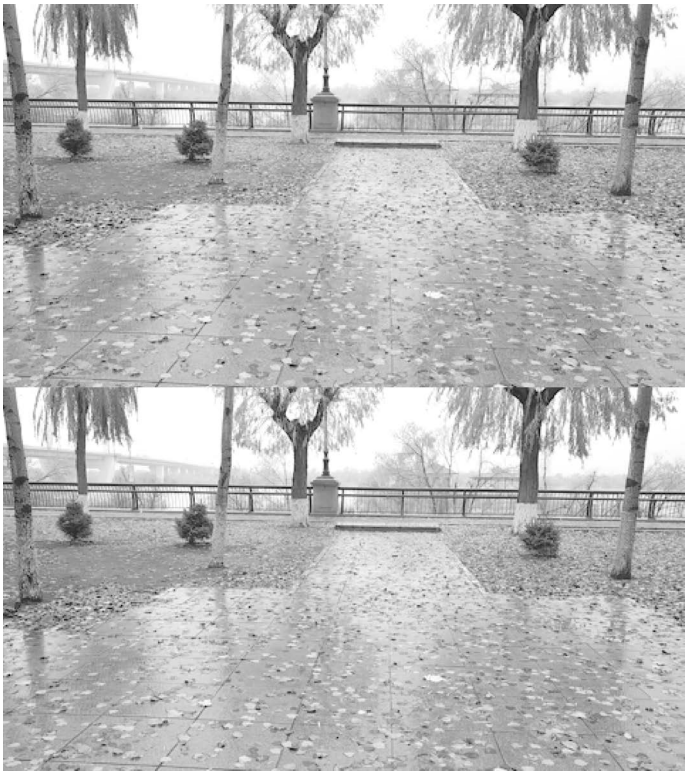

**图 5-40　图像复制**

# 5.5　空间变换

## 5.5.1　图像空间变换

### 1. 平移

imtranslate()函数用于平移图像。imtranslate()函数至少需要传入3个参数，第1个参数是要平移的图像，第2个参数是按 $x$ 轴方向平移的像素数，第3个参数是按 $y$ 轴方向平移的像素数。

将输入图像指定为input_pgm，按 $x$ 轴方向平移100像素，按 $y$ 轴方向平移200像素，平移图像，代码如下：

```
>> imtranslate(input_pgm, 100, 200)
```

平移图像的结果如图5-41所示。

**图5-41　平移图像**

此外，imtranslate()函数允许追加传入第4个参数，这个参数代表平移方式。默认的平移方式为wrap。imtranslate()函数支持的平移方式如表5-5所示。

**表5-5　imtranslate()函数支持的平移方式**

| 平移方式 | 含　义 |
|---|---|
| crop | 裁剪透视变换后的图像的中心部分，裁剪后的尺寸为源图像的尺寸 |
| wrap | 透视变换后的图像的尺寸为源图像的尺寸，将多出来的像素包回源图像 |

将输入图像指定为input_pgm，按 $x$ 轴方向平移1像素，按 $y$ 轴方向平移2像素，平移方式为crop，平移图像，代码如下：

```
>> imtranslate(input_pgm, 1, 2, "crop")
```

### 2. 旋转

imrotate()函数用于旋转图像。imrotate()函数至少需要传入两个参数，第1个参数是图像，第2个参数是旋转角度(逆时针)。

将输入图像指定为 input_png,将旋转角度指定为 45°,旋转图像,代码如下:

```
>> imrotate(input_png, 45)
```

旋转图像的结果如图 5-42 所示。

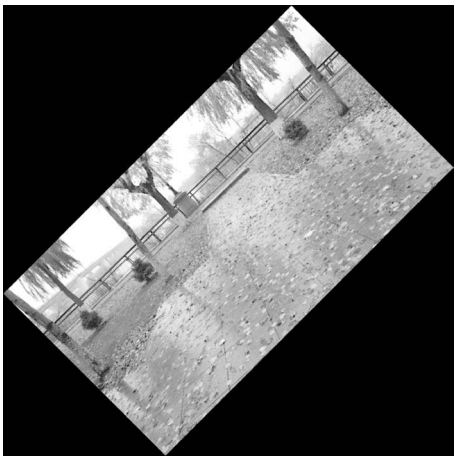

**图 5-42　旋转图像**

此外,imrotate()函数允许追加传入第 3 个参数,这个参数代表插值方式。默认的插值方式为 linear。imrotate()函数支持的插值方式如表 5-6 所示。

**表 5-6　imrotate()函数支持的插值方式**

| 插值方式 | 含　义 |
|---|---|
| linear,bilinear,triangle | 线性插值 |
| pchip | 分段三次 Hermite 插值 |
| nearest | 最近邻插值 |
| cubic,bicubic | 三次插值 |

将输入图像指定为 input_png,将旋转角度指定为 45°,将插值方式指定为 bilinear,旋转图像,代码如下:

```
>> imrotate(input_png, 45, "bilinear")
```

此外,imrotate()函数允许追加传入第 4 个参数,这个参数代表旋转方式。默认的旋转方式为 loose。imrotate()函数支持的旋转方式如表 5-7 所示。

**表 5-7　imrotate()函数支持的旋转方式**

| 旋转方式 | 含　义 |
|---|---|
| loose | 直接返回透视变换后的图像 |
| crop | 裁剪透视变换后的图像的中心部分,裁剪后的尺寸为源图像的尺寸 |

将输入图像指定为 input_png,将旋转角度指定为 45°,将插值方式指定为 bilinear,并将旋转方式指定为 crop,旋转图像,代码如下:

```
>> imrotate(input_png, 45, "bilinear", "crop")
```

此外，imrotate()函数允许追加传入第 5 个参数，这个参数代表填充颜色。默认的填充颜色为 0。

将输入图像指定为 input_png，将旋转角度指定为 45°，将插值方式指定为 bilinear，将旋转方式指定为 crop，并将填充颜色指定为 1，旋转图像，代码如下：

```
>> imrotate(input_png, 45, "bilinear", "crop", 1)
```

OpenCV 库可以用于图像旋转，顺时针旋转 90°。使用 OpenCV 库顺时针旋转 90°，代码如下：

```
#第 5 章/rotate_90_clockwise.py
import sys
import cv2

def rotate_90_clockwise(image_path, output_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.rotate(gray, rotateCode = cv2.ROTATE_90_CLOCKWISE)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    rotate_90_clockwise(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png，将输出图像指定为 output.png，顺时针旋转 90°的代码如下：

```
>> python("rotate_90_clockwise.py", "input.png", "output.png")
```

源图顺时针旋转 90°的结果如图 5-43 所示。

**图 5-43　源图顺时针旋转 90°**

OpenCV 库可以用于图像旋转，源图逆时针旋转 90°。使用 OpenCV 库逆时针旋转 90°，代码如下：

```
#第 5 章/rotate_90_counterclockwise.py
import sys
import cv2

def rotate_90_counterclockwise(image_path, output_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.rotate(gray, rotateCode = cv2.ROTATE_90_COUNTERCLOCKWISE)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    rotate_90_counterclockwise(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png，将输出图像指定为 output.png，逆时针旋转 90°的代码如下：

```
>> python("rotate_90_counterclockwise.py", "input.png", "output.png")
```

源图逆时针旋转 90°的结果如图 5-44 所示。

**图 5-44　源图逆时针旋转 90°**

OpenCV 库可以用于图像旋转，源图旋转 180°。使用 OpenCV 库旋转 180°，代码如下：

```
#第 5 章/rotate_180.py
import sys
import cv2

def rotate_180(image_path, output_path):
```

```
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.rotate(gray, rotateCode = cv2.ROTATE_180)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    rotate_180(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png，将输出图像指定为 output.png，旋转 180°的代码如下：

```
>> python("rotate_180.py", "input.png", "output.png")
```

源图旋转 180°的结果如图 5-45 所示。

**图 5-45　源图旋转 180°**

**3. 缩放**

imresize()函数用于缩放图像。imresize()函数至少需要传入两个参数，第 1 个参数是图像，第 2 个参数是缩放倍数。

将输入图像指定为 input_png，将缩放倍数指定为 0.5，缩放图像，代码如下：

```
>> imresize(input_png, 0.5)
```

此外，如果第 2 个参数是一个二元矩阵，则第 1 个参数是图像，第 2 个参数是缩放尺寸。

将输入图像指定为 input_png，将缩放尺寸指定为[100,300]，缩放图像，代码如下：

```
>> imresize(input_png, [100, 300])
```

缩放的结果如图 5-46 所示。

**图 5-46　缩放**

此外，imresize()函数允许追加传入第 3 个参数，这个参数代表插值方式。默认的插值方式为 cubic。imresize()函数支持的插值方式如表 5-8 所示。

**表 5-8　imresize()函数支持的插值方式**

| 插值方式 | 含　义 |
|---|---|
| nearest，box | 最近邻插值 |
| linear，bilinear，triangle | 双线性插值 |
| cubic，bicubic | 双三次插值 |

将输入图像指定为 input_png，将缩放尺寸指定为[100，200]，并将插值方式指定为 bilinear，缩放图像，代码如下：

```
>> imresize(input_png, [100, 200], "bilinear")
```

此外，imresize()函数允许追加传入的键-值对参数，如表 5-9 所示。

**表 5-9　imresize()函数允许追加传入的键-值对参数**

| 键　参　数 | 含　义 |
|---|---|
| Antialiasing | 如果设为 true，并且宽度或高度的缩放倍数小于 1，则在宽度或高度上应用抗锯齿 |
| Method | 与插值方式的参数相同 |
| OutputSize | 输出的图像尺寸；<br>值为一个二元矩阵 |
| Scale | 输出的图像缩放倍数；<br>值为一个标量或一个二元矩阵 |

将输入图像指定为 input_png，将缩放尺寸指定为[100，200]，并将插值方式指定为 bilinear，应用抗锯齿，缩放图像，代码如下：

```
>> imresize(input_png, [100, 200], "bilinear", "Antialiasing", true)
```

OpenCV 库可以用于图像缩放，缩放到特定尺寸。使用 OpenCV 库将图像缩放到特定尺寸的代码如下：

```
#第 5 章/resize_dsize.py
import sys
import cv2

def resize_dsize(image_path, output_path, dsize = (100, 200)):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.resize(gray, dsize = dsize)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    resize_dsize(sys.argv[1], sys.argv[2],
                 eval(sys.argv[3]))
```

将输入图像指定为 input.png，将输出图像指定为 output.png，将缩放尺寸指定为

(100,200),将图像缩放到特定尺寸的代码如下：

```
>> python("repeat.py", "input.png", "output.png", "(100, 200)")
```

OpenCV 库可以用于图像缩放,按水平方向或垂直方向缩放某个倍数。使用 OpenCV 库按水平方向或垂直方向缩放某个倍数的代码如下：

```
#第 5 章/resize_fx_fy.py
import sys
import cv2

def resize_fx_fy(image_path, output_path, fx = 1, fy = 1):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.resize(gray, fx = fx, fy = fy)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    resize_fx_fy(sys.argv[1], sys.argv[2],
                 float(sys.argv[3]),
                    float(sys.argv[4]))
```

将输入图像指定为 input.png,将输出图像指定为 output.png,按水平方向放大为 2 倍,按垂直方向放大为 3 倍,将图像缩放到特定尺寸的代码如下：

```
>> python("repeat.py", "input.png", "output.png", "2", "3")
```

OpenCV 库可以用于图像缩放,按指定的插值方式缩放。OpenCV 库支持的插值方式如表 5-10 所示。

**表 5-10　OpenCV 库支持的插值方式**

| 插 值 方 式 | 含　义 |
|---|---|
| INTER_NEAREST | 最近邻插值 |
| INTER_LINEAR | 线性插值 |
| INTER_CUBIC | 三次插值 |
| INTER_AREA | 像素面积关系插值 |
| INTER_LANCZOS4 | Lanczos 插值 |
| INTER_LINEAR_EXACT | 位精确线性插值 |
| INTER_NEAREST_EXACT | 位精确最近邻插值 |
| INTER_MAX | 插补码掩码 |
| WARP_FILL_OUTLIERS | 插值时填充目标图像的所有像素 |
| WARP_INVERSE_MAP | 插值的结果反向 |
| WARP_RELATIVE_MAP | — |

使用 OpenCV 库将图像缩放到特定尺寸的代码如下：

```
#第 5 章/resize_interp.py
import sys
```

```
import cv2

def resize_interp(image_path, output_path, dsize = (100, 200),
                  interpolation = cv2.INTER_LINEAR):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.resize(gray, dsize = dsize,
                     interpolation = interpolation)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    resize_interp(sys.argv[1], sys.argv[2],
                  eval(sys.argv[3]),
                     eval(sys.argv[4]))
```

将输入图像指定为 input.png,将输出图像指定为 output.png,将缩放尺寸指定为(100,200),将插值方式指定为 cv2.INTER_LINEAR,将图像缩放到特定尺寸的代码如下:

```
>> python("repeat.py", "input.png", "output.png", "(100, 200)",
"cv2.INTER_LINEAR")
```

**4. 切变**

imshear()函数用于切变图像。imshear()函数至少需要传入 3 个参数,第 1 个参数是图像,第 2 个参数是切变的轴($x$ 或 $y$),第 3 个参数是坡度。

将输入图像指定为 input_pgm,将切变的轴指定为 $x$,将坡度指定为 1.5,切变图像,代码如下:

```
>> imshear(input_pgm, "x", 1.5)
```

切变的结果如图 5-47 所示。

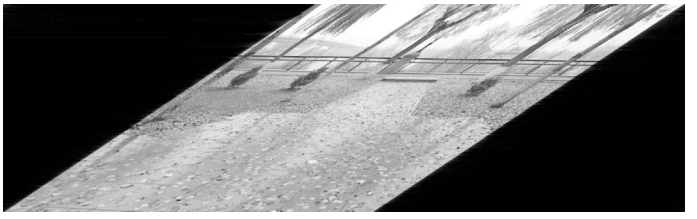

**图 5-47 切变**

此外,imshear()函数允许追加传入第 4 个参数,这个参数代表切变方式。默认的切变方式为 loose。imshear()函数支持的切变方式如表 5-11 所示。

**表 5-11 imshear()函数支持的切变方式**

| 切变方式 | 含义 |
|---|---|
| loose | 直接返回透视变换后的图像 |
| crop | 裁剪透视变换后的图像的中心部分,裁剪后的尺寸为源图像的尺寸 |
| wrap | 透视变换后的图像的尺寸为源图像的尺寸,将多出来的像素包回源图像 |

将输入图像指定为 input_png，将切变的轴指定为 $x$，将坡度指定为 1.5，将切变方式指定为 crop，切变图像，代码如下：

```
>> imshear(input_png, "x", 1.5, "crop")
```

此外，imshear()函数允许追加传入第 5 个参数，这个参数代表启用或禁用 fft 频移。如果此参数为 1，则禁用 fft 频移。默认启用 fft 频移。

将输入图像指定为 input_png，将切变的轴指定为 $x$，将坡度指定为 1.5，将切变方式指定为 crop，禁用 fft 频移，切变图像，代码如下：

```
>> imshear(input_png, "x", 1.5, "crop", 1)
```

**5. 应用任意空间变换矩阵**

imremap()函数用于将任意空间变换矩阵应用到图像。imremap()函数至少需要传入 3 个参数，第 1 个参数是图像，第 2 个参数是应用于 $x$ 轴方向的映射矩阵，第 3 个参数是应用于 $y$ 轴方向的映射矩阵。

将输入图像指定为 input_png，应用于 $x$ 轴方向的映射矩阵为 ones(size(input_png)(1),size(input_png)(2)) * 2，应用于 $y$ 轴方向的映射矩阵为 ones(size(input_png)(1),size(input_png)(2)) * 3，应用此空间变换，代码如下：

```
>> imremap(input_png, ones(size(input_png)(1), size(input_png)(2)) * 2,
ones(size(input_png)(1), size(input_png)(2)) * 3)
```

此外，imremap()函数允许追加传入第 4 个参数，这个参数代表插值方式。默认的插值方式为 linear。imremap()函数支持的插值方式如表 5-12 所示。

**表 5-12　imremap()函数支持的插值方式**

| 插值方式 | 含　义 |
|---|---|
| linear,bilinear,triangle | 线性插值 |
| pchip | 分段三次 Hermite 插值 |
| nearest | 最近邻插值 |
| cubic,bicubic | 三次插值 |
| spline | 样条插值 |

将输入图像指定为 input_png，应用于 $x$ 轴方向的映射矩阵为 ones(size(input_png)(1),size(input_png)(2)) * 2，应用于 $y$ 轴方向的映射矩阵为 ones(size(input_png)(1),size(input_png)(2)) * 3，将插值方式指定为 bilinear，应用此空间变换，代码如下：

```
>> imremap(input_png, ones(size(a)(1), size(a)(2)) * 2, ones(size(a)(1),
size(a)(2)) * 3, "bilinear")
```

此外，imremap()函数允许追加传入第 5 个参数，这个参数代表填充颜色。默认的填充颜色为 0。

将输入图像指定为 input_png，应用于 $x$ 轴方向的映射矩阵为 ones(size(a)(1),size(a)(2)) * 2，

应用于 $y$ 轴方向的映射矩阵为 ones(size(a)(1),size(a)(2)) * 3,将插值方式指定为 bilinear,将填充颜色指定为 1,应用此空间变换,代码如下:

```
>> imremap(input_png, ones(size(input_png)(1), size(input_png)(2)) * 2,
ones(size(input_png)(1), size(input_png)(2)) * 3, "bilinear", 1)
```

**6. 应用任意空间变换对象**

imtransform()函数用于将任意空间变换对象应用到图像。imtransform()函数至少需要传入两个参数,第 1 个参数是图像,第 2 个参数是调用 maketform()函数返回的空间变换对象。

将输入图像指定为 input_png,将空间变换对象指定为 $t$,应用此空间变换,代码如下:

```
>> imtransform(input_png, t)
```

**表 5-13 imtransform()函数支持的插值方式**

| 插值方式 | 含义 |
|---|---|
| bicubic | 双三次插值 |
| bilinear | 双线性插值 |
| nearest | 最近邻插值 |

此外,imtransform()函数允许追加传入插值方式。默认的插值方式为 bilinear。imtransform()函数支持的插值方式如表 5-13 所示。

将输入图像指定为 input_png,将空间变换对象指定为 $t$,将插值方式指定为 bilinear,旋转图像,代码如下:

```
>> imtransform(input_png, t)
```

此外,imtransform()函数允许追加传入键-值对参数。imtransform()函数支持的键-值对参数如表 5-14 所示。

**表 5-14 imtransform()函数支持的键-值对参数**

| 键参数 | 含义 |
|---|---|
| udata | 指定输入空间的水平限制范围;<br>值为一个二元矩阵,代表最小值和最大值 |
| vdata | 指定输入空间的垂直限制范围;<br>值为一个二元矩阵,代表最小值和最大值 |
| xdata | 指定需要的输出空间的水平限制范围;<br>值为一个二元矩阵,代表开始行和结束行 |
| ydata | 指定需要的输出空间的垂直限制范围;<br>值为一个二元矩阵,代表开始列和结束列 |
| xyscale | 指定输出空间中像素的大小;<br>如果值为一个标量,则代表每个像素的宽和高都缩放这个倍数,<br>如果值为一个二元矩阵[x,y],则代表每个像素的宽度缩放 x 倍且高度缩放 y 倍 |
| size | 指定输出图像的尺寸;<br>值为一个二元矩阵,代表最小值和最大值;<br>指定此参数将覆盖 xyscale 参数 |
| fillvalues | 填充颜色 |

## 5.5.2　点的空间变换

**1. 对点应用正向变换**

tformfwd()函数用于对一个或多个点应用正向变换。tformfwd()函数至少需要传入两个参数，第1个参数是调用maketform()函数返回的空间变换对象，第2个参数是$n\times2$的矩阵，代表多个点。

将空间变换对象指定为$t$，将点指定为[1 2;3 4;5 6]，应用正向变换，代码如下：

```
>> tformfwd(t, [1 2;3 4;5 6])
```

此外，tformfwd()函数允许传入3个参数，第1个参数是调用maketform()函数返回的空间变换对象，第2个参数是$n\times1$的矩阵(代表多个点的$x$坐标)，第3个参数是$n\times1$的矩阵(代表多个点的$y$坐标)。

将空间变换对象指定为$t$，将点的$x$坐标指定为[1 2 3]，将点的$y$坐标指定为[4 5 6]，应用正向变换，代码如下：

```
>> tformfwd(t, [1 2 3], [4 5 6])
```

**2. 对点应用逆变换**

tforminv()函数用于对一个或多个点应用逆变换。tforminv()函数至少需要传入两个参数，第1个参数是调用maketform()函数返回的空间变换对象，第2个参数是$n\times2$的矩阵，代表多个点。

将空间变换对象指定为$t$，将点指定为[1 2;3 4;5 6]，应用逆变换，代码如下：

```
>> tforminv(t, [1 2;3 4;5 6])
```

此外，tforminv()函数允许传入3个参数，第1个参数是调用maketform()函数返回的空间变换对象，第2个参数是$n\times1$的矩阵(代表多个点的$x$坐标)，第3个参数是$n\times1$的矩阵(代表多个点的$y$坐标)。

将空间变换对象指定为$t$，将点的$x$坐标指定为[1 2 3]，将点的$y$坐标指定为[4 5 6]，应用逆变换，代码如下：

```
>> tforminv(t, [1 2 3], [4 5 6])
```

## 5.5.3　空间变换对象

**1. 二维仿射变换对象**

affine2d()函数用于创建二维仿射变换对象。affine2d()函数允许不传入参数进行调用，此时将按照eye(3)仿射变换矩阵创建二维仿射变换对象。

创建默认的二维仿射变换对象，代码如下：

```
>> affine2d()
```

此外，affine2d()函数允许传入1个参数，这个参数代表的是一个3×3的仿射变换矩阵。

将仿射变换矩阵指定为[0.1 0.2 0；0.3 0.4 0；0 0 1]，创建二维仿射变换对象，代码如下：

```
>> affine2d([0.1 0.2 0; 0.3 0.4 0; 0 0 1])
```

**2. 三维仿射变换对象**

affine3d()函数用于创建三维仿射变换对象。affine3d()函数允许不传入参数进行调用，此时将按照eye(4)仿射变换矩阵创建三维仿射变换对象。

创建默认的三维仿射变换对象，代码如下：

```
>> affine3d()
```

此外，affine3d()函数允许传入1个参数，这个参数代表的是一个4×4的仿射变换矩阵。

将仿射变换矩阵指定为[0.1 0.2 0.3 0；0.4 0.5 0.6 0；0.7 0.8 0.9 0；0 0 0 1]，创建三维仿射变换对象，代码如下：

```
>> affine3d([0.1 0.2 0.3 0; 0.4 0.5 0.6 0; 0.7 0.8 0.9 0; 0 0 0 1])
```

**3. 坐标变换对象**

cp2tform()函数用于创建与内部空间坐标系和外部空间坐标系相关的坐标变换对象。cp2tform()函数允许传入3个参数，第1个参数是$n\times2$的矩阵(代表多个点在内部空间坐标系的坐标)，第2个参数是$n\times2$的矩阵(代表多个点在外部空间坐标系的坐标)，第3个参数是变换类型。

cp2tform()函数支持的变换类型如表5-15所示。

**表5-15 cp2tform()函数支持的变换类型**

| 变换类型 | 含义 | 变换类型 | 含义 |
|---|---|---|---|
| nonreflective similarity | 非反射相似形变换 | projective | 投影变换 |
| similarity | 相似形变换 | polynomial | 多项式变换 |
| affine | 仿射变换 | | |

将多个点在内部空间坐标系的坐标指定为[1,2,3;4,5,6]，将多个点在外部空间坐标系的坐标指定为[10,20,30;40,50,60]，将变换类型指定为非反射相似形变换，创建坐标变换对象，代码如下：

```
>> cp2tform([1,2,3;4,5,6], [10,20,30;40,50,60], "nonreflective
similarity")
```

此外，cp2tform()函数允许追加传入第4个参数，这个参数代表多项式的阶数。多项式变换可选2、3或4阶多项式。

将多个点在内部空间坐标系的坐标指定为[1,2,3;4,5,6]，将多个点在外部空间坐标系

的坐标指定为[10,20,30;40,50,60],将变换类型指定为多项式变换,将多项式的阶数指定为2,创建坐标变换对象,代码如下:

```
>> cp2tform([1,2,3;4,5,6], [10,20,30;40,50,60], "polynomial", 2)
```

**4. 任意变换对象**

maketform()函数用于创建任意变换对象。maketform()函数允许传入两个参数,第1个参数是变换类型,第2个参数是变换矩阵。

maketform()函数支持的变换类型如表5-16所示。

**表5-16　maketform()函数支持的变换类型**

| 变换类型 | 含　义 |
|---|---|
| projective | 投影变换 |
| affine | 仿射变换 |
| custom | 自定义变换 |

将变换类型指定为仿射变换,将变换矩阵指定为[1,2,3;4,5,6;7,8,9],创建坐标变换对象,代码如下:

```
>> maketform("affine", [1,2,3;4,5,6;7,8,9])
```

**注意**:如果将变换类型指定为自定义变换,则不允许只传入两个参数。

此外,maketform()函数允许传入3个参数,第1个参数是变换类型,第2个参数是指定多个点在内部空间坐标系的坐标,第3个参数是指定多个点在外部空间坐标系的坐标。

将变换类型指定为仿射变换,将多个点在内部空间坐标系的坐标指定为[1,2,3;4,5,6],将多个点在外部空间坐标系的坐标指定为[10,20,30;40,50,60],创建坐标变换对象,代码如下:

```
>> maketform("affine", [1,2,3;4,5,6], [10,20,30;40,50,60])
```

**注意**:如果将变换类型指定为投影变换,则在内部空间坐标系的坐标和在外部空间坐标系的坐标一般为4×2的矩阵,即图像的4个角点;如果将变换类型指定为仿射变换,则在内部空间坐标系的坐标和在外部空间坐标系的坐标一般为3×2的矩阵,即图像的3个角点;如果将变换类型指定为自定义变换,则不允许只传入3个参数。

此外,maketform()函数允许传入6个参数,第1个参数必须是custom,第2个参数是多个点在内部空间坐标系的维数,第3个参数是多个点在外部空间坐标系的维数,第4个参数是正向变换函数,第5个参数是逆变换函数,第6个参数是一个包含正向变换矩阵和逆变换矩阵的结构体,其中T代表正向变换矩阵,Tinv代表逆变换矩阵。

将变换类型指定为自定义变换,将多个点在内部空间坐标系的维数指定为2,将多个点在外部空间坐标系的维数指定为2,将正向变换函数指定为@forward,将逆向变换函数指定为@inverse,将变换矩阵的结构体指定为struct("T",[1,2,3;4,5,6;7,8,9],"Tinv",[9,8,7;6,5,4;3,2,1]),创建坐标变换对象,代码如下:

```
>> maketform("custom", 2, 2, @forward, @inverse, struct("T",
[1,2,3;4,5,6;7,8,9], "Tinv", [9,8,7;6,5,4;3,2,1]))
```

### 5.5.4 仿射变换

OpenCV 库可以用于仿射变换。使用 OpenCV 库进行仿射变换的代码如下：

```
#第 5 章/affine.py
import sys
import cv2
import numpy as np

def affine(image_path, output_path, m = np.array([[1, 2, 3], [4, 5, 6]])):
    image = cv2.imread(image_path)
    h, w = image.shape[:2]
    ret = cv2.warpAffine(image, m = m, dsize = (w, h))
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    affine(sys.argv[1], sys.argv[2], eval(sys.argv[3]))
```

将输入图像指定为 input.png，将输出图像指定为 output.png，将仿射变换矩阵指定为 np.array([[1,2,30],[4,0.5,60]])，仿射变换的代码如下：

```
>> python("affine.py", "input.png", "output.png", "\"np.float32([[1, 2,
30], [4, 0.5, 60]])\"")
```

仿射变换的结果如图 5-48 所示。

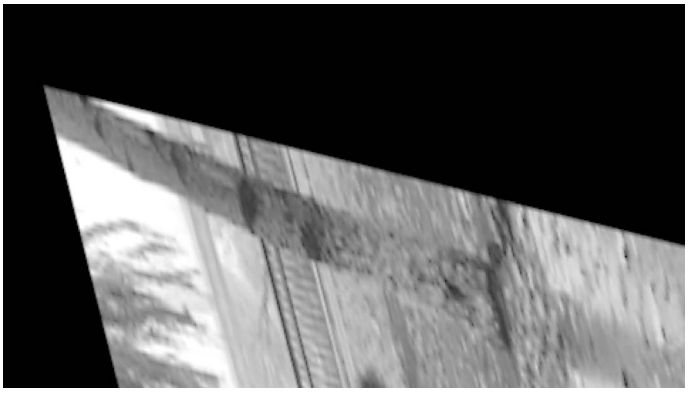

**图 5-48 仿射变换**

### 5.5.5 透视变换

imperspectivewarp()函数用于对图像进行透视变换。imperspectivewarp()函数至少需要传入两个参数，第 1 个参数是图像，第 2 个参数是变换矩阵。

将输入图像指定为 input_png，将变换矩阵指定为 diag([1,2,3])，对图像进行透视变换，代码如下：

```
>> imperspectivewarp(input_png, diag([1, 2, 3]))
```

透视变换的结果如图 5-49 所示。

此外，imperspectivewarp()函数允许追加传入第 3 个参数，这个参数代表插值方式。默认的插值方式为 linear。imperspectivewarp()函数支持的插值方式如表 5-17 所示。

图 5-49　透视变换

表 5-17　imperspectivewarp()函数支持的插值方式

| 插值方式 | 含　义 |
|---|---|
| linear | 线性插值 |
| cubic | 三次插值 |
| nearest | 最近邻插值 |
| bicubic | 双三次插值 |
| bilinear | 双线性插值 |
| triangle | 三角插值 |
| box | 盒子插值 |

将输入图像指定为 input_png，将变换矩阵指定为 diag([1,1,0.5])，将插值方式指定为 bilinear，对图像进行透视变换，代码如下：

```
>> imperspectivewarp(input_png, diag([1, 1, 0.5]), "bilinear")
```

此外，imperspectivewarp()函数允许追加传入第 4 个参数，这个参数代表透视方式。默认的透视方式为 loose。imperspectivewarp()函数支持的透视方式如表 5-18 所示。

表 5-18　imperspectivewarp()函数支持的透视方式

| 透视方式 | 含　义 |
|---|---|
| loose | 直接返回透视变换后的图像 |
| crop | 裁剪透视变换后的图像的中心部分，裁剪后的尺寸为源图像的尺寸 |
| same | 返回透视变换后的图像，但保持源图像的坐标系统不变 |

将输入图像指定为 input_png，将变换矩阵指定为 diag([1,1,0.5])，将插值方式指定为 bilinear，并将透视方式指定为 crop，对图像进行透视变换，代码如下：

```
>> imperspectivewarp(input_png, diag([1, 1, 0.5]), "bilinear", "crop")
```

此外，imperspectivewarp()函数允许追加传入第 5 个参数，这个参数代表填充颜色。默认的填充颜色为 0。

将输入图像指定为 input_png，将变换矩阵指定为 diag([1,1,0.5])，将插值方式指定为 bilinear，将透视方式指定为 crop，并将填充颜色指定为 1，对图像进行透视变换，代码如下：

```
>> imperspectivewarp(input_png, diag([1, 1, 0.5]), "bilinear", "crop", 1)
```

OpenCV 库可以用于透视变换。使用 OpenCV 库对图像进行透视变换的代码如下：

```
#第 5 章/warp_perspective.py
import sys
import cv2
import numpy as np

def warp_perspective(image_path, output_path,
                             transform_matrix = np.float32(
                                 [[1,2,3],[4,5,6],[7,8,9]]
                                 )):
    image = cv2.imread(image_path)
    h, w = image.shape[:2]
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.warpPerspective(gray, transform_matrix, dsize = (w, h))
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
    warp_perspective(sys.argv[1], sys.argv[2], eval(sys.argv[3]))
```

将输入图像指定为 input.png，将输出图像指定为 output.png，并将变换矩阵指定为 np.float32([[1,2,3],[4,5,6],[7,8,9]])，对图像进行透视变换的代码如下：

```
>> python("warp_perspective.py", "input.png", "output.png",
"\"np.float32([[1,2,3],[4,5,6],[7,8,9]])\"")
```

OpenCV 库可以对每个像素独立进行透视变换。使用 OpenCV 库对每个像素独立进行透视变换的代码如下：

```
#第 5 章/perspective_transform.py
import sys
import cv2
import numpy as np

def perspective_transform(image_path, output_path,
                             transform_matrix = np.float32(
[[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]
                                 )):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.perspectiveTransform(gray, transform_matrix)
    cv2.imwrite(output_path, ret)

if __name__ == "__main__":
        perspective_transform(sys.argv[1], sys.argv[2],
eval(sys.argv[3]))
```

将输入图像指定为 input.png，将输出图像指定为 output.png，并将变换矩阵指定为

np.float32([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])，对每个像素独立进行透视变换的代码如下：

```
>> python("perspective_transform.py", "input.png", "output.png",
"\"np.float32([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])\"")
```

## 5.5.6　角点检测

### 1. 角点的特征值和特征向量

OpenCV 库可以用于角点检测，返回角点的特征值和特征向量。使用 OpenCV 库返回角点的特征值和特征向量的代码如下：

```
#第 5 章/corner.py
import sys
import cv2
import numpy as np

def corner(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    points = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.cornerEigenValsAndVecs(points, 5, 3)
    print(ret)
    return ret

if __name__ == "__main__":
    corner(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png，将输出图像指定为 output.png，返回角点的特征值和特征向量的代码如下：

```
>> python("corner.py", "input.png", "output.png")
ans = [[[ 1.14501943e-03   2.89813412e-04  -8.96205366e-01   4.43639398e-01
     -4.43639398e-01  -8.96205366e-01]
  [ 1.03405549e-03   2.60062166e-04  -9.00474906e-01   4.34908032e-01
     -4.34908032e-01  -9.00474906e-01]
  [ 1.39679387e-03   2.99169216e-04  -9.87369776e-01   1.58432588e-01
     -1.58432588e-01  -9.87369776e-01]
  ...
  [ 4.25450417e-05   5.12894258e-06   9.93507147e-01  -1.13769636e-01
     -1.13769636e-01  -9.93507147e-01]
  [ 4.16319466e-07   1.98828516e-07   3.82690489e-01   9.23876643e-01
     -9.23876643e-01   3.82690489e-01]
  [ 0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00]]

[[ 9.46259301e-04   2.87573901e-04  -9.09257412e-01   4.16234195e-01
     -4.16234225e-01  -9.09257412e-01]
```

```
    [ 9.67575936e-04   2.36576379e-04  -9.22334969e-01   3.86391252e-01
      -3.86391252e-01  -9.22334969e-01]
    [ 1.46711455e-03   2.71447527e-04  -9.84809518e-01   1.73638284e-01
      -1.73638284e-01  -9.84809518e-01]
    ...
    [ 2.85709375e-05   8.10732308e-06   9.71283734e-01  -2.37924263e-01
      -2.37924263e-01  -9.71283734e-01]
    [ 1.93442520e-06   4.49317724e-07   1.84384286e-01  -9.82854247e-01
      -9.82854247e-01  -1.84384286e-01]
    [ 1.61052674e-06   2.34963906e-07   2.29751453e-01  -9.73249316e-01
      -9.73249316e-01  -2.29751453e-01]]

  [[ 9.45304579e-04   2.18401678e-04  -8.69600356e-01   4.93756235e-01
      -4.93756235e-01  -8.69600356e-01]
    [ 1.16077729e-03   2.00314622e-04  -9.18134987e-01   3.96267712e-01
      -3.96267712e-01  -9.18134987e-01]
    [ 1.69677264e-03   2.41866335e-04  -9.77319956e-01   2.11768135e-01
      -2.11768135e-01  -9.77319956e-01]
    ...
    [ 2.49586101e-05   7.87500903e-06   9.97551382e-01  -6.99375048e-02
      -6.99375048e-02  -9.97551382e-01]
    [ 4.28618023e-06   1.63470531e-06   5.95803916e-01  -8.03129911e-01
      -8.03129911e-01  -5.95803916e-01]
    [ 4.53862503e-06   1.30537205e-06   5.34740269e-01  -8.45016479e-01
      -8.45016479e-01  -5.34740269e-01]]

  ...

  [[ 6.33239746e-03   2.24168855e-03  -7.41683424e-01   6.70750082e-01
      -6.70750082e-01  -7.41683424e-01]
    [ 6.01595826e-03   2.50284327e-03  -3.64496887e-01   9.31204617e-01
      -9.31204617e-01  -3.64496887e-01]
    [ 8.80898908e-03   2.86775082e-03  -2.06344556e-02   9.99787092e-01
      -9.99787092e-01  -2.06344556e-02]
    ...
    [ 2.71266000e-03   1.26088841e-03  -1.91743568e-01   9.81445074e-01
      -9.81445074e-01  -1.91743568e-01]
    [ 2.13197363e-03   1.14768790e-03  -6.02409720e-01   7.98187017e-01
      -7.98187017e-01  -6.02409720e-01]
    [ 1.80175714e-03   1.08728535e-03  -8.99555743e-01   4.36805993e-01
      -4.36805993e-01  -8.99555743e-01]]

  [[ 5.83464280e-03   1.50439242e-04  -6.08798683e-01   7.93324769e-01
      -7.93324769e-01  -6.08798683e-01]
    [ 5.84169850e-03   7.06552470e-04  -3.51162165e-01   9.36314642e-01
      -9.36314642e-01  -3.51162165e-01]
    [ 8.54591280e-03   1.40787510e-03  -9.47326571e-02   9.95502770e-01
      -9.95502770e-01  -9.47326571e-02]
    ...
```

```
  [ 2.13315850e-03  9.45041946e-04 -2.35608131e-01  9.71848130e-01
    -9.71848130e-01 -2.35608131e-01]
  [ 1.81241590e-03  6.39102480e-04 -4.74356115e-01  8.80333066e-01
    -8.80333066e-01 -4.74356115e-01]
  [ 1.65940716e-03  6.97071024e-04 -6.24990284e-01  7.80632555e-01
    -7.80632555e-01 -6.24990284e-01]]

 [[ 1.67782127e-03  1.33482128e-04 -4.45609659e-01  8.95227373e-01
    -8.95227373e-01 -4.45609659e-01]
  [ 2.15225830e-03  2.54047482e-04 -1.83926851e-01  9.82939959e-01
    -9.82939959e-01 -1.83926851e-01]
  [ 2.82179099e-03  4.76479356e-04 -2.38750707e-02  9.99714971e-01
    -9.99714971e-01 -2.38750707e-02]
  ...
  [ 1.15284999e-03  4.56838869e-04 -9.56762731e-01  2.90869534e-01
    -2.90869534e-01 -9.56762731e-01]
  [ 1.01499236e-03  2.52827711e-04 -8.97153497e-01  4.41718936e-01
    -4.41718966e-01 -8.97153497e-01]
  [ 1.46499742e-03  2.25737065e-04 -9.17474627e-01  3.97794306e-01
    -3.97794306e-01 -9.17474627e-01]]]
```

**2. Harris 角点检测**

OpenCV 库可以用于 Harris 角点检测。使用 OpenCV 库对图像 Harris 角点进行检测的代码如下：

```
#第 5 章/corner_harris.py
import sys
import cv2
import numpy as np

def corner_harris(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    points = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.cornerHarris(points, 3, 5, 0.04)
    print(ret)
    return ret

if __name__ == "__main__":
    corner_harris(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png，将输出图像指定为 output.png，对图像 Harris 角点进行检测的代码如下：

```
>> python("corner_harris.py", "input.png", "output.png")
ans = [[ 3.0927035e-06  4.7796252e-06 -2.2270428e-06 ... -4.1788122e-11
    1.0521255e-14  0.0000000e+00]
 [ 4.7725794e-06  7.6710521e-06  1.2352284e-06 ...  4.3537188e-12
    2.0133755e-13 -3.2392790e-14]
```

```
[ 3.9198103e-06   8.0588870e-06  -8.9221794e-07 ...   4.9946569e-10
    3.3147929e-12  -2.6629567e-12]
...
[ 4.6312362e-06  -2.0985994e-05   1.8438735e-04 ...   8.9653549e-05
    3.5211277e-05   1.0233603e-05]
[ -4.8357424e-07  -5.3127724e-06  -6.1018727e-06 ...   2.3416495e-05
    1.0241789e-05   4.0832965e-06]
[ 6.6005763e-11   1.2821244e-09   4.0634571e-09 ...   7.7605046e-07
    8.5554493e-06   5.1563320e-06]]
```

**3. 最小特征向量**

OpenCV 库可以用于角点检测，返回最小特征向量。使用 OpenCV 库返回最小特征向量的代码如下：

```
#第 5 章/corner_min_eigen_val.py
import sys
import cv2
import numpy as np

def corner_min_eigen_val(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    points = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret = cv2.cornerMinEigenVal(points, 30)
    print(ret)
    return ret

if __name__ == "__main__":
    corner_min_eigen_val(sys.argv[1], sys.argv[2])
```

将输入图像指定为 input.png，将输出图像指定为 output.png，返回最小特征向量的代码如下：

```
>> python("corner_min_eigen_val.py", "input.png", "output.png")
ans = [[0.00086989 0.00086989 0.00086872 ... 0.01370446 0.01338646 0.01265368]
  [0.00086989 0.00086989 0.00086872 ... 0.01370446 0.01338646 0.01265368]
  [0.00087691 0.00087691 0.00087573 ... 0.01332753 0.0131539  0.01261878]
  ...
  [0.00317586 0.00317586 0.00317231 ... 0.00174782 0.00171562 0.00168094]
  [0.003131   0.003131   0.003121   ... 0.00175749 0.00172268 0.00168424]
  [0.00318256 0.00318256 0.0031609  ... 0.00185442 0.00180687 0.00175334]]
```

## 5.6 二值图像打包解包

### 5.6.1 二值图像打包

bwpack()函数用于打包二值图像，每 32 位二值图像会被打包为一个 uint32 数字，打包

后的图像可能会占用更小的空间。bwpack()函数允许传入1个参数,这个参数代表二值图像。

打包图像input_pbm,代码如下:

```
>> input_pbm_packed = bwpack(input_pbm)
```

此外,bitpack()函数也可以用于打包二值图像。bitpack()函数允许传入两个参数,第1个参数是由logical类型的变量组成的矩阵,第2个参数是打包后的类型。bitpack()函数将1个logical类型的变量视为1位,按打包后的类型将每若干位数的logical类型的变量打包为一个数字。打包后的类型和位数对照表如表5-19所示。

**表5-19　打包后的类型和位数对照表**

| 打包后的类型 | 含　义 | 位　数 |
|---|---|---|
| double | 双精度浮点型 | 64位 |
| single | 单精度浮点型 | 32位 |
| double complex | 双精度浮点型,复数 | 128位 |
| single complex | 单精度浮点型,复数 | 64位 |
| char | 字符型 | 8位 |
| int8 | 8位整型 | 8位 |
| int16 | 16位整型 | 16位 |
| int32 | 32位整型 | 32位 |
| int64 | 64位整型 | 64位 |
| uint8 | 无符号8位整型 | 8位 |
| uint16 | 无符号16位整型 | 16位 |
| uint32 | 无符号32位整型 | 32位 |
| uint64 | 无符号64位整型 | 64位 |

**注意**:bitpack()函数在打包时,logical类型的变量必须是位数的整数倍,而图像的像素数不一定恰好是位数的整数倍,所以要进行额外处理。在解包时也同样要进行额外处理。

将打包后的类型指定为double(64位),使用bitpack()函数打包图像input_pbm,代码如下:

```
>> dims = size(input_pbm);
>> out_rows = ceil(dims(1) / 64);
>> in_rows = out_rows * 64;
>> result = resize (input_pbm, [in_nrows dims(2:end)]);
>> result = reshape (bitpack(result(:), "double"), [in_nrows dims(2:end)]);
```

## 5.6.2　二值图像解包

bwunpack()函数用于解包二值图像,每个uint32数字会被解包为32位二值图像。

bwunpack()函数允许传入 1 个参数,这个参数代表二值图像。

解包图像 input_pbm_packed,代码如下:

```
>> bwunpack(input_pbm_packed)
```

此外,bitunpack()函数也可以用于解包二值图像。bitunpack()函数允许传入 1 个参数,这个参数是由打包后的类型的变量组成的矩阵。bitunpack()函数将按打包后的类型将变量解包为 logical 类型的变量。

使用 bitunpack()函数解包图像 result,代码如下:

```
>> in_result = reshape(bitunpack(result), [in_nrows dims(2:end)]);
>> width = size(input_pbm)(1);
>> height = size(input_pbm)(2);
>> unpack_result = zeros(width, height);
>> for width_index = 1:width
>>     for height_index = 1:height
>>         unpack_result(width_index, height_index) = in_result(width_index, height_index);
>>     endfor
>> endfor
```