2019 年秋季《图像处理与分析》编程作业-03

【实现和测试图像的二维快速傅里叶变换与逆变换】

姓名：　　学号：

【问题 1】通过计算一维傅里叶变换实现图像二维快速傅里叶变换

问题说明：

实现一个函数 F=dft2D(f), 其中 f 是一个灰度源图像，F 是其对应的二维快速傅里叶变换(FFT)图像. 具体实现要求按照课上的介绍通过两轮一维傅里叶变换实现。也就是首先计算源图像每一行的一维傅里叶变换，然后对于得到的结果计算其每一列的一维傅里叶变换。如果实现采用 MATLAB, 可以直接调用函数 fft 计算一维傅里叶变换。如果采用其他语言，请选择并直接调用相应的一维傅里叶变换函数。

问题分析：

本题是通过一维 FFT 实现二维 FFT，其理论公式为：

$$F(u,v)=\sum_{x=0}^{M-1}e^{-j2\pi\frac{ux}{M}}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi\frac{vy}{N}}$$

$$=\sum_{x=0}^{M-1}F(x,v)e^{-j2\pi\frac{ux}{M}} \qquad (1\text{-}1)$$

$$F(x,v)=\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi\frac{vy}{N}} \qquad (1\text{-}2)$$

根据公式，可以先对矩阵 f(x,y)的每一行的一维数组进行一维 FFT 得到 F(x,v)矩阵；再对矩阵 F(x,v)的每一列的一维数组进行一次一维 FFT 得到 F(u,v)矩阵。

函数实现：

输入：原始图像矩阵 f

输出：经过二维 FFT 操作后的矩阵 F2

运行：（1）获取图像 f 尺寸即行数 m、列数 n；

（2）创建相同尺寸的复数型零矩阵用于存储两次一维 FFT 的结果；

（3）利用 for 循环，调用现有函数先对 f 每一行进行一维 FFT，结果存放在 F1。再对 F1 每一列进行一维 FFT 得到 F2，返回 F2 的值。（注：此函数仅得到原始的 FFT 复数结果，求模及其他变换在后述的程序中实现）

详细代码：

- 编程语言：Python3.7+
- 第三方库：Numpy 1.17.2
- 函数文件：dft2D.py
- 调用方式：import dft2D，运行 dft2D.dft2D(f)
- 相关变量：

  m, n ：输入矩阵 f 的高度及宽度；

  F1 ：输入矩阵 f 每一行进行一维 FFT 后的中间矩阵；

  F2 ：F1 每一列进行一维 FFT 后的最终结果，返回该矩阵。

```python
import numpy as np
def dft2D(f):
    #两轮一维FFT实现
    m,n=f.shape                    #输入图像的尺寸
    F1=np.zeros((m,n),dtype='complex128')#创建两个m*n的复数类型的零矩阵
    F2=np.zeros((m,n),dtype='complex128')
    for i in range(m):
        F1[i,:]=np.fft.fft(f[i,:])     #对原始图像f的第i行进行一维fft
    for j in range(n):
        F2[:,j]=np.fft.fft(F1[:,j])    #对已经进行过"行fft"的矩阵第j列进行一维fft
    return F2

```

效果展示：

运行测试代码文件 **dft2D_test.py**。为了检验函数的效果，自定义 3*3 的简单矩阵 a，接着在 python 中调用该函数并与 Numpy.fft.fft2 函数的运行结果进行比较，可以发现实验结果是正确的。详细的结果如下图：

```
a=
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

 b=dft2D(a)=
 [[ 45. +0.j          -4.5+2.59807621j  -4.5-2.59807621j]
 [-13.5+7.79422863j   0. +0.j           0. +0.j        ]
 [-13.5-7.79422863j   0. +0.j           0. +0.j        ]]

 c=np.fft.fft2(a)=
 [[ 45. +0.j          -4.5+2.59807621j  -4.5-2.59807621j]
 [-13.5+7.79422863j   0. +0.j           0. +0.j        ]
 [-13.5-7.79422863j   0. +0.j           0. +0.j        ]]

 b-c=
 [[0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j]]
```

a：原始矩阵
b：自定义FFT函数结果
c：numpy 中FFT函数结果
d：差值b-c

图 1.问题一函数检验代码 dft2D_test.py 运行结果

根据运行结果，可以看出，利用自定义的 dft2D(f)函数与 Numpy 自带函数 numpy.fft.fft2(f)对矩阵 a 做二维 FFT 得到的矩阵 b、c 相等，差值矩阵为零矩阵。

【问题 2】图像二维快速傅里叶逆变换

问题说明：

实现一个函数 f=idft2D(F), 其中 F 是一个灰度图像的傅里叶变换，f 是其对应的二维快速傅里叶逆变换 (IFFT)图像，也就是灰度源图像. 具体实现要求按照课上的介绍通过类似正向变换的方式实现。

问题分析：

这一问题实际上与问题 1 中方案类似，主要依据如下公式：

$$f(x,y)=\frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{2-1}$$

$$MNf^{*}(x,y)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F^{*}(u,v)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{2-2}$$

$$F(u,v)\xrightarrow{\text{take complex conjugate}}F^{*}(u,v) \tag{2-3}$$

$$F^{*}(u,v)\xrightarrow{DFT}MNf^{*}(x,y) \tag{2-4}$$

$$MNf^{*}(x,y)\xrightarrow{\text{divide by } MN\text{, take complex conjugate}}f(x,y) \tag{2-5}$$

根据公式，首先对输入的矩阵 F(u,v)进行共轭转换得到 F*(u,v)，接着按照【问题 1】中的的方式对 F*(u,v)进行二维 FFT 操作，此时得到矩阵 MNf*(x,y)。最后需要除去 MN 的系数，同时对矩阵进行共轭操作，将复数取模得到 f(x,y)。

函数实现：

输入：经过傅里叶变换的频域矩阵 F

输出：逆向二维 FFT 操作后的矩阵 f

运行：（1）获取复数矩阵 F 尺寸即行数 M、列数 N；

（2）对矩阵 F 取共轭得到新的矩阵 F；

（3）创建相同尺寸的复数型零矩阵 f1，f2 用于存储两次一维 FFT 的结果；

（4）利用 for 循环，调用现有函数先对 F 每一行进行一维 FFT，结果存放在 f1。再对 f1 每一列进行一维 FFT 得到 f2；

（5）对 f2 除以系数 M*N，并取共轭矩阵，得到新的 f2 作为输出

详细代码：

- 编程语言：Python3.7+
- 第三方库：Numpy 1.17.2

- 函数文件：idft2D.py
- 调用方式：import idft2D，运行 idft2D.idft2D(F)
- 相关变量：

  M,N ：输入矩阵 F 的高度与宽度；

  f1 ：F（取共轭后）每行进行一维 FFT 得到的中间矩阵；

  f2 ：f1 每一列进行一维 FFT、并除以系数 M*N、取共轭得到的结果矩阵。

```python
import numpy as np
def idft2D(F):
    #类似正向变换实现逆变换
    M,N=F.shape
    F=F.conjugate()    #对原图像取共轭
    f1=np.zeros((M,N),dtype='complex128')
    f2=np.zeros((M,N),dtype='complex128')
    for i in range(M):
        f1[i,:]=np.fft.fft(F[i,:])   #对第i行进行一维FFT
    for j in range(N):
        f2[:,j]=np.fft.fft(f1[:,j])  #对第j列进行一维FFT
    f2=np.abs((f2/(M*N)).conjugate()) #添加1/MN的系数并取共轭,最后取模

    return f2
```

效果展示：

运行测试代码文件 **idft2D_test.py**。

为了验证改逆 FFT 函数是正确的，对简单矩阵 a 使用先前编写的 dft2D(f)函数进行二维 FFT 操作得到矩阵 b；随后使用编写的 idft2D(F)函数对 b 进行还原得到矩阵 c，比较 a，c 是否一致。实际运行结果说明编写函数是正确运行的，运行的结果如下：

```
a=
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

 b=dft2D(a)=
 [[ 45. +0.j          -4.5+2.59807621j  -4.5-2.59807621j]
 [-13.5+7.79422863j   0. +0.j           0. +0.j        ]
 [-13.5-7.79422863j   0. +0.j           0. +0.j        ]]

 c=idft2D(b)=
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]

 a-c=
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

a：原始矩阵
b：对a使用自定义FFT函数结果
c：对b使用自定义逆FFT函数结果
d：差值a-c

图 2.问题二函数检验代码 idft2D_test.py 运行结果

根据运行结果，可以看出，利用自定义的 dft2D(f)函数产生的复频域矩阵经过自定义的逆傅里叶变换函数 idft2D(F)作用可以恢复到完全一致的原始矩阵。

【问题 3】测试图像二维快速傅里叶变换与逆变换

问题说明：

对于给定的输入图像 house.tif, 首先将其灰度范围通过归一化调整到[0,1]. 将此归一化的图像记为 f. 首先调用问题 1 下实现的函数 dft2D 计算其傅里叶变换，记为 F。然后调用问题 2 下的函数 idft2D 计算 F 的傅里叶逆变换，记为 g. 计算并显示误差图像 d＝f-g.

问题分析：

本题是利用问题一、问题二中的两个函数对给定的输入图像进行处理，为了比较原始图像和通过频域矩阵恢复的图像之间的差别。主要实现步骤为：

（1）归一化图像,将图像矩阵 img 中每一个像素的灰度值 img[i , j]减去矩阵中最小值 min(img)，再除以矩阵中灰度最大值与最小值的差值[max(img)-min(img)],得到归一化图像 f；

（2）对归一化图像使用问题一中的二维傅里叶变换函数 dft2D(f)得到矩阵 F；

（3）将傅里叶变换得到的复数矩阵 F 使用 idft2D(F)进行逆向傅里叶变换得到恢复的矩阵 g；

（4）计算差值 f-g 并绘制图像显示出来。

实现细节：

- 编程语言：Python3.7+
- 第三方库：Numpy 1.17.2，matplotlib3.0.2，PIL5.3.0
- 自定义模块：dft2D，idft2D
- 代码文件：ft2D_idft2D_test.py
- 相关变量：

  img ： 使用 numpy 读取的图像矩阵，图像源为’rose512.tif’文件；

  f ： 将 img 归一化后得到的矩阵；

  F ： 归一化矩阵 f 进行二维 FFT 得到的复频域矩阵；

  g ： 复频域矩阵 F 进行逆 FFT 操作返还的矩阵；

  a ： f-g 后数据类型转换为 uint8 型的矩阵，即差值图像矩阵。

详细的 python 代码如下：

```
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image
import dft2D
import idft2D

```

```
img=np.array(Image.open('assignment01_images\\rose512.tif'))
f=(img-np.min(img))/(np.max(img)-np.min(img))#归一化图像
F=dft2D.dft2D(f)
g=idft2D.idft2D(F)
a=np.array((f-g),dtype='uint8')    #计算原图像与其差值，并将差值规范为uint8型

plt.figure(figsize=(10,5))
plt.subplot(121)
plt.title('Original Image')
plt.imshow(img)
plt.subplot(122)
plt.title('f-g')
plt.imshow(a)

plt.savefig('dft_idft.jpg')
```

效果展示：

运行文件 **ft2D_idft2D_test.py**。得到结果如下图所示，左边为原始图像，右边为原始图像与经过正向 FFT 及逆向 FFT 恢复的图像的差值图像：

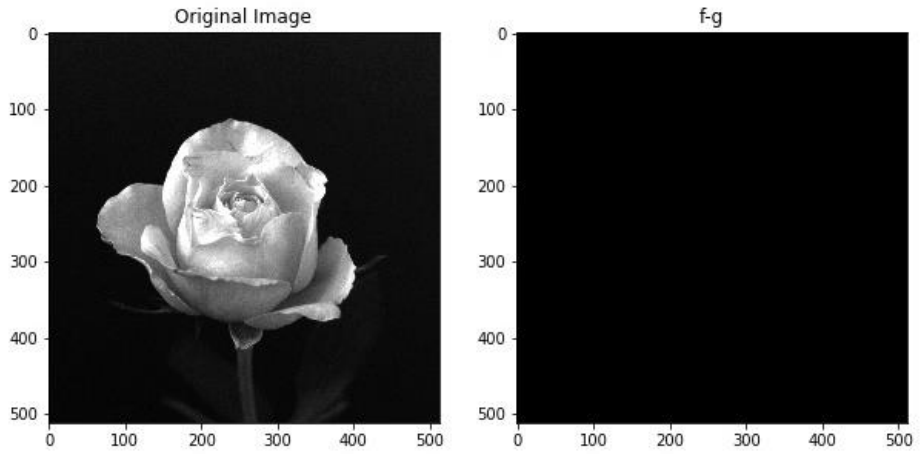

图 3.问题三代码 ft2D_idft2D_test.py 运行结果

根据上图的结果，可以看出自定义的二维傅里叶变换函数以及逆傅里叶变换函数可以成功地将空间图片转换到复频域内并还原。这说明实验的结果与理论的计算是一致的。

【问题 4】计算图像的中心化二维快速傅里叶变换与谱图像

问题说明：

我们的目标是复现下图中的结果。首先合成矩形物体图像，建议图像尺寸为 512×512，矩形位于图像中心，建议尺寸为 60 像素长，10 像素宽，灰度假设已归一化设为 1. 对于输入图像 f 计算其中心化二维傅里叶变换 F。然后计算对应的谱图像 S=log(1+abs(F)). 显示该谱图像。

问题分析：

（1）首先，需要自定义创建一个尺寸为 512×512 的矩阵，图像中央有白色的矩形，尺寸为 60*10，且经过归一化。首先，创建一个 512*512 的零矩阵（全黑）；接着对矩阵函数和列数取一半，即 512/2 取整数，得到图像的中心点坐标[512/2,512/2]。对第[512/2]-30 行至[512/2]+60 行、第[512/2]-5 列至[512/2]+10 列的子矩阵区域（60*10）赋值为 255；接着归一化处理。

（2）其次，需要实现二维傅里叶变换的中心化。这是因为直接 FFT 得到的频谱图是高频在中间、低频在四角，为了能易于观察，利用二维 FFT 的平移性质实现频谱中心化。其实现原理如下：

$$f(x)(-1)^x \Leftrightarrow F(u-M/2) \tag{4-1}$$

$$f(x,\ y)(-1)^{x+y} \Leftrightarrow F(u-M/2, v-N/2) \tag{4-2}$$

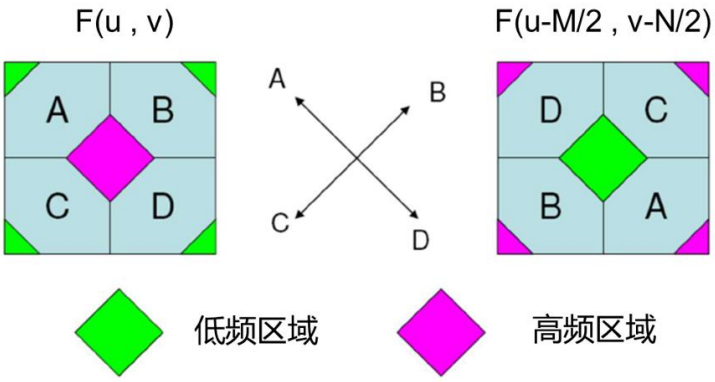

图 4.中心化频谱原理说明示意图

其原理是将原始图像矩阵 f(x,y)进行变换，乘以(-1)^(x+y)，根据傅里叶变换公式，对应于频率域的函数 F(u,v)将产生平移得到 F(u-M/2，v-N/2)。从而使得我们想要观察的低频部分频谱图更易于观察。

（3）显示对数谱图像 log(1+abs(F))。由于直流项支配者谱的值，在显示的图像中，其他灰度的动态范围被压缩了。为了显示那些细节便于观察，所以对频谱做对数变换。

实现细节：

- 编程语言：Python3.7+
- 第三方库：Numpy 1.17.2，matplotlib3.0.2
- 自定义模块：dft2D
- 运行文件：spectrum_demo.py
- 相关变量及函数：

  img_test ： 创建的 512*512 的归一化矩阵；

  test_fft ： img_test 进行二维傅里叶变换并取模得到的频谱图；

test_shift ： img_test 进行中心化的傅里叶频谱矩阵；

FFT_SHIFT(img) ： 中心化函数。

详细的 python 代码如下：

```python
import numpy as np
from matplotlib import pyplot as plt
import dft2D

#自定义中心化移位函数
def FFT_SHIFT(img):
    M,N = img.shape
    F=np.zeros((M,N))
    for i in range(M):
        for j in range(N):
            #令f(x,y)=f(x,y)*(-1)^(x+y)实现中心化
            F[i,j]=img[i,j]*np.power(-1,i+j)
    return F

#初始化矩形物体图像
img_test=np.zeros((512,512))
img_test[(512//2-30):(512//2+30),(512//2-5):(512//2+5)]=255
#归一化
img_test=(img_test-np.min(img_test))/(np.max(img_test)-np.min(img_test))

test_fft=np.abs(dft2D.dft2D(img_test))    #FFT

#对中心化的图像矩阵进行二维FFT，并取模
test_shift=np.abs(dft2D.dft2D(FFT_SHIFT(img_test)))


#绘制图像
plt.gray()
plt.figure(figsize=(10,10))
plt.subplot(221)
plt.title('Original Image')
plt.imshow(img_test)                    #源图像
plt.subplot(222)
plt.title('Spectrum')
plt.imshow(test_fft)                    #谱图像
plt.subplot(223)
plt.title('Centered Spectrum')
plt.imshow(test_shift)                  #中心化谱图像
plt.subplot(224)
plt.title('Spectrum After a Log Transformation')
plt.imshow(np.log(1+np.abs(test_shift))) #对数谱图像
```

效果展示：

运行文件 **spectrum_demo.py**。显示出原始图像、未中心化谱图像、中心化谱图像以及对数运算后的中心化谱图像，如下图所示：

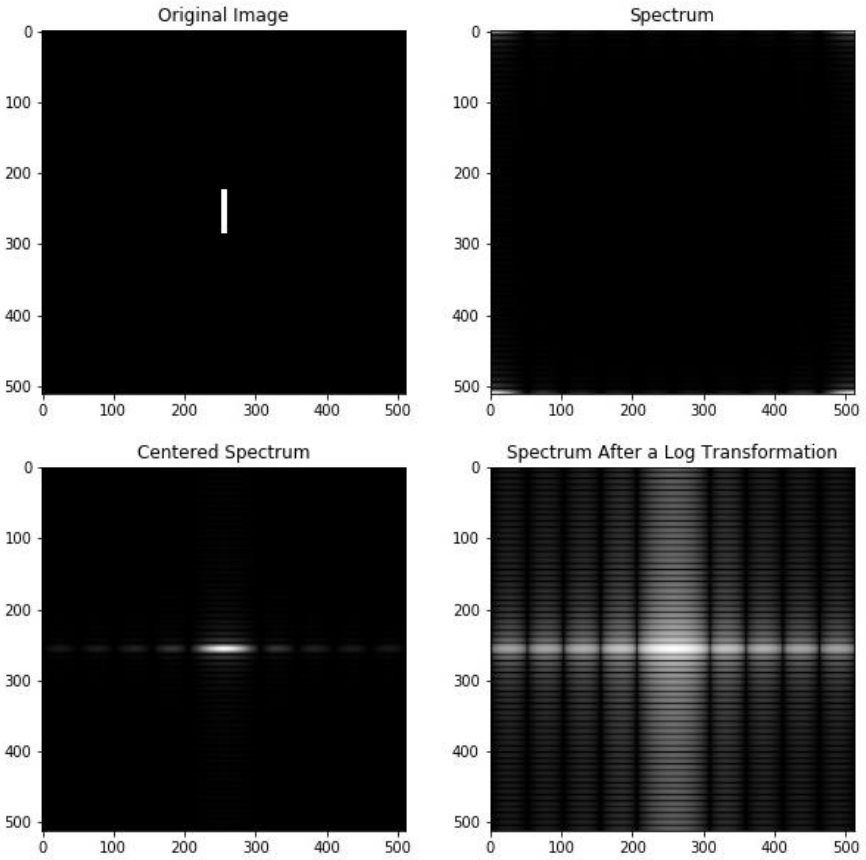

图 5.问题四代码 spectrum_demo.py 运行结果

根据上述运行结果图，可以看出：原始图像中央的矩形白块经过二维傅里叶变换后到频域内的频率成分分布在了图像的四个角上，这是因为未经中心化的频谱图像的低频部分处于图像的四个角，图像的中央对应于频谱中的高频成分。而经过中心化以后频谱在图像中央显现出了白色的断断续续的带状。由于直流项支配者谱的值，在显示的图像中，其他灰度的动态范围被压缩了。因此，在进行对数操作以后最后一张子图上显现出了清晰的频谱图像，频域内的细节得到了增强。

这个问题的实质是实现了二维矩形窗的频谱展示。一维矩形窗函数的一维傅里叶变换得到的频谱图像是一维 sinc 函数，而这里的二维矩形窗长宽不一致，对应了两个维度上不同的频域分布。由于原始图像中矩形窗的横向宽度较窄（10 个像素），因此对应于频谱图中横向主瓣较宽、旁瓣分散；而纵向的长度较长（60 个像素），因此对应的频谱中主瓣较窄且旁瓣向主瓣密集。从这一角度分析，更加容易理解二维傅里叶变换与一维傅里叶变换的关系以及二维频谱图的实际意义。