Computer Vision

Image Processing with Fourier Transform

Course Instructor:
Dr. Suman Kumar Maji

Introduction

- Image Processing is a crucial task in Computer Vision.
- For example, when we train a Deep Learning model with a small amount of image data, we need to synthesize new images using Image Processing methods to improve the performance.
- There are many methods developed for Image Processing.

Fourier Series

- The Fourier series is found by the mathematician Joseph Fourier.
- He stated that any periodic function could be expressed as a sum of infinite sines and cosines:

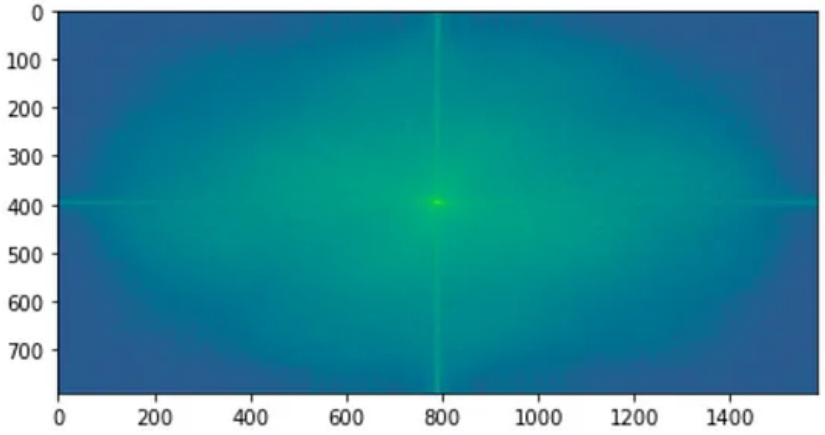
$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx)$$

- Fourier Transform is a generalization of the complex Fourier Series.
- In image processing, we use the discrete 2D Fourier Transform with formulas:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \cdot e^{i2\pi(xu/M+yv/N)}$$

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-i2\pi(xu/M+yv/N)}$$

Image in the frequency domain



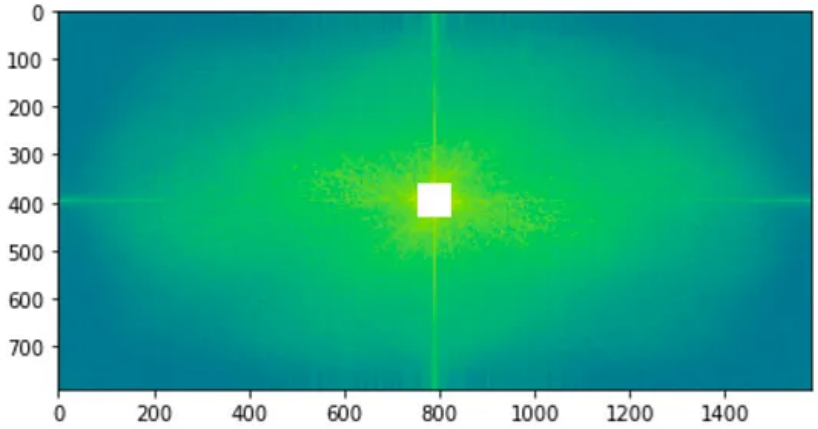
- The image on the right side is a spectrum of Fourier Transform.
- It is the heat map where the color represents the sine waves' amplitude, and position is their frequency.

cont...

- What we are doing with the 2D Fourier Transform is treating the image as a function of pixel position x and y.
- Imagine the function f(x, y) along axis (0,y):
  - So when the color jumps from "black (0)" to "white (255)", we say the color changes quickly, which means that a high amplitude and frequency sine wave contribute to that jump.
  - So the low-frequency sine waves are responsible for the image's overall color, while high-frequency waves are responsible for how color changes.

The responsibility of the high-frequency wave in an image

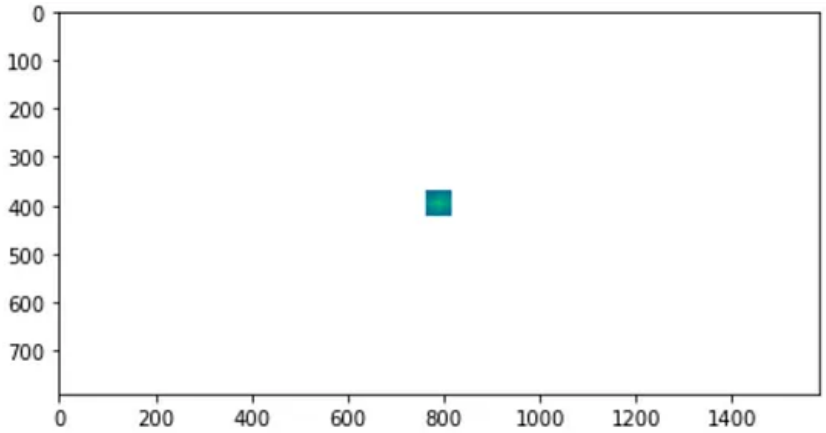
The cat image when we remove low frequency waves:



So, when we remove the low-frequency terms, we get an edge detection of the cat.

The responsibility of the low-frequency wave in an image

The cat image when we remove high frequency waves:



If we only keep the low-frequency waves, we will get a blurred image.

Fourier Transform and Convolution

- Using the Fourier Transform we can do convolution on images by just multiplication on its frequency domain.
- The convolution measures the total product in the overlapping regions of 2 functions.
- In Deep Learning, we often know about it as a convolution layer.
- While mathematically, it will look like this:

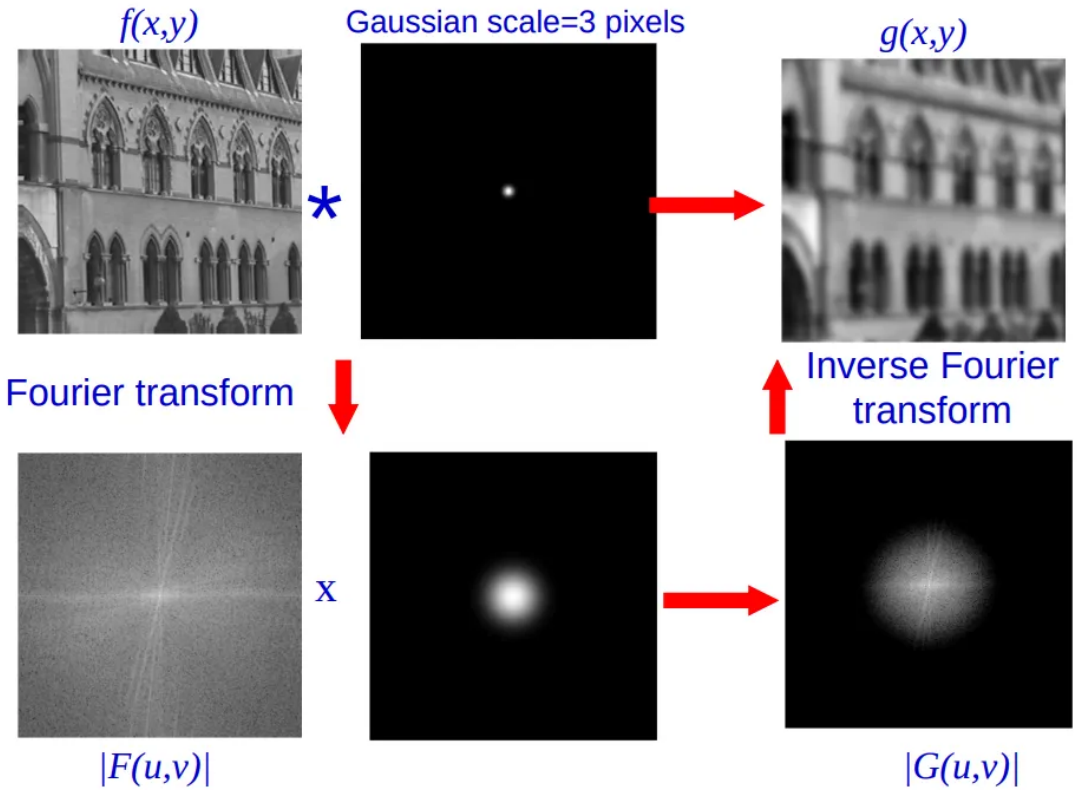
$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau.$$

- If we plug the Fourier Transform formula into this operation for f and g, we will get:

$$(f * g)(t) = F(t) \cdot G(t)$$

cont...

- With Fourier transform, we can change from computing a convolution to a multiplication.
- In the case of 2D Discrete Fourier Transform, this helps us reduce the time complexity by a lot.



Conclusion

- Fourier Transform is a powerful tool and is widely used in many applications.
- Besides the frequency representation, the Fourier Transform also produces the phase representation of the image.
- This contains the spatial information inside the image.
- However, the plot for the phases domain is less informative, and it also makes functions that affect the spatial information become more complex.

Better Edge detection and Noise reduction in images using Fourier Transform

Filters:

- Filters in image processing are just what the name suggests, **Filter**.
- They are typically a mask array of the same size as the original image which when superimposed on the ordinal image, extracts only the attributes that we are interested in.
- 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.

Types of filters

- There are mainly three types of filter used:
  1. High Pass Filter (HPF)
  2. Low Pass Filter (LPF)
  3. Band Pass Filter (BPF)
- There are various things you can do using that FFT transformed image:
  1. **Edge detection** - Using a High Pass filter or Band Pass filter
  2. **Noise Reduction** - Using a Low Pass filter
  3. **Blurring of image** - Using a Low Pass filter
  4. **Feature Extractions(In some cases)** - A mix and match of filters and some other openCV tools

High pass filter (HPF)

Here's what a HPF looks like in python - Circular HPF mask, center circle is 0, remaining all ones

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

mask = np.ones((rows, cols, 2), 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
```

Low pass filter (LPF)

LPF Filter - Circular LPF mask, center circle is 1, remaining all zeros

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

mask = np.zeros((rows, cols, 2), np.uint8)
r = 100
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
```

Band pass filter (BPF)

Band Pass Filter - Concentric circle mask, only the points living in concentric circle are ones

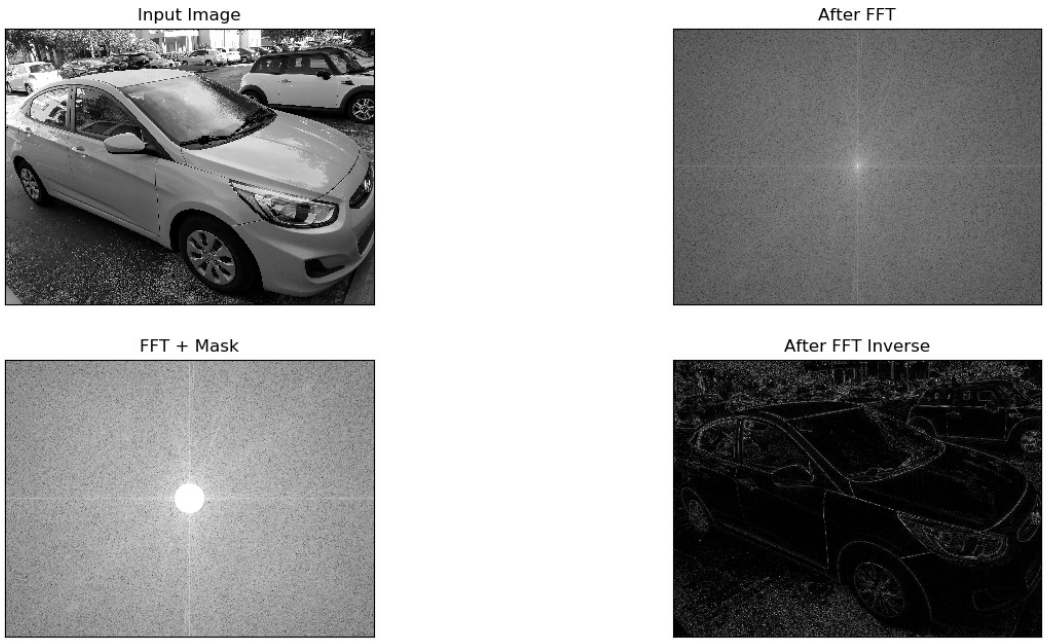
```
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)

mask = np.zeros((rows, cols, 2), np.uint8)
r_out = 80
r_in = 10
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = np.logical_and(((x - center[0]) ** 2
                            + (y - center[1]) ** 2>= r_in ** 2),
                            ((x - center[0]) ** 2
                            + (y - center[1]) ** 2<= r_out ** 2))
mask[mask_area] = 1
```

Edge Detection with High Pass Filter

- Detecting an edge in an image is of great use in the world of computer vision.
- Once we can extract edges in a image, we can use that knowledge for feature extraction or pattern detection. Edges in an image are usually made of High frequencies.
- So what we need to after taking a FFT (Fast Fourier Transform) of an image is, we apply a High Frequency Pass Filter to this FFT transformed image.
- This filter would in turn block all low frequencies and only allow high frequencies to go through.
- Finally, now if you take a inverse FFT on this filter applied image, you should see some distinct edge features in the original image.

cont...

- Below figure shows all four stages of the process and given after is the python code for the same.



- As can be seen, application of high pass filter, blocked all the low frequencies in the center and allowed only the high frequencies to pass through.
- Now since edges are usually made of low frequencies, that's we see in the resultant image.

Python code:

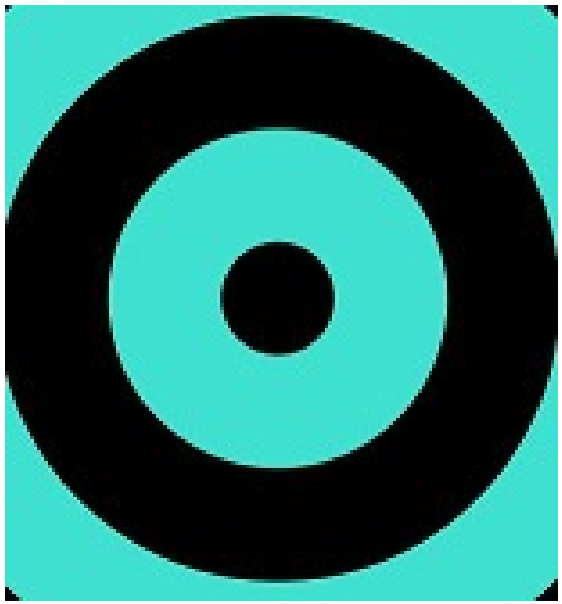
```
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)  # center

# Circular HPF mask, center circle is 0, remaining all ones

mask = np.ones((rows, cols, 2), 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

# apply mask and inverse DFT
fshift = dft_shift * mask

fshift_mask_mag = 2000 * np.log(cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1]))

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

Python code: cont...

```
plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('After FFT'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(fshift_mask_mag, cmap='gray')
plt.title('FFT + Mask'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(img_back, cmap='gray')
plt.title('After FFT Inverse'), plt.xticks([]), plt.yticks([])
plt.show()
```

Better edge detection in an image using a Band Pass Filter

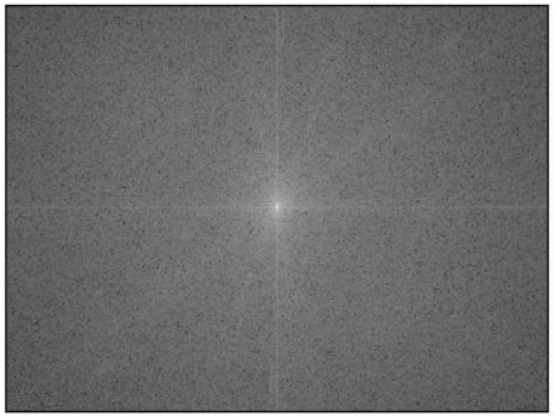
- A Band pass filter is the combination of both HPF and LPF.
- It strives to achieve a balance in not throwing away all of the low frequencies as well as all of the high frequencies.
- Or in other words, it only allows a range of frequencies to pass.
- To visualize how would the NPF mask would look like, think of it as two concentric circle with the area between two circles as one's and rest all zero's:



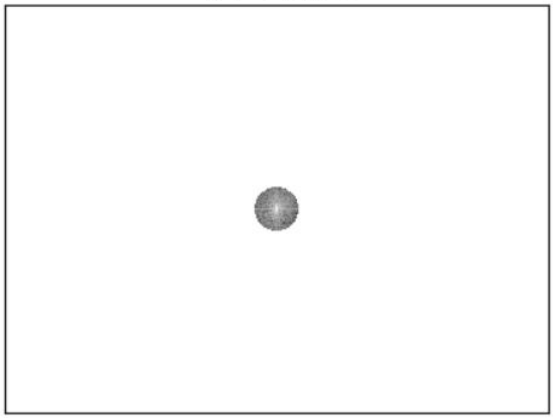
Example

Input Image

After FFT



FFT + Mask



After FFT Inverse

Side-by-side comparison between BPF and HPF

High Pass Filter

Band Pass Filter

Python code:

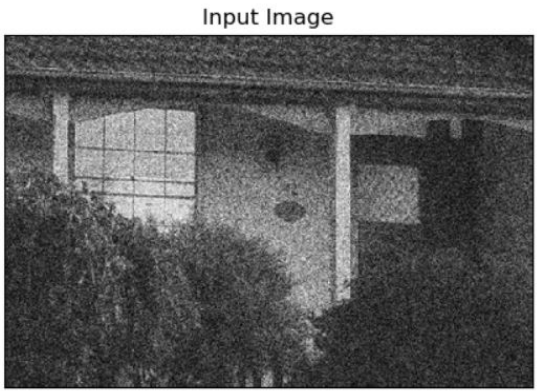
```
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)  # center

# Concentric BPF mask,with are between the two cerciles as one's, rest all zero's.
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)

mask = np.zeros((rows, cols, 2), np.uint8)
r_out = 80
r_in = 5
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]

mask_area = np.logical_and(((x - center[0]) ** 2 + (y - center[1]) ** 2 >= r_in ** 2),
                           ((x - center[0]) ** 2 + (y - center[1]) ** 2 <= r_out ** 2))
mask[mask_area] = 1

# apply mask and inverse DFT
fshift = dft_shift * mask
```

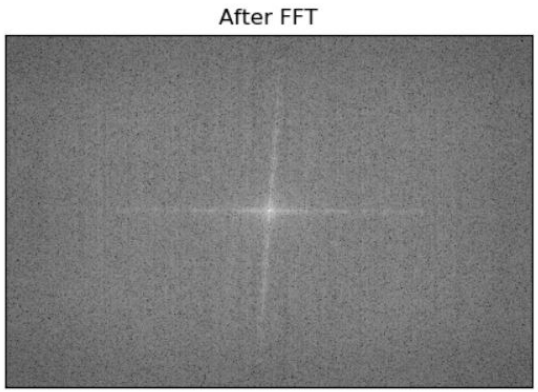
cont...

```
fshift_mask_mag = 2000 * np.log(cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1]))

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

plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('After FFT'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(fshift_mask_mag, cmap='gray')
plt.title('FFT + Mask'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(img_back, cmap='gray')
plt.title('After FFT Inverse'), plt.xticks([]), plt.yticks([])
plt.show()
```

Noise Filtering in an image using Low Pass Filter (LPF)

- Consider the following image and its fourier transform:



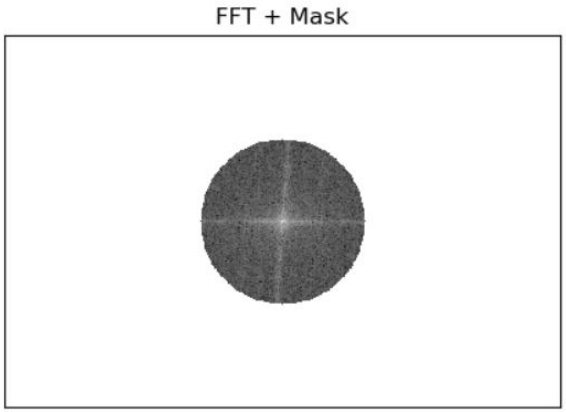


cont...

- As can be seen, the original image is quite noisy.
- We also notice a vertical and horizontal symmetry along the low frequency components in fourier transformed image.
- This is most probably due to sharp edges in the original pic.
- As we learned earlier, the high frequencies depict a sudden change of image contrast from one pixel to another in spatial domain.
- That's why they correspond to edges in the spatial domain.
- Now extending that argument, you can think of little white dots on the original image, which constitute the noise, as minuscule edges too.
- After all they follow the same pattern too i.e. sudden change of image contrast from one pixel to another in spatial domain.
- Now if we were to block majority of high frequencies, theoretically, that should help in noise reduction.

cont...

- Employing Low pass filter, we get the following result :



- As can be seen, we do see some reduced noise in the image but the lpf also took away some of the sharp feature of the image too.
- Note, if the input image contains a lot of sharp edges, like walls, pillars, house etc (like in this case), application of LPF will eat away at those features too.

Python code:

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

# Circular LPF mask, center circle is 1, remaining all zeros
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)

mask = np.zeros((rows, cols, 2), np.uint8)
r = 70
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

# apply mask and inverse DFT
fshift = dft_shift * mask

fshift_mask_mag = 2000 * np.log(cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1]))
```

cont...

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

plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('After FFT'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(fshift_mask_mag, cmap='gray')
plt.title('FFT + Mask'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(img_back, cmap='gray')
plt.title('After FFT Inverse'), plt.xticks([]), plt.yticks([])
plt.show()
```