Edges

1. canny方法

(1)参数及其解释

- 接口：

```
Edges(image).canny()
```

- 参数：
  - image——图片地址(str类型)
- 解释：将输入图片进行边缘化

(2)数学原理

Sobel算子检测方法对灰度渐变和噪声较多的图像处理效果较好，sobel算子对边缘定位不是很准确，图像的边缘不止一个像素；当对精度要求不是很高时，是一种较为常用的边缘检测方法。是带有方向的。

Sobel卷积因子为：

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

该算子包含两组3x3的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。如果以A代表原始图像，Gx及Gy分别代表经横向及纵向边缘检测的图像灰度值，其公式如下：

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

(3)方法示例及效果展示

```
from augmenters.Edges import Edges  #导包
if __name__ == '__main__':
    Edges("leaf.jpg").canny()
```

原图：

变换： Edges("leaf.jpg").canny()

Flip

1.fliplr方法

(1)参数及其解释

- 接口：

```
Flip(image).fliplr()
```

- 参数：
  - image——图片地址(str类型)
- 解释：将图片水平翻转

(2)数学原理

利用wrapAffine实现翻转

(注： `width` 代表图像的宽度； `height` 代表图像的高度)

水平翻转的变换矩阵

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} width \\ 0 \end{bmatrix}$$

$$M = \begin{bmatrix} -1 & 0 & width \\ 0 & 1 & 0 \end{bmatrix}$$

(3)方法示例及效果展示

```
from augmenters.Flip import Flip    #导包
if __name__ == '__main__':
    Flip("leaf.jpg").fliplr()
```

原图：

变换： Flip("leaf.jpg").fliplr()

2.flipud方法

(1)参数及其解释

- 接口：

```
Flip(image).flipud()
```

- 参数：
  - image——图片地址(str类型)
- 解释：将图片垂直翻转

(2)数学原理

利用wrapAffine实现翻转

(注： `width` 代表图像的宽度； `height`代表图像的高度)

垂直翻转的变换矩阵

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ height \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & height \end{bmatrix}$$

(3)方法示例及效果展示

```
from augmenters.Flip import Flip    #导包
if __name__ == '__main__':
    Flip("leaf.jpg").flipud()
```

原图：

变换： Flip("leaf.jpg").flipud()

3.flipudlr方法

(1)参数及其解释

- 接口：

```
Flip(image).flipudlr()
```

- 参数：
  - image——图片地址(str类型)
- 解释：将图像垂直水平翻转

(2)数学原理

利用wrapAffine实现翻转

(注： `width` 代表图像的宽度； `height` 代表图像的高度)

同时进行水平翻转与垂直翻转

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} width \\ height \end{bmatrix}$$

$$M = \begin{bmatrix} -1 & 0 & width \\ 0 & -1 & height \end{bmatrix}$$

(3)方法示例及效果展示

```
from augmenters.Flip import Flip    #导包
if __name__ == '__main__':
    Flip("leaf.jpg").flipudlr()
```

原图：

变换： Flip("leaf.jpg").flipudlr()

Geometric

1.rotate方法

(1)参数及其解释

- 接口：

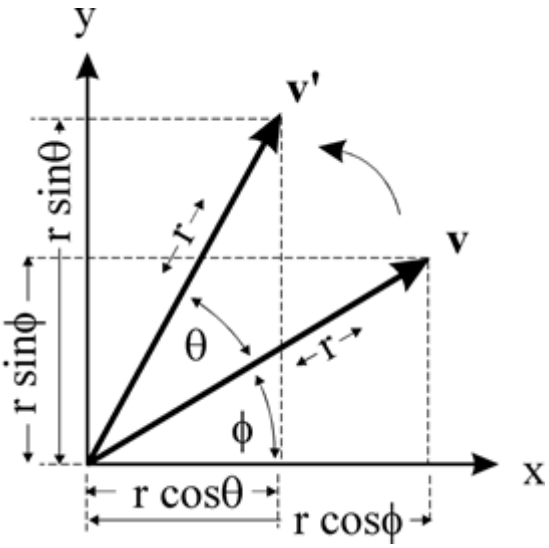
```
Geometric(image).rotate(angle,x,y)
```

- 参数：
  - image——图片地址(str类型)
  - angle——图片旋转角度(int类型)
  - x——旋转中心横坐标(默认值 = width/2 int类型)
  - y——旋转中心纵坐标(默认值 = height/2 int类型)
- 解释：确定旋转点坐标(X,Y)，对图像进行angle角度进行旋转

(2)数学原理

利用wrapAffine实现缩放

数学原理：围绕原点进行旋转



$$x = r * cos(\phi)$$

$$\begin{aligned} x' &= r * cos(\phi + \theta) \\ &= r * cos(\phi) * cos(\theta) - r * sin(\phi) * sin(\theta) \end{aligned}$$

$$y = r * sin(\phi)$$

$$\begin{aligned} y' &= r * sin(\phi + \theta) \\ &= r * sin(\phi) * cos(\theta) + r * cos(\phi) * sin(\theta) \end{aligned}$$

由此我们得出

$$x' = x * cos(\theta) - y * sin(\theta)$$

$$y' = x * sin(\theta) + y * cos(\theta)$$

所以对于矩阵变换

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$M = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \end{bmatrix}$$

> 注意，这里我们进行公式推导的时候，参照的原点是在左下角， 而在OpenCV中图像的原点在图像的左上角，所以我们在代码里面对theta取反。
>
> 我们可以利用math包中的三角函数。但是有一点需要注意 ：三角函数输入的角度是弧度制而不是角度制。
>
> 我们需要使用radians(x) 函数， 将角度转变为弧度。

```
import math
math.radians(180)
```

那么如何围绕任意点进行旋转呢？

可以先把当前的旋转中心点平移到原点处， 在原点处旋转后再平移回去。

假定旋转中心为 (cx,cy)

$M_{translation}$ 为平移矩阵 $M_{translation}^{-1}$ 为平移矩阵的逆矩阵 $M_{rotation}$ 为原点旋转矩阵

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = M_{translation}^{-1}(M_{rotation} * (M_{translation} * \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}))$$

其中

$$M_{translation} = \begin{bmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{translation}^{-1} = \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{rotation} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

所以

$$
\begin{aligned}
M &= M_{translation}^{-1} \times M_{rotation} \times M_{translation} \\
&= \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} cos(\theta) & -sin(\theta) & (1-cos(\theta)) * c_x + sin(\theta) * c_y \\ sin(\theta) & cos(\theta) & -sin(\theta) * c_x + (1-cos(\theta)) * c_y \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

(3)方法示例及效果展示

```
from augmenters.Geometric import Geometric  #导包
if __name__ == '__main__':
    Geometric("leaf.jpg").rotate(45)
    Geometric("leaf.jpg").rotate(90)
    Geometric("leaf.jpg").rotate(45,50,50)
    Geometric("leaf.jpg").rotate(90,50,50)
```

原图：

变换： 围绕中心旋转45° Geometric("leaf.jpg").rotate(45)

围绕中心旋转90° Geometric("leaf.jpg").rotate(90)

围绕坐标(50,50)旋转45° Geometric("leaf.jpg").rotate(45,50,50)

围绕坐标(50,50)旋转90° Geometric("leaf.jpg").rotate(90,50,50)

2.move方法

(1)参数及其解释

- 接口：

```
Geometric(image).move(x,y)
```

- 参数：
  - image——图片地址(str类型)
  - x——水平位移(int类型)
  - y——垂直位移(int类型)
- 解释：将图像向右平移x个单位，向下平移y个单位

(2)数学原理

平移可以说是最简单的一种空间变换。其表达式为：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

$$x' = x + b_0$$

$$y' = y + b_1$$

其中(b0,b1) 是偏移量。

(3)方法示例及效果展示

```
from augmenters.Geometric import Geometric  #导包
if __name__ == '__main__':
    Geometric("leaf.jpg").move(10,10)
```

原图：

变换： 水平位移10px 垂直位移10px Geometric("leaf.jpg").move(10,10)

3.zoom方法

(1)参数及其解释

- 接口：

```
Geometric(image).zoom(hx,wx)
```

- 参数：
  - image——图片地址(str类型)
  - wx——水平缩放因子(float类型)
  - hx——垂直缩放因子(float类型)
- 解释：将图像按照水平缩放因子wx与垂直缩放因子hx按照一定比例进行缩放

(2)数学原理

利用wrapAffine实现缩放 数学原理 对图像的伸缩变换的变换矩阵M为

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

其中，

fx：代表x轴的焦距(缩放因子)

fy：代表y轴的焦距(缩放因子)

则可以得出以下式子：

$$x' = f_x * x$$

$$y' = f_y * y$$

(3)方法示例及效果展示

```
from augmenters.Geometric import Geometric  #导包
if __name__ == '__main__':
    Geometric("leaf.jpg").zoom(1,0.5)
    Geometric("leaf.jpg").zoom(0.5, 0.5)
```

原图：

变换： 水平缩放1垂直缩放0.5 Geometric("leaf.jpg").zoom(1,0.5)

水平缩放0.5垂直缩放0.5 Geometric("leaf.jpg").zoom(0.5, 0.5)

4.crop方法

(1)参数及其解释

- 接口：

```
Geometric(image).crop(x1,y1,x2,y2)
```

- 参数：
  - image——图片地址(str类型)
  - x1——左上角横坐标(int类型)
  - y1——左上角纵坐标(int类型)
  - x2——右上角横坐标(int类型)
  - y2——左上角纵坐标(int类型)
- 解释：根据左上(x1,y1)右下角(x2,y2)坐标将图像完成裁剪

(2)数学原理

直接根据左上右下角坐标完成裁剪

(3)方法示例及效果展示

```
from augmenters.Geometric import Geometric  #导包
if __name__ == '__main__':
    Geometric("leaf.jpg").crop(20,20,100,100)
```

原图：

变换： Geometric("leaf.jpg").crop(20,20,100,100)

Imgcorruptlike

1.saltPepperNoise方法

(1)参数及其解释

- 接口：

```
Imgcorruptlike(image).saltPepperNoise(proportion)
```

- 参数：
  - image——图片地址(str类型)
  - proportion——噪音比(默认值 = 0.05 float类型)
- 解释：根据噪音比proportion在图像像素点中随机添加噪音，修改像素点为0(黑噪音)或255(白噪音)

(2)数学原理

根据噪音比在图像像素点中随机添加噪音，修改像素点为0(黑噪音)或255(白噪音)。

(3)方法示例及效果展示

```
from augmenters.Imgcorruptlike import Imgcorruptlike    #导包
if __name__ == '__main__':
    Imgcorruptlike("leaf.jpg").saltPepperNoise(0.1)
    Imgcorruptlike("leaf.jpg").saltPepperNoise(0.2)
```

原图：

变换： 噪音比为0.1 Imgcorruptlike("leaf.jpg").saltPepperNoise(0.1)

噪音比为0.2 Imgcorruptlike("leaf.jpg").saltPepperNoise(0.2)

2.gaussNoise方法

(1)参数及其解释

- 接口：

```
Imgcorruptlike(image).gaussNoise()
```

- 参数：
  - image——图片地址(str类型)
- 解释：将图像按照高斯分布进行加噪处理

(2)数学原理

高斯噪声是指它的概率密度函数服从高斯分布（即正态分布）的一类噪声

与椒盐噪声相似（Salt And Pepper Noise），高斯噪声（gauss noise）也是数字图像的一个常见噪声。

椒盐噪声是出现在随机位置、噪点深度基本固定的噪声，高斯噪声与其相反，是几乎每个点上都出现噪声、噪点深度随机的噪声。

算法步骤：

通过概率论里关于正态分布的有关知识可以很简单的得到其计算方法，高斯噪声的概率密度服从高斯分布（正态分布）其中有means（平均值）和sigma（标准方差）两个参数。

高斯分布（正态分布）：

$$f(x)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

(3)方法示例及效果展示

```
from augmenters.Imgcorruptlike import Imgcorruptlike    #导包
if __name__ == '__main__':
    Imgcorruptlike("leaf.jpg").gaussNoise()
```

原图：

变换： Imgcorruptlike("leaf.jpg").gaussNoise()

3.randomNoise方法

(1)参数及其解释

- 接口：

```
Imgcorruptlike(image).randomNoise(noise_num, noise_color)
```

- 参数：
  - image——图片地址(str类型)
  - noise_num——加噪音数量(int类型)
  - noise_color——噪音颜色（0~255 int类型）
- 解释：将图像按照设定好的噪音数量noise_num进行加噪处理，噪音颜色范围0~255

(2)数学原理

根据噪音数量在图像像素点中随机添加噪音，修改像素点为0~255。

(3)方法示例及效果展示

```
from augmenters.Imgcorruptlike import Imgcorruptlike    #导包
if __name__ == '__main__':
    Imgcorruptlike("leaf.jpg").randomNoise(3000,0)
    Imgcorruptlike("leaf.jpg").randomNoise(1000, 255)
```

原图：

变换： 加噪数量3000,加黑噪音 Imgcorruptlike("leaf.jpg").randomNoise(3000,0)

加噪数量1000,加白噪音 Imgcorruptlike("leaf.jpg").randomNoise(1000, 255)

4.fog方法

(1)参数及其解释

- 接口：

```
Imgcorruptlike(image).fog(A,beta)
```

- 参数：
  - image——图片地址(str类型)
  - A——亮度(默认值 = 0.8 float类型)
  - beta——雾的浓度(默认值 = 0.05 float类型)
- 解释：按照亮度A和浓度beta将图像雾化

(2)数学原理

改变像素点的颜色

(3)方法示例及效果展示

```
from augmenters.Imgcorruptlike import Imgcorruptlike    #导包
if __name__ == '__main__':
    Imgcorruptlike("leaf.jpg").fog()
```

原图：

变换： Imgcorruptlike("leaf.jpg").fog()