# Lab 4
# Pengolahan Citra
# Image Processing in Frequency Domain
# Senin, 19 Oktober 2020

## 1. Brief Introduction

In the frequency domain, the value and location of an image can be represented by sinusoidal relationships that depend upon the frequency of a pixel occurring within an image. It determines which pixels contain more important information and whether repeating patterns occur.

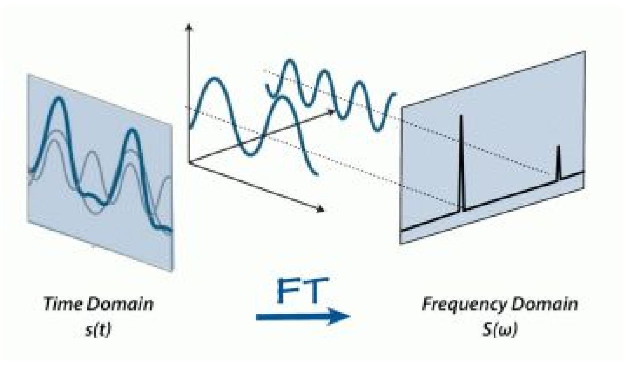


Figure 1. Transformation from time-domain to frequency-domain with FT (in image related, time-domain ~ spatial-domain)

This transformation is called “2D Discrete Fourier Transformation” or 2D-DFT. One of the implementations is called “Fast Fourier Transformation” or FFT which is considered as the current fastest DFT algorithm with complexity O(n log n). (We are not going to cover the math of DFT)

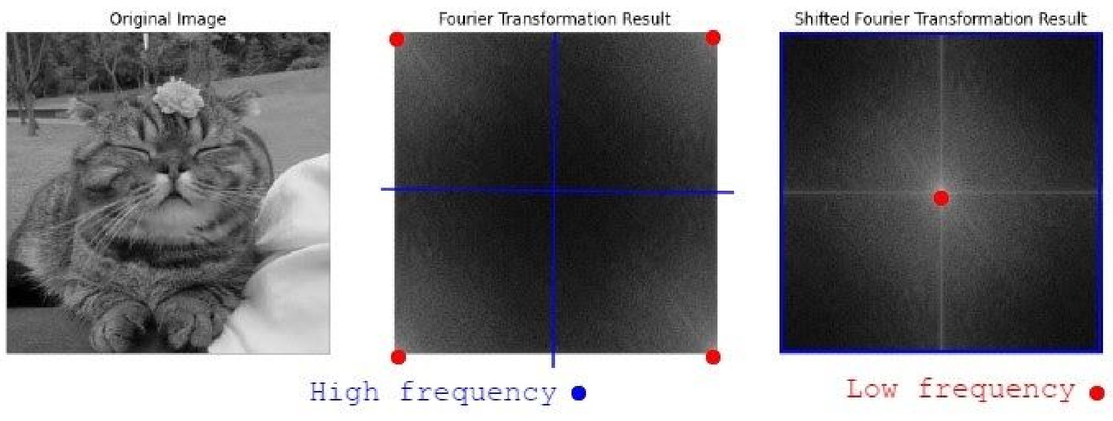

Original Image
Fourier Transformation Result
Shifted Fourier Transformation Result
High frequency ●
Low frequency ●

Figure 2. Fourier transform on grayscale image

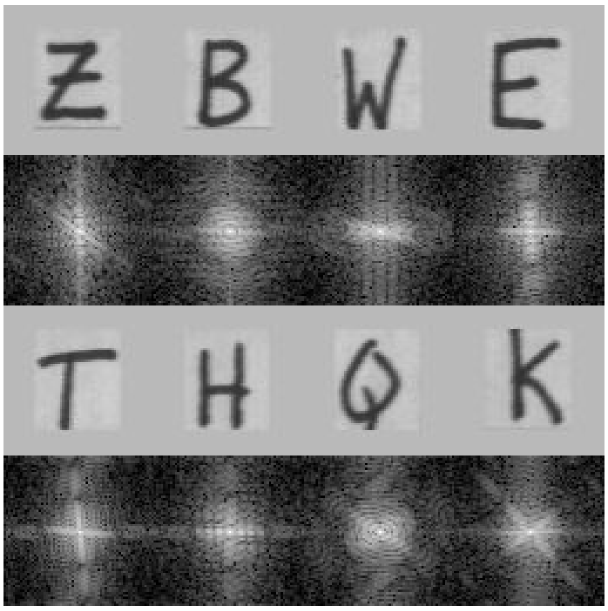


Figure 3. Fourier transform utilization for letter recognition

## 2. Fourier Representation

To transform an image into a frequency domain with FFT, we can use fftpack modules from scipy. Mathematically, this is DFT function to map images in spatial domain f(x,y) with the size of (M, N) to frequency domain F(u, v):

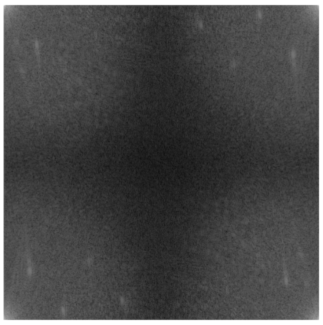
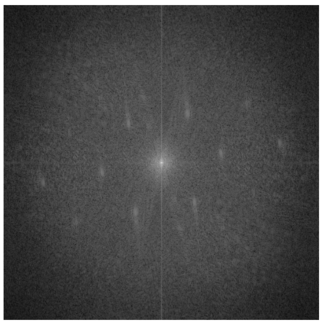
$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

Below is the mathematical function to map image in frequency domain to spatial domain using inverse DFT:

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

Here are the steps to represent an image in its frequency domain and vice versa:

| Code | Output image |
|---|---|
| <pre>import numpy as np<br>import matplotlib.pyplot as plt<br>from helper import *<br>from scipy import fftpack as fp<br>from skimage import io, color, util<br><br># Load image<br>im = color.rgb2gray(io.imread('lulu.jpg'))</pre> | |

| | |
|---|---|
| <pre># Calculate DFT<br># ft contains imaginary and real numbers<br>ft = fp.fft2(im)<br><br># Calculate magnitude of imaginary and real numbers<br>ft_norm = abs(ft)</pre> | |
| <pre># Scale image<br>ft_scale = np.log(1 + ft_norm)</pre> |  |
| <pre># Shift low frequency to the center of image<br>ft_shift = fp.fftshift(ft_scale)</pre> |  |
| <pre># Calculate IDFT<br># Can process only if imaginary and real numbers<br># provided (Pay attention to what var is used here)<br>ift = fp.ifft2(ft).real</pre> | |

## 3. Spatial vs Frequency Filtering

In spatial domain, filtering is done by convolving images with certain filtering kernels, filtering in frequency domain can be done by multiplying image's Fourier representation with certain filtering kernels. The result of this multiplication can be transformed back into the image representation using inverse DFT. Here is an example diagram of how filtering in frequency domain works.

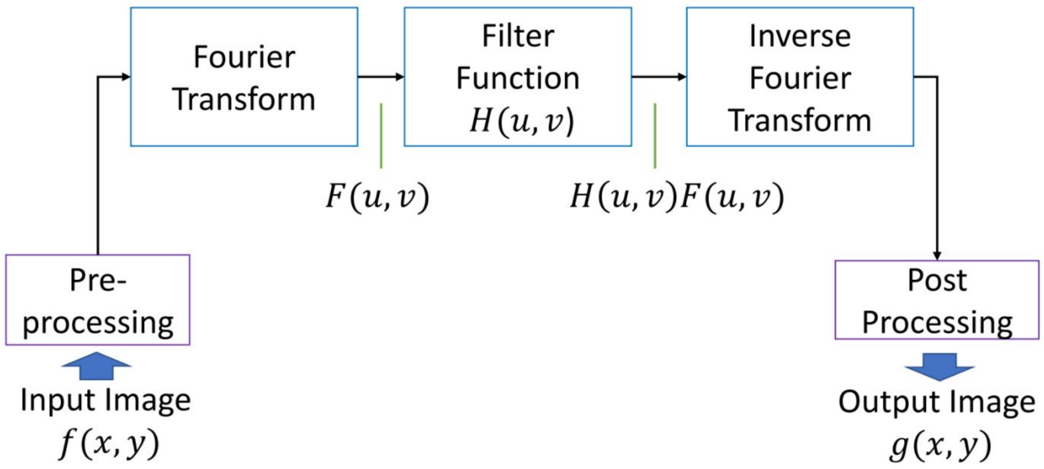
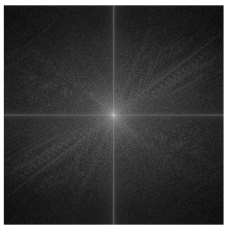
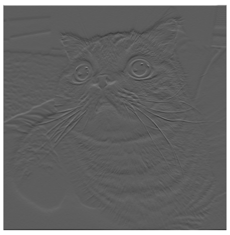
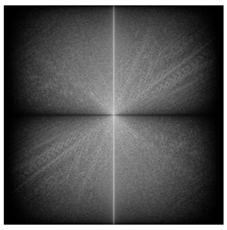
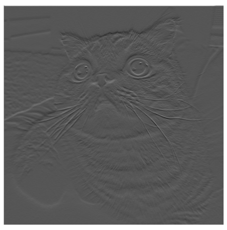


Figure 4. DFT Filtering process diagram

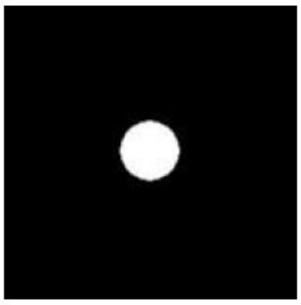
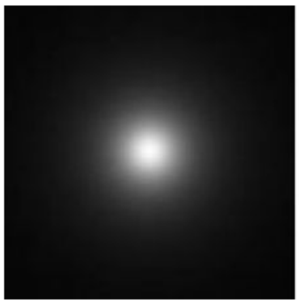
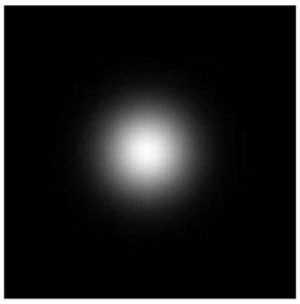
Here are the steps differences between filtering in spatial and frequency domain using sobel filter (edge detection):

| Spatial Code | Spatial Output | Frequency Code | Frequency Output |
|---|---|---|---|
| # Load image<br>im = io.imread('lulu_sobel.jpg')<br>im = color.rgb2gray(im)<br><br># Create sobel filter mask<br>kern = np.array([[1, 2, 1],<br>                  [0, 0, 0],<br>                  [-1, -2, -1]]) | | # Load image<br>im = io.imread('lulu_sobel.jpg')<br>im = color.rgb2gray(im)<br><br># Create sobel filter mask<br>kern = np.array([[1, 2, 1],<br>                  [0, 0, 0],<br>                  [-1, -2, -1]]) | |
|  |  | # Get padded image size<br>w, l = paddedsize(im.shape[0],<br>                  im.shape[1])<br><br># Calculate DFT<br>f = fp.fft2(im, (w, l))<br>h = fp.fft2(kern, (w, l))<br><br># Need to shift & scale to show image |  |
| # Convolve the image and kernel<br># Refer to scipy.signal.convolve2d<br>filtered = convolve2d(kern, im) |  | # Multiply the image and kernel<br>g = h * f |  |
|  |  | # Calculate Inverse DFT<br>ift = fp.ifft2(g).real<br><br># Remove padding<br>ift = ift[:im.shape[0],<br>:im.shape[1]] |  |

|  |  |  |  |
|---|---|---|---|
| `# Obtain magnitude for the edges`<br>`m = abs(filtered)` | | `# Obtain magnitude for the edges`<br>`m = abs(ift)` | |
| `# Binary thresholding`<br>`b = m > 0.5` | | `# Binary thresholding`<br>`b = m > 0.5` | |

## 4. Low-Pass Filtering

Low pass filters are used for blurring or smoothing an image. Suppress fourier transform value in high frequency and let low frequency through. There are 3 common low pass filtering techniques: Ideal Lowpass Filter, Butterworth Lowpass Filter and Gaussian Lowpass Filter.

| Ideal Lowpass Filter | Butterworth Lowpass Filter | Gaussian Lowpass Filter |
|---|---|---|
|  |  |  |

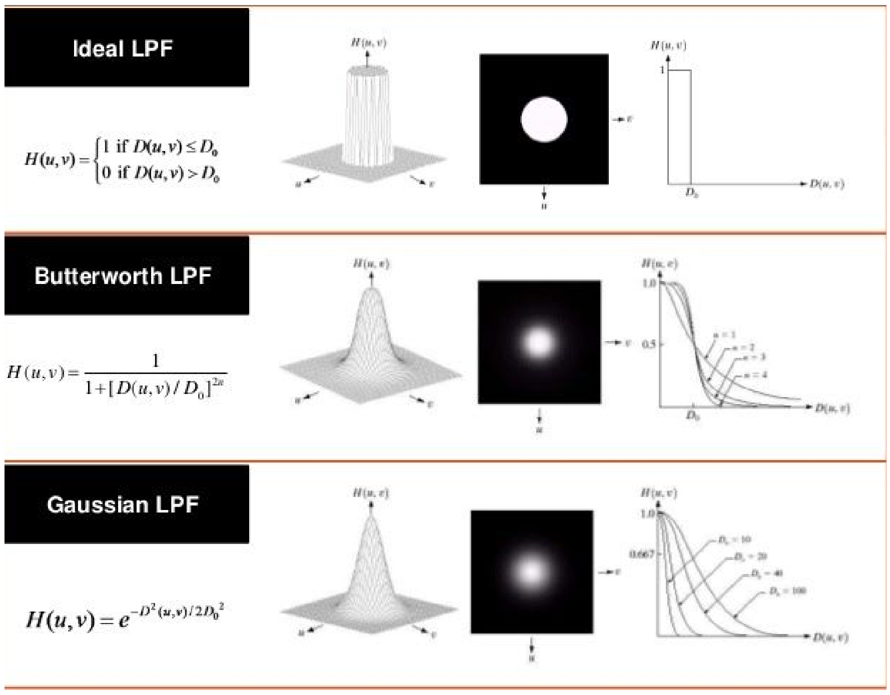
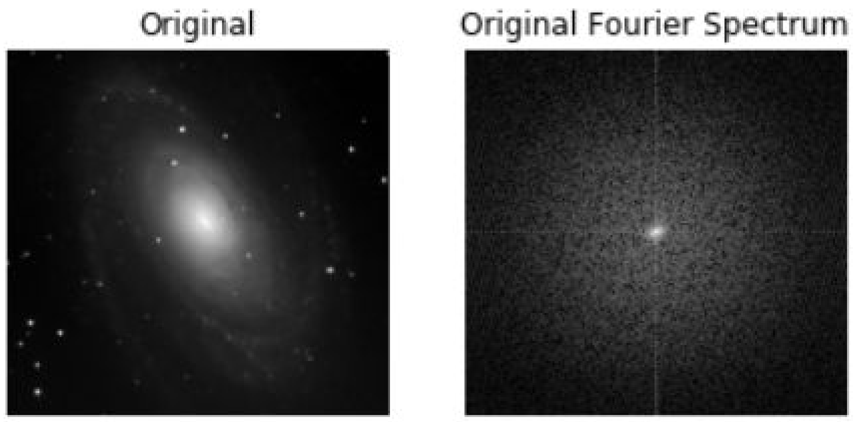

Ideal LPF
$H(u,v)=\begin{cases}1 & \text{if } D(u,v)\le D_0\\ 0 & \text{if } D(u,v)>D_0\end{cases}$
$H(u,v)$
$D_0$
$D(u,v)$
Butterworth LPF
$H(u,v)=\frac{1}{1+[D(u,v)/D_0]^{2n}}$
1.0
0.5
n = 1
n = 2
n = 3
n = 4
Gaussian LPF
$H(u,v)=e^{-D^2(u,v)/2D_0^2}$
0.667
$D_0 = 10$
$D_0 = 20$
$D_0 = 40$
$D_0 = 100$

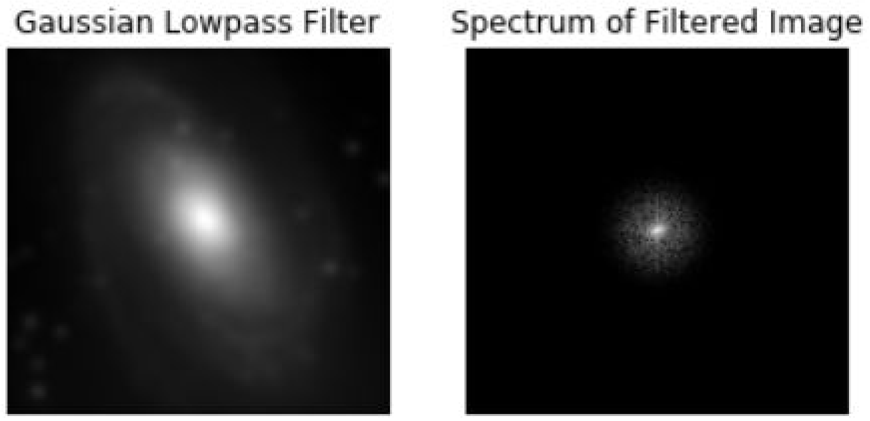
Figure 5. Types of Lowpass filter

Below are examples to apply gaussian lowpass filter to an image

```
# Load image
i1 = color.rgb2gray(io.imread('galaxy.png'))
w, l = paddedsize(i1.shape[0], i1.shape[1])

# Create gaussian lowpass filter
# lpfilter function is provided in helper.py
h = lpfilter('gaussian', w, l, 0.05 * w)

# Calculate DFT
f = fp.fft2(i1,(w,l))

# Apply lowpass filter
LPFS_galaxy = h * f

# Calculate IDFT for spatial domain transformation
LPF_galaxy = fp.ifft2(LPFS_galaxy).real
LPF_galaxy = LPF_galaxy[:i1.shape[0],:i1.shape[1]]

# Shifting for fourier spectrum display
Fc = fp.fftshift(f)
Fcf = fp.fftshift(LPFS_galaxy)

# Scaling for fourier spectrum display
S1 = np.log(1+abs(Fc))
S2 = np.log(1+abs(Fcf))
```

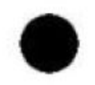
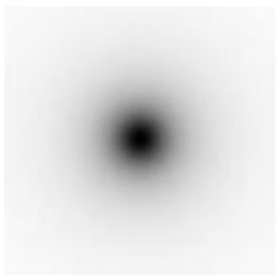
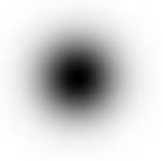
```
# Show image
plt.subplot(1,2,1); plt.imshow(i1, cmap='gray')
plt.title("Original")
plt.axis("off")

plt.subplot(1,2,2); plt.imshow(S1, cmap='gray')
plt.title("Original Fourier Spectrum")
plt.axis("off")
plt.show()
```





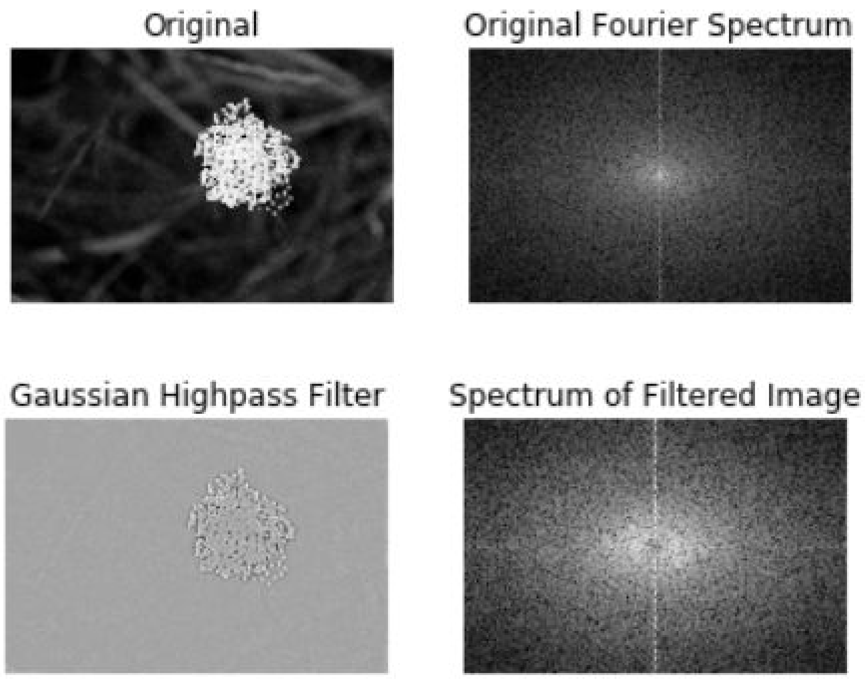
## 5. High-Pass Filtering

High pass filters are used for image sharpening. Suppress fourier transform value in low frequency and let high frequency through. There are 3 common high pass filtering techniques: Ideal Highpass Filter, Butterworth Highpass Filter and Gaussian Highpass Filter.

| Ideal HighpassFilter | Butterworth Highpass Filter | Gaussian Highpass Filter |
|---|---|---|
|  |  |  |

Below are examples to apply gaussian highpass filter to an image

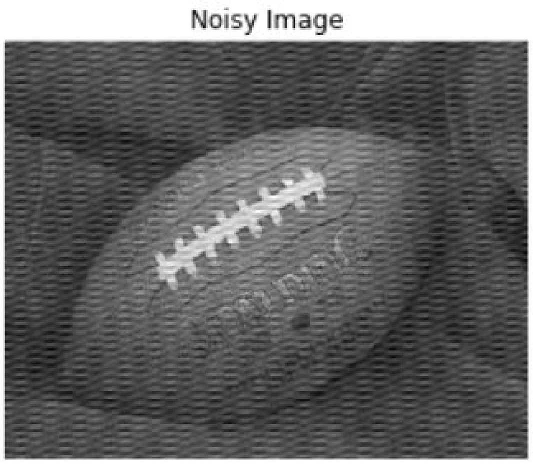
```
# Load image
i1 = color.rgb2gray(io.imread('flower.jpg'))
w, l = paddedsize(i1.shape[0], i1.shape[1])

# Create gaussian highpass filter
# hpfilter function is provided in helper.py
h = hpfilter('gaussian', w, l, 0.05 * w)

# Calculate DFT
f = fp.fft2(i1,(w,l))

# Apply highpass filter
LPFS_flower = h*f

# Calculate IDFT for spatial domain transformation
LPF_flower = fp.ifft2(LPFS_flower).real
LPF_flower = LPF_flower[:i1.shape[0],:i1.shape[1]]

# Shifting for fourier spectrum display
Fc = fp.fftshift(f)
Fcf = fp.fftshift(LPFS_flower)

# Scaling for fourier spectrum display
S1 = np.log(1+abs(Fc))
S2 = np.log(1+abs(Fcf))

# Show image
plt.subplot(1,2,1); plt.imshow(i1, cmap='gray')
plt.title("Original")
plt.axis("off")

plt.subplot(1,2,2); plt.imshow(S1, cmap='gray')
plt.title("Original Fourier Spectrum")
plt.axis("off")
plt.show()
```

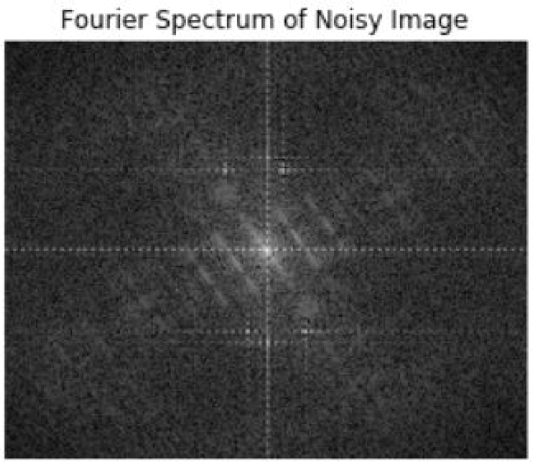


## 6. Notch Filtering

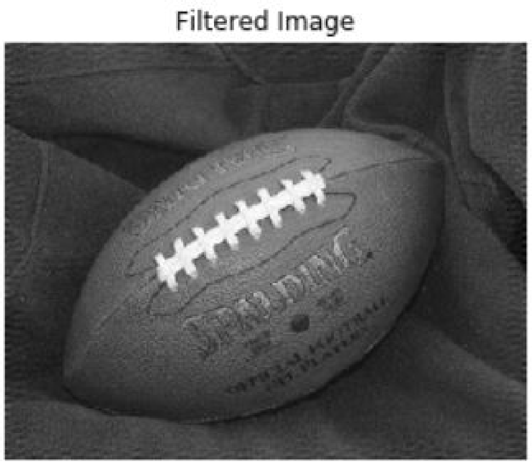
Notch filters are used to remove ”spectral” noise from an image. It is known as a band-stop filter with a narrow stopband. It will attenuate specific chosen frequency (and a few of its neighbors) and let other frequencies pass through.

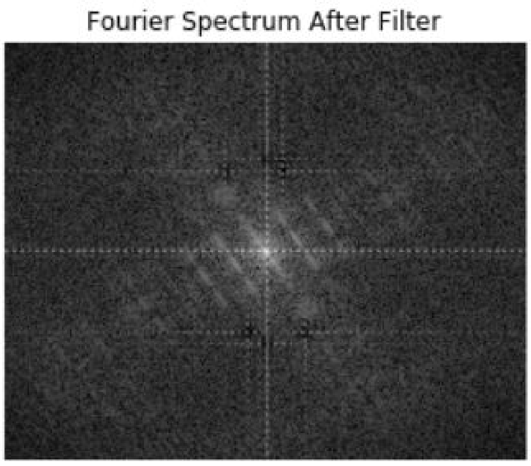
```
# Load image
football = io.imread('noiseball.png')
w, l = paddedsize(football.shape[0],football.shape[1])

# Calculate DFT
F = fp.fft2(util.img_as_float(football),(w,l))

# Scaling & Shifting for fourier spectrum display
Fc = fp.fftshift(F)
S1 = np.log(1+abs(Fc))

# Show image
plt.subplot(1,2,1); plt.imshow(football, cmap='gray')
plt.title('Noisy Image'); plt.axis("off")
plt.subplot(1,2,2); plt.imshow(S1, cmap='gray')
plt.title('Fourier Spectrum of Noisy Image');
plt.axis("off")
plt.show()
```





```
# Apply notch filter
# notch function is provided in helper.py
H1 = notch('btw', w, l, 10, 50, 100)
H2 = notch('btw', w, l, 10, 1, 400)
H3 = notch('btw', w, l, 10, 620, 100)
H4 = notch('btw', w, l, 10, 22, 414)
H5 = notch('btw', w, l, 10, 592, 414)
H6 = notch('btw', w, l, 10, 1, 114)

# Apply notch filter
FS_football = F*H1*H2*H3*H4*H5*H6

# Calculate IDFT for spatial domain transformation
F_football = fp.ifft2(FS_football).real
F_football = F_football[:football.shape[0],:football.shape[1]]

# Scaling & Shifting for fourier spectrum display
Fcf = fp.fftshift(FS_football)
S2 = np.log(1+abs(Fcf))

# Show image
plt.subplot(1,2,1); plt.imshow(F_football, cmap='gray')
plt.title('Filtered Image')
plt.axis("off")

plt.subplot(1,2,2); plt.imshow(S2, cmap='gray')
plt.title('Fourier Spectrum After Filter')
plt.axis("off")
plt.show()
```





Sources:
https://drive.google.com/drive/u/6/folders/105lvZMSbYQEt-iYDJWDm4S8cDjOPAB8N
http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/11ImageProc/13FourierFiltrationEn.pdf
https://www.cs.uregina.ca/Links/class-info/425/Lab5/lesson.html
https://www.programmersought.com/article/89491832197/

More sources to read:
http://www.princeton.edu/~cuff/ele201/kulkarni_text/frequency.pdf
https://www.cs.unm.edu/~brayer/vision/fourier.html

Image credit:
lulu_sobel (https://www.instagram.com/p/CFw1mrOhRNE/ )
lulu (https://www.picuki.com/media/2421078037246586877 )
flower(https://www.flickr.com/photos/bodomi/36282239771/)
galaxy(https://diffractionlimited.com/help/maximdl/Low-Pass_Filtering.htm)
noiseball(lab 3 pengolahan citra 2019/2020)