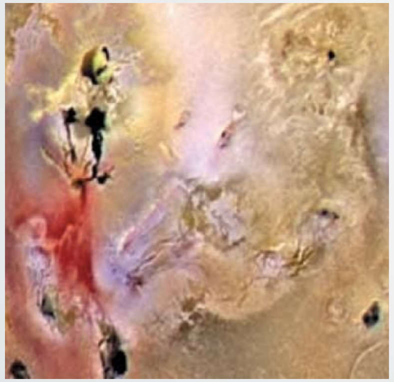







CSC 3141

IMAGE PROCESSING LABORATORY

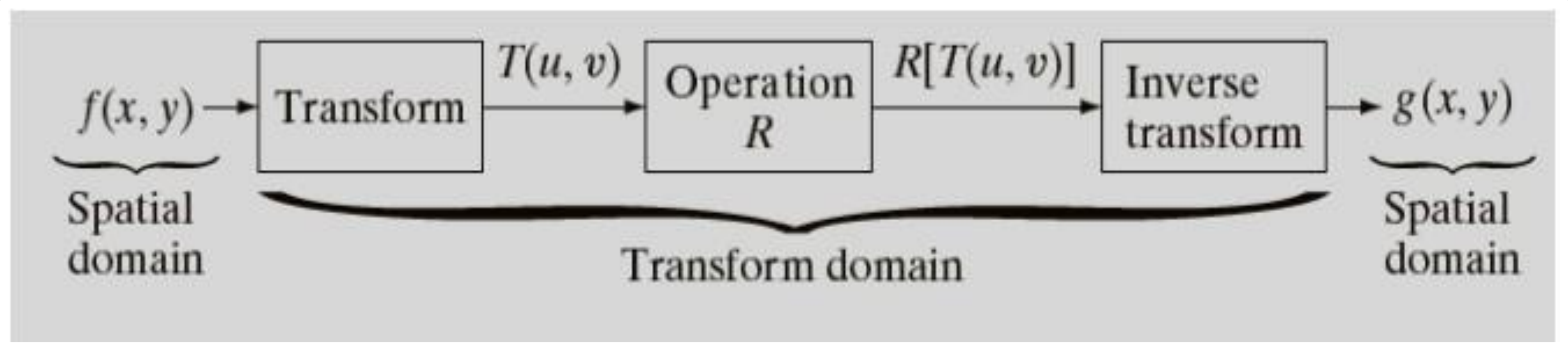
10 – Fourier Transform

Introduction to Fourier Transformation

Domain Transform in Image Processing

Fourier Transform converts an input image from spatial domain to frequency domain. The Fourier Transform is an important image processing tool which is used **to decompose an image into its sine and cosine components**. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent.

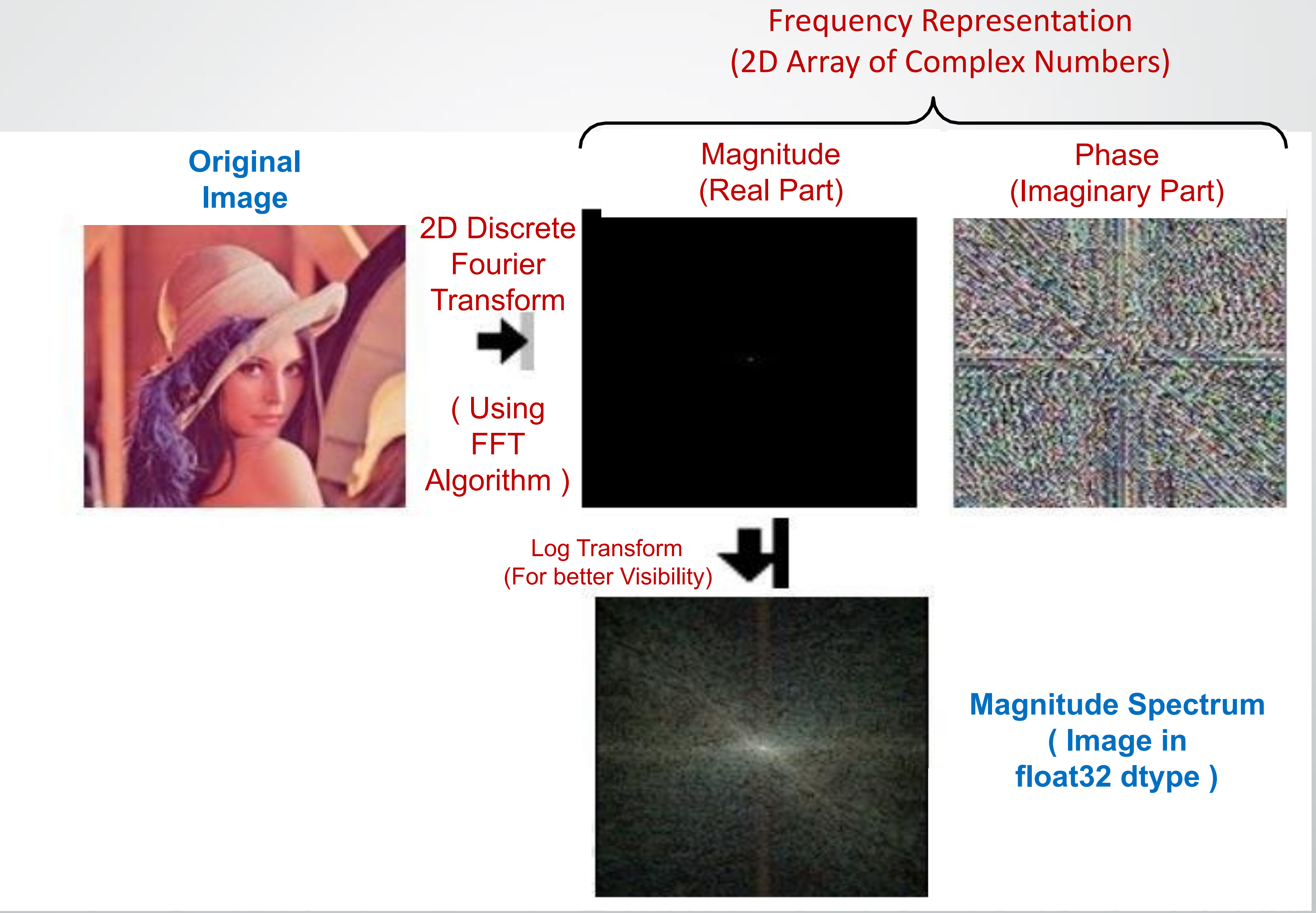
If you were to plot an image after taking its Fourier Transform, all you would see is a plot of high and low frequencies. **Low frequencies situated towards the center** of the image and high frequencies scattered around.



General Idea – Spatial to Frequency Domain Conversion

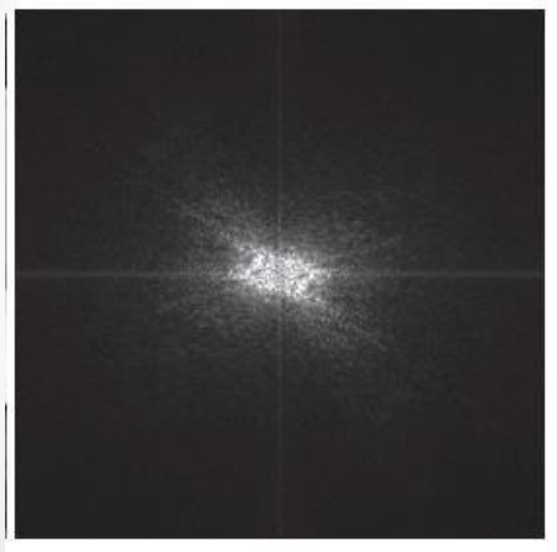
Magnitude and Phase

Direct numerical representation of the "Complex Numbers" is not very useful for image work.
By plotting the values onto a 2-dimentional plane, can convert the value into a Polar Representation consisting of 'Magnitude‘ and 'Phase‘ components.



Fourier Transform representation explained

Input Image
(Lena image)



Fourier Image
(Magnitude Spectrum)

Ref: https://www.youtube.com/watch?v=OOu5KP3Gvx0

Fourier Transform applications in image processing

The Use of Fourier transform in Image Processing is basically dominated by two areas:

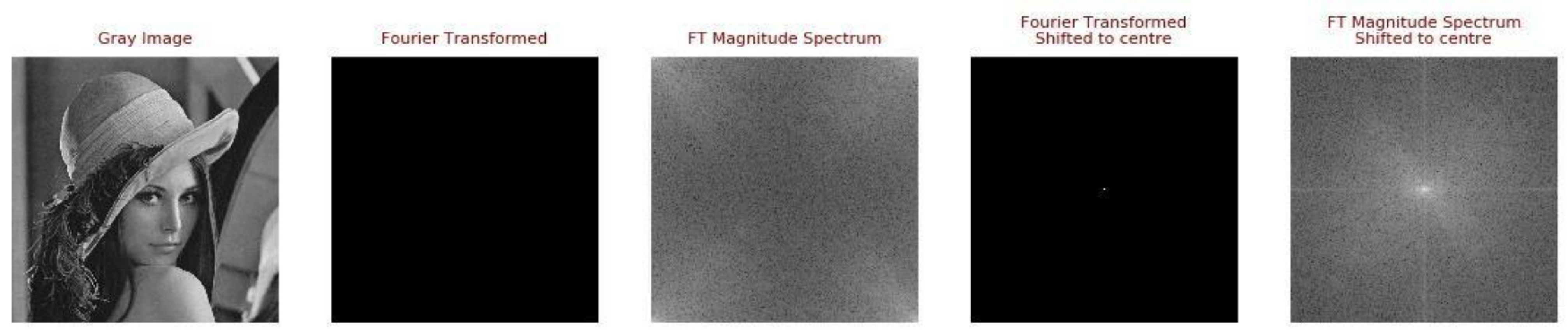
1. **OCR (Optical Character Recognition)**
2. **Noise Removal**

Ref: https://www.cs.unm.edu/~brayer/vision/fourier.html
Ref: https://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm
Ref: https://www.quora.com/How-are-Fourier-transforms-used-in-image-processing

Fourier Transform Implementations

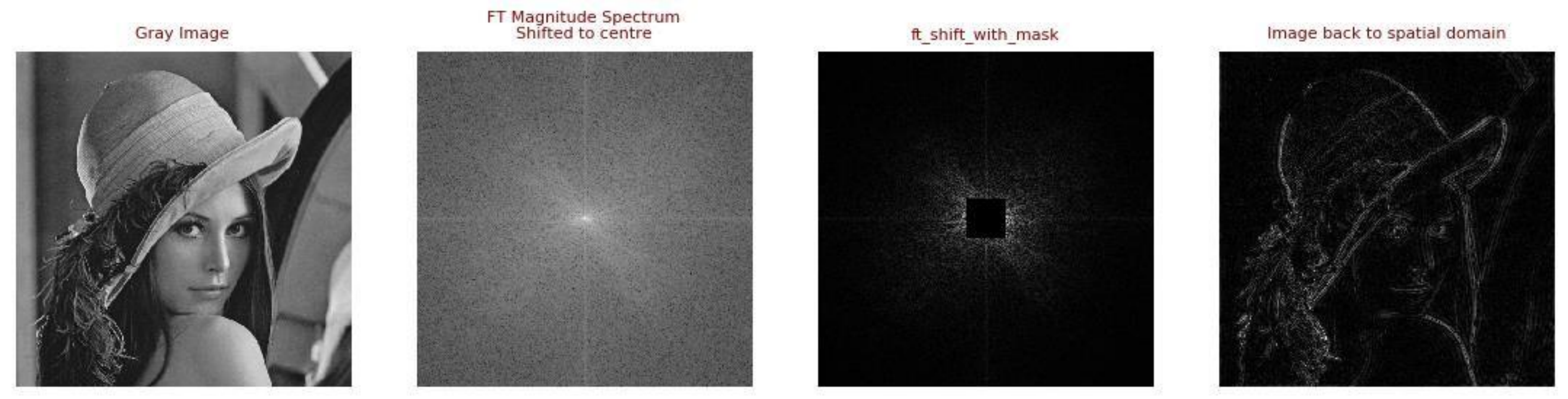
FT - Numpy implementation

```
#------ FT - Numpy Implementation ------

img = cv2.imread('images/lenna.bmp',0)

ft_img = np.fft.fft2(img)
ft_img_abs = np.abs(ft_img)

fshift_img = np.fft.fftshift(ft_img)
ft_shift_img_abs = np.abs(fshift_img)

#log transform for better visibility
magnitude_spectrum_base = 20*np.log(np.abs(ft_img))
magnitude_spectrum_shifted = 20*np.log(np.abs(fshift_img))
```



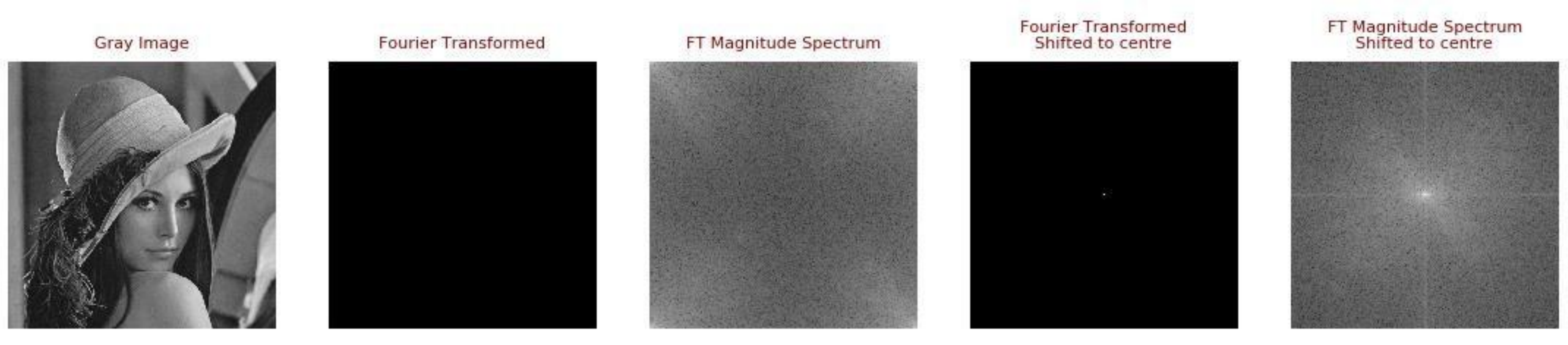
FT reversed - Numpy implementation

```
rows, cols = img.shape
crow,ccol = int(rows/2) , int(cols/2)
fshift_img[crow-30:crow+30, ccol-30:ccol+30] = 0

ft_shift_with_mask = np.abs(fshift_img)

f_ishift = np.fft.ifftshift(fshift_img)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)
```



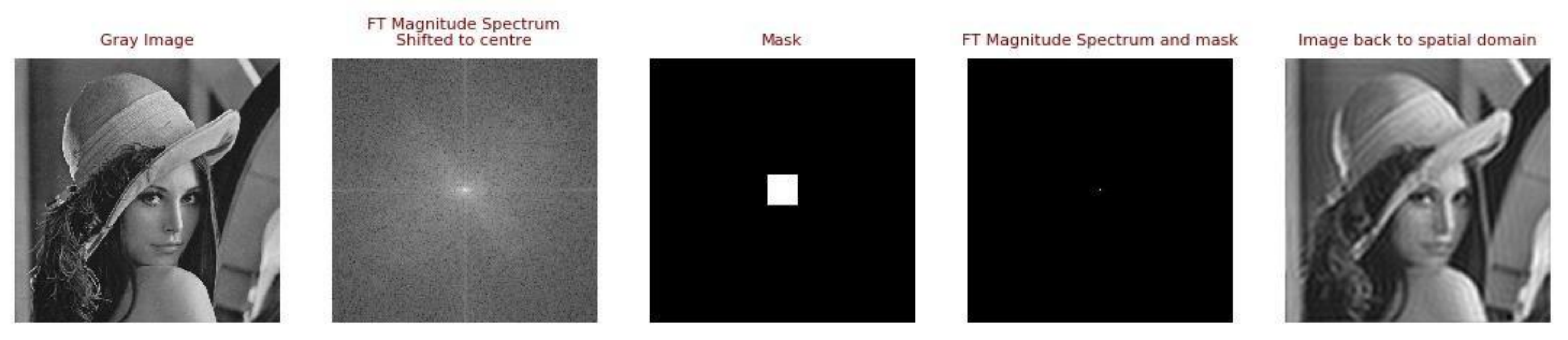
FT - OpenCV implementation

```
#------ FT - OpenCV Implementation ------

img = cv2.imread('images/lenna.bmp',0)

ft_img = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
ft_img_abs = np.abs(cv2.magnitude(ft_img[:,:,0],ft_img[:,:,1]))

fshift_img = np.fft.fftshift(ft_img)
ft_shift_img_abs = np.abs(cv2.magnitude(fshift_img[:,:,0],fshift_img[:,:,1]))

#log transform for better visibility
magnitude_spectrum_base = 20*np.log(cv2.magnitude(ft_img[:,:,0],ft_img[:,:,1]))
magnitude_spectrum_shifted = 20*np.log(cv2.magnitude(fshift_img[:,:,0],fshift_img[:,:,1]))
```



FT reversed - OpenCV implementation

```
rows, cols = img.shape
crow,ccol = int(rows/2) , int(cols/2)

# create a mask first, center square is 1, remaining all zeros (a low pass filter - blurring)
mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1

# apply mask and inverse DFT
fshift_and_mask = fshift_img*mask
fshift_and_mask_abs = np.abs(cv2.magnitude(fshift_and_mask[:,:,0],fshift_and_mask[:,:,1]))

f_ishift = np.fft.ifftshift(fshift_and_mask)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
```

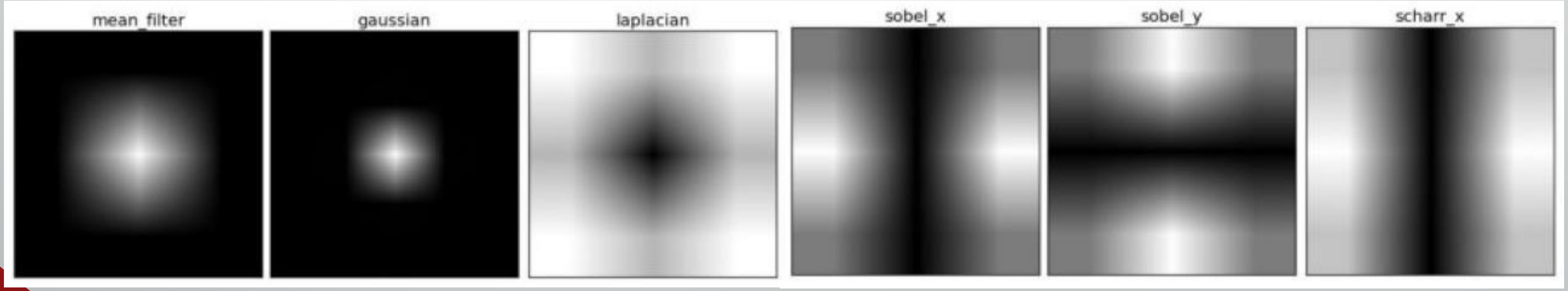


Frequency Domain Filters

Frequency Domain Filters

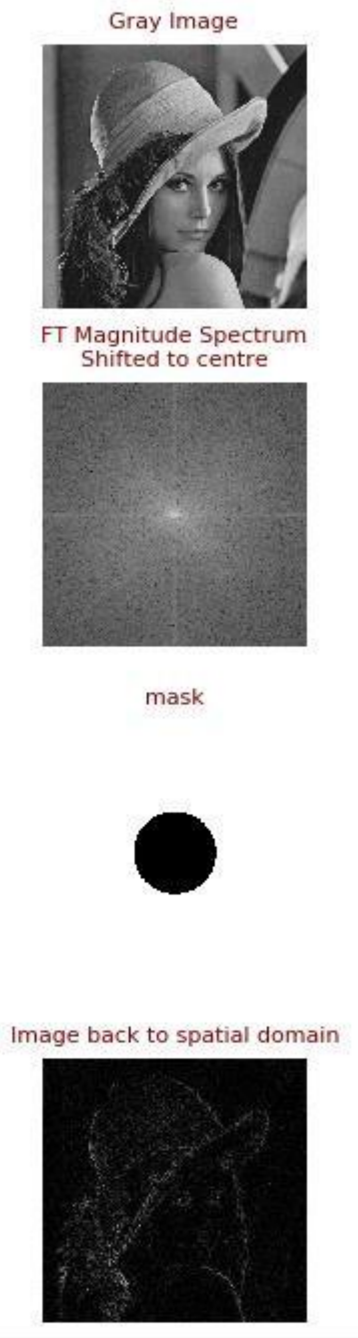
In an FFT transformed image, low frequencies are found in the center and high frequencies are scattered around, we can then create a mask array which has a circle of zeros in the center and rest all ones. Now when this mask is applied to the original image, the resultant image would only have high frequencies. This becomes quite useful as low frequencies correspond to edges in spatial domain.

Low Pass Filter (LPF) - Center area is 1, remaining all zeros
High Pass Filter (HPF) - Center area is 0, remaining all are Ones



Filters – High Pass Filter (For Edge Detection)

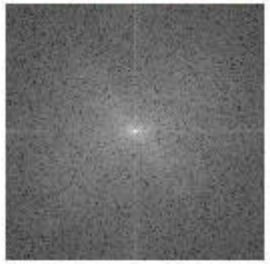
```
img = cv2.imread('images/lenna.bmp',0)

ft_img = np.fft.fft2(img)
fshift_img = np.fft.fftshift(ft_img)
ft_shift_img_abs = np.abs(fshift_img)

# log transform for better visibility
magnitude_spectrum_shifted = 20*np.log(np.abs(fshift_img))

# high pass - a circled mask
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.ones((rows, cols), np.uint8)
r = 80
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
mask[mask_area] = 0

fshift_and_mask = fshift_img*mask
ft_shift_with_mask = np.abs(fshift_and_mask)

f_ishift = np.fft.ifftshift(fshift_and_mask)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)
```



Filters – Low Pass Filter

```
img = cv2.imread('images/lenna.bmp',0)

ft_img = np.fft.fft2(img)
fshift_img = np.fft.fftshift(ft_img)
ft_shift_img_abs = np.abs(fshift_img)

# log transform for better visibility
magnitude_spectrum_shifted = 20*np.log(np.abs(fshift_img))

# high pass - a circled mask
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.zeros((rows, cols), np.uint8)
r = 80
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
mask[mask_area] = 1

fshift_and_mask = fshift_img*mask
ft_shift_with_mask = np.abs(fshift_and_mask)

f_ishift = np.fft.ifftshift(fshift_and_mask)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)
```


Gray Image


FT Magnitude Spectrum Shifted to centre

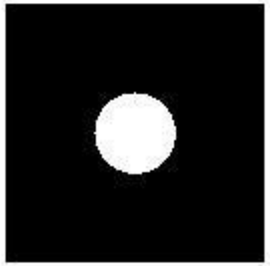

mask

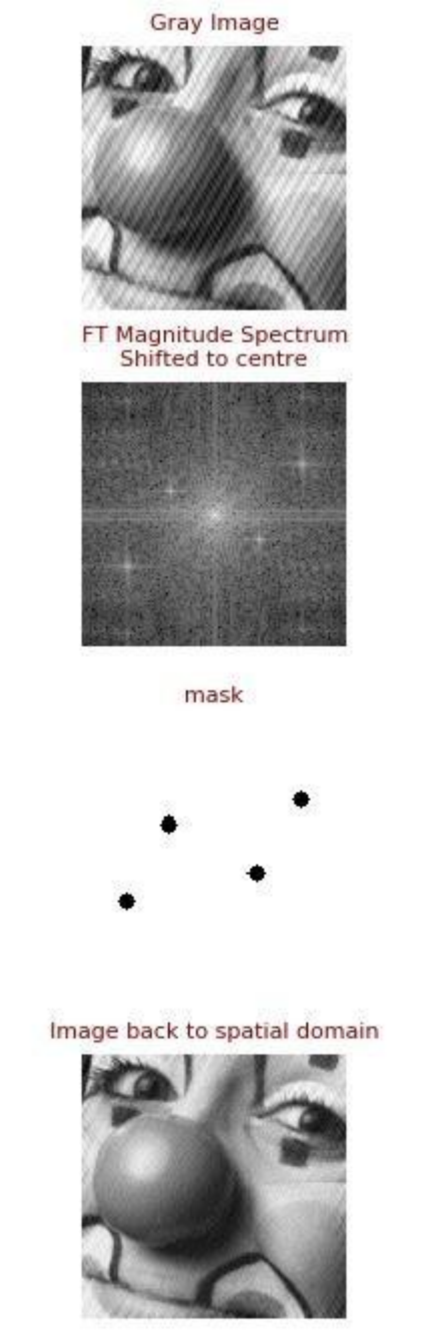

Image back to spatial domain

Removing Patterned Noise

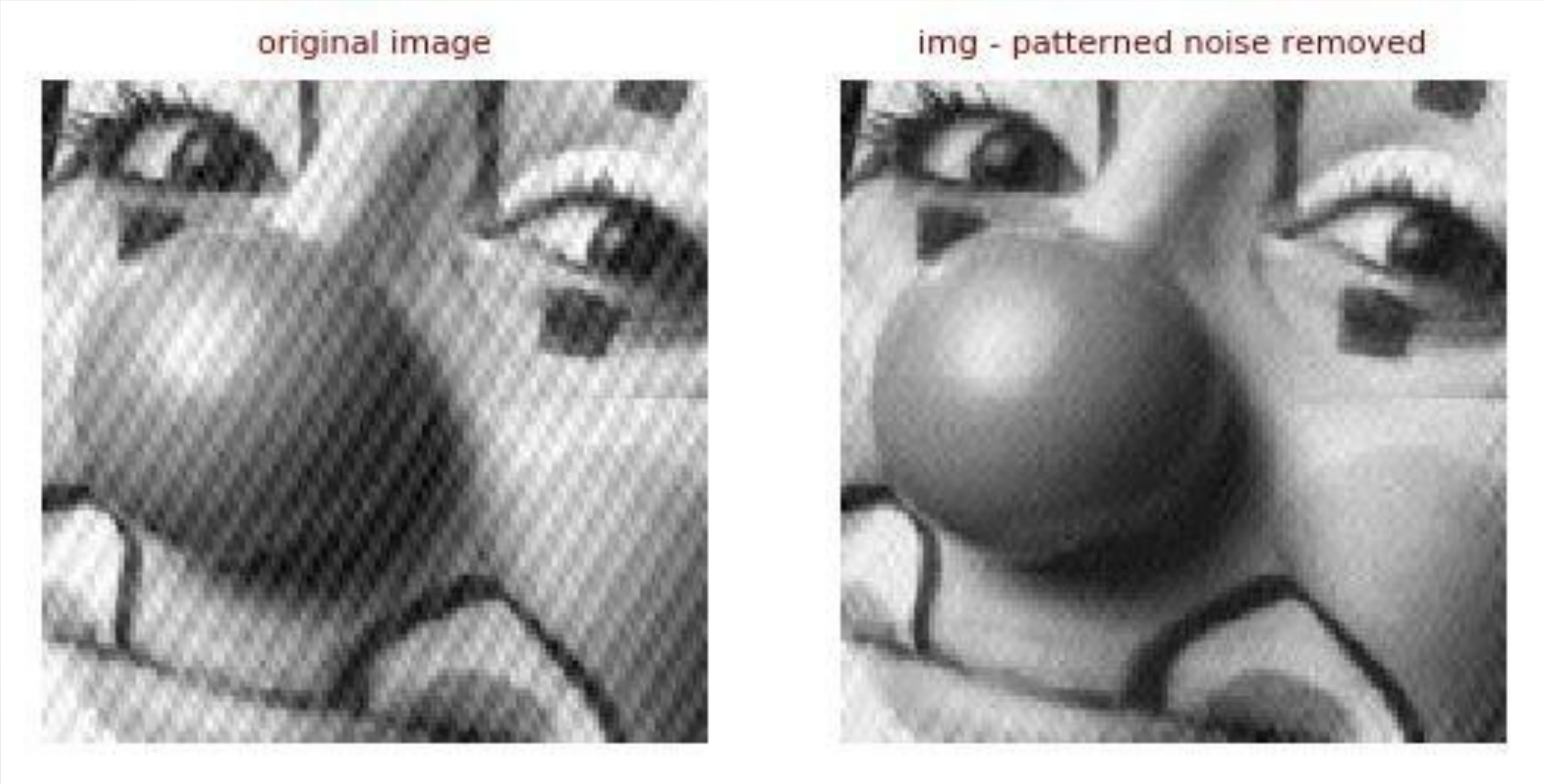
Patterned Noise removal

```
img = cv2.imread('images/clown.jpg',0)

ft_img = np.fft.fft2(img)
fshift_img = np.fft.fftshift(ft_img)
ft_shift_img_abs = np.abs(fshift_img)

# log transform for better visibility
magnitude_spectrum_shifted = 20*np.log(np.abs(fshift_img))

# band reject - a mask containing 4 black circles
rows, cols = img.shape
mask = np.ones((rows, cols), np.uint8)
r = 4

center_1,center_2,center_3,center_4 = [39,106],[51,42],[88,21],[75,84]

x, y = np.ogrid[:rows, :cols]
mask_area_1 = (x - center_1[0]) ** 2 + (y - center_1[1]) ** 2 <= r*r
mask[mask_area_1] = 0
mask_area_2 = (x - center_2[0]) ** 2 + (y - center_2[1]) ** 2 <= r*r
mask[mask_area_2] = 0
mask_area_3 = (x - center_3[0]) ** 2 + (y - center_3[1]) ** 2 <= r*r
mask[mask_area_3] = 0
mask_area_4 = (x - center_4[0]) ** 2 + (y - center_4[1]) ** 2 <= r*r
mask[mask_area_4] = 0

fshift_and_mask = fshift_img*mask
ft_shift_with_mask = np.abs(fshift_and_mask)

f_ishift = np.fft.ifftshift(fshift_and_mask)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)
```

https://www.quora.com/How-are-Fourier-transforms-used-in-image-processing



Patterned Noise removal



https://www.quora.com/How-are-Fourier-transforms-used-in-image-processing

– END –