# 傅里叶变换

## 目标：

* 使用OpenCV查找图像的傅立叶变换
* 使用Numpy中提供的FFT函数
* 傅立叶变换的一些应用
* 我们将看到以下函数：cv.dft()，cv.idft()等

## 理论

傅立叶变换用于分析各种滤波器的频率特性。对于图像，2D离散傅里叶变换（DFT）用于找到频域。称为快速傅里叶变换（FFT）的快速算法用于计算DFT。有关这些的详细信息可以在任何图像处理或信号处理教科书中找到。

对于正弦信号，x（t）= Asin（2πft），我们可以说f是信号的频率，如果采用其频域，我们可以看到f处的尖峰。如果对信号进行采样以形成离散信号，则我们得到相同的频域，但在[-π，π]或[0,2π]（或对于N点DFT的[0，N]）范围内是周期性的。你可以将图像视为在两个方向上采样的信号。因此，在X和Y方向上进行傅里叶变换可以得到图像的频率表示。

更直观地说，对于正弦信号，如果幅度在短时间内变化如此之快，则可以说它是高频信号。如果变化缓慢，则为低频信号。你可以将相同的想法扩展到图像。幅度在图像中的幅度变化很大？在边缘点，或噪音。我们可以说，边缘和噪声是图像中的高频内容。如果幅度没有太大变化，则它是低频分量。

现在我们将看到如何找到傅立叶变换。

## Numpy中的傅里叶变换

首先，我们将看到如何使用Numpy找到傅立叶变换。Numpy有一个FFT包来做到这一点。np.fft.fft2()为我们提供了一个复杂数组的频率变换。它的第一个参数是输入图像，它是灰度。第二个参数是可选的，它决定了输出数组的大小。如果它大于输入图像的大小，则在计算FFT之前用零填充输入图像。如果小于输入图像，则将裁剪输入图像。如果没有传递参数，则输出数组大小将与输入相同。

现在，一旦得到结果，零频率分量（DC分量）将位于左上角。如果要将其置于中心位置，则需要在两个方向上将结果移动$$\frac{N}{2}$$。这只是通过函数np.fft.fftshift()完成的。找到频率变换后，你可以找到幅度谱。

## 实例

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

请注意，你可以在中心看到更多更白的区域，显示低频内容更多。

所以你找到了频率变换现在你可以在频域做一些操作，比如高通滤波和重建图像，即找到逆DFT。 为此，你只需通过使用尺寸为60x60的矩形窗口进行遮罩来移除低频。 然后使用np.fft.ifftshift()应用反向移位，以便DC组件再次出现在左上角。 然后使用np.ifft2()函数找到逆FFT。 结果再次是一个复杂的数字。 你可以采取它的绝对价值。

## 实例2.

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

结果显示高通滤波是边缘检测操作。这也表明大多数图像数据存在于光谱的低频区域。无论如何，我们已经看到如何在Numpy中找到DFT，IDFT等。现在让我们看看如何在OpenCV中完成它。

如果你仔细观察结果，特别是JET颜色的最后一个图像，你可以看到一些文物（我用红色箭头标记的一个实例）。它在那里显示出一些类似波纹的结构，它被称为振铃效应。它是由我们用于遮蔽的矩形窗口引起的。此蒙版转换为sinc形状，这会导致此问题。因此矩形窗口不用于过滤。更好的选择是高斯Windows。

## OpenCV中的傅里叶变换

OpenCV为此提供了cv.dft()和cv.idft()函数。它返回与之前相同的结果，但有两个通道。第一个通道将具有结果的实部，第二个通道将具有结果的虚部。输入图像应首先转换为np.float32。

## 实例1

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

注意：你还可以使用cv.cartToPolar（），它可以一次性返回幅度和相位

所以，现在我们必须进行逆DFT。 在之前的会话中，我们创建了一个HPF，这次我们将看到如何去除图像中的高频内容，即我们将LPF应用于图像。 它实际上模糊了图像。 为此，我们首先在低频处创建具有高值（1）的掩模，即我们传递LF内容，并且在HF区域传递0。

## 实例2

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

注意：像往常一样，OpenCV函数cv.dft()和cv.idft()比Numpy函数更快。 但是Numpy功能更加用户友好

## DFT的性能优化

对于某些阵列大小，DFT计算的性能更好。 当阵列大小为2的幂时，它是最快的。 尺寸为2，3和5的乘积的阵列也可以非常有效地处理。 因此，如果你担心代码的性能，可以在找到DFT之前将数组的大小修改为任何最佳大小（通过填充零）。 对于OpenCV，你必须手动填充零。 但对于Numpy，你可以指定FFT计算的新大小，它会自动为你填充零。

那么我们如何找到这个最佳尺寸？ OpenCV为此提供了一个函数cv.getOptimalDFTSize（）。 它适用于cv.dft（）和np.fft.fft2（）。

## 优化实例1

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 为什么拉普拉斯算子是高通滤波器？

在论坛中提出了类似的问题。 问题是，为什么拉普拉斯算子是高通滤波器？ 为什么Sobel是HPF？ 第一个答案就是傅立叶变换。 只需将拉普拉斯算子的傅里叶变换用于更高尺寸的FFT。 分析一下：

import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt

# simple averaging filter without scaling parameter

mean\_filter = np.ones((3,3))

# creating a gaussian filter

x = cv.getGaussianKernel(5,10)

gaussian = x\*x.T

# different edge detecting filters

# scharr in x-direction

scharr = np.array([[-3, 0, 3],

[-10,0,10],

[-3, 0, 3]])

# sobel in x direction

sobel\_x= np.array([[-1, 0, 1],

[-2, 0, 2],

[-1, 0, 1]])

# sobel in y direction

sobel\_y= np.array([[-1,-2,-1],

[0, 0, 0],

[1, 2, 1]])

# laplacian

laplacian=np.array([[0, 1, 0],

[1,-4, 1],

[0, 1, 0]])

filters = [mean\_filter, gaussian, laplacian, sobel\_x, sobel\_y, scharr]

filter\_name = ['mean\_filter', 'gaussian','laplacian', 'sobel\_x', \

'sobel\_y', 'scharr\_x']

fft\_filters = [np.fft.fft2(x) for x in filters]

fft\_shift = [np.fft.fftshift(y) for y in fft\_filters]

mag\_spectrum = [np.log(np.abs(z)+1) for z in fft\_shift]

for i in xrange(6):

plt.subplot(2,3,i+1),plt.imshow(mag\_spectrum[i],cmap = 'gray')

plt.title(filter\_name[i]), plt.xticks([]), plt.yticks([])

plt.show()

窗口将如下图显示：

|  |
| --- |
|  |