图像处理与可视化 Homework3 报告

林子开 21307110161

2023 年 10 月 22 日

目录

1 Implementation of Spatial Filter

1.1 Smoothing

1.1.1 Introduction to My Algorithm and the Python Codes

I've completed the smoothing algorithm in two methods. The first is to take the **local average** of the kernel and the second is to make convolution with a **Gauss kernel**.

User may set different kernel size for the first method, e.g., 3×3, and the corresponding efficient, e.g., $\frac{1}{9}$, will be computed automatically. As for the Gauss kernel method, user need to set σ. The algorithm assumes the density function of Gauss distribution beyond 3σ is just 0 and will automatically calculate the kernel size.

In addition, my algorithm will pad the original figure with pixels of intensity 127 before smoothing, in order to maintain the size of smoothed figures as the original one.

Here are the Pythond codes of my smoothing algorithm. Note that I've imported **numba** to speed up my algorithm. Please install it first.

Listing 1: Python codes of my smoothing algorithm

```
1 """实现图像域基于空间滤波器的平滑操作"""
2 import numpy as np
3 from numba import jit  # 加速器
```

```
import copy
from PIL import Image
from math import floor, pi, exp

@jit # 加速!
def smooth_averayge(pixels,kernelSize):
    """
    功能: 取卷积核覆盖范围内所有像素点平均值进行平滑操作
    input:
        pixels: 图像的所有像素点, 一个矩阵
        kernelSize: 卷积核的大小
    output:
        进行平滑操作之后的图像的所有像素点
    """
    # 首先进行padding
    m = kernelSize//2 # padding时边缘增加的宽度
    P = np.ones((pixels.shape[0]+2*m,pixels.shape[1]+2*m))
    P = P*127 # 边缘增加的部分用127填充
    P[m:P.shape[0]-m, m:P.shape[1]-m] = pixels
    pixels = copy.deepcopy(P) # 深拷贝
    # 平滑操作
    for i in  range(m,P.shape[0]-m):
        for j in  range(m,P.shape[1]-m):
            pixels[i,j] = np. sum(P[i-m:i+m+1, j-m:j+m+1])/kernelSize**2
    # 只返回中间部分
    return pixels[m:P.shape[0]-m,m:P.shape[1]-m]


def smooth_Gauss(pixels,sigma):
    """
    功能: 将卷积核设为高斯核, 并根据标准差sigma自动调整卷积核大小(3倍标准差之外都设为0), 进
        行平滑化处理
    input:
        pixels: 图像的所有像素点, 一个矩阵
        sigma: 高斯核的标准差
    output:
        进行平滑操作之后的图像的所有像素点
    """
    # 首先进行padding
    # from math import floor
    kernelSize = 1+2*floor(3*sigma)
    Kernal = np.empty((kernelSize,kernelSize)) # 高斯核
    m = kernelSize//2 # padding时边缘增加的宽度
    # from math import pi, exp
    for i in  range(kernelSize):
        for j in  range(kernelSize):
            Kernal[i,j] = (1/(2*pi*sigma**2)) * exp(-((i-m)**2+(j-m)**2)/(2*sigma**2))
    # print(np.sum(Kernal))

    P = np.ones((pixels.shape[0]+2*m,pixels.shape[1]+2*m))
    P = P*127 # 边缘增加的部分用127填充
    P[m:P.shape[0]-m, m:P.shape[1]-m] = pixels
```

```
    pixels = copy.deepcopy(P) # 深拷贝
    # 平滑操作
    for i in  range(m,P.shape[0]-m):
        for j in  range(m,P.shape[1]-m):
            pixels[i,j] = np. sum(P[i-m:i+m+1, j-m:j+m+1]*Kernal) # 元素相乘，Hadarmard积
    # 只返回中间部分
    return pixels[m:P.shape[0]-m,m:P.shape[1]-m]

def main():
    myPicture = 'test.jpeg'
    im_raw = Image. open(myPicture)
    im = im_raw.convert('L')
    pixels = np.array(im)


    # 以下用取局部平均值的方法进行平滑处理
    for kernelSize in [3,5,7,9]:
        pixels2 = smooth_averayge(pixels, kernelSize)
        pixels2 = pixels2.astype(np.uint8)
        new_im = Image.fromarray(pixels2)
        new_im.save('result1/smooth_average_kernelSize='+ str(kernelSize)+'.png')


    # 以下用高斯核的方法进行平滑处理
    for sigma in [1,2,3,4]:
        pixels2 = smooth_Gauss(pixels, sigma)
        pixels2 = pixels2.astype(np.uint8)
        new_im = Image.fromarray(pixels2)
        new_im.save('result1/smooth_Gauss_sigma='+ str(sigma)+'.png')


if __name__ == '__main__':
    main()
```

1.1.2 Experiment Results

The original figure is as follows:

图 1: the original figure

The smoothed figures by the local average method are like:

(a) kernel size = 3

(b) kernel size = 5

(c) kernel size = 7

(d) kernel size = 9

图 2: smoothed figures by local average method with different kernel sizes

The smoothed figures by the Gauss kernel method are like:

(a) $\sigma = 1$

(b) $\sigma = 2$

(c) $\sigma = 3$

(d) $\sigma = 4$

图 3: smoothed figures by Gauss kernel method with different σ

We can find the man's face are blurred after smoothing.

1.2 Sharpening

1.2.1 Introduction to my Algorithm and Python Codes

I also implement two methods for sharpening algorithm. The first method is to make convolution with a **Laplacian operator** and the second one is to add a **unsharp masking** to the original figure.

For the first method, users can set the coefficient w in $g = (1 + w\nabla^2)f$ and the algorithm will automatically check if the intensity after transform is still contained in $[0, 255]$. The intensity larger than 255 will be set 255, and the intensity less than 0 will be set 0.

For the second method, the algorithm first calculate the local average base on a Gauss Kernel with $\sigma = 1$ and then get the unsharp musk. After that, add the unsharp musk to the original figure, i.e., $g = f + kh_{\text{mask}}$. User can also set the coefficient k for it and the algorithm will still check and reset the intensity within $[0, 255]$.

Similaryly, the algorithm will pad with pixels with intensity 127 before sharpening, in order to keep the figure size.

Here are the python codes:

Listing 2: Python codes

```
"""实现图像域基于空间滤波器的锐化操作"""

import numpy as np
from numba import jit
from PIL import Image
from math import floor, pi, exp

@jit # 加速!
def Sharpen_Laplacian(pixels,w):
    """
    功能：将卷积核设为拉普拉斯算子，并基于拉普拉斯运算后的结果对图像进行锐化
    input:
        pixels: 图像的所有像素点，一个矩阵
        w: 拉普拉斯算子的系数
    output:
        进行锐化操作之后的图像的所有像素点
    """
    # 构建拉普拉斯算子
    Kernal = np.zeros((3,3))
    Kernal[1,1] = -4
    Kernal[0,1] = 1
    Kernal[1,0] = 1
    Kernal[1,2] = 1
    Kernal[2,1] = 1

    # padding
    m = 1 # padding时边缘增加的宽度
    P = np.ones((pixels.shape[0]+2*m,pixels.shape[1]+2*m))
    P = P*127 # 边缘增加的部分用127填充
```

```
    P[m:P.shape[0]-m, m:P.shape[1]-m] = pixels
    Laplace = np.array(P) # 深拷贝，用于存储图像梯度值
    # 求出图像梯度值
    for i in  range(m,P.shape[0]-m):
        for j in  range(m,P.shape[1]-m):
            Laplace[i,j] = np. sum(P[i-m:i+m+1, j-m:j+m+1]*Kernal) # 元素相乘，Hadarmard积

    # 只要图像梯度中间的部分
    Laplace = Laplace[m:P.shape[0]-m, m:P.shape[1]-m]

    # 归一化
    Laplace = Laplace/np. max(Laplace)
    # 锐化
    pixels = pixels + w*Laplace

    # 检查灰度值是否超出[0,255]，超出的部分都变成0或者255
    for i in  range(pixels.shape[0]):
        for j in  range(pixels.shape[1]):
            p = pixels[i,j]
            if p < 0:
                pixels[i,j] = 0
            elif p > 255:
                pixels[i,j] = 255

    return pixels



def sharpen_highboost(pixels,sigma,k):
    """
    功能：先根据标准差sigma调整高斯核大小，基于高斯核做卷积求出局部范围内的灰度平均值，
        然后求出unsharp masking，最后将原图像的灰度加上unsharp masking函数值，
        完成对图像的锐化
    input:
        pixels：图像的所有像素点，一个矩阵
        sigma：求平均值时所用高斯核的标准差
        k：unsharp masking的系数
    output:
        进行锐化操作之后的图像的所有像素点
    """
    # 首先构建高斯核用于求平均值
    # from math import floor
    kernelSize = 1 + 2*floor(3*sigma) # 高斯核的大小
    Kernal = np.empty((kernelSize,kernelSize)) # 高斯核
    m = kernelSize//2 # padding时边缘增加的宽度
    # from math import pi, exp
    for i in  range(kernelSize):
        for j in  range(kernelSize):
            Kernal[i,j] = (1/(2*pi*sigma**2)) * exp(-((i-m)**2+(j-m)**2)/(2*sigma**2))

    # # only for test
    # Kernal = np.ones((kernelSize,kernelSize))
```

```
    # Kernal = Kernal/np.sum(Kernal)

    # padding
    P = np.ones((pixels.shape[0]+2*m,pixels.shape[1]+2*m))
    P = P*127 # 边缘增加的部分用127填充
    P[m:P.shape[0]-m, m:P.shape[1]-m] = pixels
    unsharp_mask = np.array(P) # 深拷贝

    # 求平均值，并计算unsharp_mask
    for i in  range(m,P.shape[0]-m):
        for j in  range(m,P.shape[1]-m):
            unsharp_mask[i,j] = P[i,j] - np. sum(P[i-m:i+m+1, j-m:j+m+1]*Kernal)

    # 只要中间部分
    unsharp_mask = unsharp_mask[m:P.shape[0]-m,m:P.shape[1]-m]
    # 归一化
    unsharp_mask = unsharp_mask/np. max(unsharp_mask)
    # 锐化
    pixels = pixels + k*unsharp_mask

    # 检查灰度值是否超出[0,255]，超出的部分都变成0或者255
    for i in  range(pixels.shape[0]):
        for j in  range(pixels.shape[1]):
            p = pixels[i,j]
            if p < 0:
                pixels[i,j] = 0
            elif p > 255:
                pixels[i,j] = 255

    return pixels


def main():
    myPicture = 'test2.png'
    im_raw = Image. open(myPicture)
    im = im_raw.convert('L')
    pixels = np.array(im)

    ## 以下基于拉普拉斯算子进行锐化
    for w in [-20,-50,-80,-110]:
        pixels2 = Sharpen_Laplacian(pixels, w)
        pixels2 = pixels2.astype(np.uint8)
        new_im = Image.fromarray(pixels2)
        new_im.save('result2/sharpen_Laplacian_w='+ str(w)+'.png')

    ## 以下基于highboost的方法进行锐化
    sigma = 1 # 高斯核的标准差
    for k in [20,50,80,110]:
        pixels2 = sharpen_highboost(pixels, sigma, k)
        pixels2 = pixels2.astype(np.uint8)
        new_im = Image.fromarray(pixels2)
        new_im.save('result2/sharpen_highboost_sigma='+ str(sigma)+'_k='+ str(k)+'.png')
```

```
134
135
136 if __name__ == '__main__':
137     main()
```

We take a picture of the surface of planet as the test figure. It is like:

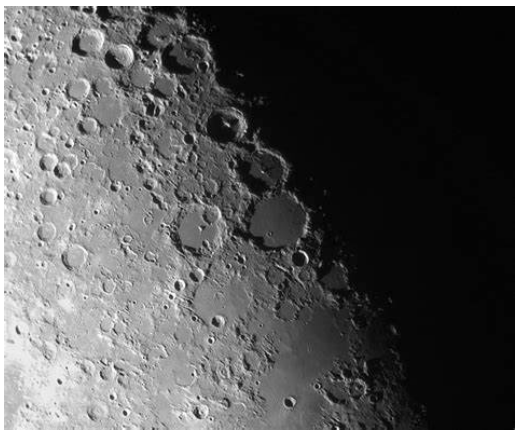
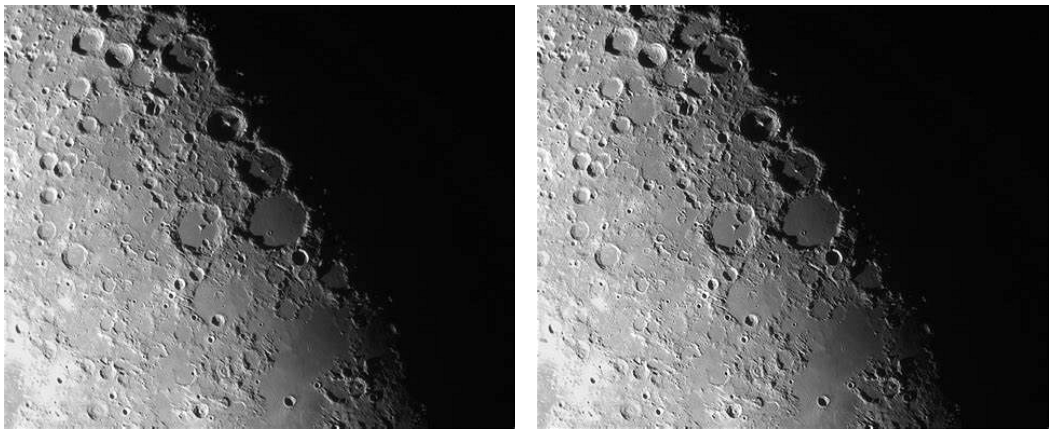


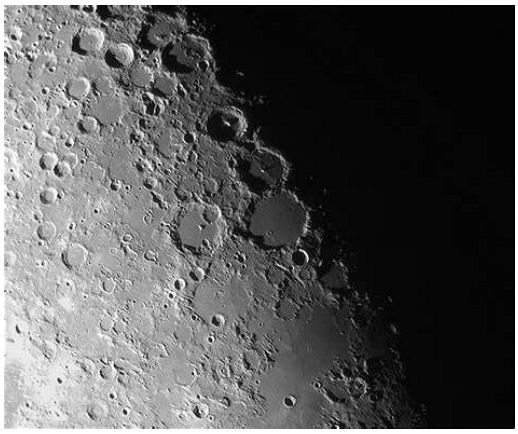
图 4: the original figure

And the results by Laplacian operator are as follows:

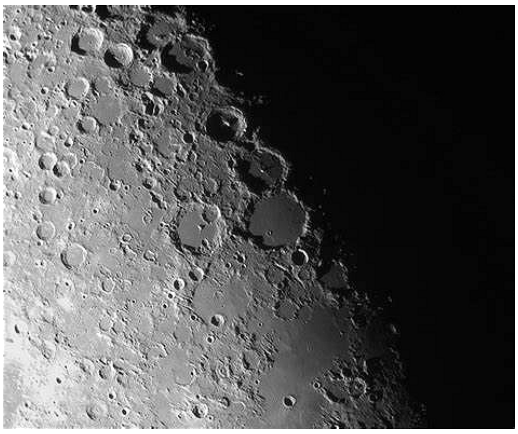


(a) w = -20

(b) w = -50



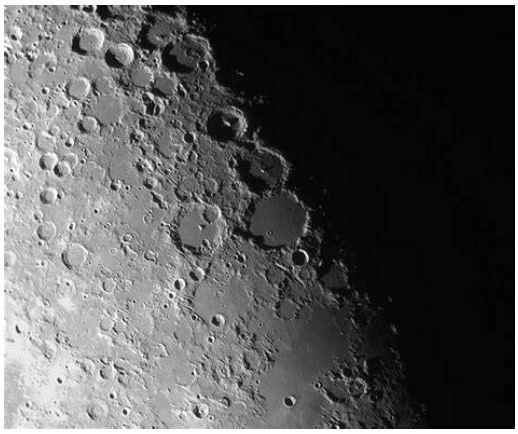
(c) w = -80



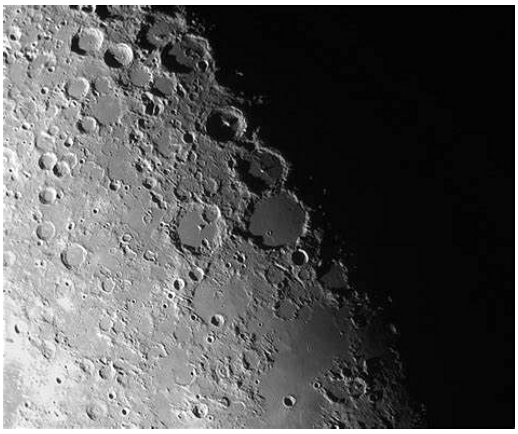
(d) w = -110

图 5: sharpened figures by Laplacian operator method with different w

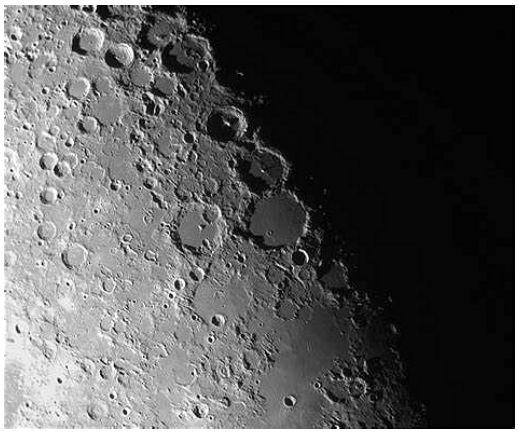
The results by adding unsharp musk are as follows:



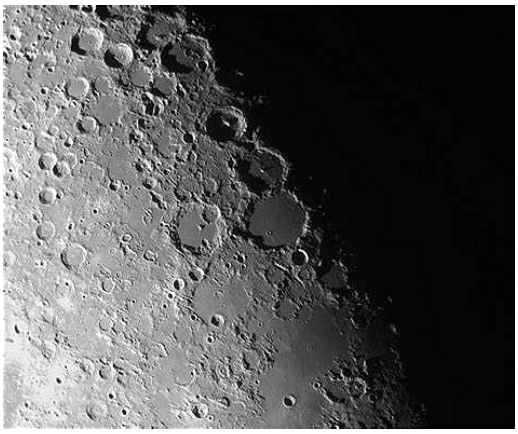
(a) $k = 20$



(b) $k = 50$



(c) $k = 80$



(d) $k = 110$

图 6: sharpened figures by adding unsharp musk with different k

We can find the sharpened figures show more details on the surface of the planet.

2 Proofs about Frequence Transform

2.1 Impulse Train after Frequence Transform

An impulse train is:

$$s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta T)$$

We can re-write $s_{\Delta T}(t)$ as Fourier series

$$s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} c_n \mathrm{e}^{j\frac{2\pi n}{\Delta T}t}$$

where

$$c_n = \frac{1}{\Delta}\int_{-\Delta T/2}^{\Delta T/2} s_{\Delta T}(t)\mathrm{e}^{-j\frac{2\pi n}{\Delta T}t}\,\mathrm{d}t$$

Since only $\delta(t)$ is contained in $[-\Delta T/2, \Delta T/2]$, therefore

$$c_n = \frac{1}{\Delta}\int_{-\Delta T/2}^{\Delta T/2} \delta(t)\mathrm{e}^{-j\frac{2\pi n}{\Delta T}t}\,\mathrm{d}t$$

$$= \frac{1}{\Delta T} \mathrm{e}^{0}$$
$$= \frac{1}{\Delta T}$$

Hence

$$s_{\Delta T}(t) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \mathrm{e}^{j\frac{2\pi n}{\Delta T}t}$$

Now derive the Fourier transform of $\mathrm{e}^{j\frac{2\pi n}{\Delta T}t}$. Since

$$\mathscr{F}^{-1}\{\ \delta(\mu-\mu_0)\} = \int_{-\infty}^{\infty} \delta(\mu-\mu_0)\mathrm{e}^{j2\pi\mu t}\,\mathrm{d}\mu$$
$$= \mathrm{e}^{j2\pi\mu_0 t}$$

therefore,

$$\mathscr{F}\left\{\mathrm{e}^{j2\pi\mu_0 t}\right\} = \delta(\mu-\mu_0)$$
$$\mathscr{F}\left\{\mathrm{e}^{j\frac{2\pi n}{\Delta T}t}\right\} = \delta(\mu-\frac{n}{\Delta T})$$

Finally we can obtain the Fourier transform of an impulse series

$$\mathscr{F}\{s_{\Delta T}(t)\} = \mathscr{F}\left\{\frac{1}{\Delta T}\sum_{n=-\infty}^{\infty}\mathrm{e}^{j\frac{2\pi n}{\Delta T}t}\right\}$$
$$= \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty}\mathscr{F}\left\{\mathrm{e}^{j\frac{2\pi n}{\Delta T}t}\right\}$$
$$= \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty}\delta(\mu-\frac{n}{\Delta T})$$

2.2 The Discrete Frequence Transform of a Real Signal

Suppose $f(x)$ is a real function, hence $\{f_n\}$ are also real. the DFT of $f(x)$ is:

$$F(m) = \sum_{n=0}^{M-1} f_n \mathrm{e}^{-j2\pi mn/M}, \quad m = 0, 1, 2, \ldots, M-1$$

And

$$F(-m) = \sum_{n=0}^{M-1} f_n \mathrm{e}^{j2\pi mn/M}$$

its conjucate is

$$\overline{F(-m)} = \overline{\sum_{n=0}^{M-1} f_n \mathrm{e}^{j2\pi mn/M}} = \sum_{n=0}^{M-1} f_n \mathrm{e}^{-j2\pi mn/M} = F(m)$$

Now we come to the conclusion that the DFT of real function $f(x)$ is conjugate symmetric.

2.3 The convolution theorem of 2-Dimension

The proof of the convolution theorem of 2-dimension is as follows:

$\mathscr{F}\{f(x,y) * h(x,y)\}(u,v)$

$$
\begin{aligned}
&= \mathscr{F}\left\{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)h(x-m,y-n)\right\}(u,v) \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)h(x-m,y-n)\exp\left\{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right\} \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)h(x-m,y-n)\exp\left\{-j2\pi\left(\frac{u(x-m)}{M}+\frac{v(y-n)}{N}+\frac{um}{M}+\frac{vn}{N}\right)\right\} \\
&= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(m,n)h(x-m,y-n)\exp\left\{-j2\pi\left(\frac{u(x-m)}{M}+\frac{v(y-n)}{N}+\frac{um}{M}+\frac{vn}{N}\right)\right\} \\
&= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)\exp\left\{-j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)\right\}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} h(x-m,y-n)\exp\left\{-j2\pi\left(\frac{u(x-m)}{M}+\frac{v(y-n)}{N}\right)\right\}
\end{aligned}
$$

Since both $h(x,y)$ **and** $\exp\left\{-j2\pi\left(\frac{u(x-m)}{M}+\frac{v(y-n)}{N}\right)\right\}$ **are periodic**, then we attain

$$
\begin{aligned}
&\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} h(x-m,y-n)\exp\left\{-j2\pi\left(\frac{u(x-m)}{M}+\frac{v(y-n)}{N}\right)\right\} \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} h(x,y)\exp\left\{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right\} \\
&= \mathscr{F}\left\{h(x,y)\right\}(u,v) \\
&= H(u,v)
\end{aligned}
$$

Therefore

$$
\begin{aligned}
&\mathscr{F}\{f(x,y)*h(x,y)\}(u,v) \\
&= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)\exp\left\{-j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)\right\}H(u,v) \\
&= \mathscr{F}\left\{f(m,n)\right\}(u,v)\cdot H(u,v) \\
&= F(u,v)\cdot H(u,v)
\end{aligned}
$$

On the other hand

$$
\begin{aligned}
&\mathscr{F}^{-1}\{F(u,v)*H(u,v)\}(x,y) \\
&= \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}\left\{F(u,v)*H(u,v)\right\}\exp\left\{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right\} \\
&= \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1} F(p,q)H(u-p,v-q)\exp\left\{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right\} \\
&= \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1} F(p,q)H(u-p,v-q)\exp\left\{j2\pi\left(\frac{(u-p)x}{M}+\frac{(v-q)y}{N}+\frac{px}{M}+\frac{qy}{N}\right)\right\} \\
&= \frac{1}{MN}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1} F(p,q)\exp\left\{j2\pi\left(\frac{px}{M}+\frac{qy}{N}\right)\right\}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} H(u-p,v-q)\exp\left\{j2\pi\left(\frac{(u-p)x}{M}+\frac{(v-q)y}{N}\right)\right\} \\
&= \frac{1}{MN}\left[MNf(x,y)\right]\left[MNh(x,y)\right] \\
&= MNf(x,y)\cdot h(x,y)
\end{aligned}
$$

Now we have completed the proof.