Problem 3

Digital Image Processing

肖逸群 116033910112

Filtering in frequency domain

问题描述

> Implement the ideal, Butterworth and Gaussian lowpass and highpass filters and test them under different parameters using characters_test_pattern.tif.

算法思想

1. 读取原始图像并转换为灰度图像，然后利用二维傅里叶变换将其转变到频率域的表示，这里考虑到效率原因，我使用了快速傅里叶变换。
   根据定义可知，二维傅里叶变换的公式如下：

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

   二维傅里叶反变换的公式如下：

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

2. 在进行傅里叶变换后，我们得到了频率域的表示，然后使用各种不同的滤波器来进行滤波，即对傅里叶变换后得到的矩阵进行卷积计算。
3. 本题要求包含了三种滤波算法的高通和低通滤波模型，高通滤波器的作用是使得高频部分保留，而低频部分衰减，因而高通滤波器将会减少平滑区域里头的灰度变化，突出边缘区域的灰度变化。低通滤波器的作用是使得低频部分得以保留，而高频部分衰减，因而低通滤波器将会减少边缘区域的灰度变化，突出平滑区域的灰度变化。
4. 本题用到的几种滤波器模型定义如下：
   a. **理想滤波器**
      二维理想低通滤波器定义如下：

$$H(u,v) = \begin{cases} 1, & D(u,v) \leqslant D_0 \\ 0, & D(u,v) > D_0 \end{cases}$$

      二维理想高通滤波器定义如下：

$$H(u,v)=\begin{cases}0, & D(u,v)\leqslant D_0\\ 1, & D(u,v)>D_0\end{cases}$$

其中D0是一个正常数，即半径，D(u,v)是频率域中点(u,v)与频率矩形中心的距离，即：

$$D(u,v)=\left[(u-P/2)^2+(v-Q/2)^2\right]^{1/2}$$

P,Q为图片填充后的尺寸，在我的算法中，我另P,Q分别为图片高和宽的2倍。
理想滤波器在半径D0的圆内，所有频率无衰减地通过，而圆外部分则完全被滤除。

b. **Butterworth 滤波器**
Butterworth低通滤波器的定义如下：

$$H(u,v)=\frac{1}{1+[\frac{D(u,v)}{D_0}]^{2n}}$$

Butterworth高通滤波器的定义如下：

$$H(u,v)=\frac{1}{1+[\frac{D_0}{D(u,v)}]^{2n}}$$

D与D0的定义同上。
Butterworth滤波器在通过频率和滤除频率之间是连续的，与理想滤波器的间断截止不同。

c. **高斯滤波器**
高斯低通滤波器的定义如下：

$$H(u,v)=e^{-\frac{D^2(u,v)}{2D_0^2}}$$

高斯高通滤波器的定义如下：

$$H(u,v)=1-e^{-\frac{D^2(u,v)}{2D_0^2}}$$

D与D0的定义同上。
高斯滤波器相比之前两个滤波器更加平滑，对微小物体和细线条使用高斯滤波器的结果也会较为清晰。

源码分析

- 读取原始图像，对原始图像进行各种滤波变换，最后展示图像

```
def main():
    originalImage = Image.open('./resource/characters_test_pattern.tif')
    radius = 15
```

```python

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    plt.subplot(2, 2, 2)
    idealLowpassImage = Frequency.idealLowpass(originalImage, radius)
    plt.imshow(idealLowpassImage, cmap = plt.get_cmap('gray'))
    plt.title('Ideal Lowpass With Radius = %s' % radius)

    plt.subplot(2, 2, 3)
    butterworthLowpassImage = Frequency.butterworthLowpass(originalImage, radius, 2)
    plt.imshow(butterworthLowpassImage, cmap = plt.get_cmap('gray'))
    plt.title('Butterworth Lowpass With Radius = %s' % radius)

    plt.subplot(2, 2, 4)
    gaussianLowpassImage = Frequency.gaussianLowpass(originalImage, radius)
    plt.imshow(gaussianLowpassImage, cmap = plt.get_cmap('gray'))
    plt.title('Gaussian Lowpass With Radius = %s' % radius)

    plt.show()

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    plt.subplot(2, 2, 2)
    idealHighpassImage = Frequency.idealHighpass(originalImage, radius)
    plt.imshow(idealHighpassImage, cmap = plt.get_cmap('gray'))
    plt.title('Ideal Highpass With Radius = %s' % radius)

    plt.subplot(2, 2, 3)
    butterworthHighpassImage = Frequency.butterworthHighpass(originalImage, radius, 2)
    plt.imshow(butterworthHighpassImage, cmap = plt.get_cmap('gray'))
    plt.title('Butterworth Highpass With Radius = %s' % radius)

    plt.subplot(2, 2, 4)
    gaussianHighpassImage = Frequency.gaussianHighpass(originalImage, radius)
    plt.imshow(gaussianHighpassImage, cmap = plt.get_cmap('gray'))
    plt.title('Gaussian Highpass With Radius = %s' % radius)

    plt.show()
```

- 快速傅里叶变换函数

```python
def FFT(originalImage, matrix):

    m, n = originalImage.size
    p = 2 * m
    q = 2 * n

    f = np.zeros((q, p), dtype = np.int)
    originalImageMatrix = originalImage.load()
    for x in range(m):
        for y in range(n):
            f[y][x] = originalImageMatrix[x, y] * ((-1) ** (x + y))

    F = np.fft.fft2(f)
    for x in range(p):
        for y in range(q):
            F[y][x] = F[y][x] * matrix[y][x]
```

```python

    G = np.fft.ifft2(F)

    outputImage = originalImage.copy()
    resultImageMatrix = outputImage.load()
    for x in range(m):
        for y in range(n):
            resultImageMatrix[x, y] = int( G[y][x].real * ((-1) ** (x + y)) )

    return outputImage
```

- 理想低通滤波变换和理想高通滤波变换

```python
def idealLowpass(originalImage, radius):

    m, n = originalImage.size
    p = 2 * m
    q = 2 * n

    matrix = np.zeros((q, p), dtype = np.int)

    for i in range(q):
        for j in range(p):
            D = ((i - n) ** 2 + (j - m) ** 2) ** 0.5
            if D <= radius:
                matrix[i][j] = 1

    resultImage = FFT(originalImage, matrix)
    return resultImage

def idealHighpass(originalImage, radius):

    m, n = originalImage.size
    p = 2 * m
    q = 2 * n
    matrix = np.zeros((q, p), dtype=np.int)

    for i in range(q):
        for j in range(p):
            D = ((i - n) ** 2 + (j - m) ** 2) ** 0.5
            if D > radius:
                matrix[i][j] = 1

    resultImage = FFT(originalImage, matrix)
    return resultImage
```

- Butterworth低通滤波变换和Butterworth高通滤波变换

```python
def butterworthLowpass(originalImage, radius, order):

    m, n = originalImage.size
    p = 2 * m
    q = 2 * n
    matrix = np.zeros((q, p))

    for i in range(0, q):
        for j in range(0, p):
            D = ((i - n) ** 2 + (j - m) ** 2) ** 0.5
            if radius != 0:
                matrix[i][j] = 1 / (1 + (D / radius) ** (2 * order))
            elif D == 0:
```

```
                matrix[i][j] = 1

    resultImage = FFT(originalImage, matrix)
    return resultImage

def butterworthHighpass(originalImage, radius, order):

    m, n = originalImage.size
    p = 2 * m
    q = 2 * n
    matrix = np.zeros((q, p))

    for i in range(q):
        for j in range(p):
            D = ((i - n) ** 2 + (j - m) ** 2) ** 0.5
            if D != 0:
                matrix[i][j] = 1 / (1 + (radius / D) ** (2 * order))

    resultImage = FFT(originalImage, matrix)
    return resultImage
```

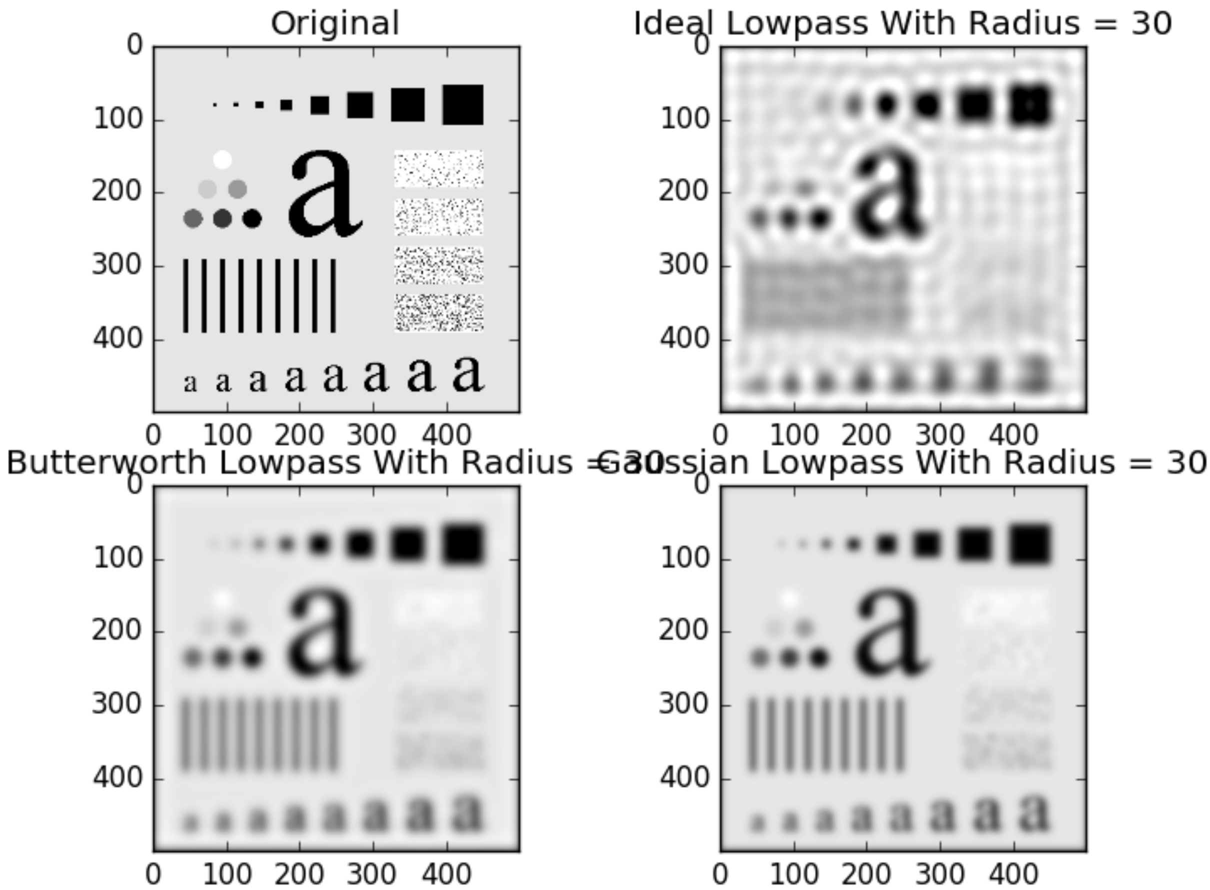
- 高斯低通变换和高斯高通变换

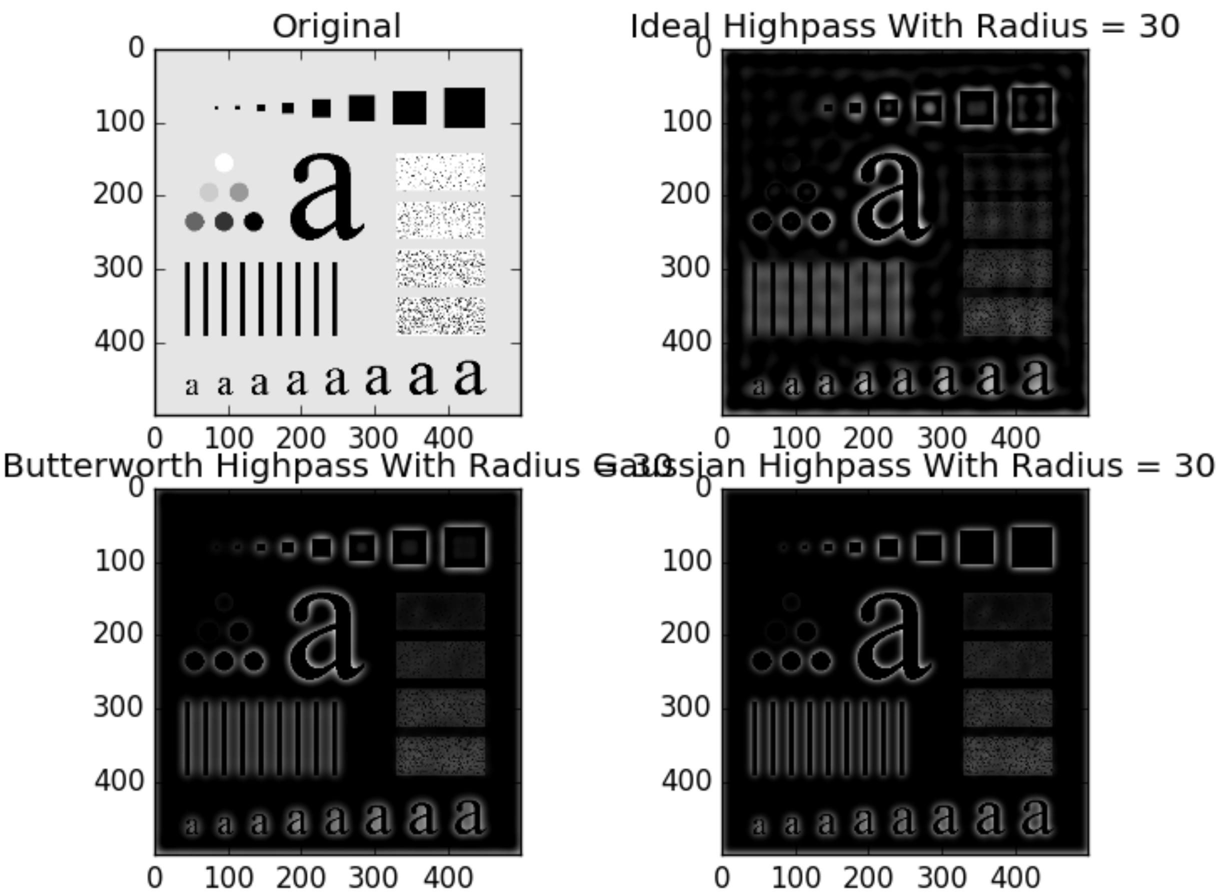
```
def butterworthLowpass(originalImage, radius, order):

    m, n = originalImage.size
    p = 2 * m
    q = 2 * n
    matrix = np.zeros((q, p))

    for i in range(0, q):
        for j in range(0, p):
            D = ((i - n) ** 2 + (j - m) ** 2) ** 0.5
            if radius != 0:
                matrix[i][j] = 1 / (1 + (D / radius) ** (2 * order))
            elif D == 0:
                matrix[i][j] = 1

    resultImage = FFT(originalImage, matrix)
    return resultImage

def butterworthHighpass(originalImage, radius, order):

    m, n = originalImage.size
    p = 2 * m
    q = 2 * n
    matrix = np.zeros((q, p))

    for i in range(q):
        for j in range(p):
            D = ((i - n) ** 2 + (j - m) ** 2) ** 0.5
            if D != 0:
                matrix[i][j] = 1 / (1 + (radius / D) ** (2 * order))

    resultImage = FFT(originalImage, matrix)
    return resultImage
```

实验结果

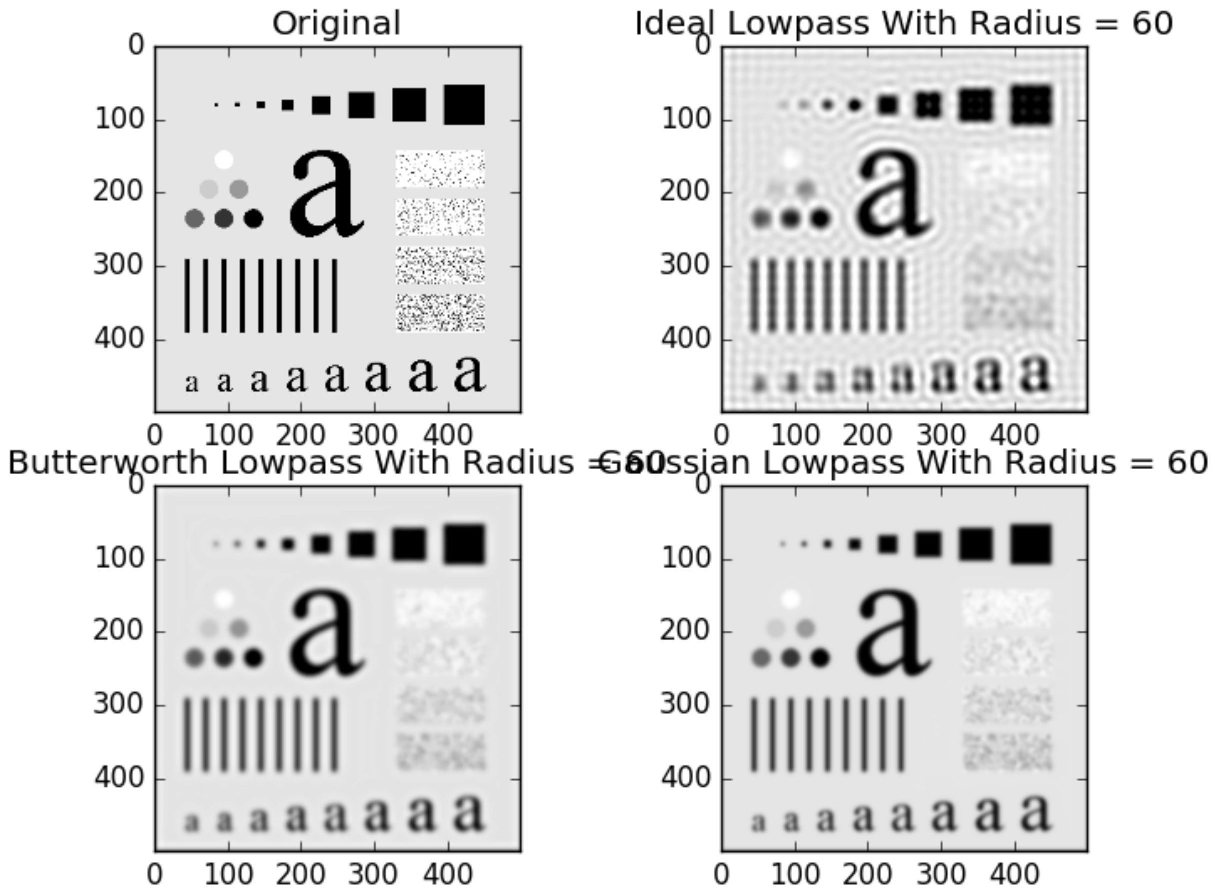
实验结果的输出如下列图片所示：

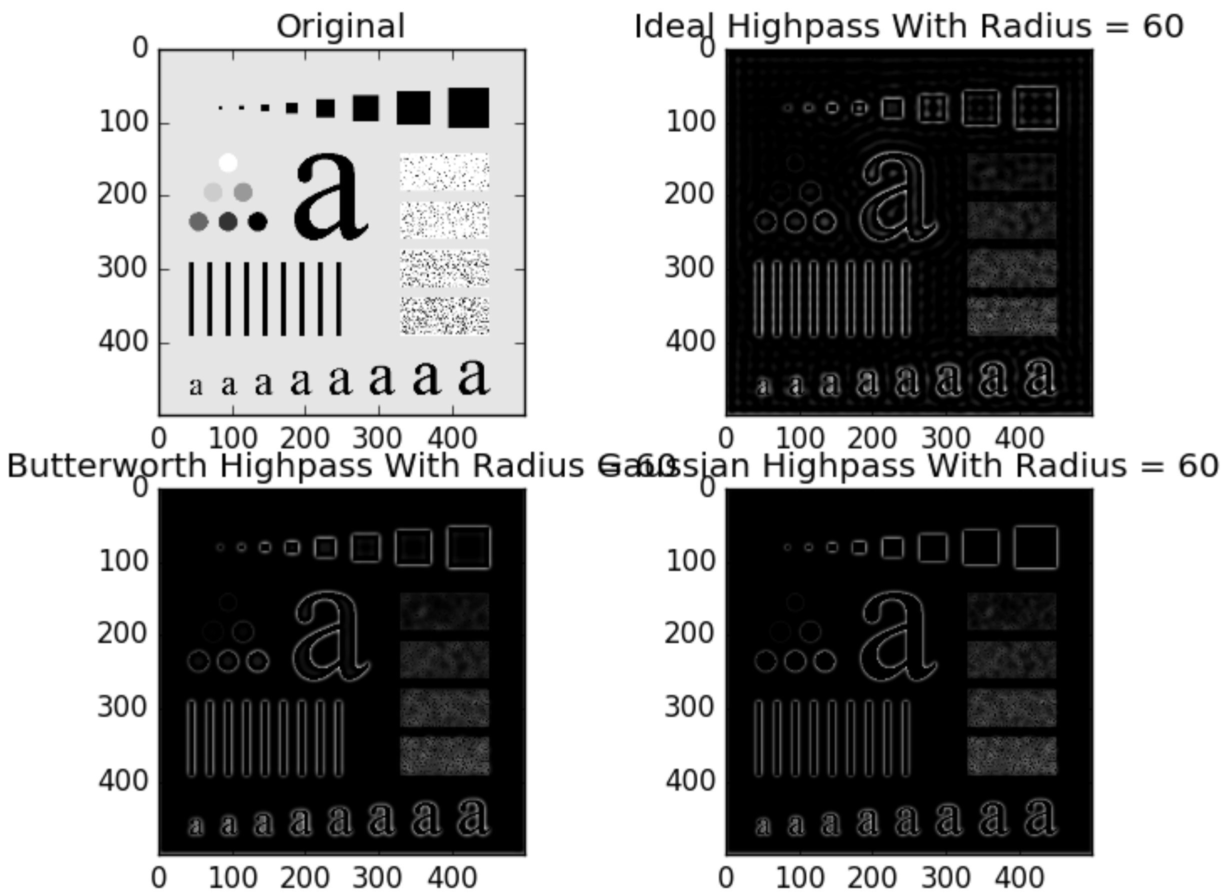
- 半径为30时：


Original
Ideal Lowpass With Radius = 30
Butterworth Lowpass With Radius = 30
Gaussian Lowpass With Radius = 30
0
100
200
300
400

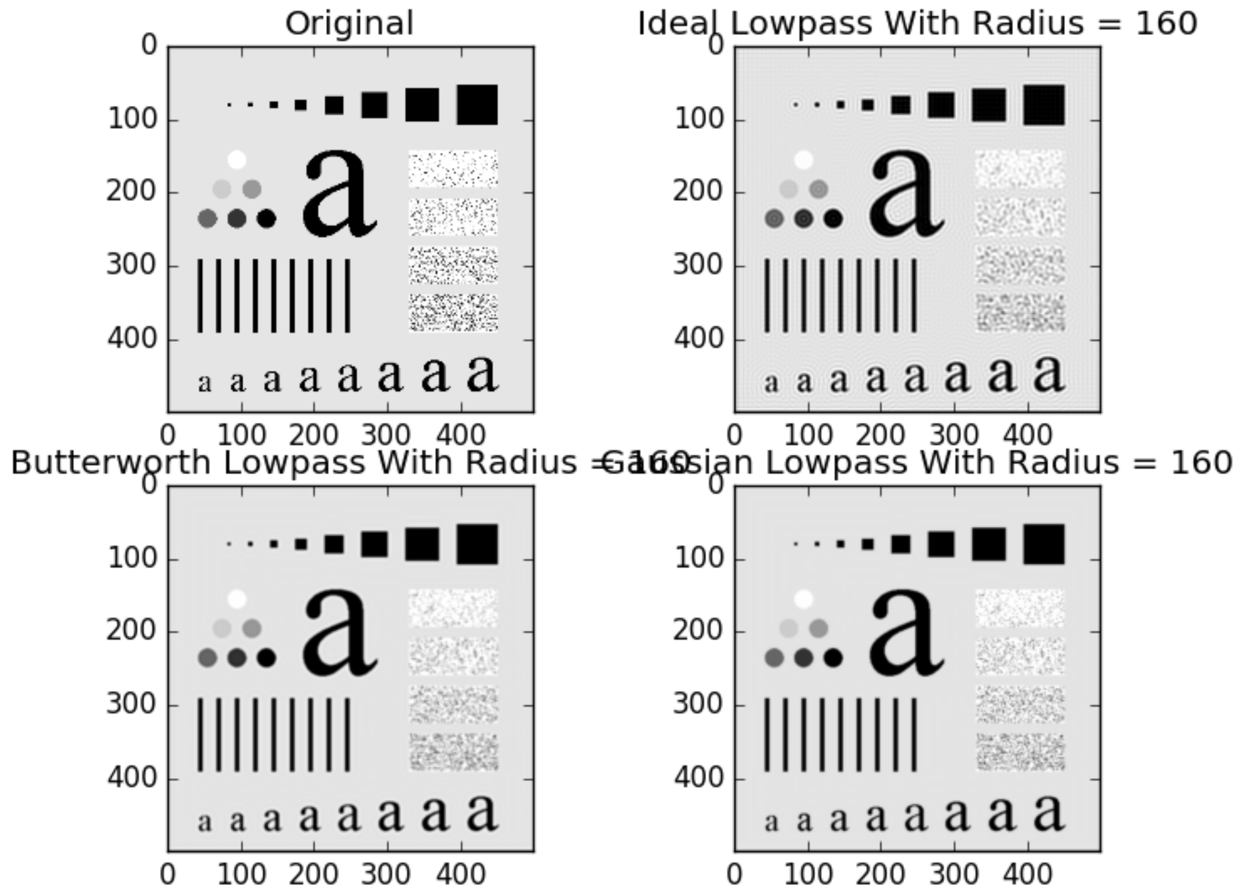

Original
Ideal Highpass With Radius = 30
Butterworth Highpass With Radius = 30
Gaussian Highpass With Radius = 30
0
100
200
300
400

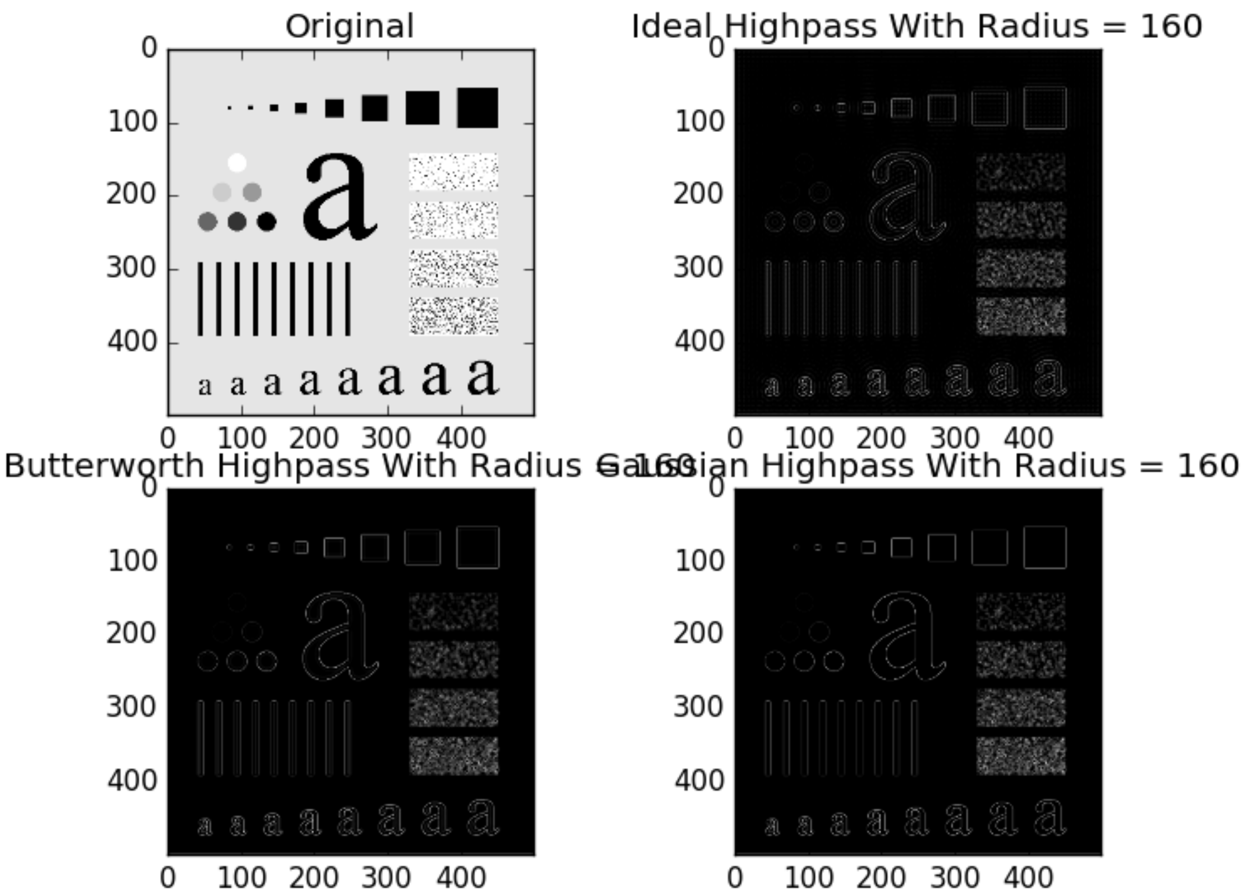
- 半径为60时：


Original
Ideal Lowpass With Radius = 60
Butterworth Lowpass With Radius = 60
Gaussian Lowpass With Radius = 60


Original
Ideal Highpass With Radius = 60
Butterworth Highpass With Radius = 60
Gaussian Highpass With Radius = 60

- 半径为160时：


Original
Ideal Lowpass With Radius = 160
Butterworth Lowpass With Radius = 160
Gaussian Lowpass With Radius = 160


Original
Ideal Highpass With Radius = 160
Butterworth Highpass With Radius = 160
Gaussian Highpass With Radius = 160

由结果可知，当半径较小时候，理想滤波器因为其在半径之外的部分全部滤除，导致结果较为模糊，而高斯滤波器因为其通过频率和截断频率间最为平滑，故结果最为清晰。
随着半径增大，保留下来的频率范围也随之增大，大到一定范围的时候几种滤波模型所呈现的效果并无太大区别。