数字图像处理作业三

1.1 旋转

我们可以看出来这一套流程很像频率域滤波的步骤，只是它没有扩充以及裁剪的步骤，以及它的滤波函数实际的作用是将离散傅立叶变换后得到的结果取共轭 原图像可以用函数f(x,y)来表示，对它做傅立叶变换之后得到的结果是F(u,v)，对它取共轭得到的结果F*(u,v)(不是乘号)根据性质我们可以知道它等于F(-u,-v),也就是说我们的f(x,y)的傅立叶变换是根据原点成共轭对称的，但是我们经过前面成-1^(x+y)的操作，已经将对称中心从原点转移到了(M/2，N/2)(令原图像的大小为MN) 所以在这里，F*(u,v)=F(M/2-u,N/2-v) 接下来对F(M/2-u,N/2-v)做傅立叶反变换，并再次乘上-1^(x+y)之后，得到的新图像中g(x,y)=f(M-x,N-y) 于是新图像就与原图像成中心对称，对称中心为图像的中心点

1.2 傅立叶谱

在图2.2与图2.4中，我们可以看出图2.4多了两条亮带，一条是横向的，一条是纵向的，这两条亮带贯穿了整幅频谱图 在图像中，高频成分表示灰度值变化快的地方，图2.3与图2.1的不同之处在于图2.3在白色背景下拓展了一圈黑边，这样的话图2.3不仅比图2.1多了高频部分(白色背景和黑色边框交接的地方)，还多了低频部分(黑色边框内部)

1.3低通和高通

H(u,v)如下

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

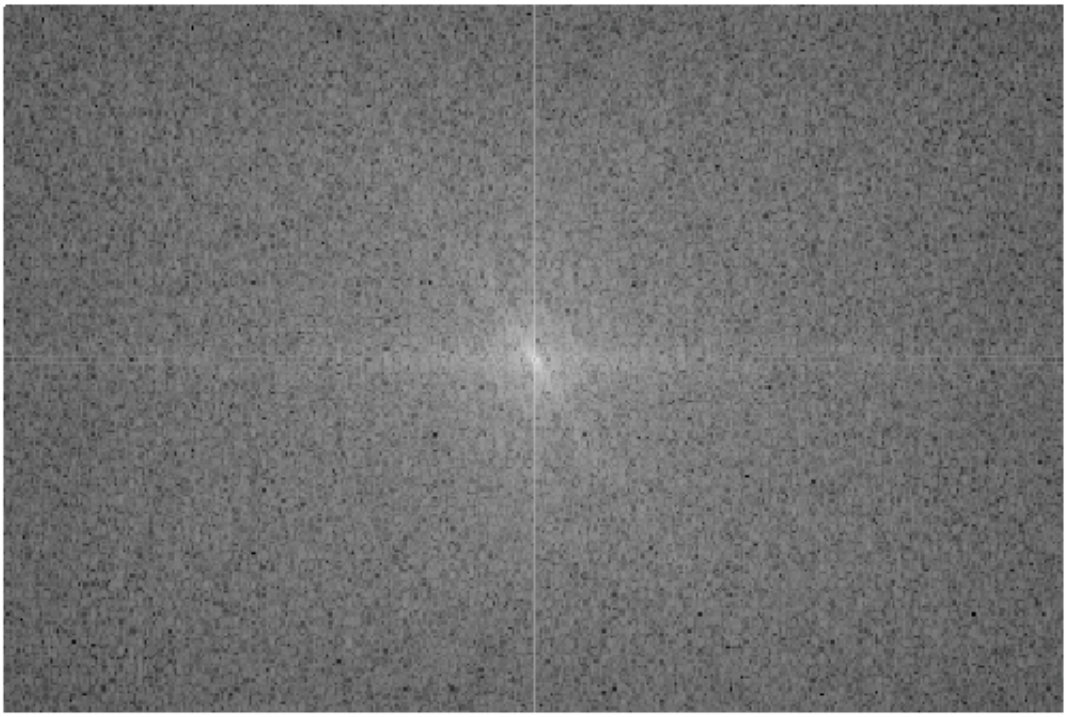
所以取原图的一部分

$$\begin{bmatrix} f(x-1,y-1) & f(x-1,y) & f(x-1,y+1) \\ f(x,y-1) & f(x,y) & f(x,y+1) \\ f(x+1,y-1) & f(x+1,y) & f(x+1,y+1) \end{bmatrix}$$

令f(x,y)在变换之后的图像中对应的点是g(x,y) 则将上面的H(u,v)与f矩阵做卷积之后，可得 g(x,y)=f(x-1,y)+f(x,y-1)+f(x,y+1)+f(x+1,y)-4f(x,y)　G(x,y)=F(u,v)*($e^{2\pi ju/m}+e^{2\pi jv/m}+e^{-2\pi ju/m}+e^{-2\pi jv/m}$-4)　H(u,v)=$e^{2\pi ju/m}+e^{2\pi jv/m}+e^{-2\pi ju/m}+e^{-2\pi jv/m}$-4 所以滤波器为 h(u,v)=2[cos(2πu/m)+cos(2πv/n)-4] 这个滤波器是高通的 考虑D(u,v)=$[(u-P/2)^2+(v-Q/2)^2]^{1/2}$ =$[(u-m)^2+(v-n)^2]^{1/2}$ 当u趋近m,v趋近n时，即D(u,v)趋近0时 h(u,v)趋近0，所以该滤波器是高通滤波器

2.1傅立叶变换

写一个函数来执行二维离散傅里叶变换(DFT)或逆离散傅里叶变换(IDFT).该函数原型为“dft2d(input_img，flags)→output_img”，返回给定输入的DFT/IDFT结果。“flags”是一个参数，用于指定是否需要DFT或IDFT。 对于报告，请加载您的输入，并使用您的程序 1.执行DFT并在报告上手工粘贴（居中）傅里叶光谱。(10分)



2.根据最后一个问题的结果执行IDFT，并将真实的部分粘贴在报告上。注意：真实的部分应该非常类似于您的输入图像。（为什么？想一想)(10分)

3.详细讨论如何在不到2页的情况下实现DFT/IDFT。请关注算法部分。不要在报告中广泛复制/粘贴你的代码，因为你的代码也提交了。（20分） 首先需要对图像进行中心化 即g(x,y)=f(x,y)*$(-1)^{x+y}$

```
for x in range(m):
        for y in range(n):
            output[x,y]=image[x, y]*((-1) ** (x+y))
```

正常的二维离散傅立叶变换如下 F(u,v)=$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}$f(x,y)$e^{-2\pi(ju/M+vy/N)}$ 用代码实现的话如下

```
    h=image.shape[0]
    w=image.shape[1]
    output=np.zeros((h,w),np.complex)
    for u in range(h):
        for v in range(w):
            s=complex(0)
            for x in range(h):
                for y in range(w):
                    s +=image[x,y]*np.exp(-2j*np.pi*
(u*x/float(h)+v*y/float(w)))
            output[u,v]=s
    return output
```

但是这是一个四重循环，我们的图片是384*256的大小，这样循环一遍的话需要的运算量接近100亿，在实际操作中显然不现实 对此进行改进的话我们按照书中给出的方法，将四重循环改成两个三重循环，虽然跑一次也需要几分钟，并且其中风扇呼呼作响，但是好歹程序运行时间在我们的接受范围中 F(u,v)=$\sum_{x=0}^{M-1}$F(u,v)$e^{-2j\pi ux/M}$ F(x,v)=$\sum_{y=0}^{N-1}$f(x,y)$e^{-2j\pi uy/N}$ 这样的话，四重循环的运算量就被减小到两个三重循环 F(x,v)=$\sum_{y=0}^{N-1}$f(x,y)$e^{-2j\pi uy/N}$:

```
for x in range(M):
        for v in range(N):
            for y in range(N):
                s+=image[y,x]*np.exp(-1.j*2*np.pi*v*y/N)
            temp[v,x]=s
            s=complex(0)
```

$F(u,v)=\sum_{x=0}^{M-1}F(u,v)e^{-2j\pi ux/M}$:

```
 for v in range(N):
        for u in range(M):
            for x in range(M):
                t+=temp[v,x]*np.exp(-1.j*2*np.pi*u*x/M)
            output[v,u]=t
            t=complex(0)
```

在我们傅立叶变换之后，求出来的图像的每一个像素点都是复数，为了让得出来的频谱图更加好看一些， 我们将每个复数的模加1取对数(加1是为了防止出现对0取对数的情况) `result = np.log(1 + np.abs(dft))` 傅立叶反变换的公式为 $f(x,y)=1/MN\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{-2j\pi(ux/M+vy/N)}$ 对其两边取共轭，并乘MN得到 $MNf^*(x,y)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F^*(u,v)e^{-2j\pi(ux/M+vy/N)}$ 这样的话就可以使用正变换的手段来进行傅立叶反变换 对其取正变换之后再进行中心化

```
temp=np.zeros((h,w),np.complex)
        t=0
        for i in range (h):
            for j in range (w):
                t=np.conjugate(image[i,j])
                temp[i,j]=t
        temp2=dft_2d(temp)
        temp3=np.zeros((h,w),np.complex)
        for i in range (h):
            for j in range (w):
                temp3[i,j]=np.conjugate((temp2[i,j]/(h*w)))
        result3=center(temp3)
```

以上就是傅立叶变换以及傅立叶反变换的实现过程

2.3频率域滤波

编写一个在频域中执行fi抖动的函数。函数原型是“filter2d_freq(Input_img，filter)→Output_img”，其中“filter”是给定的filter。如果有必要，您可以修改原型。根据卷积定理，频域上的fi抖动要求对给定的图像和filter应用dft/fft，然后对它们进行乘，然后再加fi/IFFT，得到fi抖动的结果。因此，应该很容易基于“dft2d”(或“fft2d”)实现“filter2d_freq”。 1.用5×5平均filter平滑输入图像。将结果粘贴到报表上。(8

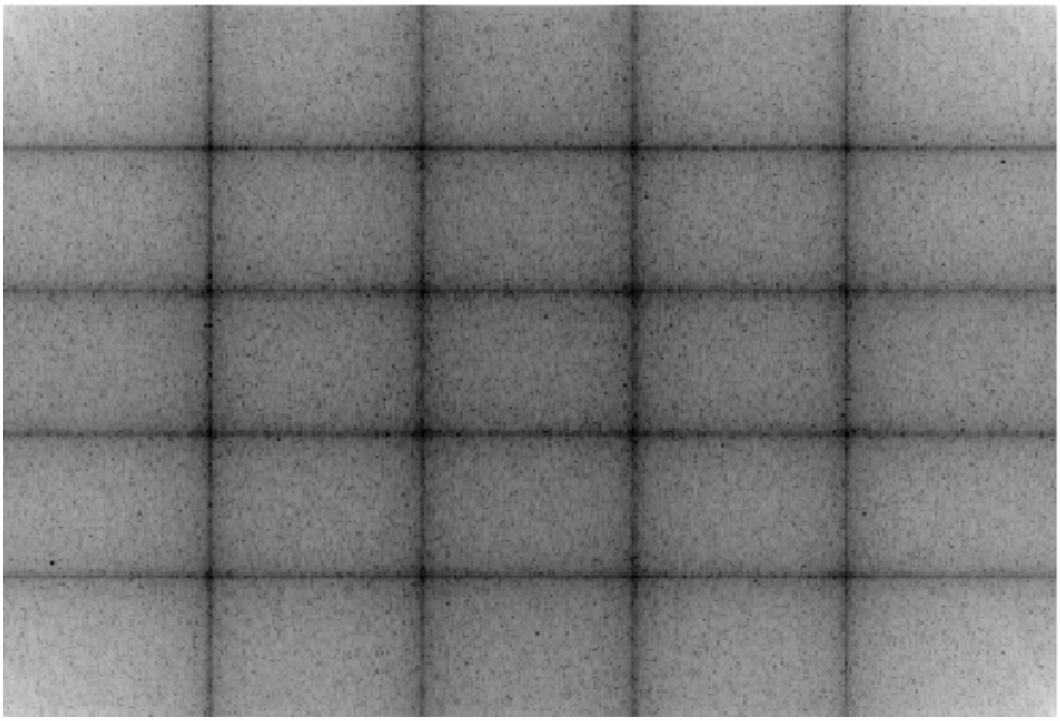
分）  2. 用以下3×3 拉普拉斯 filter 对输入图像进行锐化，然后粘贴结果。（8 分）

 3. 详细讨论如何在不到2页的篇幅内实现操作功能。（14分） 在上一个问题中，我们实现了dft与idft 在这里我们要实现的是频率域的滤波 基本步骤如下

图像为h * w,滤波器为m * n，则填充之后的图像与滤波器尺寸均为(h+m,w+n) 填充的具体方法是在右下角补0 下面是填充的基本操作，n和m为填充之后的尺寸

```
def padding(image,n,m):
    N=image.shape[0]
    M=image.shape[1]
    output=np.zeros((n,m))
    for i in range(n):
        for j in range(m):
            output[i,j]=0
    for i in range(N):
        for j in range(M):
            output[i,j]=image[i,j]
    return output
```

对原图像以及滤波器填充之后做傅立叶变换，得到的结果Fuv,Huv，将他们相乘之后得到结果Guv是滤波之后的结果，对其取log，则可以得到它的频率域图像



接下来我们对Guv做频率域反变换以及中心化，最后再对图像进行裁剪即可得到结果 操作如下

```
iimg=dft2d(Guv,1)
iimg = center(iimg)
output=np.zeros((h,w))
for i in range(h):
        for j in range(w):
            output[i,j]=iimg[i+2,j+2]
```

而拉普拉斯锐化的步骤与均值滤波类似，只不过填充的尺寸以及滤波器不同