中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

学生姓名：　丁晓琪

学生学号：　22336057

专业名称：　计科

编程作业1：

1. 图片加入椒盐噪声：

- 理论：这里取$P_s = 0.2, P_p = 0.2$
- 实现：对原图像的每个像素随机一个0-1的概率p。当p<=0.2时，该像素位置加上盐粒噪声；当p>0.2且p<=0.4时，该像素位置加上胡椒噪声；p为其他值，维持原样

```
#添加椒盐噪声
def Add_SP_noise(noise_matrix):
    M,N=noise_matrix.shape
    for i in range(0,M):
        for j in range(0,N):
            random_num=np.random.rand()
            if(random_num<=0.2):
                noise_matrix[i][j]=255
            elif random_num>0.2 and random_num<=0.4:
                noise_matrix[i][j]=0
```

2.中值滤波

- 理论：

$$\hat{f}(x,y) = \underset{(r,c)\in S_{xy}}{median}\{g(r,c)\}$$

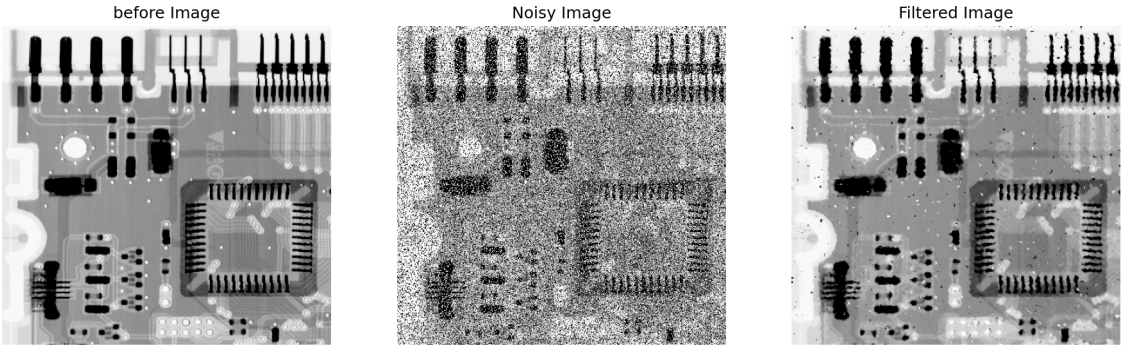
- 实现：为了保持滤波后图像的大小不变，这里对图像镜像拓展

```
def Median_Filtering(noise_matrix,kernel_size):
    # 输入：
    # noise_matrix:需要中值滤波的图像
    # kernel_size: 中值滤波核的大小
    M,N=noise_matrix.shape
    filtered_matrix = np.zeros_like(noise_matrix,
dtype=noise_matrix.dtype)
```

```
    # 1. 拓展边界,镜像扩展
    pad_size=kernel_size//2
    padded_matrix=np.pad(noise_matrix,((pad_size, pad_size), (pad_size, pad_size)), mode='reflect')
    print(padded_matrix.shape)
    # 2. 取每个窗口内的中值为该像素的滤波后的值
    for i in range(pad_size,M+pad_size):
        for j in range(pad_size,N+pad_size):
            window = padded_matrix[i-pad_size:i+pad_size, j-pad_size:j+pad_size]
            median_value=np.median(window)
            filtered_matrix[i-pad_size,j-pad_size]=median_value
    return filtered_matrix
```

3. 实验结果

中值滤波核大小为 5*5：可见添加椒盐噪声后图片中分布着不规则的黑白像素点，中值滤波后大部分消失，但是和5.10(b)比较仍然存在明显的少数椒盐噪声，可能是椒盐噪声分布过于密集，中值滤波无法完全去除。



编程作业2：

注意：由于这里图像的大小并不是2的幂次方，且自己实现傅里叶变换的效率过低，这里直接使用numpy库

1. 运动模糊

- 理论：

$$H(u,v)=\frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]e^{-j\pi(ua+vb)} \qquad (5.77)$$

- 实现：
  - 步骤：求图像的傅里叶变换，将图像的频域表示乘运动模糊退化核得到退化后的图像的频域表示，再做傅里叶逆变换得到模糊后的图像

```
    # 运动模糊
    DFT_gray_matrix=np.fft.fft2(gray_matrix) #默认中心化了
    Motion_Blur_matrix=Motion_Blur(M,N,1,0.1,0.1)*DFT_gray_matrix
    IFFT_Motion_Blur_matrix=np.real(np.fft.ifft2(Motion_Blur_matrix))
```

  - 计算运动模糊退化核的实现：由于np.fft.fft2会实现图像的频谱中心化，这里退化核也要频谱中心化

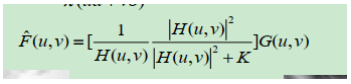
```python
def Motion_Blur(M,N, T, a, b):
    # 功能：计算运动模糊的退化滤波核
    # 输入：滤波核大小M，N，运动模糊参数T，a，b
    # 输出：频域上的运动模糊的滤波核
    H = np.zeros((M,N), dtype=np.complex128)
    for u in range(M):
        for v in range(N):
            # 要中心化的频率坐标
            tm = np.pi * ((u-M//2) * a + (v-N//2) * b)
            if abs(tm) < 1e-10:  # 避免除 0 错误
                H_u_v = T
            else:
                H_u_v = T * (np.sin(tm) * np.exp(-1j * tm)) / tm
            H[u,v] =  H_u_v

    return H

```

2. 添加高斯噪声

```python
    # 添加高斯噪声
    final_degenerate_image= Add_Gaussion_noise(IFFT_Motion_Blur_matrix,0,10)
def Add_Gaussion_noise(image,mean,variance):
    # 功能：对输入图像image加高斯噪声
    # 输入：图像的空域表达image,高斯噪声的参数mean,variance
    # 输出：加入高斯噪声后的图像result--matrix
    M,N=image.shape
    sigma=variance**0.5
    gauss=np.random.normal(mean,sigma,(M,N))
    gauss=gauss.reshape(M,N)
    result_matrix=image+gauss
    return result_matrix
```

3. 使用维纳滤波器恢复图像

- 理论：



- 实现：
    - 公式中的K用加上噪声后的图像的功率谱密度/原始图像的功率谱密度近似
      计算图像的功率谱的实现如下：

```python
def estimate_power_spectra(DFT_gray_matrix,G_u_v):
    # 功能：噪声功率谱密度和原始图像的功率谱密度。
    # 输入：原图像的频域表达 DFT_gray_matrix，加入噪声后的频域表达 G_u_v
    power_spectra_image = np.square(np.abs(DFT_gray_matrix))
    power_spectra_noise = np.square(np.abs(G_u_v))
    return power_spectra_image, power_spectra_noise
```

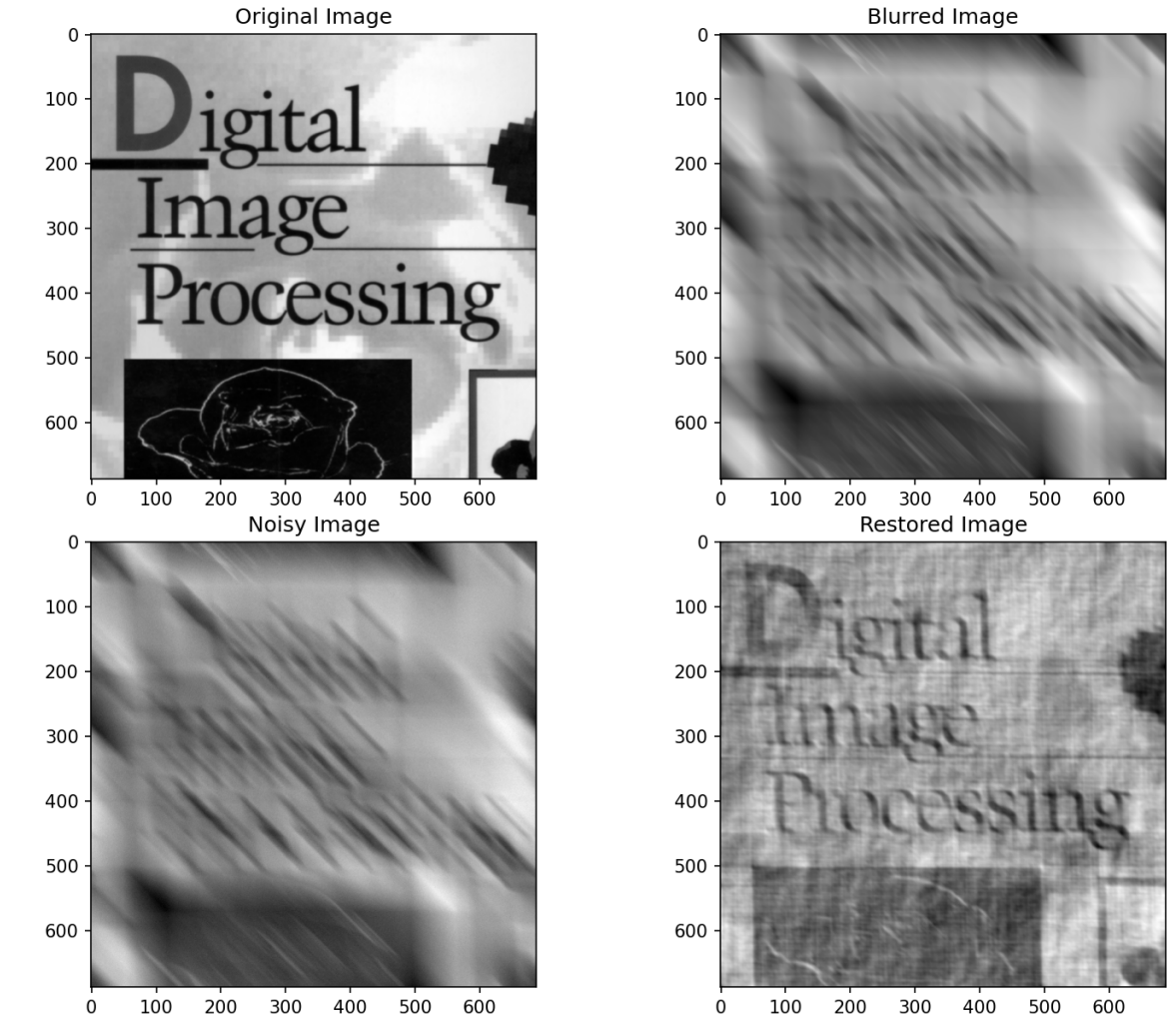
- 维纳滤波的实现：可以将公式转换为下述形式，先计算维纳滤波核，再计算滤波核乘上噪声图像得到滤波后的图像

$$\frac{H^*(u,v)}{|H(u,v)|^2+\frac{S_\eta(u,v)}{S_f(u,v)}}$$

```python
def Wiener_filtering(G_u_v,noise_power,image_power):
    # 功能：对退化和加上噪声的图像的频域G_u_v做维纳斯滤波
    # 输入：noise_power,image_power都是功率谱密度
    # 输出：对G--u_v加上维纳滤波
    M,N=G_u_v.shape
    H=Motion_Blur(M,N,1,0.1,0.1)
    H_conj = np.conj(H)
    H_abs=np.abs(H)
    H_abs_sq=np.square(H_abs)

    S_eta=noise_power
    S_f=image_power
    factor=S_eta/S_f
    wiener_filtering=H_conj / (H_abs_sq + factor)

    F_restore=wiener_filtering*G_u_v
    return F_restore
```

4. 实验结果

运动模糊后的图像可见在+45度的方向上模糊，维纳斯滤波后可见图像更加清晰，能够看清楚图片中的字母


Original Image
Blurred Image
Noisy Image
Restored Image
Digital
Image
Processing
0
100
200
300
400
500
600