**OpenCV--傅里叶变换**

**【演练目的】**

1.掌握使用OpenCV 对图像进行傅里叶变换

2.了解使用Numpy 中FFT（快速傅里叶变换）函数

**【傅里叶变换原理】**

Joseph Fourier是一位18世纪的法国数学家，在数学领域，他认为一切事物均可以用波形来描述，所有观察到的波形也可以有一系列的简单的且频率不同的正弦曲线相加得到，也就是说，人们看到的波形是由多个波形相加得到的。

傅里叶变换经常被用来分析不同滤波器的频率特性。可以使用2D 离散傅里叶变换(DFT) 分析图像的频域特性。实现DFT 的一个快速算法被称为快速傅里叶变换（FFT）。对于一个正弦信号：x (t= Asin (2ft), 它的频率为f，如果把这个信号转到它的频域表示，会在频率f 中看到一个峰值。如果信号是由采样产生的离散信号组成，会得到类似的频谱图，只不过前面是连续的，现在是离散。你可以把图像想象成沿着两个方向采集的信号。所以对图像同时进行X 方向和Y 方向的傅里叶变换，就会得到这幅图像的频域表示（频谱图）。

更直观一点，对于一个正弦信号，如果它的幅度变化非常快，可以说他是高频信号，如果变化非常慢，称之为低频信号。可以把这种想法应用到图像中，图像那里的幅度变化非常大呢？边界点或者噪声。所以边界和噪声是图像中的高频分量（注意这里的高频是指变化非常快，而非出现的次数多）。如果没有如此大的幅度变化称之为低频分量。

**Numpy 中的傅里叶变换**

首先使用Numpy 进行傅里叶变换。Numpy 中的FFT 包可以实现快速傅里叶变换。函数np.fft.fft2() 可以对信号进行频率转换，输出结果是一个复杂的数组。函数的第一个参数是输入图像，要求是灰度格式。第二个参数是可选的, 决定输出数组的大小。输出数组的大小和输入图像大小一样。如果输出结果比输入图像大，输入图像就需要在进行FFT 前补0。如果输出结果比输入图像小的话，输入图像就会被切割。

**OpenCV 中的傅里叶变换**

OpenCV 中相应的函数是cv2.dft() 和cv2.idft()。输出的结果是双通道的。第一个通道是结果的实数部分，第二个通道是结果的虚数部分。输入图像要首先转换成np.float32 格式.

**【演练环境】**

Python 3.x

PyCharm

OpenCV-python

**【演练内容】**

 掌握Numpy 中的傅里叶变换、OpenCV的傅里叶变换。

**【演练步骤】**

1.打开终端，进入对应的演练文件夹

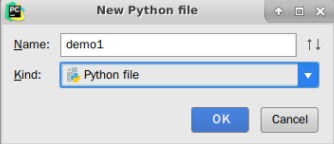
2.所需图片数据

1. opencv15.zip

解压数据

3.打开PyCharm开发工具，新建项目，取名为FourierTransform

4.在FourierTransform项目下新建python文件，右键-【New】-【Python File】，取名为demo1



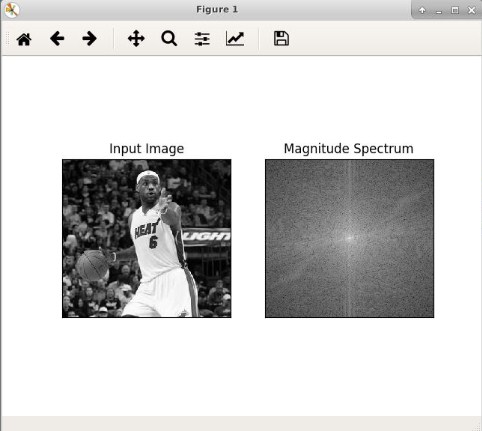
**《Numpy 中的傅里叶变换》**

5.Numpy 中的FFT 包可以实现快速傅里叶变换。函数np.fft.fft2() 可以对信号进行频率转换，输出结果是一个复杂的数组。

频率为0 的部分(直流分量)在输出图像的左上角。(2D傅里叶变换F(x,y)的F(0,0)位置在图像的左上角,F(0,0)表示的是图像灰度的均值)如果想让它(直流分量)在输出图像的中心，还需要将结果沿两个方向平移 N/2 。函数 np.fft.fftshift() 可以实现这一步。

1. # coding=utf-8
2. **import** cv2
3. **import** numpy as np
4. from matplotlib **import** pyplot as plt
5. img = cv2.imread("data/nba.jpg", 0)
6. f = np.fft.fft2(img)
7. fshift = np.fft.fftshift(f)
8. # 构建振幅图
9. # 先取绝对值，表示取模。取对数，将数据范围变小
10. magnitude\_spectrum = 20\*np.log(np.abs(fshift))
11. print(magnitude\_spectrum)
12. plt.subplot(121),plt.imshow(img, cmap = 'gray')
13. plt.title('Input Image'), plt.xticks([]), plt.yticks([])
14. plt.subplot(122),plt.imshow(magnitude\_spectrum , cmap = 'gray')
15. plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
16. plt.show()

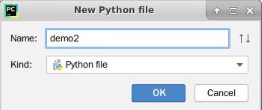
右键运行demo1.py，结果如下：



看到输出结果的中心部分更白(亮)，这说明低频分量更多。

6.进行频域变换,例如高通滤波和重建图像(DFT 的逆变换)。

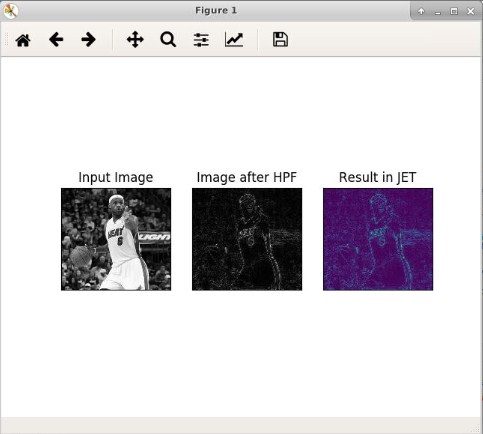
在FourierTransform项目下新建python文件，右键-【New】-【Python File】，取名为demo2



可以使用一个60x60 的矩形窗口对图像进行掩模操作从而去除低频分量。然后再使用函数np.fft.ifftshift() 进行逆平移操作，所以现在直流分量又回到左上角了，然后使用函数 np.ifft2() 进行 FFT 逆变换。

1. # coding=utf-8
2. **import** cv2
3. **import** numpy as np
4. from matplotlib **import** pyplot as plt
5. img = cv2.imread("data/nba.jpg", 0)
6. f = np.fft.fft2(img) #得到结果为复数矩阵
7. fshift = np.fft.fftshift(f) #直接取中心
8. rows, cols = img.shape
9. crow,ccol = int(rows/2), int(cols/2)
10. fshift[crow-30:crow+30, ccol-30:ccol+30] = 0 #蒙板大小60×60
11. f\_ishift = np.fft.ifftshift(fshift)
12. img\_back = np.fft.ifft2(f\_ishift) #使用FFT逆变换，此时结果仍然是复数
13. img\_back = np.abs(img\_back) # 取绝对值
14. plt.subplot(131),plt.imshow(img, cmap = 'gray')
15. plt.title('Input Image'), plt.xticks([]), plt.yticks([])
16. plt.subplot(132),plt.imshow(img\_back, cmap = 'gray')
17. plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
18. plt.subplot(133),plt.imshow(img\_back)
19. plt.title('Result in JET'), plt.xticks([]), plt.yticks([])
20. plt.show()

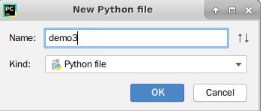
右键运行demo2.py，结果如下：



上图的结果显示高通滤波其实是一种边界检测操作，同时图像中的大部分数据集中在频谱图的低频区域。

**《OpenCV 中的傅里叶变换》**

7.在FourierTransform项目下新建python文件，右键-【New】-【Python File】，取名为demo3



8.OpenCV 中相应的函数是 cv2.dft() 和 cv2.idft()。和前面输出的结果一样，但是是双通道的。第一个通道是结果的实数部分，第二个通道是结果的虚数部分。输入图像要首先转换成 np.float32 格式。

1. # coding=utf-8
2. **import** cv2
3. **import** numpy as np
4. from matplotlib **import** pyplot as plt
5. img = cv2.imread("data/nba.jpg", 0)
6. dft = cv2.dft(np.float32(img),flags = cv2.DFT\_COMPLEX\_OUTPUT)
7. dft\_shift = np.fft.fftshift(dft)
8. magnitude\_spectrum = 20\*np.log(cv2.magnitude(dft\_shift[:,:,0],dft\_shift[:,:,1])) #频谱图
9. plt.subplot(121),plt.imshow(img, cmap = 'gray')
10. plt.title('Input Image'), plt.xticks([]), plt.yticks([])
11. plt.subplot(122),plt.imshow(magnitude\_spectrum, cmap = 'gray')
12. plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
13. plt.show()

使用函数 cv2.cartToPolar()会同时得到幅度和相位,此函数也是直角坐标转换为极坐标的函数。

9.现在做逆 DFT。在前面的部分实现了一个 HPF(高通滤波)，现在做 LPF(低通滤波)将高频部分去除。其实就是对图像进行模糊操作。首先需要构建一个掩模，与低频区域对应的地方设置为 1，与高频区域对应的地方设置为 0。

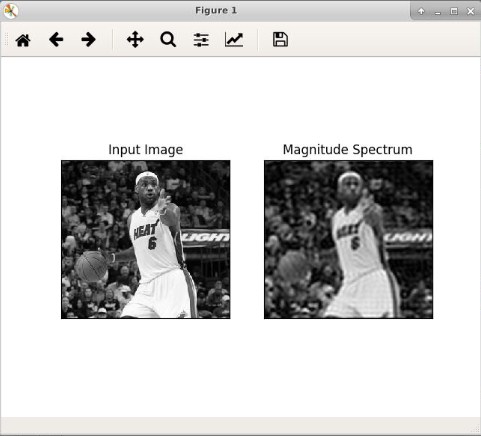
在demo3.py中追加以下代码：

1. rows, cols = img.shape
2. crow,ccol = rows/2 , cols/2
3. # 首先创建一个掩膜, 中心为1, 其他部分均为0
4. mask = np.zeros((rows,cols,2),np.uint8)
5. mask[crow-30:crow+30, ccol-30:ccol+30] = 1
6. # 使用mask 进行DFT
7. fshift = dft\_shift\*mask
8. f\_ishift = np.fft.ifftshift(fshift)
9. img\_back = cv2.idft(f\_ishift)
10. img\_back = cv2.magnitude(img\_back[:,:,0],img\_back[:,:,1])

demo3.py的完整代码如下：

1. # coding=utf-8
2. **import** cv2
3. **import** numpy as np
4. from matplotlib **import** pyplot as plt
5. img = cv2.imread("data/nba.jpg", 0)
6. dft = cv2.dft(np.float32(img),flags = cv2.DFT\_COMPLEX\_OUTPUT)
7. dft\_shift = np.fft.fftshift(dft)
8. rows, cols = img.shape
9. crow,ccol = rows//2 , cols//2
10. # 首先创建一个掩膜, 中心区为1, 其他部分均为0
11. mask = np.zeros((rows,cols,2),np.uint8)
12. mask[crow-30:crow+30, ccol-30:ccol+30] = 1
13. # 使用mask 进行DFT
14. fshift = dft\_shift\*mask
15. f\_ishift = np.fft.ifftshift(fshift)
16. img\_back = cv2.idft(f\_ishift)
17. img\_back = cv2.magnitude(img\_back[:,:,0],img\_back[:,:,1])
18. plt.subplot(121),plt.imshow(img, cmap = 'gray')
19. plt.title('Input Image'), plt.xticks([]), plt.yticks([])
20. plt.subplot(122),plt.imshow(img\_back, cmap = 'gray')
21. plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
22. plt.show()

右键运行demo2.py，结果如下：



注意:OpenCV 中的函数 cv2.dft() 和 cv2.idft() 要比 Numpy 快。但是Numpy 函数更加用户友好。

以上！