HW3: 频率域滤波

1 习题

1.1 离散傅里叶变换对 (10 分)

课本上的DFT的变换公式为：

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

IDFT:

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

我们可以计算$F(0,0)$的值：

$$F(0,0)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)$$

那么我们可以把图像中的所有点的灰度值加起来得到结果 `gray`

1. 如果$F(0,0)==gray$, 则$\frac{1}{MN}$包含在IDFT前。
2. 如果$MN*F(0,0)==gray$，则$\frac{1}{MN}$包含在DFT前。
3. 如果$F(0,0)==\frac{1}{\sqrt{MN}}*gray$, 则以$\frac{1}{\sqrt{MN}}$分别包含在两项的前面。

1.2 傅里叶频谱(15 分)

1(b)与**1(c)**对应的傅里叶频谱一样，填充的目的是为了在DFT的周期间建立缓冲，最后在通过傅里叶变换转到频率域。频率域的傅里叶变换具有周期性，两幅图在平面上无限拼接后呈现的结果是一样的，所以求出的傅里叶频谱相同。

1.3 频率域滤波器(15 分)

1) 设 $g(x,y)$ 是空间域中卷积后结果.

$$g(x,y)=\begin{bmatrix} f(x-1,y-1) & f(x-1,y) & f(x-1,y+1) \\ f(x,y-1) & f(x,y) & f(x,y+1) \\ f(x+1,y-1) & f(x+1,y) & f(x+1,y+1) \end{bmatrix} \star \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$= \frac{1}{4}[f(x-1,y)+f(x,y-1)+f(x,y+1)+f(x+1,y)]$$

由平移性质，左右分别作傅立叶变换：

$$G(u,v)=\frac{1}{4}(e^{j2\pi u/M}+e^{j2\pi v/N}+e^{-j2\pi u/M}+e^{-j2\pi v/N})\cdot F(u,v)$$

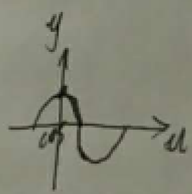
$$= H(u,v) * F(u,v).$$

$$\therefore H(u,v)=\frac{1}{4}(e^{j2\pi u/M}+e^{j2\pi v/N}+e^{-j2\pi u/M}+e^{-j2\pi v/N})$$

$$= \frac{1}{2}[\cos(2\pi\frac{u}{M})+\cos(2\pi\frac{v}{N})] \quad ①$$

2) 考虑①式，v 一定时，u 从 $0\to M-1$ 时. $2\pi\frac{u}{M}$ ~~从 0→1→0→-1~~

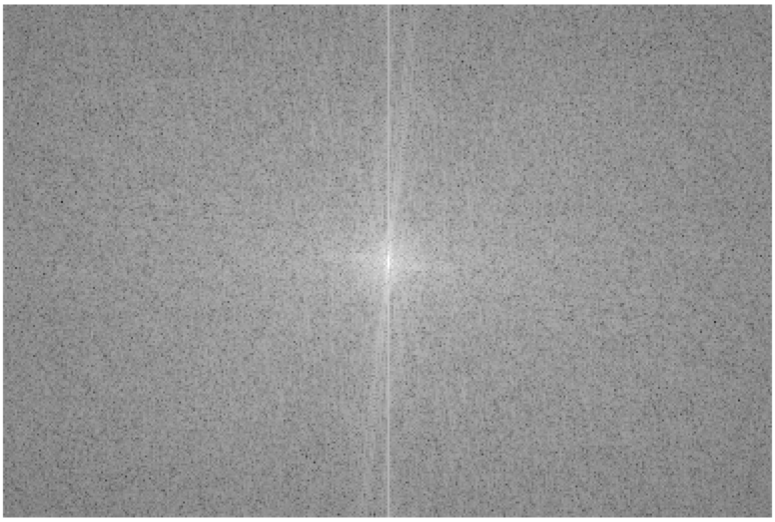
$2\pi\frac{u}{M}$ 从 $0\to 2\pi$，$\cos(2\pi\frac{u}{M})$ 从 $1\to 0\to -1\to 0\to 1$



，u 远离原点时，$H(u,v)\downarrow$，这是低通滤波器的特征.

2 编程题

2.2 傅里叶变换(30 分)

1. 

Fourier_Spectrum_img

2. 

dft_and_idft_result

3. 关于 `dft2d` 的计算，朴素的算法是使用课本上的原始公式，进行一个四重循环，但是实际上的运行时间非常长，因为图片本身的大小是(256*384)，每次遍历的话耗时非常长，加上使用的python实现，所以使用朴素的算法进行暴力求解显然行不通。
   课本4.11有一个优化的方法，利用二维DFT的可分性，把二维的DFT拆成了分别对行和列进行一维的DFT，从而减少了一层循环，运行效率提升到可以接受的范围。具体实现的算法如下：

   1. 根据要求，通过flags来判读是进行正变换还是逆变换。
   2. 对于正变换来说，先对输入矩阵的每一行进行一维的DFT处理，调用 `dft1d(f)` 来实现，然后对处理后的矩阵的每一列也进行一维的DFT处理，从而完成二维DFT的处理。
   3. 对于逆变换来说，同理，先对输入矩阵的每一行进行一维的DFT逆处理，调用 `idft1d(F)` 来实现，然后对处理后的矩阵的每一列也进行一维的DFT逆处理，从而完成二维DFT的逆处理。

4. 为了代码的简便，对于元操作`dft1d(f)`，使用矩阵运算的表达方式， 先构造好关于W_M的矩阵，然后直接用W和f相乘即可得到一维DFT变换

```python
def dft1d(f):
    """
    使用矩阵运算的方法来计算一维的傅里叶变换
    """
    M = f.shape[0]
    W = np.array([[np.exp(-1j*2*np.pi*u*x/M) for x in range(M)] for u in range(M)])
    return W.dot(f)
```

`idft1d(F)`也是同理，只是W矩阵中的细节每一项指数上多一个负号，计算完后每个元素都要除以MN即可。

2.4 频率域滤波(30 分)

1.

meanfilter3x3

meanfilter7x7

meanfilter11x11

2. 

拉普拉斯锐化

3. 滤波操作，在实现了dft2d的基础上，变的容易实现。按照课本提供的频率域滤波流程小结实现基本即可，有一些地方需要微调。

    1. 给定一幅大小为M*N的输入图像f(x,y), 滤波器大小为C*D，得到填充参数P和Q，P 不小于 M + C - 1, Q不小于N + D - 1。这里我统一使用最小值，因为使用普通的 `dft2d` 的话，应该尽量减少运算的规模。
    2. 对f(x, y)右下方补充必要数量的0，形成P*Q的图像$f_p(x, y)$
    3. 每个元素乘$(-1)^{x+y}$进行中心化
    4. 计算来自步骤3的图像的DFT，得到F(u,v)
    5. 对滤波器h(x,y)右下方补充必要数量的0，形成P*Q的矩阵$h_p(x, y)$
    6. 将h(x,y)的中心位置移到$h_p(x, y)$的左上角原点，使用循环移动的方法，目的是为了保持空间域与频率域滤波结果的一致性。
    7. 计算来自步骤6的矩阵的DFT，得到H(u,v)
    8. 计算$G(u, v) = H(u, v) * F(u, v)$
    9. 将G(u,v)作傅里叶反变换，并且进行中心化还原，取左上角的[M,N]大小的矩阵部分，得到处理后的图像g(x, y)
    10. 为了使g(x,y)正常显示，还需要进行标定操作。