Image and Video Processing Lab
Lab 2: Image Filtering in Frequency Domain

Aruna Shaju Kollannur (SC21M111)

Sub: Image and Video Processing Lab
Department: Avionics(DSP)
Date of Lab sheet: October 5, 2021
Date of Submission: October 20, 2021

Question 1: Display the power spectrum of 'alphabet.tif ' and compute the amount of power enclosed within radii of 10, 30, 60, 160, and 460 pixels with respect to the full-size spectrum.

Aim

Immplementing algorithm to calculate the Power spectrum of a given image and determine the power enclosed within a given pixel radius.

Discussion

Filtering techniques in the frequency domain are based on modifying the Fourier transform to achieve a specific objective, and then computing the inverse DFT to get us back to the spatial domain. For a given (a padded) digital image, f (x, y), of size P x Q pixels,
- Each pixel is multiplied with

$$(-1)^{(x+y)}$$

to centre the transform
- Computations with filter is done with center at (P/ 2,Q/ 2) as radius 0.

Algorithm

```
Step 1: Start
Step 2: Read the image
Step 3: Obtain the height and width of the image.
Step 4: Zero pad the image to size (2*height, 2*width ) dimensions.
Step 5: Multiply each pixel with (-1)^(x+y) to centre the DFT.
Step 6: Take the DFT of the image.
Step 7: Obtain the total power contained and power spectrum in the image.
Step 8: Run a loop to obtain the power enclosed within different radius measures.
Step 9: Display power enclosed and power spectrum results.
```

Program Code

```
"""
Display the power spectrum of 'alphabet.tif ' and compute the amount of
power enclosed within radii of 10, 30, 60, 160, and 460 pixels with respect to
the full-size spectrum.
```

```
"""
import numpy as np
import cv2
import math
import matplotlib.pyplot as plt

def Ideal_LPF_Filter(Image,cutoff):
    height, width = Image.shape

    Ideal_LPF = np.full((height, width),0)

    # Creating the Ideal Low Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d = math.sqrt((i-int(height)/2)**2 + (j-int(width)/2)**2)
             if (d <= cutoff):
                 Ideal_LPF[i,j] = 1

    Spectrum_covered = Ideal_LPF*Image
    return Spectrum_covered

def Total_Power(Image):
    height, width = Image.shape
    power_image = 0

    for i in range(height):
        for j in range(width):
           power_image = power_image + (abs(Image[i,j])**2)

    return power_image

def Power_Contained(Image, radius, Total_Power_image):
    height, width = Image.shape
    power_radius = 0

    for i in range(height):
        for j in range(width):
             d = math.sqrt((i-int(height)/2)**2 + (j-int(width)/2)**2)
             if (d <= radius):
                 power_radius = power_radius + (abs(Image[i,j])**2)

    alpha = (power_radius/Total_Power_image)*100

    return alpha

Alphabet_Image = cv2.imread("alphabet.tif", 0)
height, width = Alphabet_Image.shape

# Zero Padding
Alphabet_pad = np.pad(np.array(Alphabet_Image), ((int(height/2),int(height/2)),
               (int(width/2),int(width/2))),'constant')
Alphabet_pad_centre = np.full((2*height, 2*width),0)

# Centering
for i in range(height*2):
    for j in range(width*2):
```

```
        Alphabet_pad_centre[i,j] = Alphabet_pad[i,j]*((-1)**(i+j))

Alphabet_Image_FFT = np.fft.fft2(Alphabet_pad_centre)

Total_Power_Alphabet_Pad = Total_Power(Alphabet_Image_FFT)

radius_10 = Power_Contained(Alphabet_Image_FFT, 10 , Total_Power_Alphabet_Pad)
power_spectrum_10 = Ideal_LPF_Filter(Alphabet_Image_FFT, 10)
print(f' Power containd in 10 pixel radius is = {radius_10} %')

radius_30 = Power_Contained(Alphabet_Image_FFT, 30 , Total_Power_Alphabet_Pad)
power_spectrum_30 = Ideal_LPF_Filter(Alphabet_Image_FFT, 30)
print(f' Power containd in 30 pixel radius is = {radius_30} %')

radius_60 = Power_Contained(Alphabet_Image_FFT, 60 , Total_Power_Alphabet_Pad)
power_spectrum_60 = Ideal_LPF_Filter(Alphabet_Image_FFT, 60)
print(f' Power containd in 60 pixel radius is = {radius_60} %')

radius_160 = Power_Contained(Alphabet_Image_FFT, 160 , Total_Power_Alphabet_Pad)
power_spectrum_160 = Ideal_LPF_Filter(Alphabet_Image_FFT, 160)
print(f' Power containd in 160 pixel radius is = {radius_160} %')

radius_460 = Power_Contained(Alphabet_Image_FFT, 460 , Total_Power_Alphabet_Pad)
power_spectrum_460 = Ideal_LPF_Filter(Alphabet_Image_FFT, 460)
print(f' Power containd in 460 pixel radius is = {radius_460} %')

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(Alphabet_Image_FFT)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(power_spectrum_10)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(power_spectrum_30)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(power_spectrum_60)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(power_spectrum_160)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(power_spectrum_460)), cmap='gray');
```

Result

```
 Power containd in 10 pixel radius is = 86.9904451066706 %
 Power containd in 30 pixel radius is = 93.08230076141847 %
 Power containd in 60 pixel radius is = 95.6690435869182 %
 Power containd in 160 pixel radius is = 97.77493855071388 %
 Power containd in 460 pixel radius is = 99.1825569515614 %
```

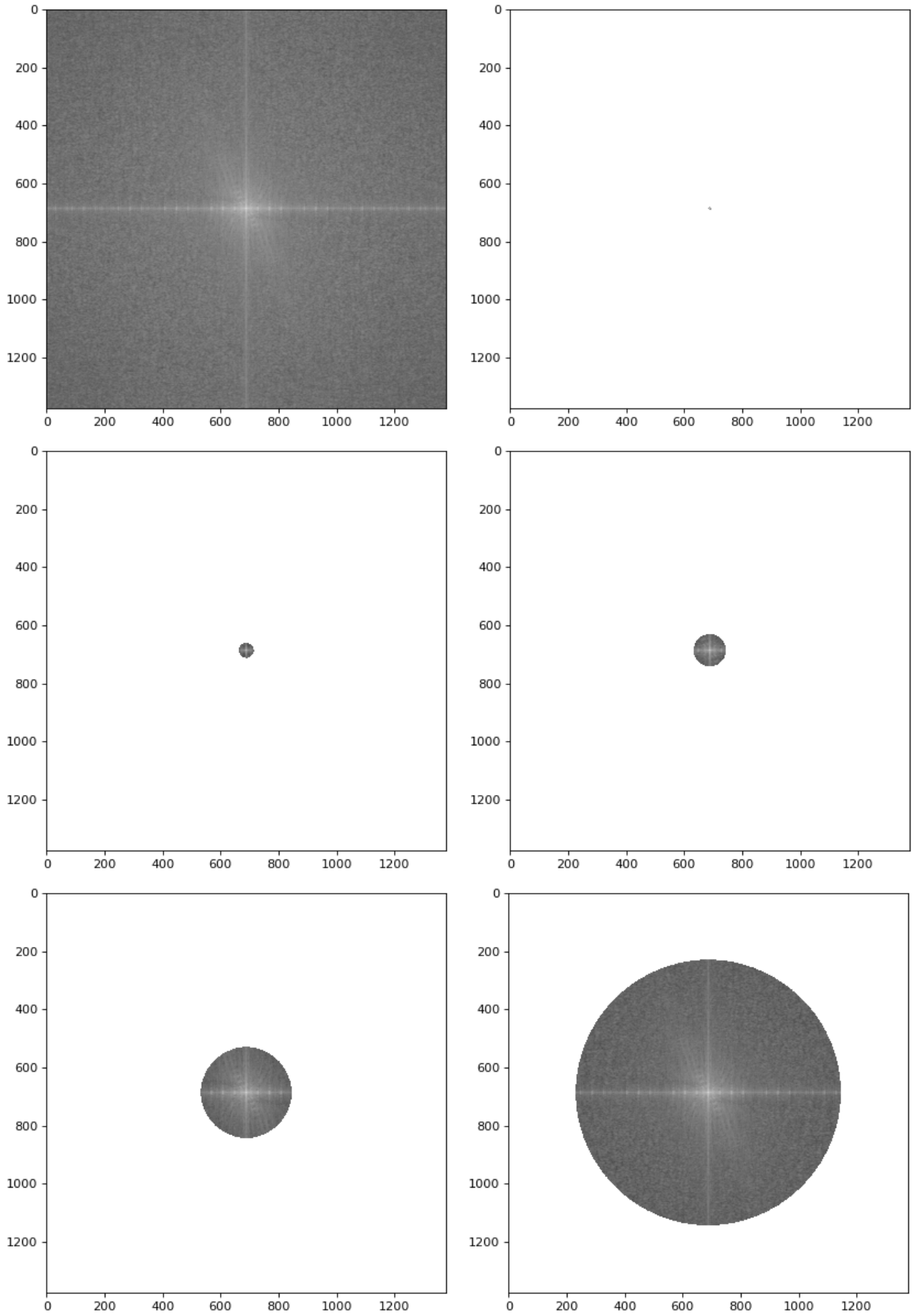

Figure 1: Alphabet Image Power enclosed: (a)Total spectrum (b)radius = 10(c)radius = 30 (d)radius = 60 (e)radius = 160 (f)radius = 460

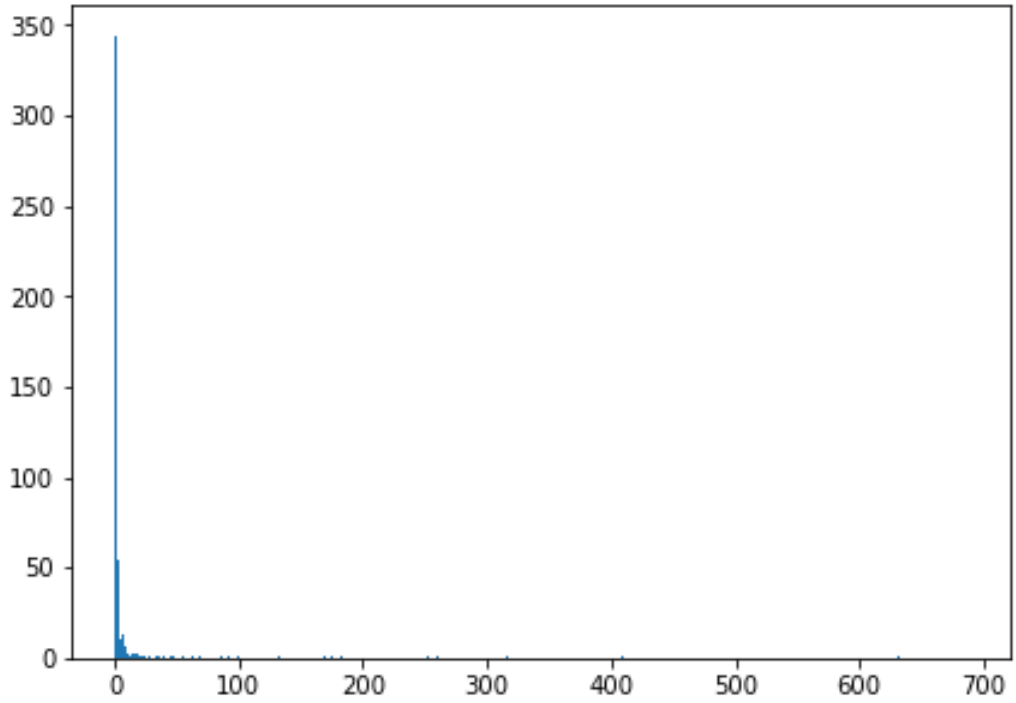

Figure 2: Power Spectrum (normalized power vs frequency

Inference

Image power with respect to frequency components can be calculated with centre of DFT representing lowest frequency component. In alphabet image, lower frequency components contained maximum power of image.

Question 2: Perform following lowpass filtering(LPF) operations on 'alphabet.tif ' with cutoff frequencies set at radii values 10, 30, 60, 160, and 460:
(a) Ideal LPF
(b) Gaussian LPF
(c) Butterworth LPF(compare the effect of filter order)

Aim

Perform lowpass filtering with Ideal , Gaussian and Butterworth LPF with different cutoff frequencies

Discussion

Low pass filtering involves passing of low frequency components and attenuating the high frequency components (e.g. edges and other sharp intensity transitions).

Three filters under consideration are:ideal, Butterworth, and Gaussian. These three categories cover the range from very sharp (ideal) to very smooth (Gaussian) filtering. The shape of a Butterworth filter is controlled by a parameter called the filter order. For large values of this parameter, the Butterworth filter approaches the ideal filter. For lower values, the Butterworth filter is more like a Gaussian filter. Thus, the Butterworth filter provides a transition between two "extremes."

Algorithm

```
Step 1: Start
Step 2: Read the image
Step 3: Obtain the height and width of the image.
Step 4: Mirror pad the image to size (2*height, 2*width ) dimensions.
Step 5: Multiply each pixel with (-1)^(x+y) to centre the DFT.
Step 6: Take the DFT of the image.
Step 7: Create the low pass filter mask for Ideal, Gaussian and Butterworth(different orders).
Step 8: Multiply the Image DFT with respective mask to get Low pass output in frequency domain.
Step 9: Take IDFT of above output, decentre the image and extract the image from the padded
output image.
Step 10: Show the outputs and compare the same.
```

Program Code

```
import numpy as np
import cv2
import math
import matplotlib.pyplot as plt


def Ideal_LPF(Image,cutoff):
    height, width = Image.shape

    Ideal_LPF = np.full((height, width),0)

    # Creating the Ideal Low Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d = math.sqrt((i-int(height)/2)**2 + (j-int(width)/2)**2)
             if (d <= cutoff):
                 Ideal_LPF[i,j] = 1

    Ideal_Lowpass_output = Filter_Output(Image, Ideal_LPF)
    return Ideal_Lowpass_output


def Gaussian_LPF(Image, cutoff):
    height, width = Image.shape

    Gaussian_LPF = Image.copy()

    # Creating the Ideal Low Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d_square = (i-int(height)/2)**2 + (j-int(width)/2)**2
             temp = -d_square/(2*(cutoff**2))
             Gaussian_LPF[i,j] = math.exp(temp)


    Ideal_Lowpass_output = Filter_Output(Image, Gaussian_LPF)

    return Ideal_Lowpass_output
```

```
def Butterworth_LPF(Image, cutoff,order):
    height, width = Image.shape

    Butterworth_LPF = Image.copy()

    # Creating the Butterworth Low Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d = math.sqrt((i-int(height)/2)**2 + (j-int(width)/2)**2)
             temp = (d/cutoff)**(2*order)
             Butterworth_LPF[i,j] = 1/(1+temp)

    Ideal_Lowpass_output = Filter_Output(Image, Butterworth_LPF)

    return Ideal_Lowpass_output
def Filter_Output(Image, Mask):
    height, width = Image.shape

    Low_Pass_output = Image*Mask
    Lowpass_output_IFFT = np.fft.ifft2(Low_Pass_output)


    # Decentering
    for i in range(height):
     for j in range(width):

        Lowpass_output_IFFT[i,j] = Lowpass_output_IFFT[i,j]*((-1)**(i+j))

    Lowpass_output_image = Lowpass_output_IFFT[(int(height/2) - int(height/4)):
                          (int(height/2) + int(height/4)), (int(width/2) - int(width/4)):
                          (int(width/2) + int(width/4)) ]

    # Discarding imaginary values of image and limiting into gray scale range


    Lowpass_output_image = np.clip(Lowpass_output_image, 0, 255)
    Lowpass_output_image = Lowpass_output_image.astype('uint8')


    return Lowpass_output_image

Alphabet_Image = cv2.imread("alphabet.tif", 0)
height, width = Alphabet_Image.shape

# Mirror Padding
Alphabet_pad = np.pad(np.array(Alphabet_Image), ((int(height/2),int(height/2)),
               (int(width/2),int(width/2))),'symmetric')
Alphabet_pad_centre = np.full((2*height, 2*width),0)
# Centering
for i in range(height*2):
    for j in range(width*2):

        Alphabet_pad_centre[i,j] = Alphabet_pad[i,j]*((-1)**(i+j))

Alphabet_Image_FFT = np.fft.fft2(Alphabet_pad_centre)

Ideal_Lowpass_10 = Ideal_LPF(Alphabet_Image_FFT, 10)
```

```
cv2.imwrite('Ideal_Lowpass_10.tif',Ideal_Lowpass_10)

Ideal_Lowpass_30 = Ideal_LPF(Alphabet_Image_FFT, 30)
cv2.imwrite('Ideal_Lowpass_30.tif',Ideal_Lowpass_30)

Ideal_Lowpass_60 = Ideal_LPF(Alphabet_Image_FFT, 60)
cv2.imwrite('Ideal_Lowpass_60.tif',Ideal_Lowpass_60)

Ideal_Lowpass_160 = Ideal_LPF(Alphabet_Image_FFT, 160)
cv2.imwrite('Ideal_Lowpass_160.tif',Ideal_Lowpass_160)

Ideal_Lowpass_460 = Ideal_LPF(Alphabet_Image_FFT, 460)
cv2.imwrite('Ideal_Lowpass_460.tif',Ideal_Lowpass_460)

Gaussian_Lowpass_10 = Gaussian_LPF(Alphabet_Image_FFT, 10)
cv2.imwrite('Gaussian_Lowpass_10.tif',Gaussian_Lowpass_10)

Gaussian_Lowpass_30 = Gaussian_LPF(Alphabet_Image_FFT, 30)
cv2.imwrite('Gaussian_Lowpass_30.tif',Gaussian_Lowpass_30)

Gaussian_Lowpass_60 = Gaussian_LPF(Alphabet_Image_FFT, 60)
cv2.imwrite('Gaussian_Lowpass_60.tif',Gaussian_Lowpass_60)

Gaussian_Lowpass_160 = Gaussian_LPF(Alphabet_Image_FFT, 160)
cv2.imwrite('Gaussian_Lowpass_160.tif',Gaussian_Lowpass_160)

Gaussian_Lowpass_460 = Gaussian_LPF(Alphabet_Image_FFT, 460)
cv2.imwrite('Gaussian_Lowpass_460.tif',Gaussian_Lowpass_460)

Butterworth_Lowpass_10_1 = Butterworth_LPF(Alphabet_Image_FFT, 10,1)
cv2.imwrite('Butterworth_Lowpass_10_1.tif',Butterworth_Lowpass_10_1)

Butterworth_Lowpass_30_1 = Butterworth_LPF(Alphabet_Image_FFT, 30,1)
cv2.imwrite('Butterworth_Lowpass_30_1.tif',Butterworth_Lowpass_30_1)

Butterworth_Lowpass_60_1 = Butterworth_LPF(Alphabet_Image_FFT, 60,1)
cv2.imwrite('Butterworth_Lowpass_60_1.tif',Butterworth_Lowpass_60_1)

Butterworth_Lowpass_160_1 = Butterworth_LPF(Alphabet_Image_FFT, 160,1)
cv2.imwrite('Butterworth_Lowpass_160_1.tif',Butterworth_Lowpass_160_1)

Butterworth_Lowpass_460_1 = Butterworth_LPF(Alphabet_Image_FFT, 460,1)
cv2.imwrite('Butterworth_Lowpass_460_1.tif',Butterworth_Lowpass_460_1)

Butterworth_Lowpass_10_3 = Butterworth_LPF(Alphabet_Image_FFT, 10,3)
cv2.imwrite('Butterworth_Lowpass_10_3.tif',Butterworth_Lowpass_10_3)

Butterworth_Lowpass_30_3 = Butterworth_LPF(Alphabet_Image_FFT, 30,3)
cv2.imwrite('Butterworth_Lowpass_30_3.tif',Butterworth_Lowpass_30_3)

Butterworth_Lowpass_60_3 = Butterworth_LPF(Alphabet_Image_FFT, 60,3)
cv2.imwrite('Butterworth_Lowpass_60_3.tif',Butterworth_Lowpass_60_3)

Butterworth_Lowpass_160_3 = Butterworth_LPF(Alphabet_Image_FFT, 160,3)
cv2.imwrite('Butterworth_Lowpass_160_3.tif',Butterworth_Lowpass_160_3)

Butterworth_Lowpass_460_3 = Butterworth_LPF(Alphabet_Image_FFT, 460,3)
cv2.imwrite('Butterworth_Lowpass_460_3.tif',Butterworth_Lowpass_460_3)
```

```
Butterworth_Lowpass_10_5 = Butterworth_LPF(Alphabet_Image_FFT, 10,5)
cv2.imwrite('Butterworth_Lowpass_10_5.png',Butterworth_Lowpass_10_5)

Butterworth_Lowpass_30_5 = Butterworth_LPF(Alphabet_Image_FFT, 30,5)
cv2.imwrite('Butterworth_Lowpass_30_5.png',Butterworth_Lowpass_30_5)

Butterworth_Lowpass_60_5 = Butterworth_LPF(Alphabet_Image_FFT, 60,3)
cv2.imwrite('Butterworth_Lowpass_60_5.png',Butterworth_Lowpass_60_5)

Butterworth_Lowpass_160_5 = Butterworth_LPF(Alphabet_Image_FFT, 160,5)
cv2.imwrite('Butterworth_Lowpass_160_5.png',Butterworth_Lowpass_160_5)

Butterworth_Lowpass_460_5 = Butterworth_LPF(Alphabet_Image_FFT, 460,5)
cv2.imwrite('Butterworth_Lowpass_460_5.png',Butterworth_Lowpass_460_5)
```

Result

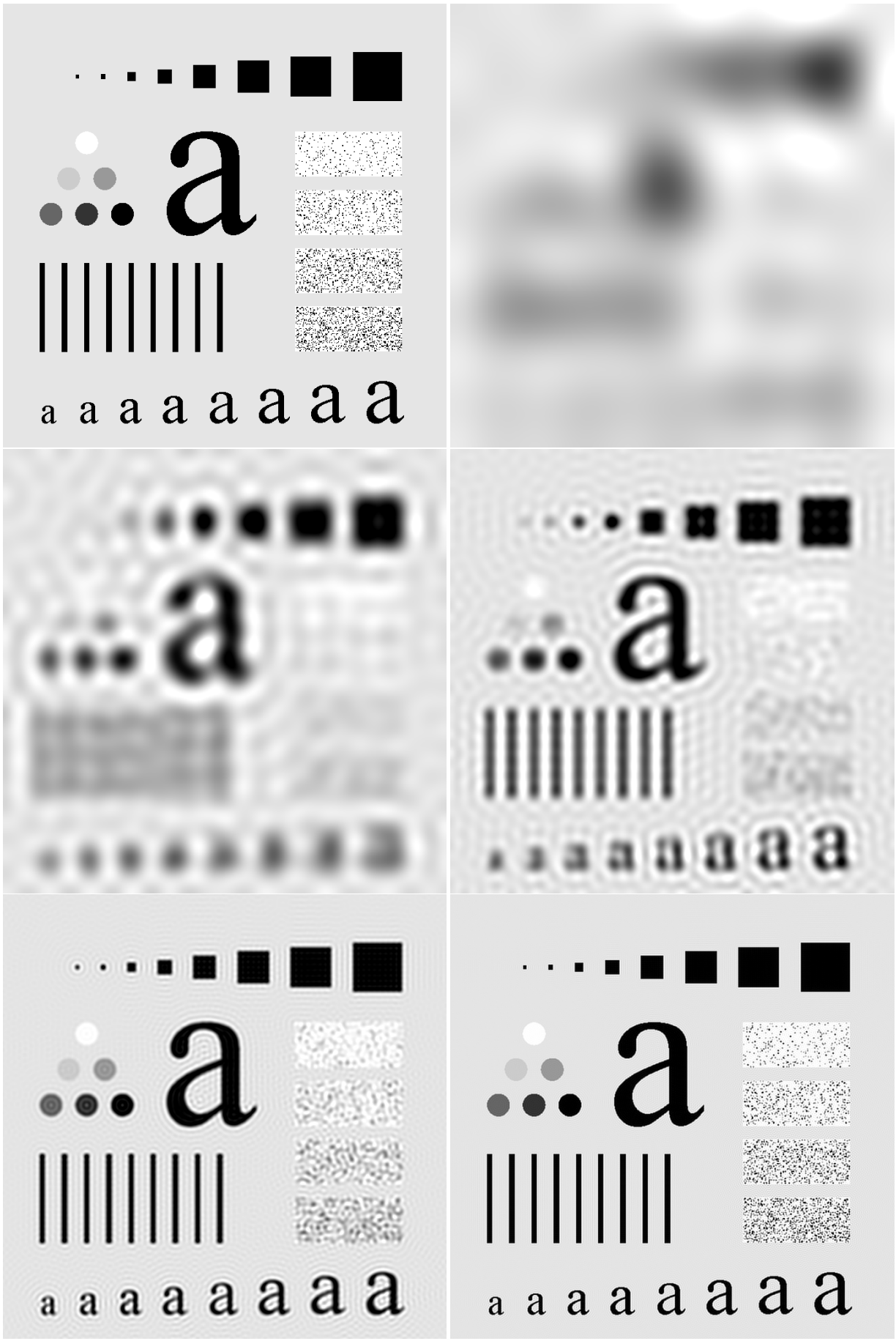

Figure 3: Ideal Low pass filter : (a)Original Image (b)cutoff = 10(c)cutoff 30 (d)cutoff = 60 (e)cutoff = 160 (f)cutoff = 460

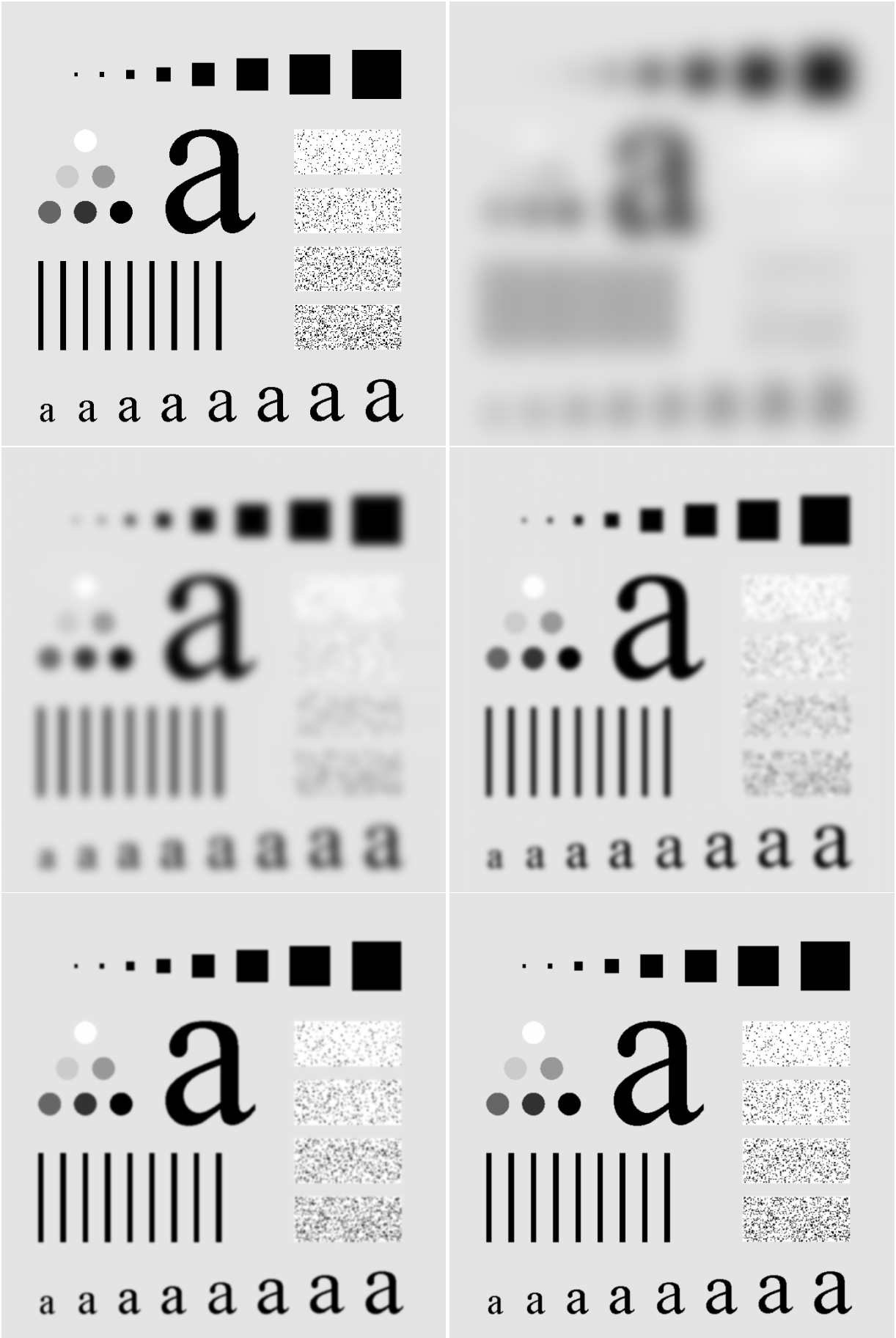

Figure 4: Gaussian Low pass filter : (a)Original Image (b)cutoff = 10(c)cutoff 30 (d)cutoff = 60 (e)cutoff = 160 (f)cutoff = 460

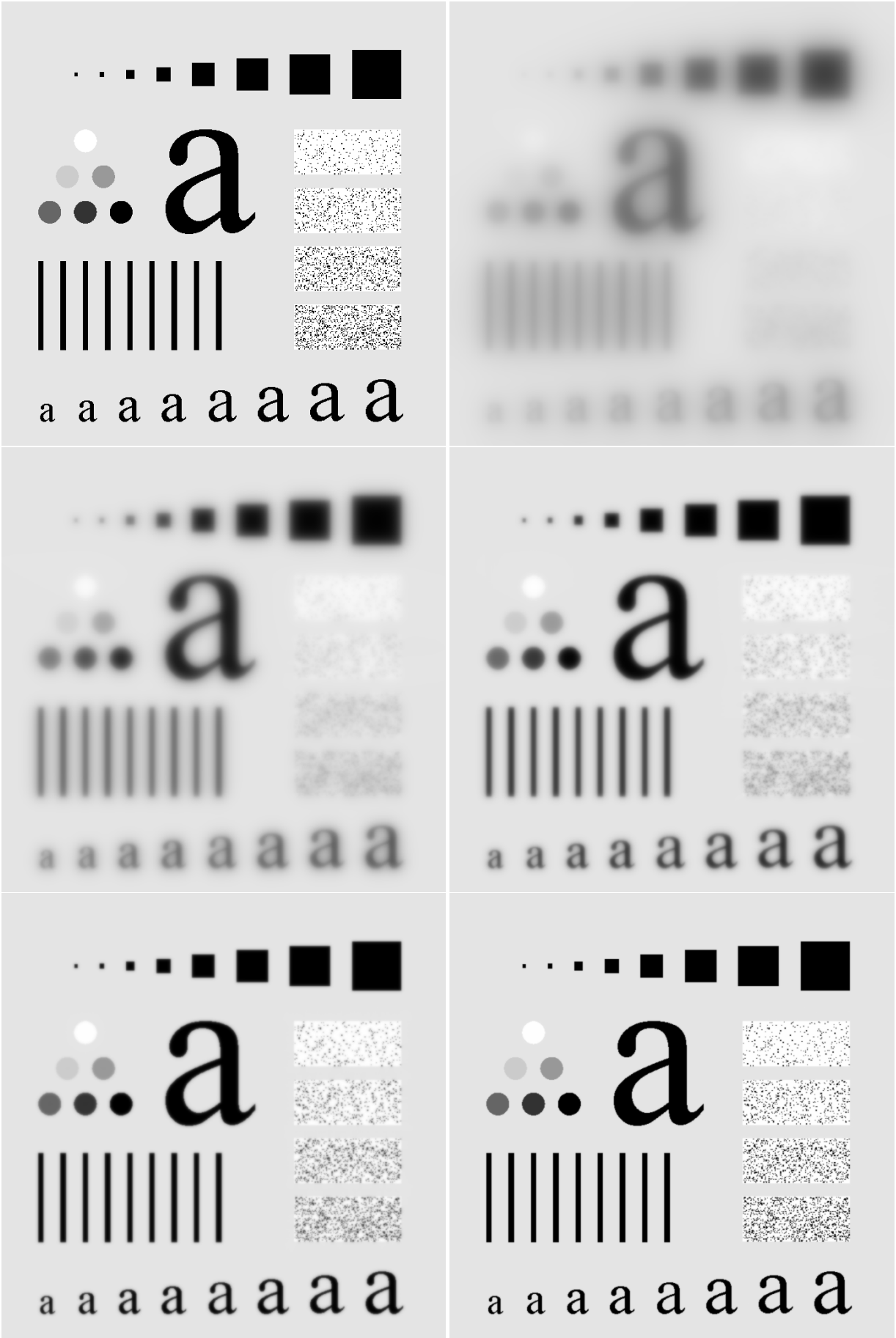

Figure 5: Butterworth Low pass filter - Order 1 : (a)Original Image (b)cutoff = 10(c)cutoff 30 (d)cutoff = 60 (e)cutoff = 160 (f)cutoff = 460

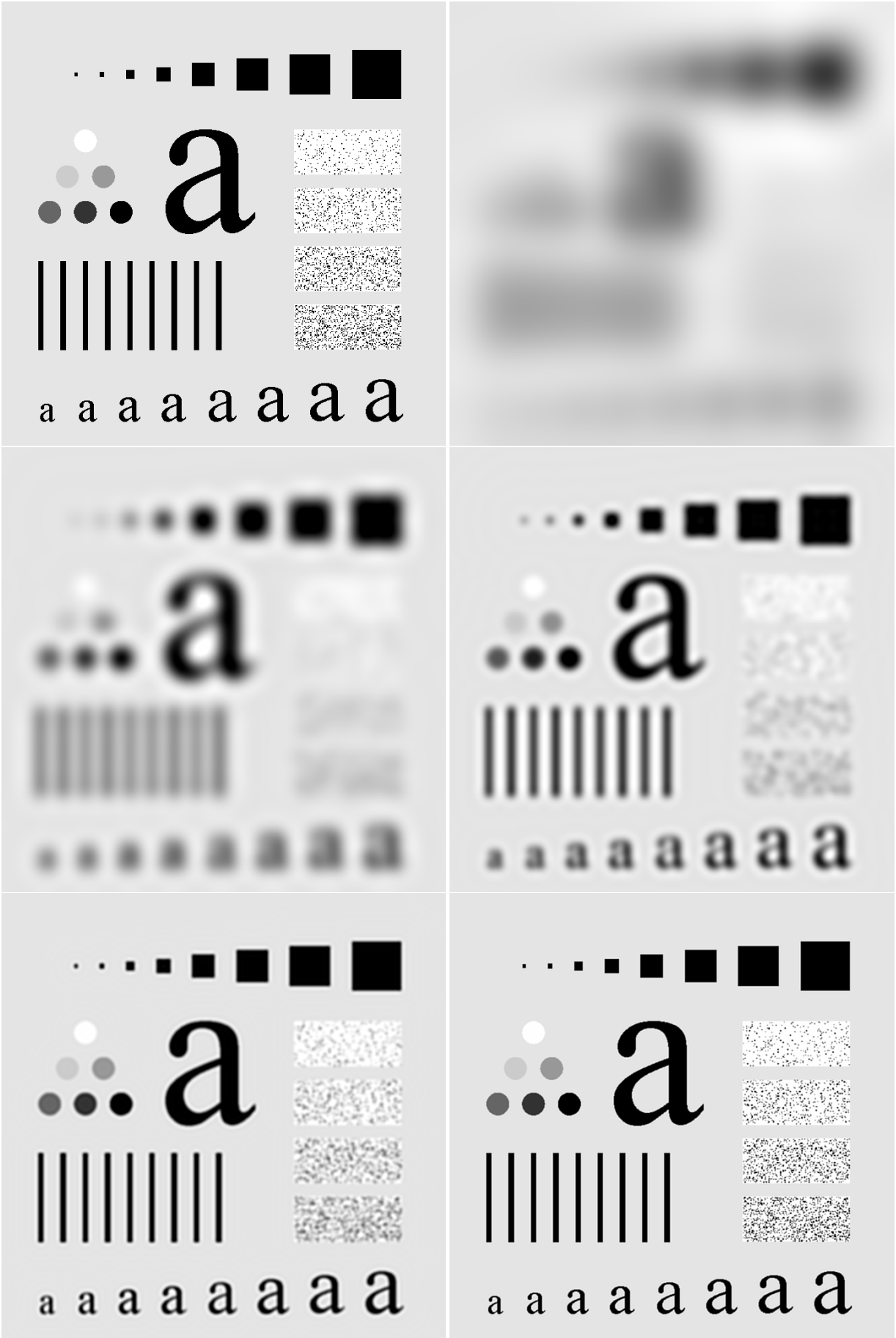

Figure 6: Butterworth Low pass filter - Order 3 : (a)Original Image (b)cutoff = 10(c)cutoff 30 (d)cutoff = 60 (e)cutoff = 160 (f)cutoff = 460

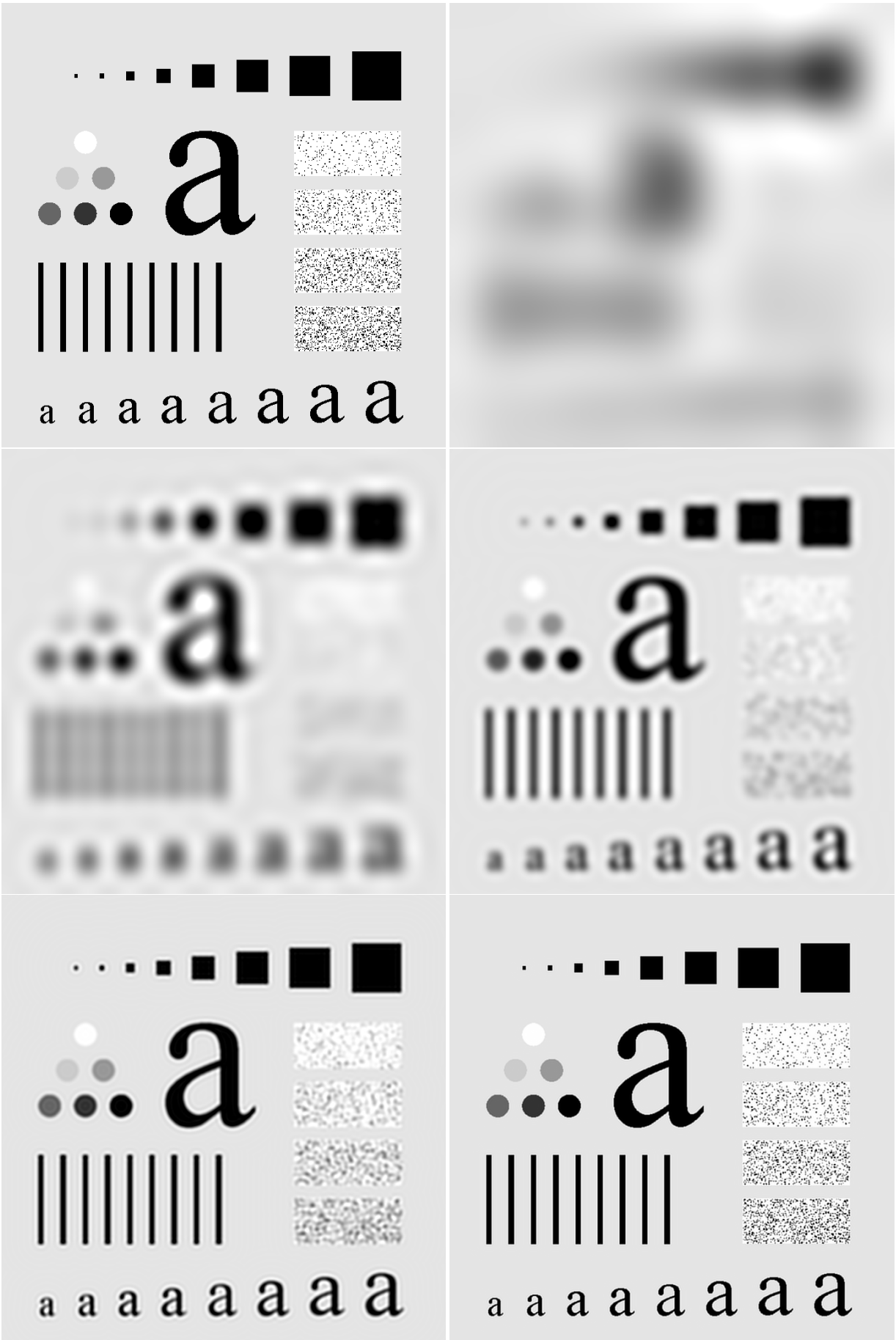

Figure 7: Butterworth Low pass filter - Order 5: (a)Original Image (b)cutoff = 10(c)cutoff 30 (d)cutoff = 60 (e)cutoff = 160 (f)cutoff = 460

Inference

Blurring increases as cutoff frequency decreases(i.e. lesser higher frequency components passed through).

Gaussian Low Pass filters allow a smooth blurring of Image compared to Ideal Low Pass Filter.

Comparing Butterworth Filters, the blurring effect increases with order number as it approaches ideal frequency characteristics of ideal lowpass filter.

Question 3: Perform following highpass filtering(HPF) operations on 'alphabet.tif ' using cutoff frequency 60 and 160:
(a) Ideal HPF
(b) Gaussian HPF
(c) Butterworth HPF(compare the effect of filter order)

Aim

Perform highpass filtering with Ideal , Gaussian and Butterworth HPF with different cutoff frequencies

Discussion

The edges and other abrupt changes in intensities are associated with high-frequency components hence, image sharpening can be achieved in the frequency domain by highpass filtering, which attenuates low frequencies components without disturbing high-frequencies in the Fourier transform.

Algorithm

Step 1: Start
Step 2: Read the image
Step 3: Obtain the height and width of the image.
Step 4: Mirror pad the image to size (2*height, 2*width) dimensions.
Step 5: Multiply each pixel with (-1)^(x+y) to centre the DFT.
Step 6: Take the DFT of the image.
Step 7: Create the high pass filter masks for Ideal, Gaussian and Butterworth(different orders).
Step 8: Multiply the Image DFT with respective mask to get High pass output in frequency domain.
Step 9: Take IDFT of above output, decentre the image and extract the image from the padded output image.
Step 10: Show the outputs and compare the same.

Program Code

```
import numpy as np
import cv2
import math
import matplotlib.pyplot as plt


def Ideal_HPF(Image,cutoff):

    height, width = Image.shape
    Ideal_HPF = np.full((height, width),0)
```

```
    # Creating the Ideal High Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d = math.sqrt((i-int(height)/2)**2 + (j-int(width)/2)**2)
             if (d >= cutoff):
                 Ideal_HPF[i,j] = 1

    Ideal_Highpass_output = Filter_Output(Image, Ideal_HPF)
    return Ideal_Highpass_output


def Gaussian_HPF(Image, cutoff):
    height, width = Image.shape

    Gaussian_HPF = Image.copy()

    # Creating the Gaussian Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d_square = (i-int(height)/2)**2 + (j-int(width)/2)**2
             temp = -d_square/(2*(cutoff**2))
             Gaussian_HPF[i,j] = 1 - math.exp(temp)


    Ideal_Highpass_output = Filter_Output(Image, Gaussian_HPF)

    return Ideal_Highpass_output

def Butterworth_HPF(Image, cutoff,order):
    height, width = Image.shape

    Butterworth_HPF = Image.copy()

    # Creating the Butterworth High Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d = math.sqrt((i-int(height)/2)**2 + (j-int(width)/2)**2)

             # To combat zero division error
             if(d!=0):
                 temp = (cutoff/d)**(2*order)
                 Butterworth_HPF[i,j] = 1/(1+temp)

             else:
                 Butterworth_HPF[i,j] = 0

    Ideal_Highpass_output = Filter_Output(Image, Butterworth_HPF)

    return Ideal_Highpass_output

def Filter_Output(Image, Mask):
    height, width = Image.shape

    High_Pass_output = Image*Mask
```

```
    Highpass_output_IFFT = np.fft.ifft2(High_Pass_output)


    # Decentering
    for i in range(height):
     for j in range(width):

        Highpass_output_IFFT[i,j] = Highpass_output_IFFT[i,j]*((-1)**(i+j))

    Highpass_output_image = Highpass_output_IFFT[(int(height/2) - int(height/4)):
                            (int(height/2) + int(height/4)), (int(width/2) - int(width/4)):
                            (int(width/2) + int(width/4)) ]

    # Discarding imaginary values of image and limiting into gray scale range

    Highpass_output_image = np.clip(Highpass_output_image, 0, 255)
    Highpass_output_image = Highpass_output_image.astype('uint8')

    return Highpass_output_image

Alphabet_Image = cv2.imread("alphabet.tif", 0)
height, width = Alphabet_Image.shape

# Mirror Padding
Alphabet_pad = np.pad(np.array(Alphabet_Image), ((int(height/2),int(height/2)),
               (int(width/2),int(width/2))),'symmetric')

Alphabet_pad_centre = np.full((2*height, 2*width),0)

# Centering
for i in range(height*2):
    for j in range(width*2):

        Alphabet_pad_centre[i,j] = Alphabet_pad[i,j]*((-1)**(i+j))

Alphabet_Image_FFT = np.fft.fft2(Alphabet_pad_centre)

Ideal_Highpass_60 = Ideal_HPF(Alphabet_Image_FFT, 60)
cv2.imshow('Ideal_Highpass_60', abs(Ideal_Highpass_60))
cv2.imwrite('Ideal_Highpass_60.tif',Ideal_Highpass_60)
cv2.waitKey(0)

Ideal_Highpass_160 = Ideal_HPF(Alphabet_Image_FFT, 160)
cv2.imshow('Ideal_Highpass_160', abs(Ideal_Highpass_160))
cv2.imwrite('Ideal_Highpass_160.tif',Ideal_Highpass_160)
cv2.waitKey(0)

Gaussian_Highpass_60 = Gaussian_HPF(Alphabet_Image_FFT, 60)
cv2.imshow('Gaussian_Highpass_60', abs(Gaussian_Highpass_60))
cv2.imwrite('Gaussian_Highpass_60.tif',Gaussian_Highpass_60)
cv2.waitKey(0)

Gaussian_Highpass_160 = Gaussian_HPF(Alphabet_Image_FFT, 160)
cv2.imshow('Gaussian_Highpass_160', abs(Gaussian_Highpass_160))
cv2.imwrite('Gaussian_Highpass_160.tif',Gaussian_Highpass_160)
cv2.waitKey(0)

Butterworth_Highpass_60_1 = Butterworth_HPF(Alphabet_Image_FFT, 60,1)
```

```
cv2.imshow('Butterworth_Highpass_60_1', abs(Butterworth_Highpass_60_1))
cv2.imwrite('Butterworth_Highpass_60_1.tif',Butterworth_Highpass_60_1)
cv2.waitKey(0)

Butterworth_Highpass_160_1 = Butterworth_HPF(Alphabet_Image_FFT, 160,1)
cv2.imshow('Butterworth_Highpass_160_1', abs(Butterworth_Highpass_160_1))
cv2.imwrite('Butterworth_Highpass_160_1.tif',Butterworth_Highpass_160_1)
cv2.waitKey(0)

Butterworth_Highpass_60_3 = Butterworth_HPF(Alphabet_Image_FFT, 60,3)
cv2.imshow('Butterworth_Highpass_60_3', abs(Butterworth_Highpass_60_3))
cv2.imwrite('Butterworth_Highpass_60_3.tif',Butterworth_Highpass_60_3)
cv2.waitKey(0)

Butterworth_Highpass_160_3 = Butterworth_HPF(Alphabet_Image_FFT, 160,3)
cv2.imshow('Butterworth_Highpass_160_3', abs(Butterworth_Highpass_160_3))
cv2.imwrite('Butterworth_Highpass_160_3.tif',Butterworth_Highpass_160_3)
cv2.waitKey(0)


Butterworth_Highpass_60_5 = Butterworth_HPF(Alphabet_Image_FFT, 60,5)
cv2.imshow('Butterworth_Highpass_60_5', abs(Butterworth_Highpass_60_5))
cv2.imwrite('Butterworth_Highpass_60_5.tif',Butterworth_Highpass_60_5)
cv2.waitKey(0)

Butterworth_Highpass_160_5 = Butterworth_HPF(Alphabet_Image_FFT, 160,5)
cv2.imshow('Butterworth_Highpass_160_5', abs(Butterworth_Highpass_160_5))
cv2.imwrite('Butterworth_Highpass_160_5.tif',Butterworth_Highpass_160_5)
cv2.waitKey(0)
```

Result

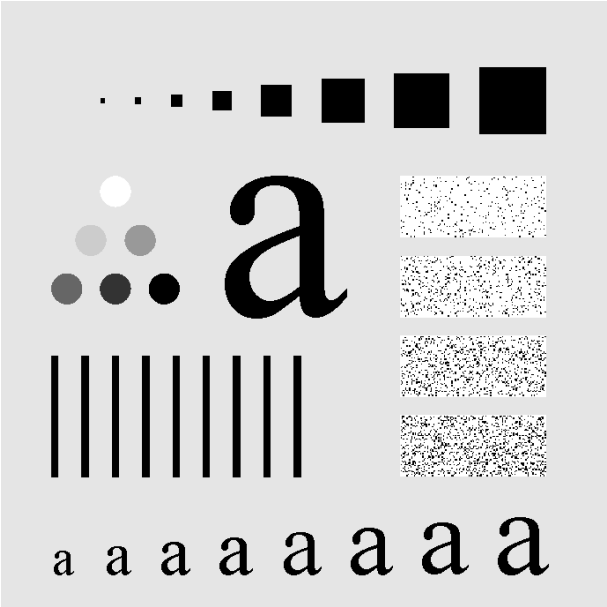

Figure 8: Original Image : alphabet.tif

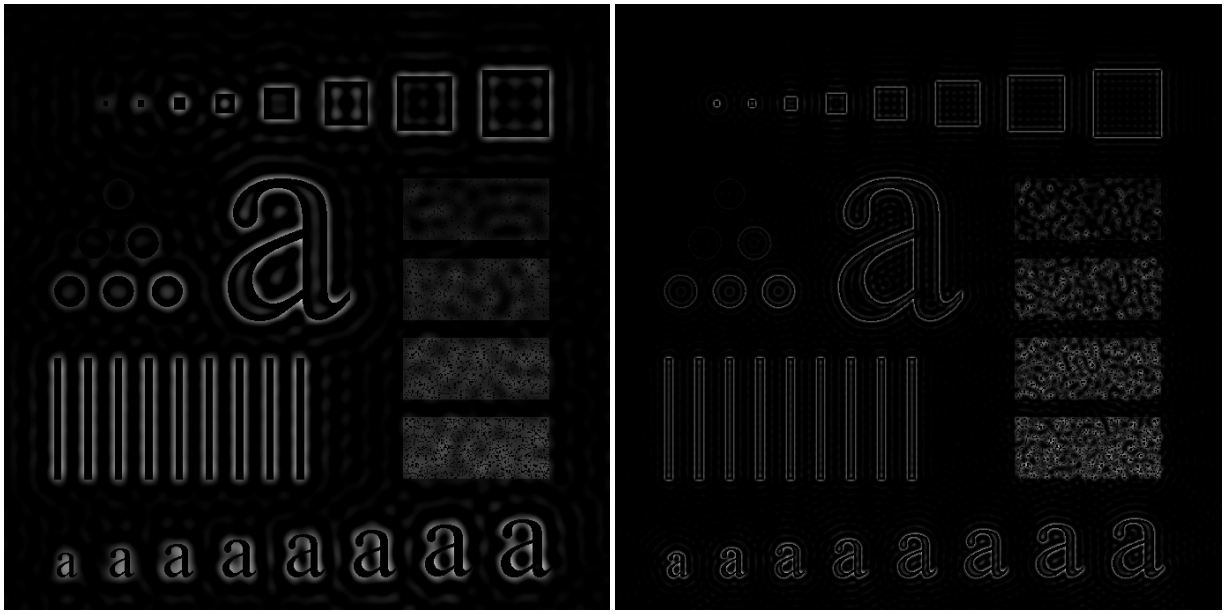

Figure 9: Ideal High pass filter : (a)cutoff = 60 (b)cutoff = 160

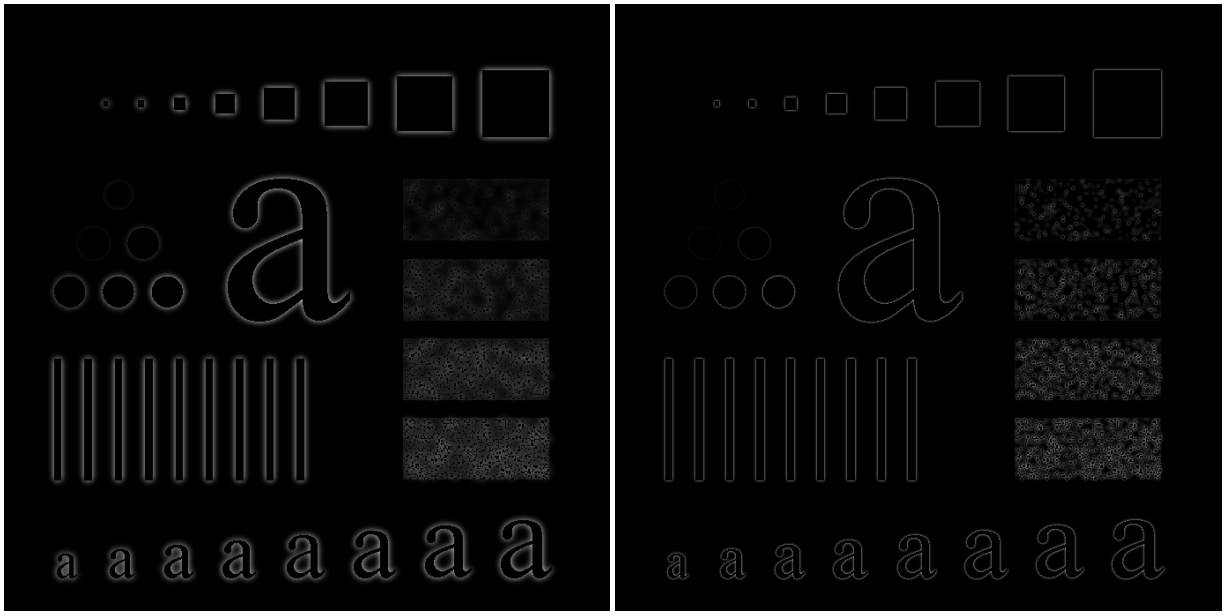

Figure 10: Gaussian High pass filter : (a)cutoff = 60 (b)cutoff = 160

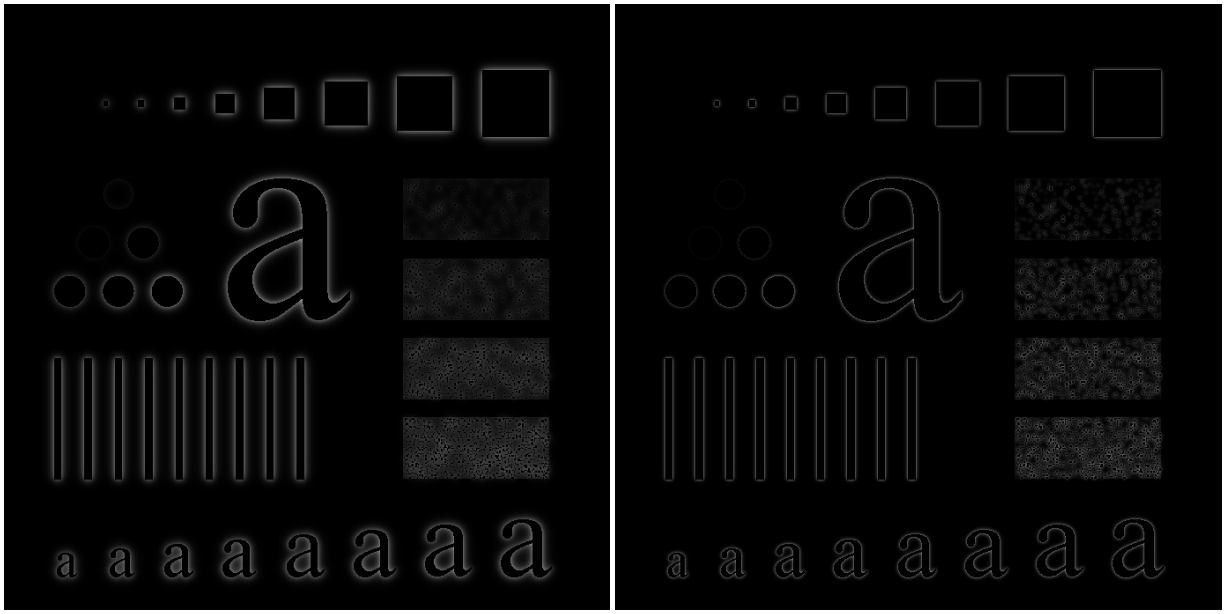

Figure 11: Butterworth High pass filter- Order 1 : (a)cutoff = 60 (b)cutoff = 160

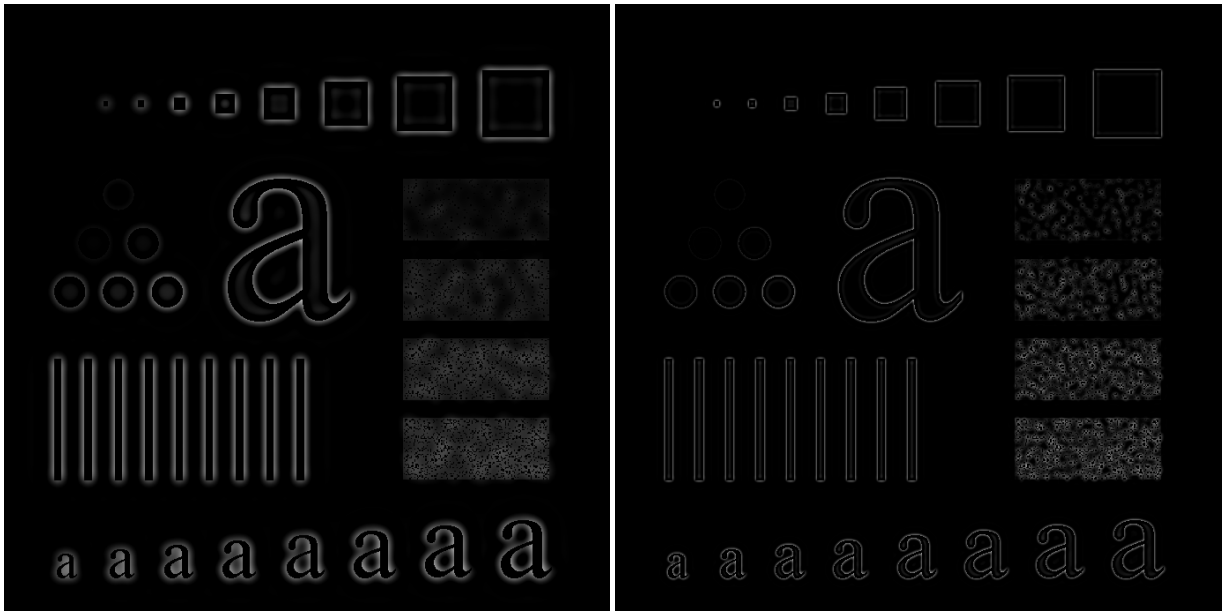

Figure 12: Butterworth High pass filter- Order 3 : (a)cutoff = 60 (b)cutoff = 160

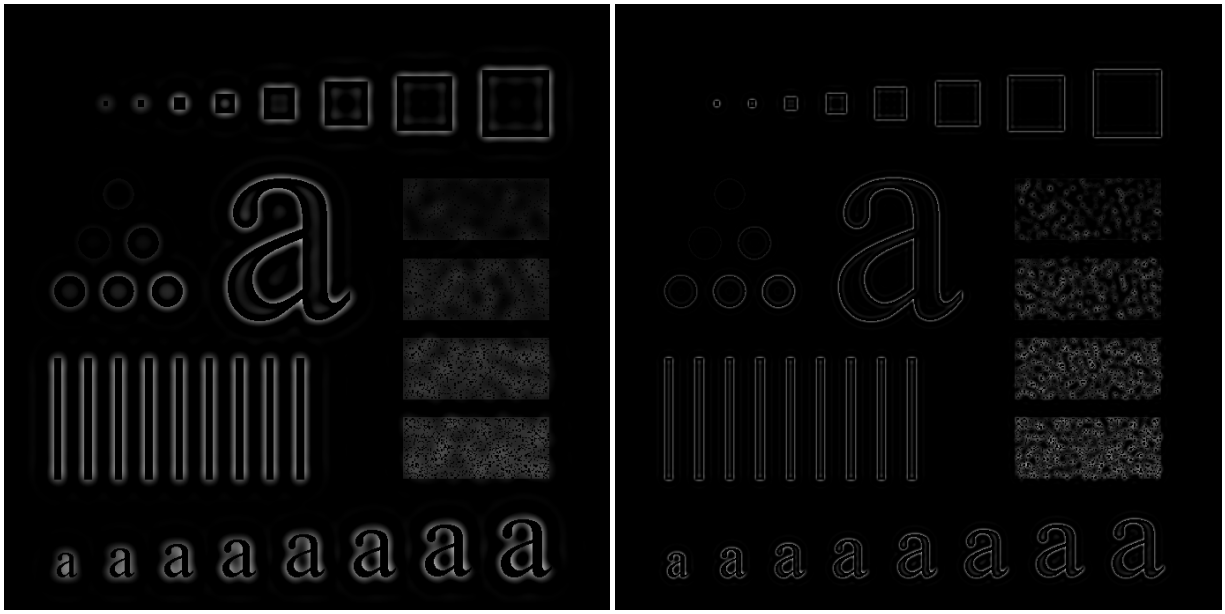

Figure 13: Butterworth High pass filter- Order 5 : (a)cutoff = 60 (b)cutoff = 160

Inference

Sharpening of image (attenuation of lower frequency components) increases as cutoff frequency decreases(i.e. lesser lower frequency components
passed through).

Gaussian High Pass filters allow a smooth sharpening of Image compared to Ideal High Pass Filter.

Comparing Butterworth Filters, the sharpening effect increases with order number as it approaches ideal frequency characteristics of ideal highpass filter.

Question 4: Perform image enhancement on 'moon.tif ' using Laplacian in the frequency domain.

Aim

Image enhancement using Laplacian in frequency domain

Discussion

The Laplacian can be implemented in the frequency domain with respect to the center of the frequency rectangle, using the transfer function

$$H(u,v) = 4pi^2 D(u,v)^2$$

(u,v) being distance of pixel position from centre
This mask is then used in image enhancement

Algorithm

Step 1: Start
Step 2: Read the image
Step 3: Obtain the height and width of the image.

```
Step 4: Zero pad the image to size (2*height, 2*width ) dimensions.
Step 5: Multiply each pixel with (-1)^(x+y) to centre the DFT.
Step 6: Take the DFT of the image.
Step 7: Create the Laplacian mask
Step 8: Multiply the Image DFT with respective mask to get Laplacian output.
Step 9: Take IDFT of above output and  decentre the output.
output image.
Step 10:Normalize the values of Laplacian output.
Step 11: Subtract the Laplacian output from the original padded image.
Step 10: Display the enhanced image.
```

Program Code

```
mport numpy as np
import cv2
import math
import matplotlib.pyplot as plt

def Laplacian_Mask(Image):
    height, width = Image.shape

    Laplacian = np.full((height, width),0)

    # Creating the Laplacian Mask

    for i in range(height):
        for j in range(width):
             d = math.sqrt((i-int(height)/2)**2 + (j-int(width)/2)**2)
             Laplacian[i,j] = -1*(4*(math.pi**2)*(d**2))

    Laplacian_output = Image*Laplacian
    Laplacian_output_IFFT = np.fft.ifft2(Laplacian_output)


    # Decentering
    for i in range(height):
     for j in range(width):

        Laplacian_output_IFFT[i,j] = Laplacian_output_IFFT[i,j]*((-1)**(i+j))

    # Normalizing values of Laplacian mask output
    Laplacian_output_image  = cv2.normalize(np.float32(Laplacian_output_IFFT)  , None,
                             -255, 255, norm_type=cv2.NORM_MINMAX)
    Laplacian_output_adjusted  = Laplacian_output_image [(int(height/2) - int(height/4)):
                                (int(height/2) + int(height/4)),
                                (int(width/2) - int(width/4)):  (int(width/2) + int(width/4)) ]


    plt.imshow(np.log(abs(Laplacian_output_adjusted)), cmap='gray');

    return Laplacian_output_image

Moon_Image = cv2.imread("moon.tif", 0)
height, width = Moon_Image.shape

# Zero Padding
Moon_pad = np.pad(np.array(Moon_Image), ((int(height/2),int(height/2)),(int(width/2),
          int(width/2))),'symmetric')
```

```
Moon_pad_centre = np.full((2*height, 2*width),0)

# Centering
for i in range(height*2):
    for j in range(width*2):

        Moon_pad_centre[i,j] = Moon_pad[i,j]*((-1)**(i+j))

Moon_Image_FFT = np.fft.fft2(Moon_pad_centre)

Laplacian_output = Laplacian_Mask(Moon_Image_FFT)
Laplacian_output_adjusted = np.clip(Laplacian_output , 0, 255)
Laplacian_output_adjusted = Laplacian_output_adjusted.astype('uint8')

plt.imshow(Laplacian_output_adjusted, cmap='gray');
cv2.imwrite('Laplacian_Mask.tif',Laplacian_output_adjusted)

Laplacian_Enhancement =  Moon_pad - Laplacian_output

Laplacian_Enhancement_output = Laplacian_Enhancement[(int(height) - int(height/2)):
                              (int(height) + int(height/2)),
                              (int(width) - int(width/2)):(int(width) + int(width/2)) ]
Laplacian_Enhancement_output = np.clip(Laplacian_Enhancement_output, 0, 255)
Laplacian_Enhancement_output = Laplacian_Enhancement_output.astype('uint8')

plt.imshow(Laplacian_Enhancement_output, cmap='gray');
cv2.imwrite('Laplacian_Enhancement.tif',Laplacian_Enhancement_output)
```

Result

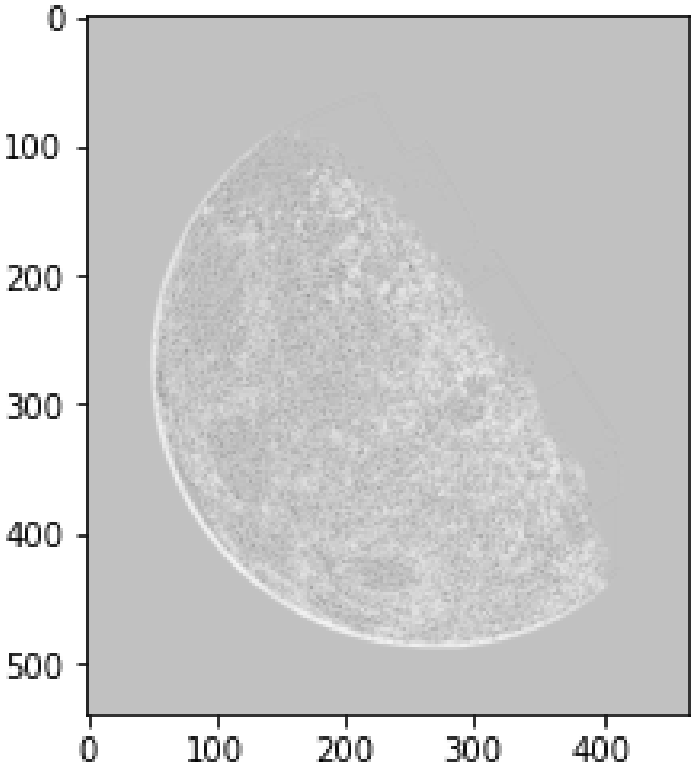

Figure 14: Laplacian Mask (Normalized)

Figure 15: (a)Original Image(b) Enhanced Image

Inference

```
Laplacian filter output can be used in image enhancement in frequency domain.
```

Question 5: Perform the following image enhancement operations on 'chest.tif '.
(a) Unsharp masking
(b) Highboost filtering
(c) High-frequency emphasis filtering

Aim

Perform image enhancement using Unsharp masking, Highboost filtering and High frequency emphasis filtering

Discussion

In Unsharp Masking, the unsharp mask is given by:

$$\text{gmask}(x,y) = f(x,y) - \text{flp}(x,y),$$

where flp is output of image through low pass filter

$$g(x,y) = f(x,y) + k*\text{gmask}$$

where k = 1

In Highboost filtering, the output is given by:

$$g(x,y) = f(x,y) + k*\text{gmask}$$

where k is greater than 1

In High Frequency Emphasis Filtering, the output is given by:

$$g(x,y) = \text{IDFT1} + k*H(u, v)]F(u, v)$$

where H(u,v) is the highpass filter output in frequency domain

Algorithm

```
Step 1: Start
Step 2: Read the image
Step 3: Obtain the height and width of the image.
Step 4: Mirror pad the image to size (2*height, 2*width ) dimensions.
Step 5: Multiply each pixel with (-1)^(x+y) to centre the DFT.
Step 6: Take the DFT of the image.
Step 7: Obtain the Lowpass filter output of image and
perform unsharp masking and Highboost filtering.
Step 8: Obtain the Highpass filter output of image and
perform High Frequency Emphasis Filtering accordingly
Step 9: Display the outputs
```

Program Code

```
import numpy as np
import cv2
import math
import matplotlib.pyplot as plt

def Gaussian_LPF(Image, cutoff):
    height, width = Image.shape

    Gaussian_LPF = Image.copy()

    # Creating the Ideal Low Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d_square = (i-int(height)/2)**2 + (j-int(width)/2)**2
             temp = -d_square/(2*(cutoff**2))
             Gaussian_LPF[i,j] = math.exp(temp)


    Ideal_Lowpass_output = Filter_Output(Image, Gaussian_LPF)

    plt.figure(num=None, figsize=(8, 6), dpi=80)
    plt.imshow(np.log(abs(Gaussian_LPF)), cmap='gray');

    return Ideal_Lowpass_output

def High_Freq_Emphasis(Image, cutoff):
    height, width = Image.shape

    Gaussian_HPF = Image.copy()

    # Creating the Ideal Low Pass Filter Mask

    for i in range(height):
        for j in range(width):
             d_square = (i-int(height)/2)**2 + (j-int(width)/2)**2
             temp = -d_square/(2*(cutoff**2))
             Gaussian_HPF[i,j] = 1 - math.exp(temp)

    High_Pass_output = Image*Gaussian_HPF

    High_Freq_Emp_output = Image + 2*High_Pass_output
    High_Freq_Emp_output = np.fft.ifft2(High_Freq_Emp_output)


    # Decentering
    for i in range(height):
     for j in range(width):

        High_Freq_Emp_output[i,j] = High_Freq_Emp_output[i,j]*((-1)**(i+j))

    High_Freq_Emp_output = High_Freq_Emp_output[(int(height/2) - int(height/4)):
                          (int(height/2) + int(height/4)), (int(width/2) - int(width/4)):
                          (int(width/2) + int(width/4)) ]

    # Discarding imaginary values of image and limiting into gray scale range
```

```
    High_Freq_Emp_output= np.clip(High_Freq_Emp_output, 0, 255)
    High_Freq_Emp_output = High_Freq_Emp_output.astype('uint8')

    return High_Freq_Emp_output

def Filter_Output(Image, Mask):
    height, width = Image.shape

    Low_Pass_output = Image*Mask
    Lowpass_output_IFFT = np.fft.ifft2(Low_Pass_output)


    # Decentering
    for i in range(height):
     for j in range(width):

        Lowpass_output_IFFT[i,j] = Lowpass_output_IFFT[i,j]*((-1)**(i+j))

    Lowpass_output_image = Lowpass_output_IFFT[(int(height/2) - int(height/4)):
                           (int(height/2) + int(height/4)), (int(width/2) - int(width/4)):
                           (int(width/2) + int(width/4)) ]

    # Discarding imaginary values of image and limiting into gray scale range


    Lowpass_output_image = np.clip(Lowpass_output_image, 0, 255)
    Lowpass_output_image = Lowpass_output_image.astype('uint8')


    return Lowpass_output_image

Chest_Image = cv2.imread("chest.tif", 0)
height, width = Chest_Image.shape

# Mirror Padding
Chest_pad = np.pad(np.array(Chest_Image), ((int(height/2),int(height/2)),
           (int(width/2),int(width/2))),'symmetric')

Chest_pad_centre = np.full((2*height, 2*width),0)

# Centering
for i in range(height*2):
    for j in range(width*2):

        Chest_pad_centre[i,j] = Chest_pad[i,j]*((-1)**(i+j))

Chest_Image_FFT = np.fft.fft2(Chest_pad_centre)

Chest_pad_centre = Chest_pad_centre[(int(height) - int(height/2)):(int(height) + int(height/2)),
                  (int(width) - int(width/2)):(int(width) + int(width/2)) ]
Lowpass_Output = Gaussian_LPF(Chest_Image_FFT , 60)
G_mask = Chest_pad_centre - Lowpass_Output


unsharp_output = Chest_pad_centre + G_mask
unsharp_output = np.clip(unsharp_output, 0, 255)
unsharp_output = unsharp_output.astype('uint8')
```

```
highboost_output = Chest_pad_centre + 2*G_mask
highboost_output = np.clip(highboost_output, 0, 255)
highboost_output = highboost_output.astype('uint8')

high_freq_emphasis = High_Freq_Emphasis(Chest_Image_FFT , 60)

plt.subplot(3, 1, 1)
plt.imshow(unsharp_output, cmap="gray")
cv2.imwrite('unsharp_output.tif',unsharp_output)

plt.subplot(3, 1, 2)
plt.imshow(highboost_output , cmap="gray")
cv2.imwrite('highboost_output .tif',highboost_output )

plt.subplot(3,1,3)
plt.imshow(high_freq_emphasis , cmap="gray")
cv2.imwrite('high_freq_emphasis.tif',high_freq_emphasis)
```

Result

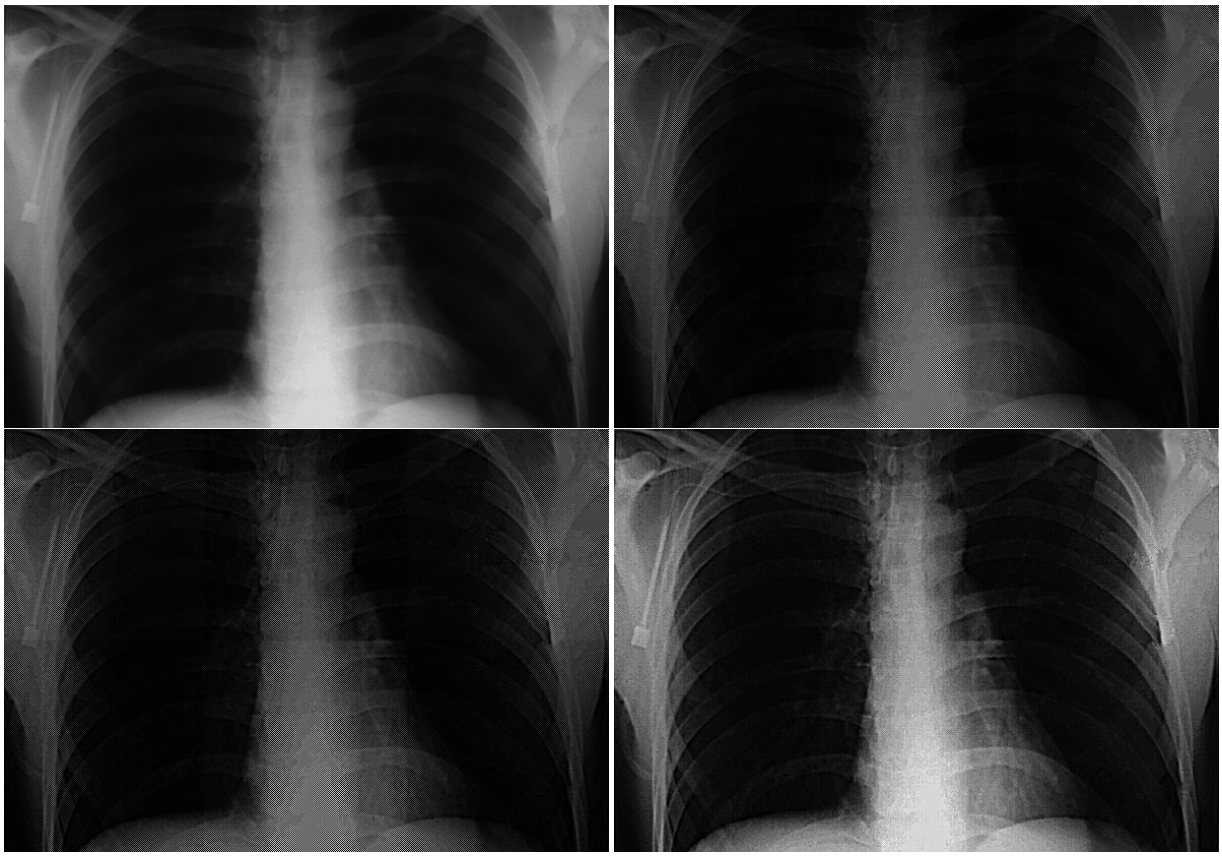

Figure 16: Chest Image (a)Original (b)Unsharp Masking (c)Highboost Filtering (k = 2)(d)High Frequency Emphasis Filtering (k = 2)

Inference

High Frequency Emphasis Filtering provides higher enhancement compared to Unsharp Masking and Highboost Filtering.