图像处理第二次作业
——python实现十一种滤波器

3017218110　吴一汶

算数均值滤波器：

```
def means_filter(input_image, filter_size):
    # 算数均值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m= input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n= input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num, m - pad_num):
        for j in range(pad_num, n - pad_num):
            output_image[i, j] = np.sum(filter_template * input_image_cp[i - pad_num:i + pad_num +
1, j - pad_num:j + pad_num + 1]) / (filter_size **2)
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

几何均值滤波器：

```
def geometric_filter(input_image, filter_size):
    # 几何均值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m= input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n= input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            output_image[i,j]=np.prod(filter_template * input_image_cp[i - pad_num:i + pad_num +
1, j - pad_num:j + pad_num + 1]) ** (1/filter_size **2)
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

谐波均值滤波器：

```
def harmonic_filter(input_image, filter_size):
```

```
    # 谐波均值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            if(np.sum(filter_template*input_image_cp[i - pad_num:i + pad_num + 1, j - pad_num:j +
pad_num + 1])!=0):
                output_image[i, j] =filter_size**2/np.sum(1/(filter_template*input_image_cp[i -
pad_num:i + pad_num + 1, j - pad_num:j + pad_num + 1]))
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image

逆谐波均值滤波器：
def reverse_harmonic_filter(input_image, filter_size,q):
    # 逆谐波均值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            if np.sum((filter_template*input_image_cp[i - pad_num:i + pad_num + 1, j - pad_num:j +
pad_num + 1])**q)!=0:
                output_image[i, j] =np.sum((filter_template*input_image_cp[i - pad_num:i + pad_num +
1, j - pad_num:j + pad_num + 1])**(q+1))/np.sum((filter_template*input_image_cp[i - pad_num:i
+ pad_num + 1, j - pad_num:j + pad_num + 1])**q)
            else:
                output_image[i,j]=0.0
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image

中值滤波器：
def median_filter(input_image, filter_size):
    # 中值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
```

```
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            output_image[i, j] =np.median((filter_template*input_image_cp[i - pad_num:i + pad_num
+ 1, j - pad_num:j + pad_num + 1]))
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

最大值滤波器：

```
def max_filter(input_image, filter_size):
    # 最大值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            output_image[i, j] =np.max((filter_template*input_image_cp[i - pad_num:i + pad_num +
1, j - pad_num:j + pad_num + 1]))
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

最小值滤波器：

```
def min_filter(input_image, filter_size):
    # 最小值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
```

```
        for j in range(pad_num,n-pad_num):
            output_image[i, j] =np.min((filter_template*input_image_cp[i - pad_num:i + pad_num +
1, j - pad_num:j + pad_num + 1]))
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

中点滤波器：

```
def midpoint_filter(input_image, filter_size):
    # 中点滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            output_image[i, j] =(np.max((filter_template*input_image_cp[i - pad_num:i + pad_num +
1, j - pad_num:j + pad_num + 1]))+np.min((filter_template*input_image_cp[i - pad_num:i +
pad_num + 1, j - pad_num:j + pad_num + 1])))/2
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

阿尔法均值滤波器

```
def alpha_avg_filter(input_image, filter_size,d):
    # 阿尔法均值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    sum=0
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            sortedArr=np.sort(input_image_cp[i - pad_num:i + pad_num + 1, j - pad_num:j +
pad_num + 1],axis=None)
            for k in range(0, int(d/2)-1):
                sum=sum+sortedArr[k]+sortedArr[sortedArr.shape[0]-k-1]
            output_image[i, j] =np.sum(filter_template*(input_image_cp[i - pad_num:i + pad_num +
1, j - pad_num:j + pad_num + 1]-sum))/(filter_size**2-d)
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
```

```
    return output_image
```

自适应滤波器：

```
def adaptive_filter(input_image, filter_size):
    # 自适应滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            avg=np.mean(filter_template*input_image_cp[i - pad_num:i + pad_num + 1, j -
pad_num:j + pad_num + 1])
            v=np.var(filter_template*input_image_cp[i - pad_num:i + pad_num + 1, j - pad_num:j +
pad_num + 1])
            if(v!=0):
                output_image[i,j]=input_image_cp[i,j]-255**2*0.01/v*(input_image_cp[i,j]-avg)
            else:output_image[i,j]=input_image_cp[i,j]
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

自适应中值滤波器：

```
def adaptive_median_filter(input_image, filter_size):
    # 自适应中值滤波器
    input_image_cp = np.copy(input_image)  # 输入图像的副本
    pad_num = int((filter_size - 1) / 2)  # 输入图像需要填充的尺寸
    input_image_cp = np.pad(input_image_cp, (pad_num, pad_num), mode="constant",
constant_values=0)  # 填充输入图像
    m = input_image_cp.shape[0]  # 获取填充后的输入图像的大小
    n = input_image_cp.shape[1]  # 获取填充后的输入图像的大小
    output_image = np.copy(input_image_cp)  # 输出图像
    filter_template = np.ones((filter_size, filter_size))  # 空间滤波器模板

    for i in range(pad_num,m-pad_num):
        for j in range(pad_num,n-pad_num):
            len=1
            while len<=filter_size:
                zmin = np.min(filter_template * input_image_cp[i - int((len-1)/2):i + int((len-1)/2)+1, j -
int((len-1)/2):j + int((len-1)/2) + 1])
                zmax = np.max(filter_template * input_image_cp[i - int((len-1)/2):i + int((len-1)/2) + 1, j
- int((len-1)/2):j + int((len-1)/2) + 1])
```

```
                zmedian = np.median(filter_template * input_image_cp[i - int((len-1)/2):i + int((len-1)/2)
+ 1, j - int((len-1)/2):j + int((len-1)/2) + 1])
                A1 = zmedian - zmin
                A2 = zmedian - zmax
                B1 = input_image_cp[i, j] - zmin
                B2 = input_image_cp[i, j] - zmax
                if((A1>0)&(A2<0)):
                    break
                else:
                    len=len+2
                if len==filter_size+1 :
                    output_image[i,j]=input_image_cp[i,j]
            if ((B1>0)&(B2<0)):
                output_image[i, j] = input_image_cp[i, j]
                while len <= filter_size:
                    zmin = np.min(filter_template*input_image_cp[i - int((len-1)/2):i + int((len-1)/2) + 1, j
- int((len-1)/2):j + int((len-1)/2) + 1])
                    zmax = np.max(filter_template*input_image_cp[i - int((len-1)/2):i + int((len-1)/2) + 1,
j - int((len-1)/2):j + int((len-1)/2) + 1])
                    zmedian = np.median( filter_template * input_image_cp[i - int((len - 1) / 2):i + int((len
- 1) / 2) + 1,j - int((len - 1) / 2):j + int((len - 1) / 2) + 1])
                    A1 = zmedian - zmin
                    A2 = zmedian - zmax
                    B1 = input_image_cp[i, j] - zmin
                    B2 = input_image_cp[i, j] - zmax
                    if ((A1 > 0) & (A2 < 0)):
                        break
                    else:
                        len = len + 2
                    if len == filter_size + 1 :
                        output_image[i, j] = input_image_cp[i, j]
            else:
                output_image[i, j] = zmedian
    output_image = output_image[pad_num:m - pad_num, pad_num:n - pad_num]  # 裁剪
    return output_image
```

result：

原图像与经过处理之后的输出结果：

means

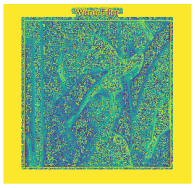
geometric

harmonic

median

adaptive_median

adaptive



max

min

midpoint

reverse_harmonic

alpha_avg