CSC589 Introduction to Computer Vision
Lecture 8

Applications of Fourier Transform, Sampling

Bei Xiao

Last Lecture

- Fourier transform and frequency domain
  - Fourier transformation in 1d signal
  - Fourier transform of images
  - Fourier transform in Python

- Reminder: Read your text book 3.4. Homework 2 is out and will be Due on Thursday, Feb 19th (extended). Start early!

Today's lecture

- Review of Fourier Transform
- Filtering in Frequency domain
- Python exercises Here is a useful tutorial on fft with numpy: http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_transforms/py_fourier_transform/py_fourier_transform.html

Next Lecture: Guest Lecture

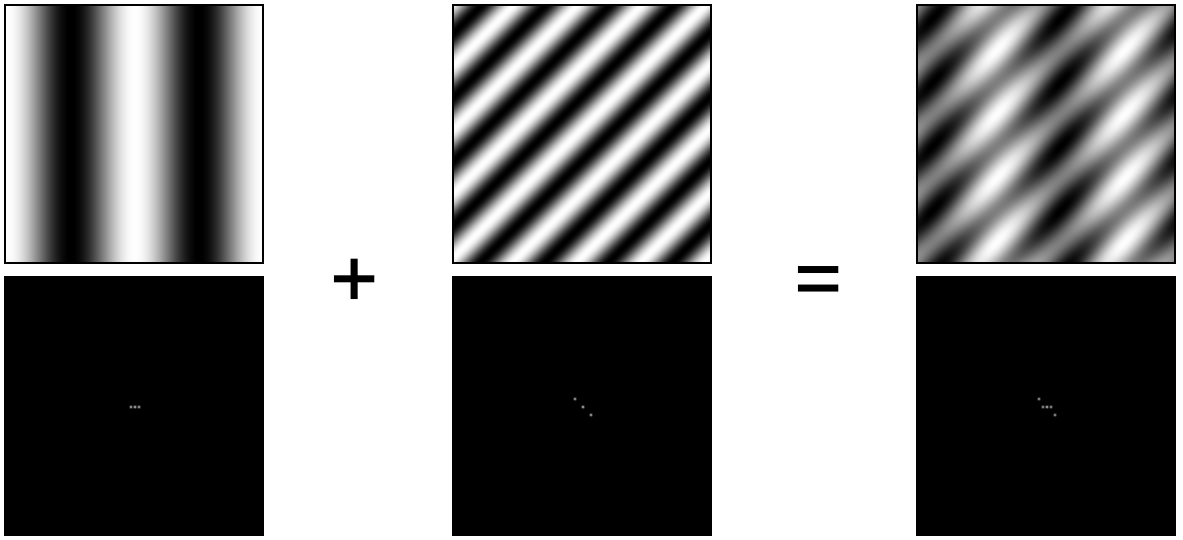
- Dr. Katerina Fragkiadaki:
- "Video Segmentation and Multi-object tracking in the Era of Deep Learning!"

- She will also give a talk on video understanding at 4-5pm on Thursday.

MATLAB or Python

- We can choose again for the main tool for the rest of the course.
- Python appear to be difficult, have a sharper learning curve, but it is a general purpose language.
- MATLAB has much more tutorials online (especially for image processing) and easier user interface. However, you have to pay $100 for a student' license or access through university's VPN with limited license.
- You can try it out and run some demo codes (I will upload) and we can vote again. I posted a survey on blackboard.
- For now, we continue with demos in Python.
- Homework 2 can be done either with Python or MATLAB.

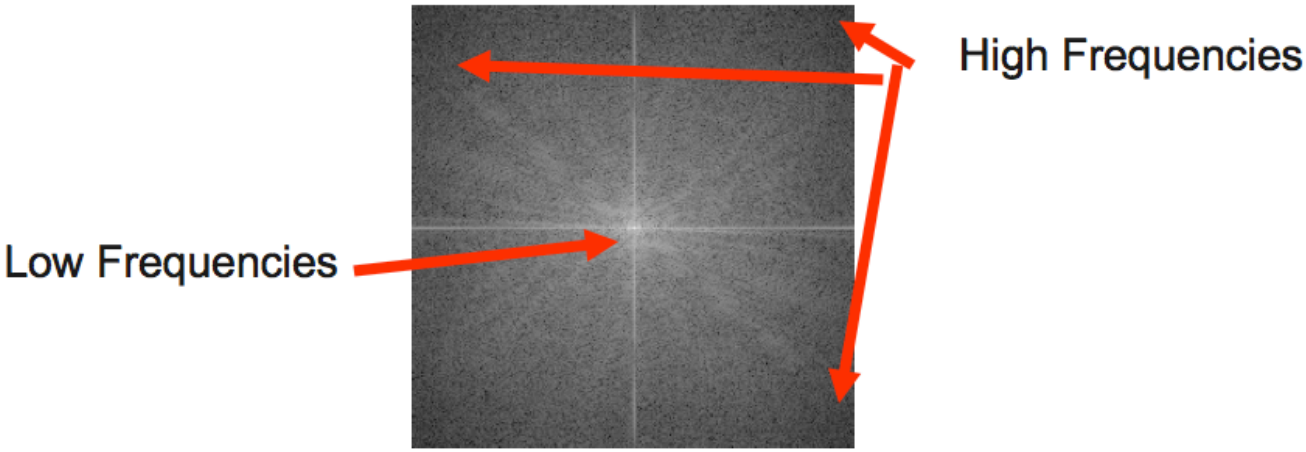
Signals can be composed



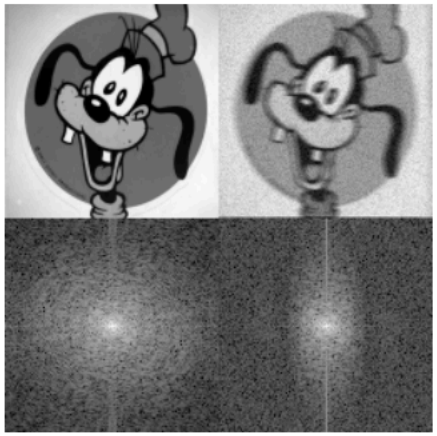
http://sharp.bu.edu/~slehar/fourier/fourier.html#filtering
More: http://www.cs.unm.edu/~brayer/vision/fourier.html

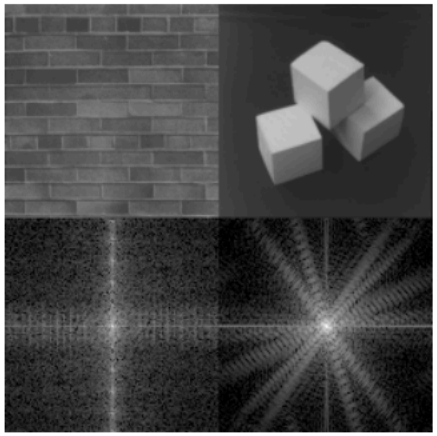
Fourier Transform

- Essentially, we are figuring out how to build the image out of these sinusoids of different frequency
- Each pixel in the transformed image corresponds to the amount of a sinusoid of a particular sinusoid that is needed

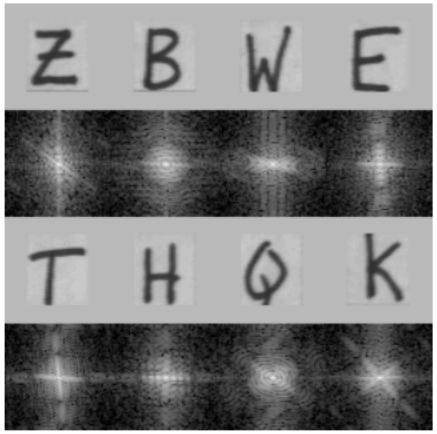


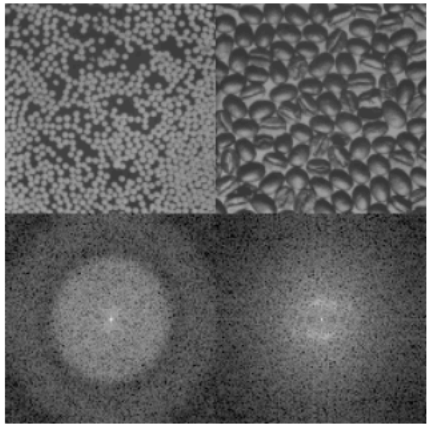
FFT magnitude spectrum of images

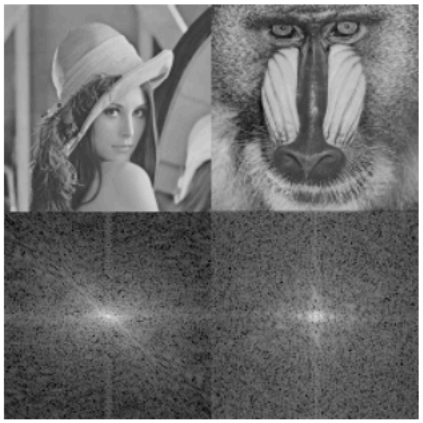




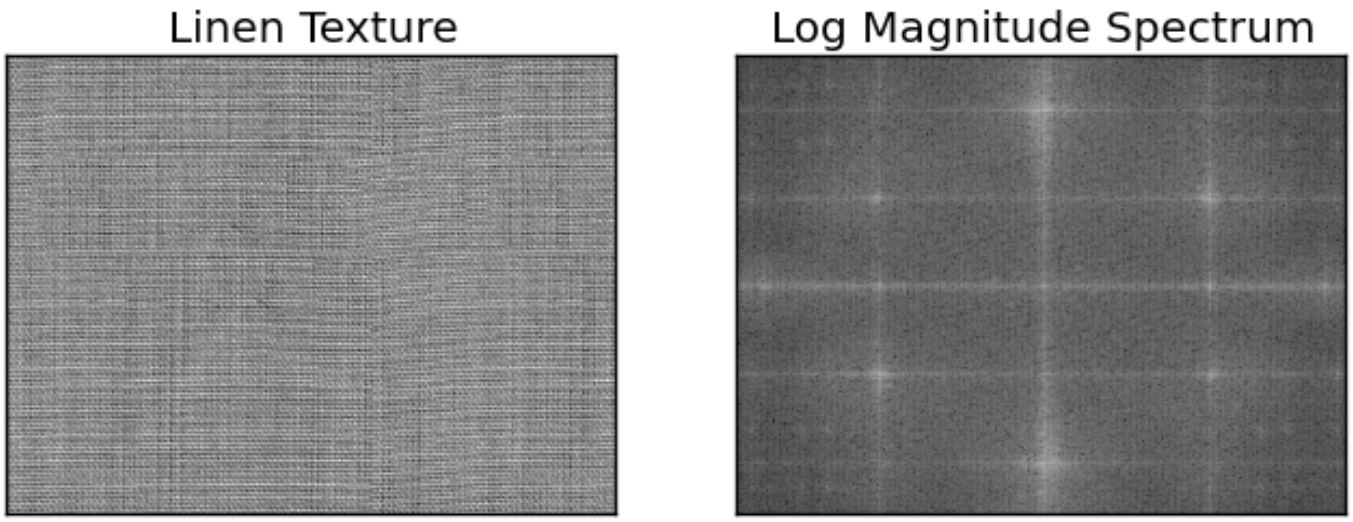
- notice a bright band going to high frequencies perpendicular to the strong edges in the image
- Anytime an image has a strong-contrast, sharp edge the gray values must change very rapidly. It takes lots of high frequency power to follow such an edge so there is usually such a line in its magnitude spectrum.





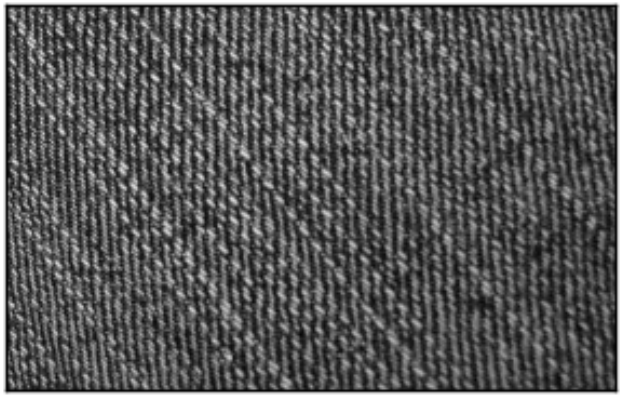


Application: Texture Analysis

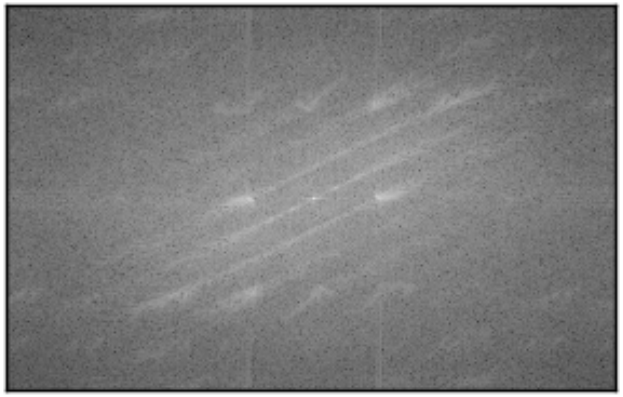


Application: Texture analysis

Jean Texture

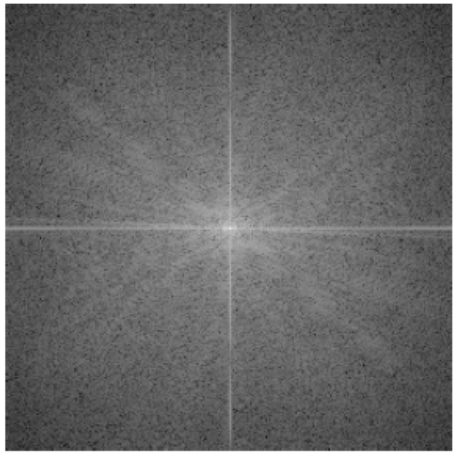


Log Magnitude Spectrum

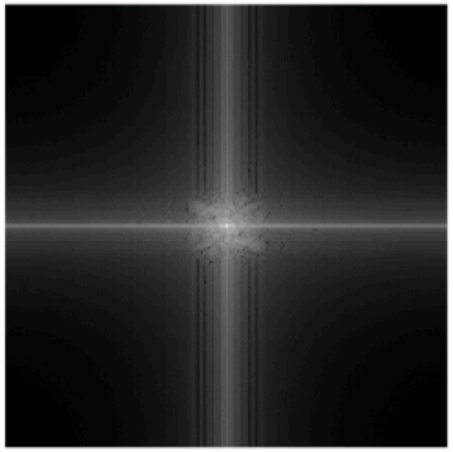


What are the high frequencies?

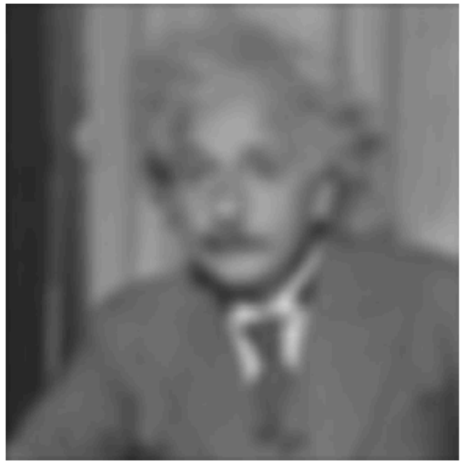
What if we remove the high frequencies?



Old Spectrum



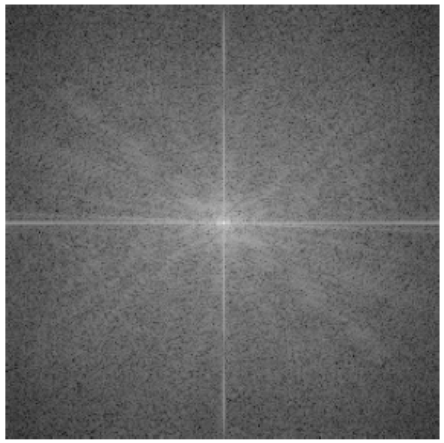
New Spectrum



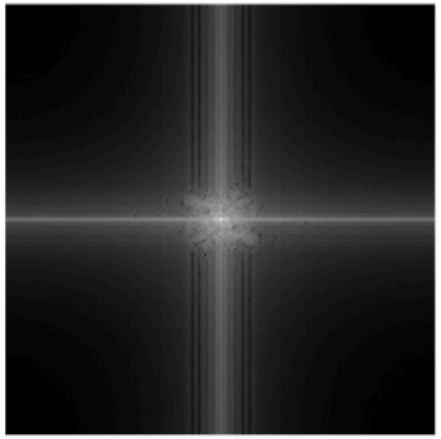
How will the new image look?

What are the high frequencies?

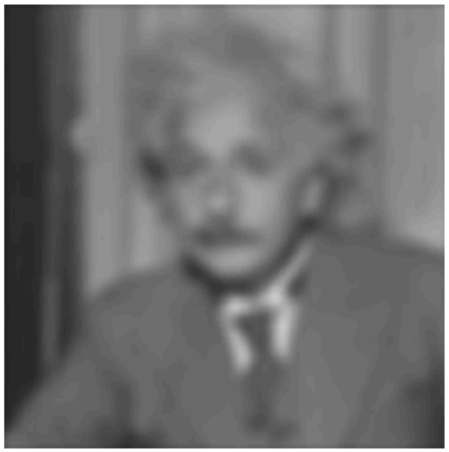
Removing the high frequencies makes the image look blurry



Old Spectrum



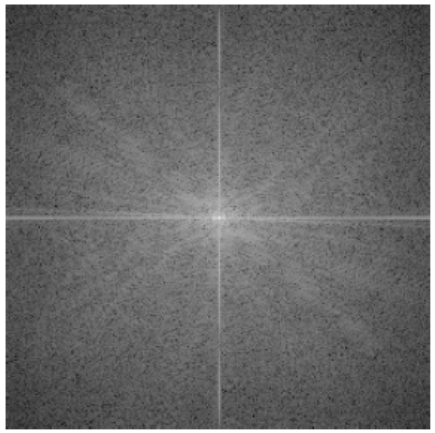
New Spectrum



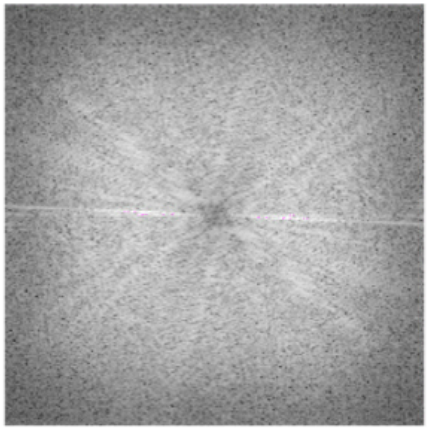
Try building a sharp edge out of low-frequency sinusoids

What are the low frequencies?

What if we remove the low frequencies?



Old Spectrum

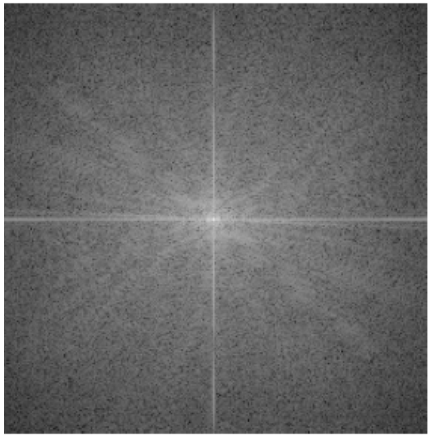


New Spectrum

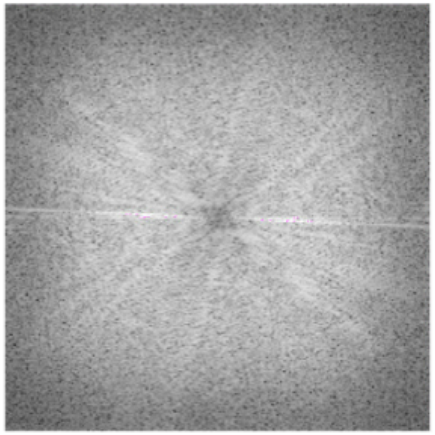
How will the new image look?

What are the low frequencies?

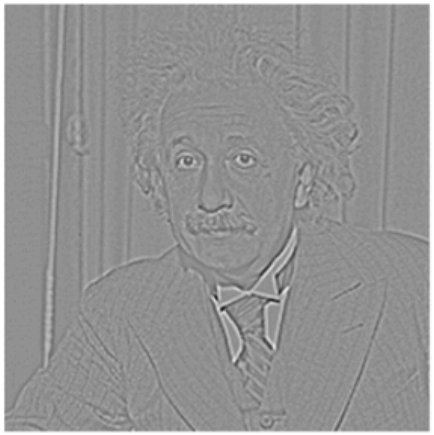
What if we remove the low frequencies?



Old Spectrum



New Spectrum



How will the new image look?

FFT in Python

FFT

```
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 30*np.log(np.abs(fshift))
phase = np.angle(fshift)
```

Inverse FFT

```
rows, cols = img.shape
crow, ccol = rows/2 , cols/2 # center of the image
 fshift[crow-5:crow+5, ccol-5:ccol+5] = 0
 f_ishift = np.fft.ifftshift(fshift)
 img_back = np.fft.ifft2(f_ishift)
```

2D discrete Fourier transform

- The discrete FT for a discrete signal f(x) with N values is given by:

$$F(u) = \frac{1}{N} \sum_{x=0..N-1} f(x) e^{-i 2\Pi ux / N}$$

$$F(u).re = \frac{1}{N} \sum_{x=0..N-1} f(x) \cos(-2\Pi ux / N)$$

$$F(u).im = \frac{1}{N} \sum_{x=0..N-1} f(x) \sin(-2\Pi ux / N)$$

Working with DFT (Discrete Fourier Transform)

- Is the complex part bothering you yet?
- Let's look at a different representation
- Every complex number can also be represented as: $Z = x+jy = re^{j\theta}$

- r- magnitude (real number), r = abs(Z) or $\sqrt{x^2 + y^2}$
- Θ-phase

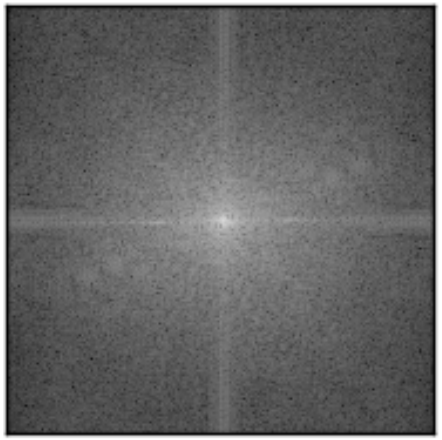
Phase and Magnitude

- Fourier transform of a real function is complex
  - Difficult to plot, visualize, instead we can think of phase and magnitude of the transform
- Phase is the phase of the complex transform
- Magnitude is the magnitude of the complex transform
- Curious fact
- All natural images have about the same magnitude transform Demonstration
- Take two pictures, swap the phase transform, compute the inverse, what does the result look like?

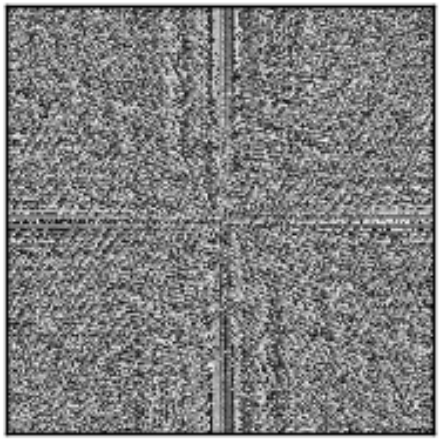
Phase and Magnitude

Input Image

Log Magnitude Spectrum



Phase



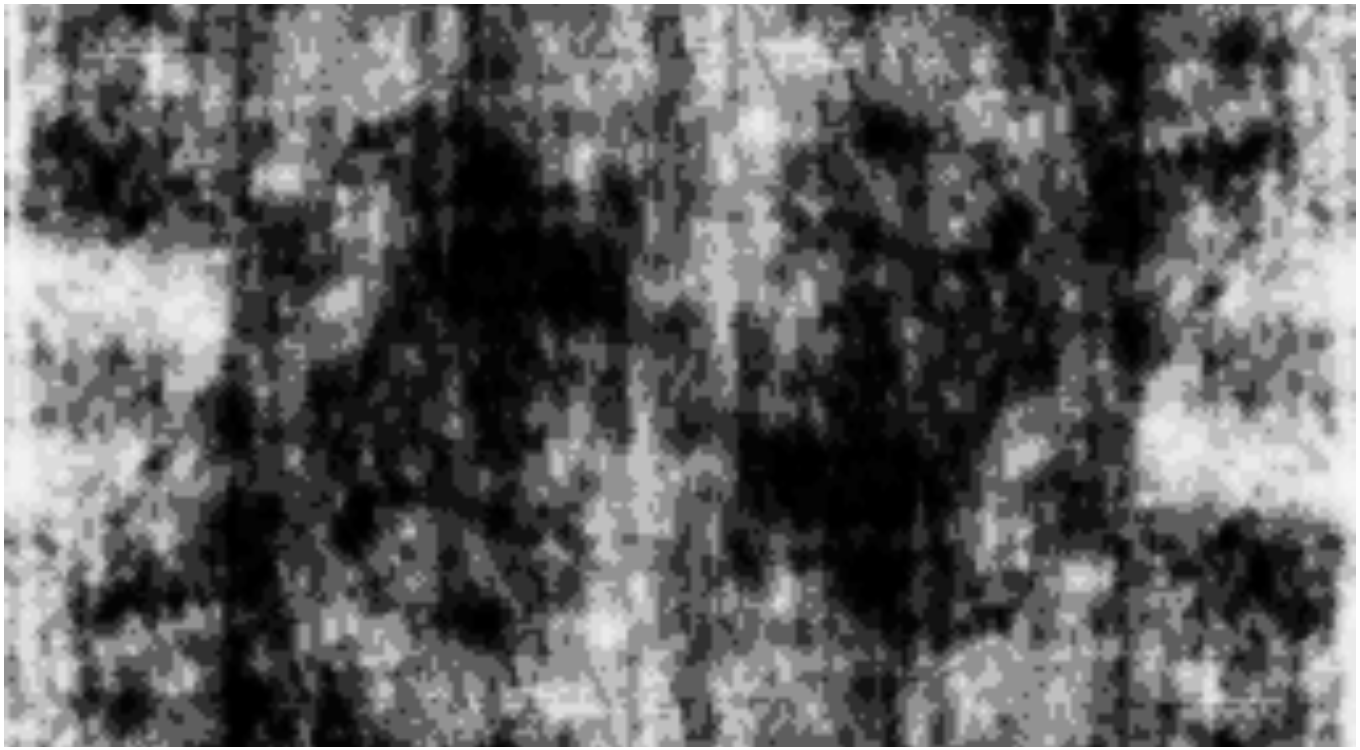
Not much new image, but

What happens if you reverse Fourier without the phase?

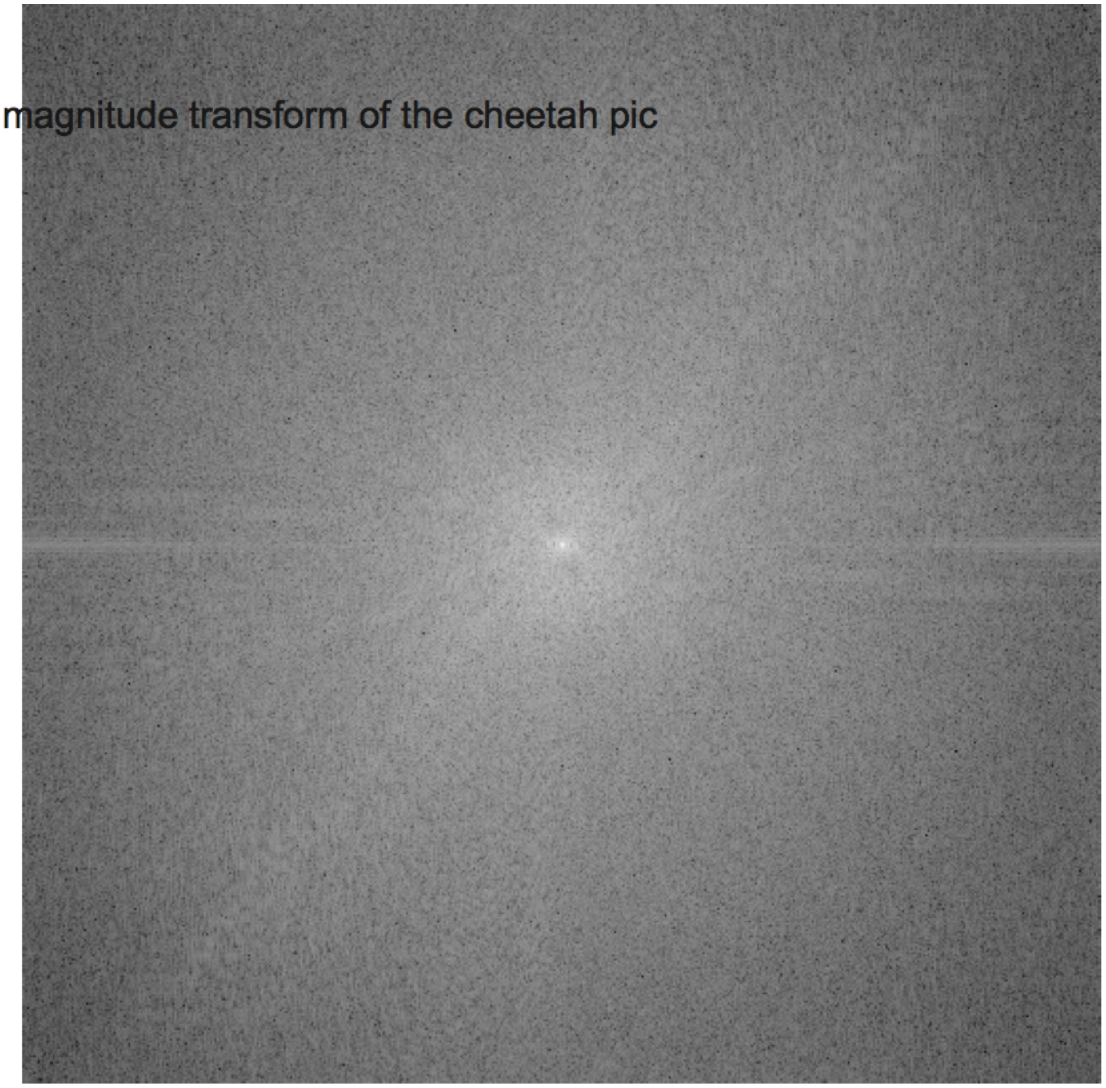
- Open an image
- Compute magnitude and phase of the image
- Reverse the ONLY magnitude back
- Image histogram equalize the reversed FFT image
- Display the image
- What happens?

What happens if you reverse Fourier without the phase?

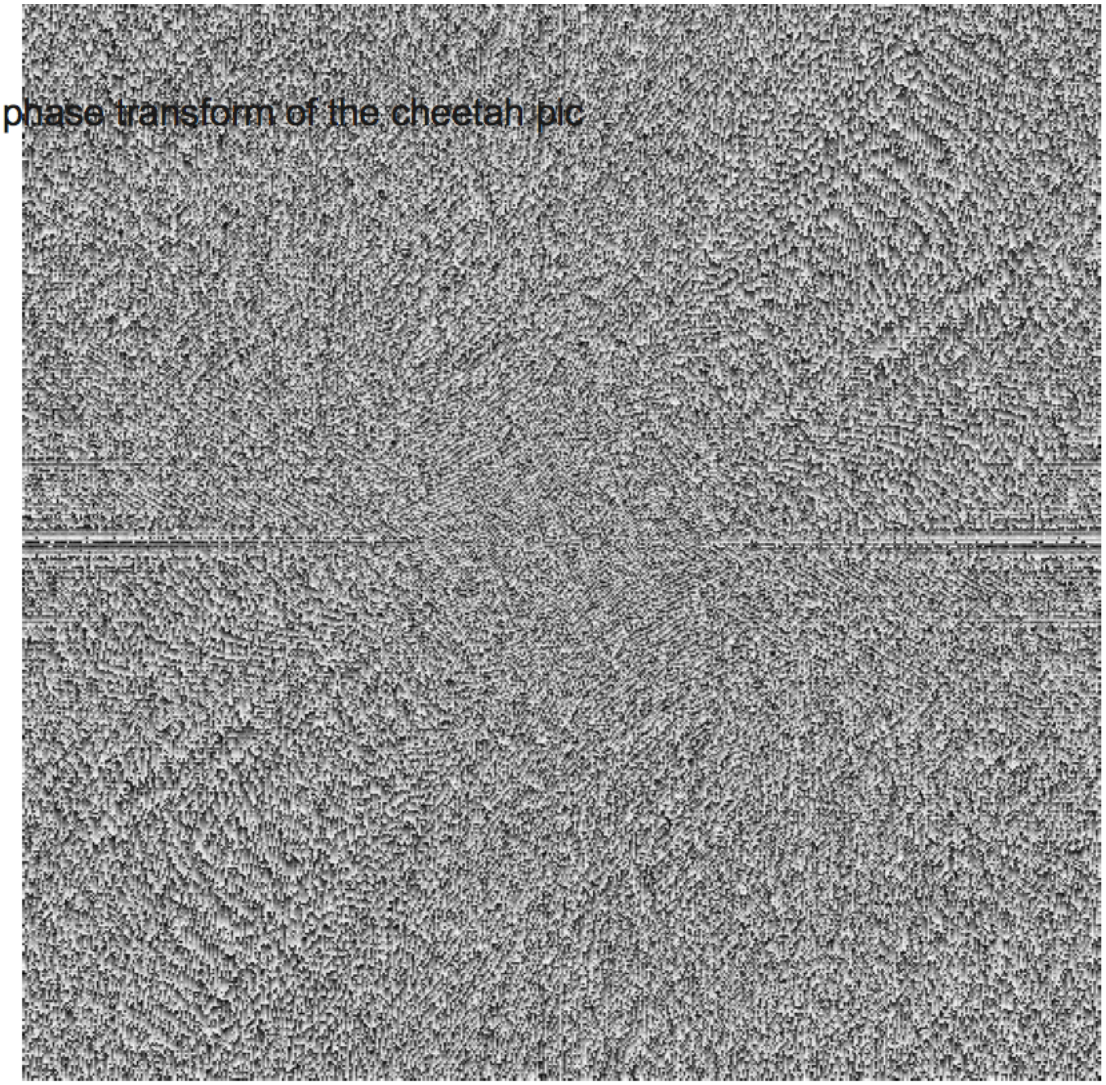
Same frequency, abstract nonsense



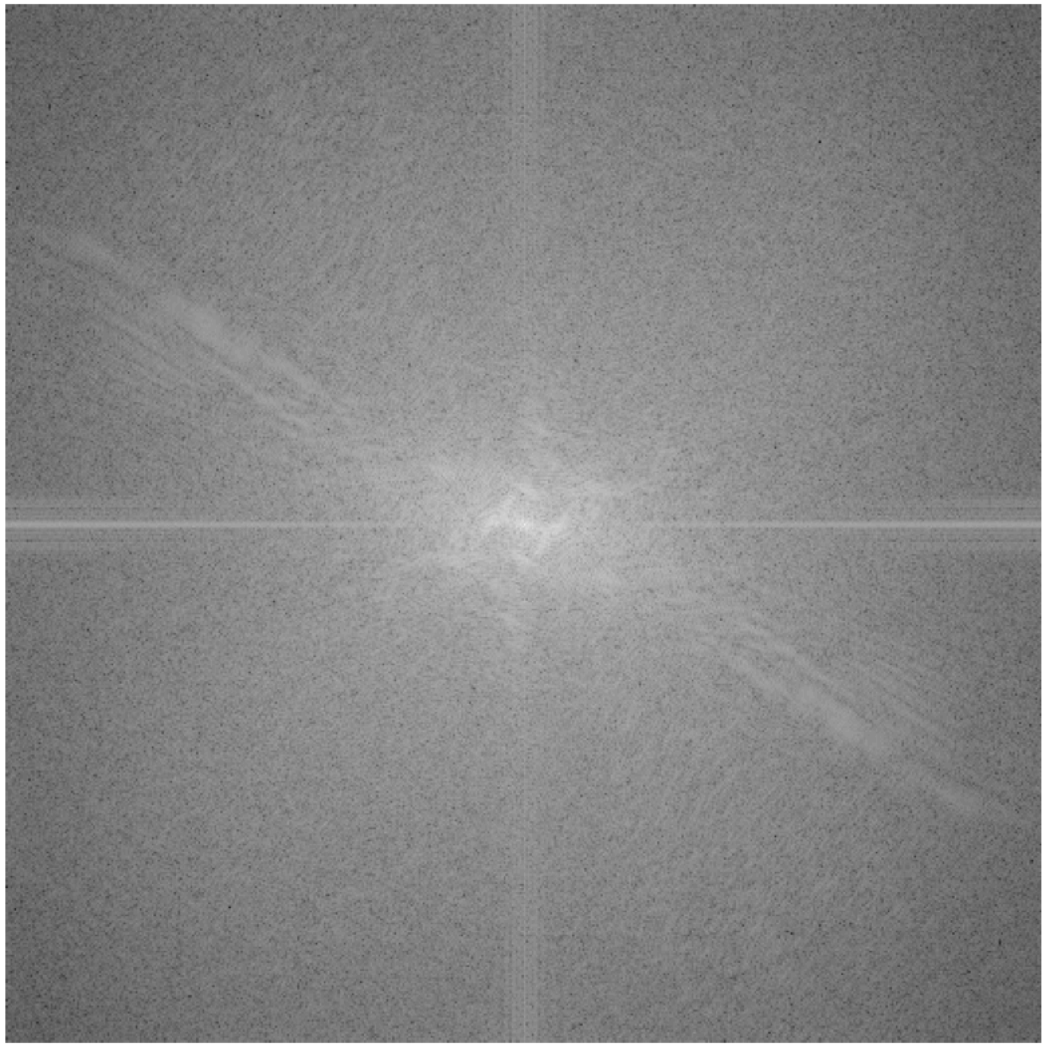
This is the magnitude transform of the cheetah pic



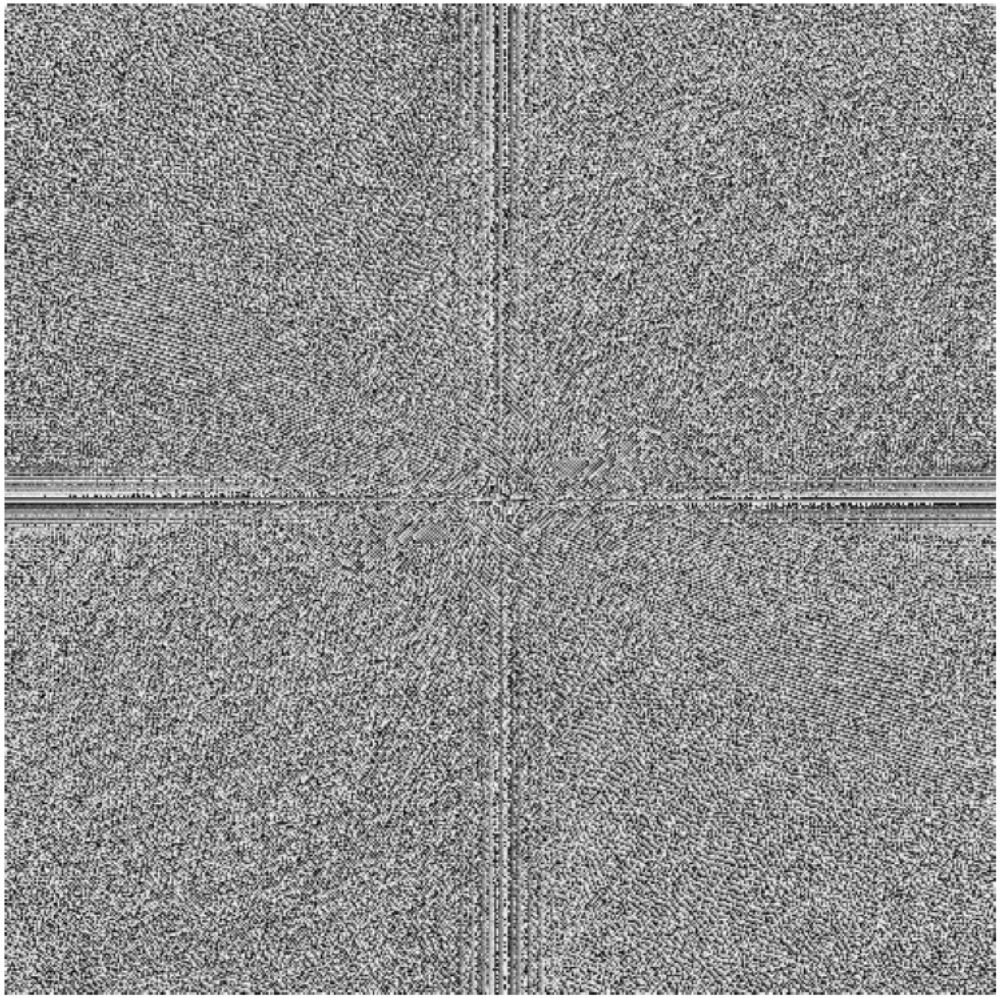
This is the phase transform of the cheetah pic



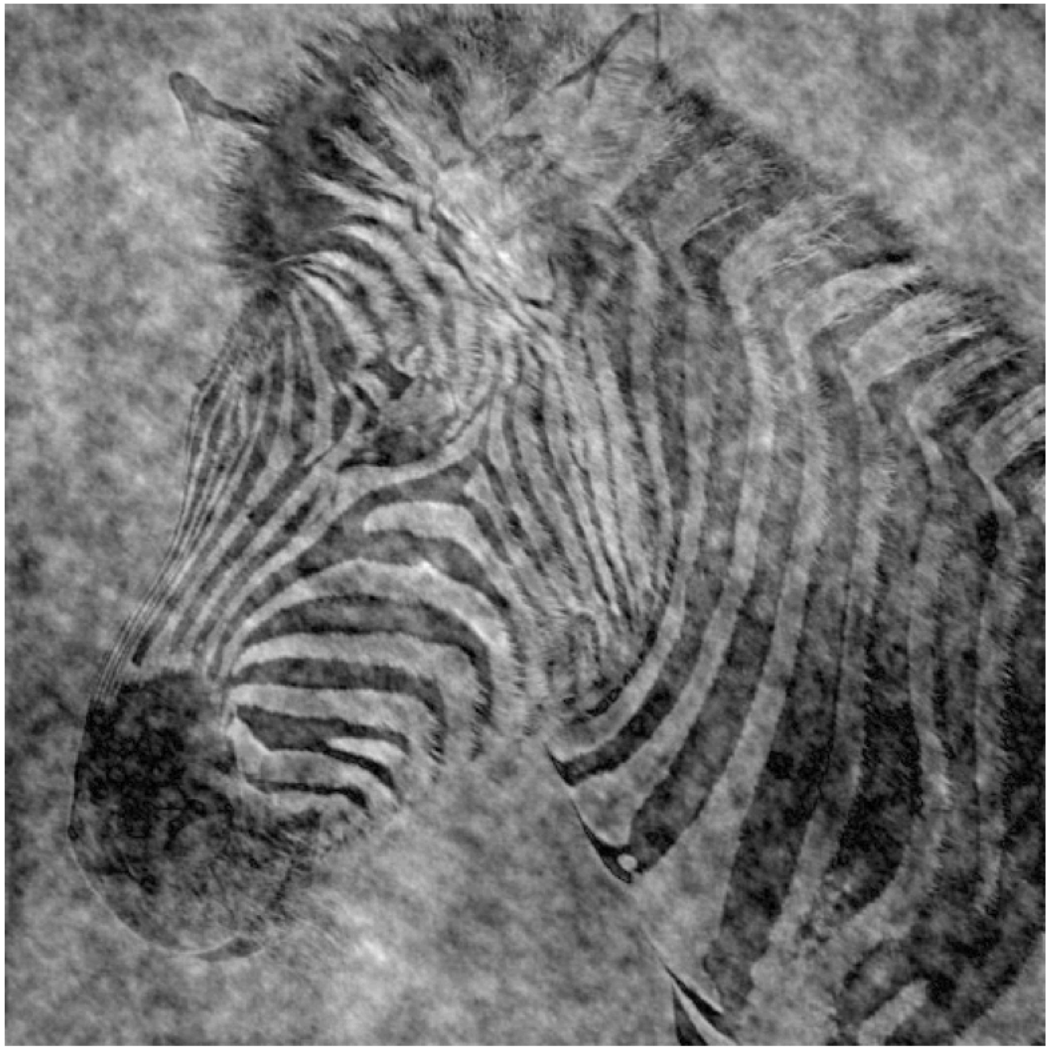
This is the magnitude transform of the zebra pic



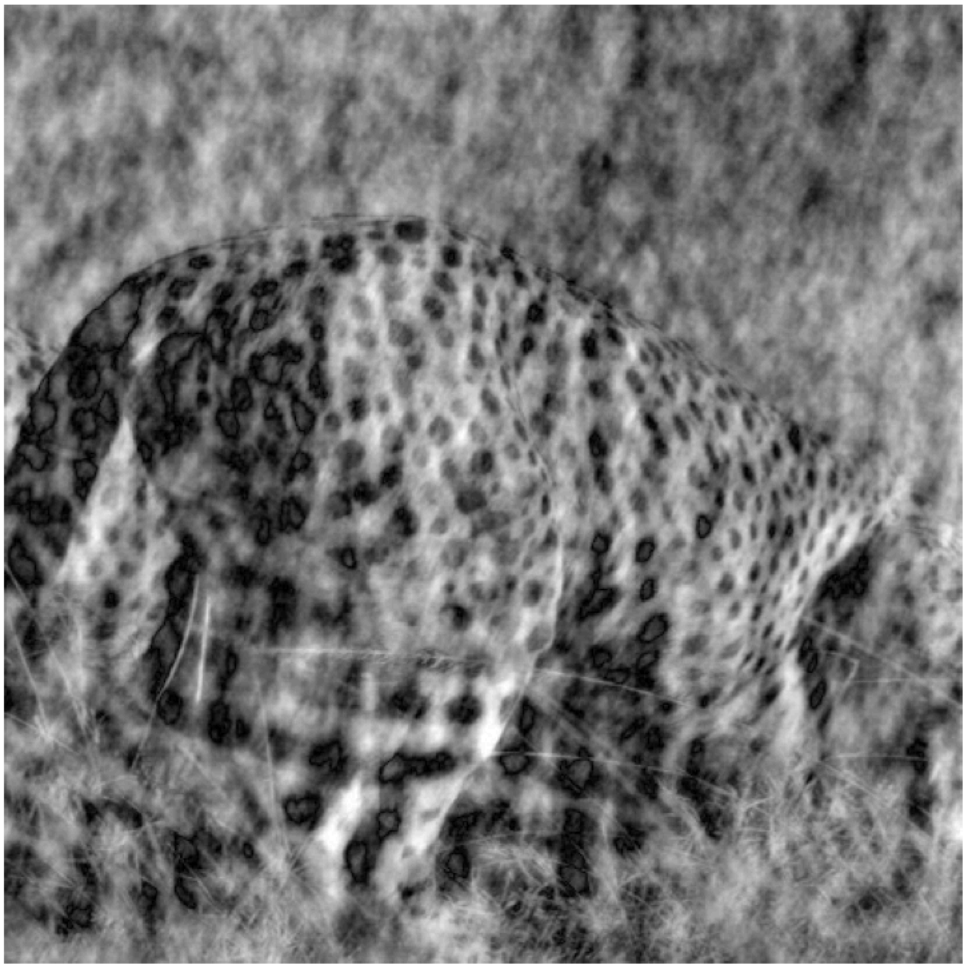
This is the phase transform of the zebra pic



Reconstruction with zebra phase, cheetah magnitude



Reconstruction with cheetah phase, zebra magnitude



The Convolution Theorem

- The Fourier transform of the convolution of two functions is the product of their Fourier transforms

$$\mathrm{F}[g * h] = \mathrm{F}[g]\mathrm{F}[h]$$

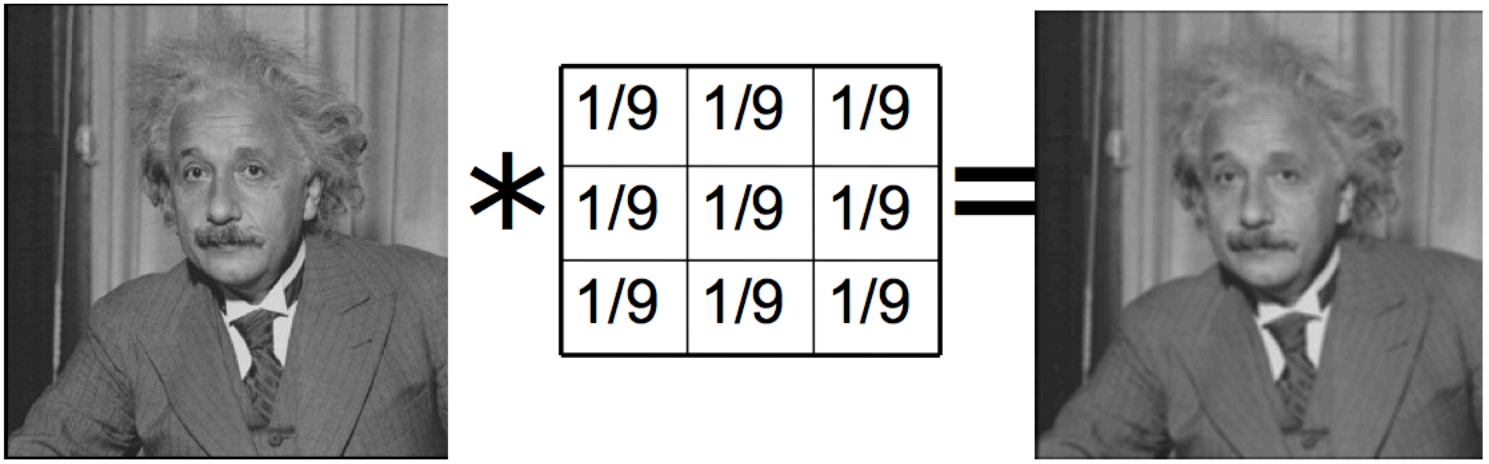
- **Convolution** in spatial domain is equivalent to **multiplication** in frequency domain!

$$g * h = \mathrm{F}^{-1}[\mathrm{F}[g]\mathrm{F}[h]]$$

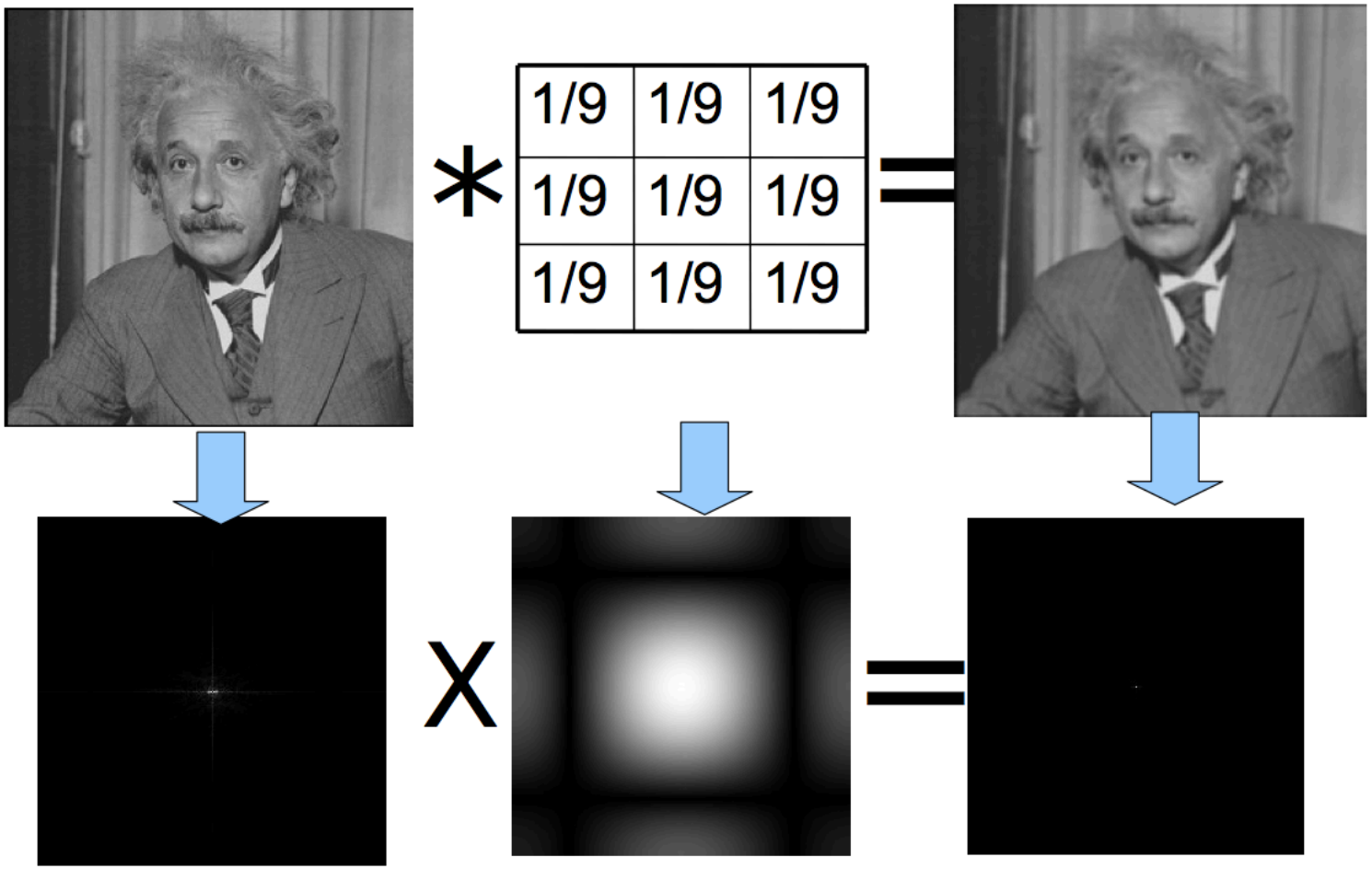
This is extremely handy!

Back to averaging

Remember:



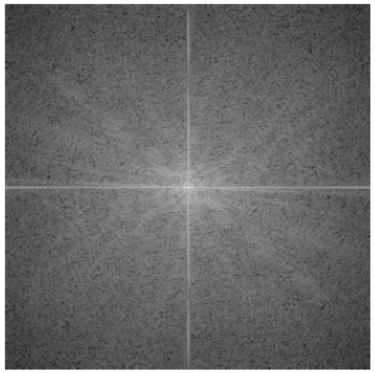
Back to averaging



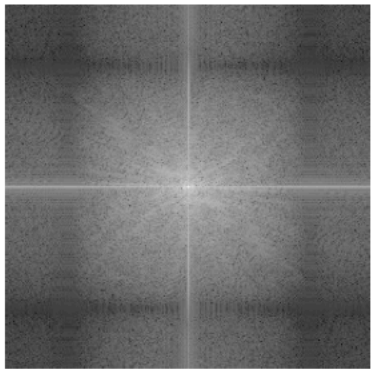
Filtering

- We pixel-wise multiply the DFT (Discrete Fourier Transform) of the input image by the DFT of the filter
- Frequencies where the magnitude of the response of the filter are new zero (black in the images) will be removed.

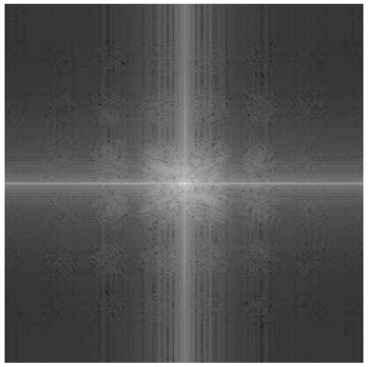
Take the log to rescale brightness



Unfiltered Spectrum

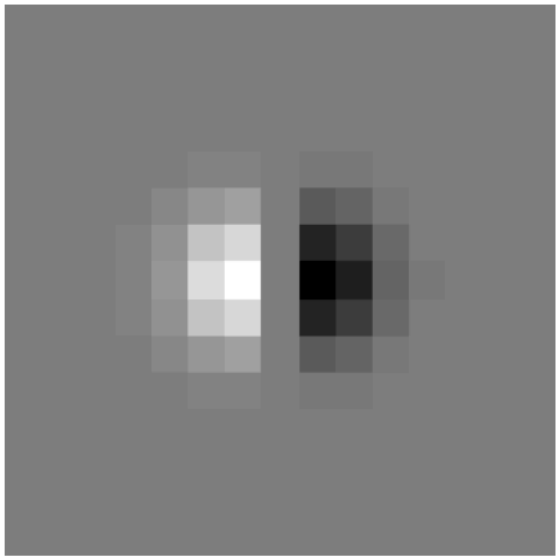


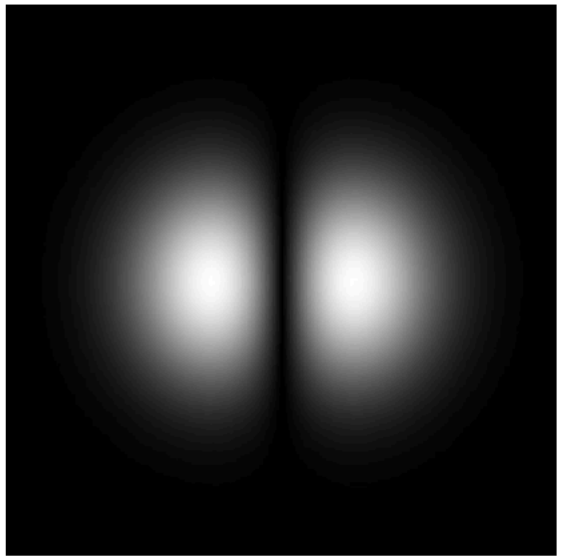
Log Spectrum after 3x3 averaging



Log-Spectrum after 7x7 averaging

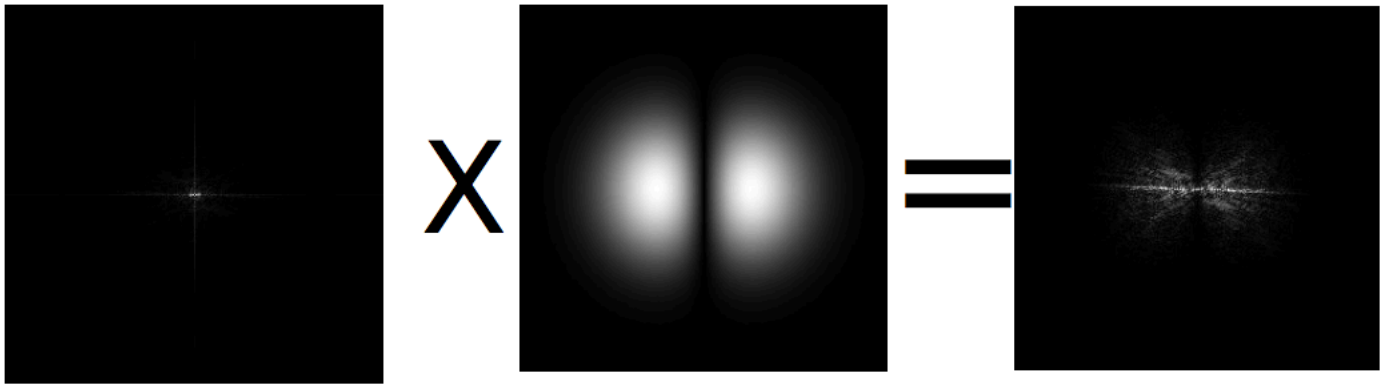
First, the filter





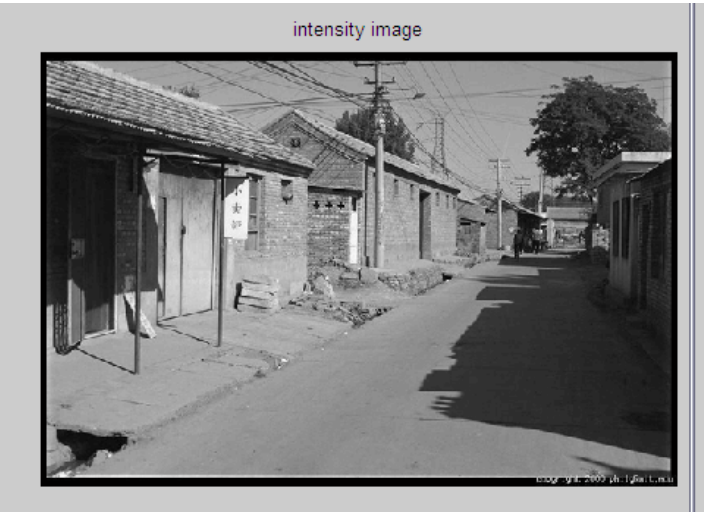
Magnitude of the DFT

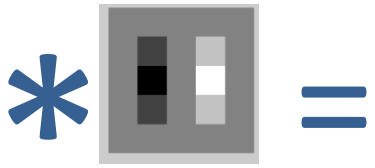
After Filtering

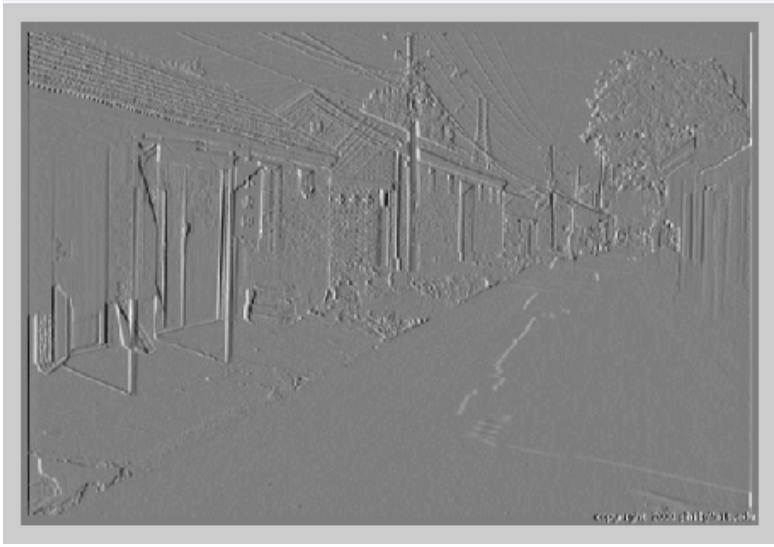


Filtering in spatial dom

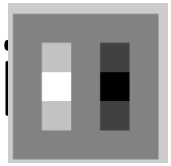
1	0	-1
2	0	-2
1	0	-1



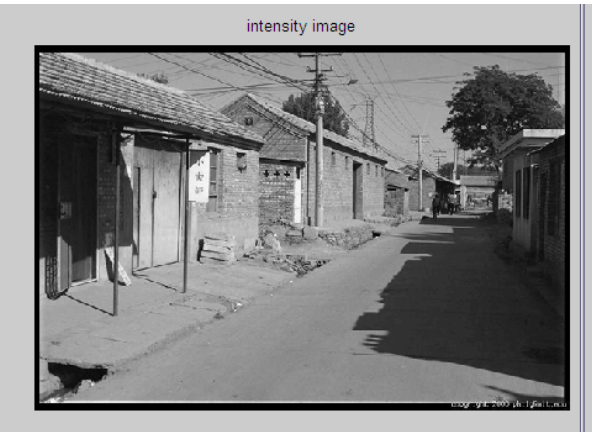




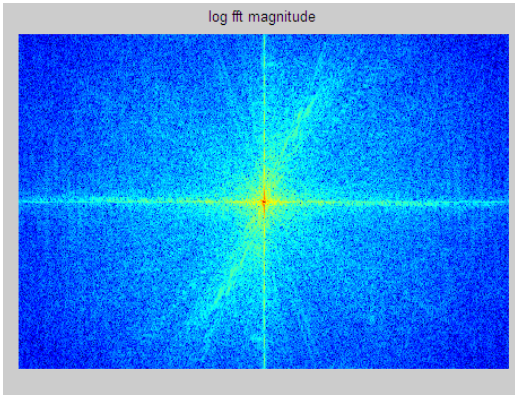
Filtering in frequency domai



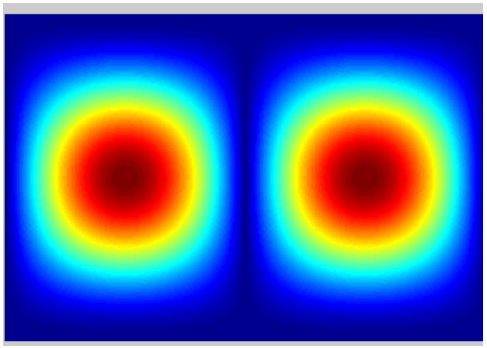
FFT



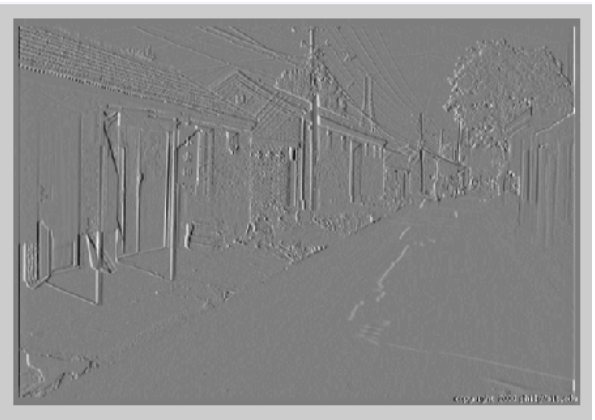
FFT



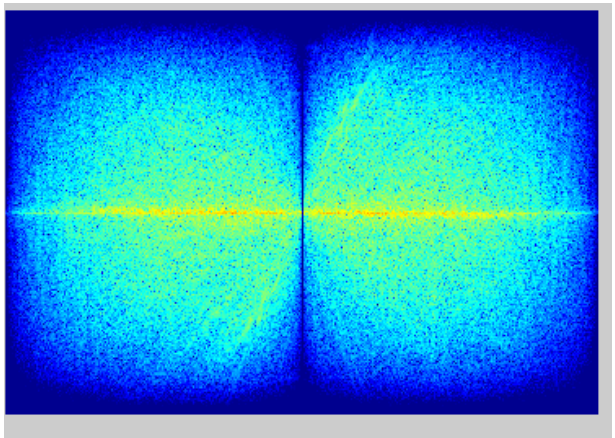
×



=



Inverse FFT

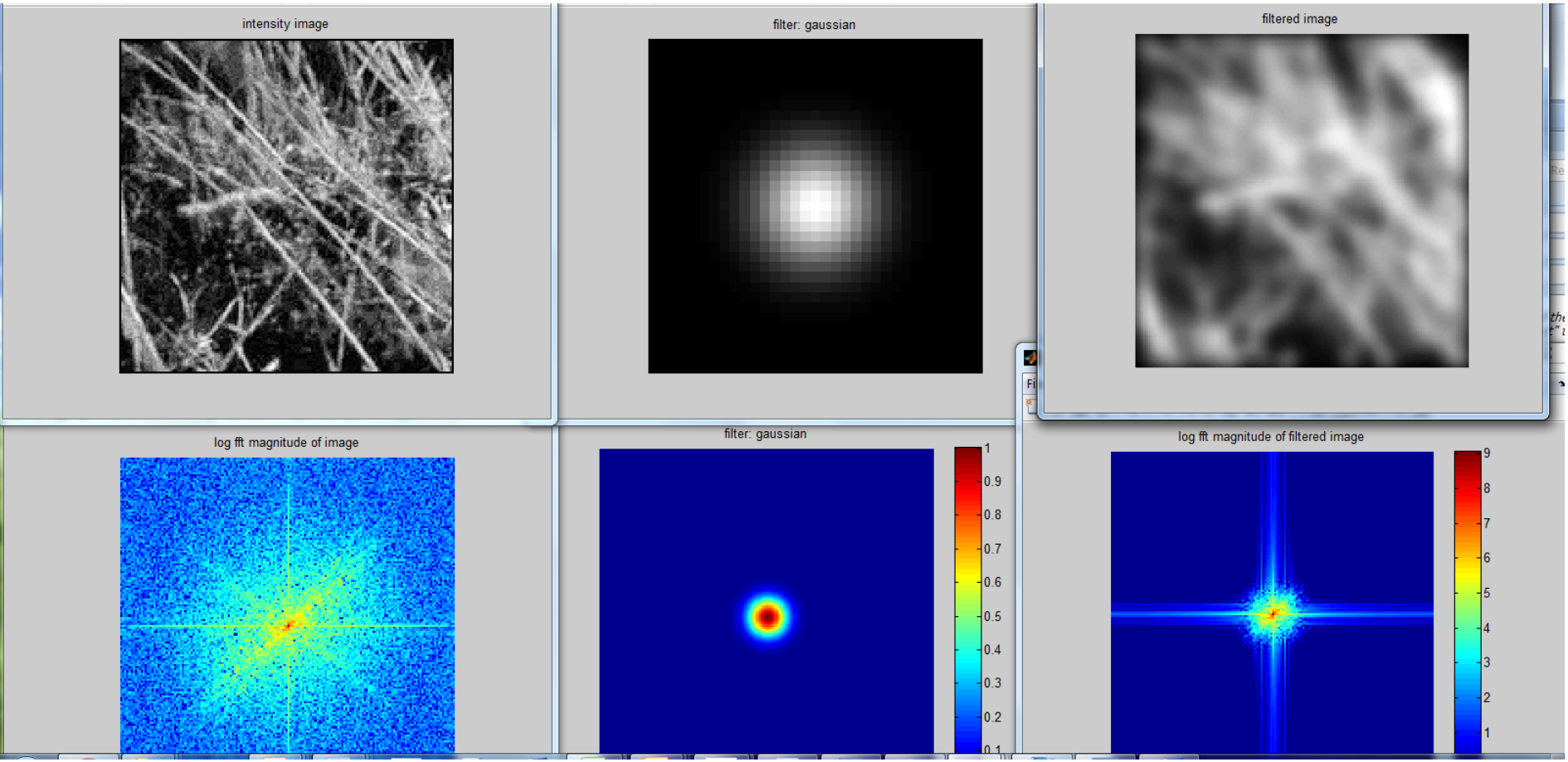


Slide: Hoiem

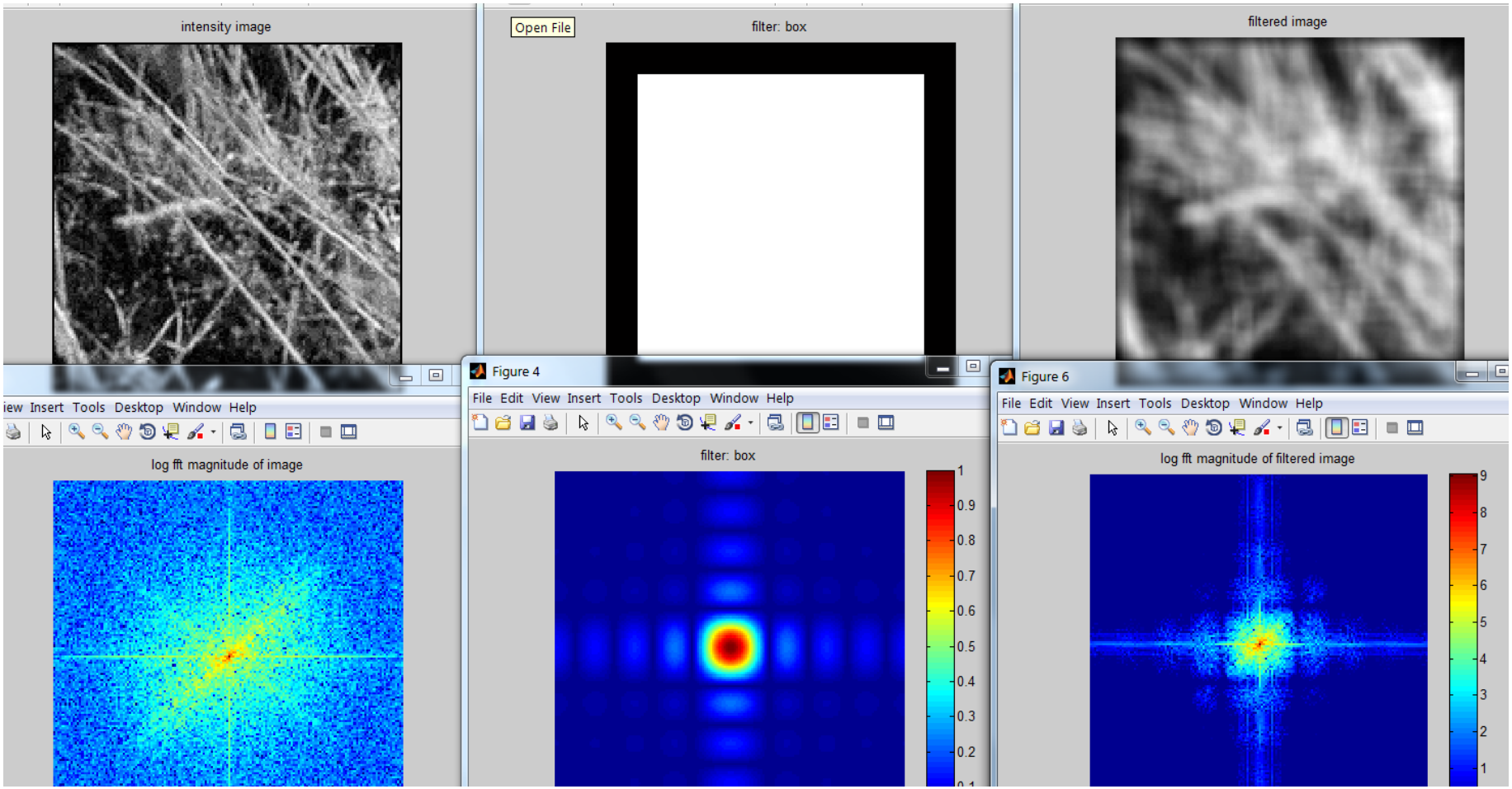
Filtering with FFT in Python

```
img = plt.imread('cheetah.png')

# prepare an 1-D Gaussian convolution kernel
t = np.linspace(-10, 10, 30)
bump = np.exp(-0.1*t**2)
bump /= np.trapz(bump) # normalize the integral to 1
# make a 2-D kernel out of it
kernel = bump[:,np.newaxis] * bump[np.newaxis,:]

# padded fourier transform, with the same shape as the image
kernel_ft = np.fft.fft2(kernel, s=img.shape[:2], axes=(0, 1))

# convolve
img_ft = np.fft.fft2(img, axes=(0, 1))
img2_ft = kernel_ft[:,:,np.newaxis] * img_ft
img2 = np.fft.ifft2(img2_ft, axes=(0, 1)).real

# clip values to range
img2 = np.clip(img2, 0, 1)


# plot output
plt.imshow(img2)
```

Gaussian


intensity image
filter: gaussian
filtered image
log fft magnitude of image
filter: gaussian
log fft magnitude of filtered image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
9
8
7
6
5
4
3
2
1

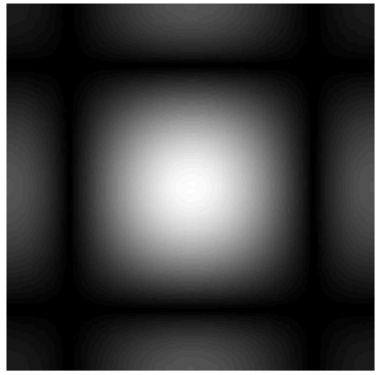
Box Filter


intensity image
Open File
filter: box
filtered image
Figure 4
File Edit View Insert Tools Desktop Window Help
filter: box
Figure 6
File Edit View Insert Tools Desktop Window Help
log fft magnitude of filtered image
log fft magnitude of image

Vocabulary

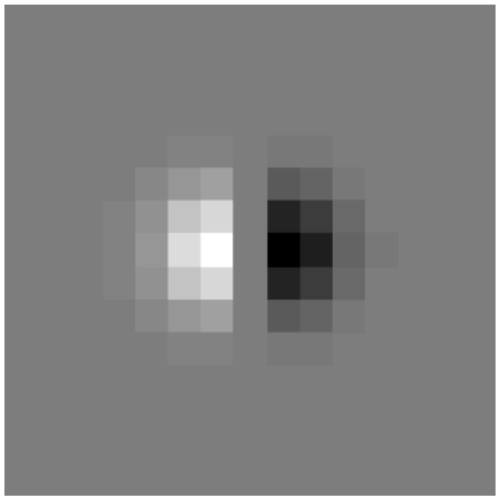
Low Pass Filter:

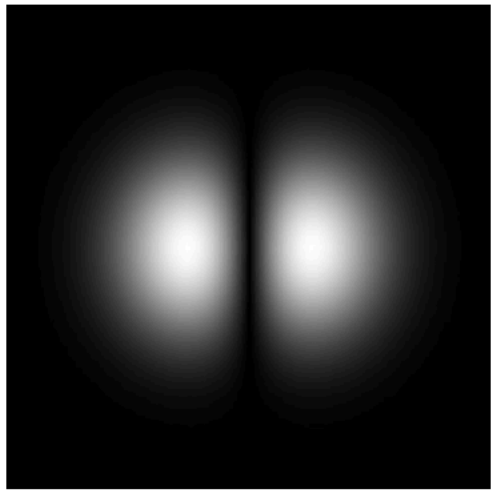
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



Vocabulary

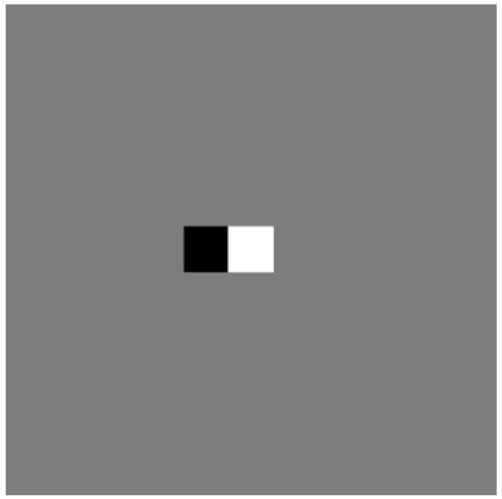
Band-pass Filter:

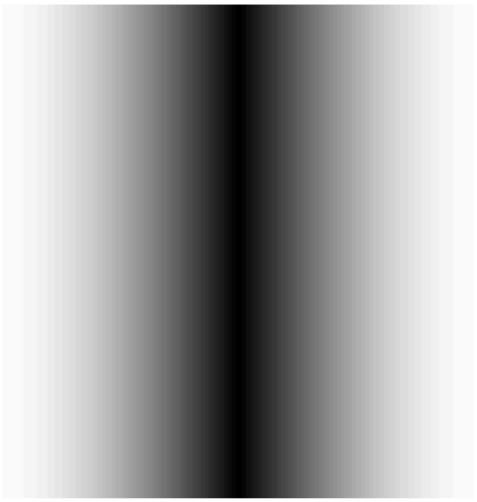




Vocabulary

High-pass Filter:





Exercise 1

- Download the FFTAnalysis.py from blackboard.
- Use the einstein.png
- Right now the image removes low frequency (center) of the images.
- Can you modify the code that you are removing the high frequency (low pass) the image? You can do this either by directly modifying the DFT or use a filter.

Exercise 2

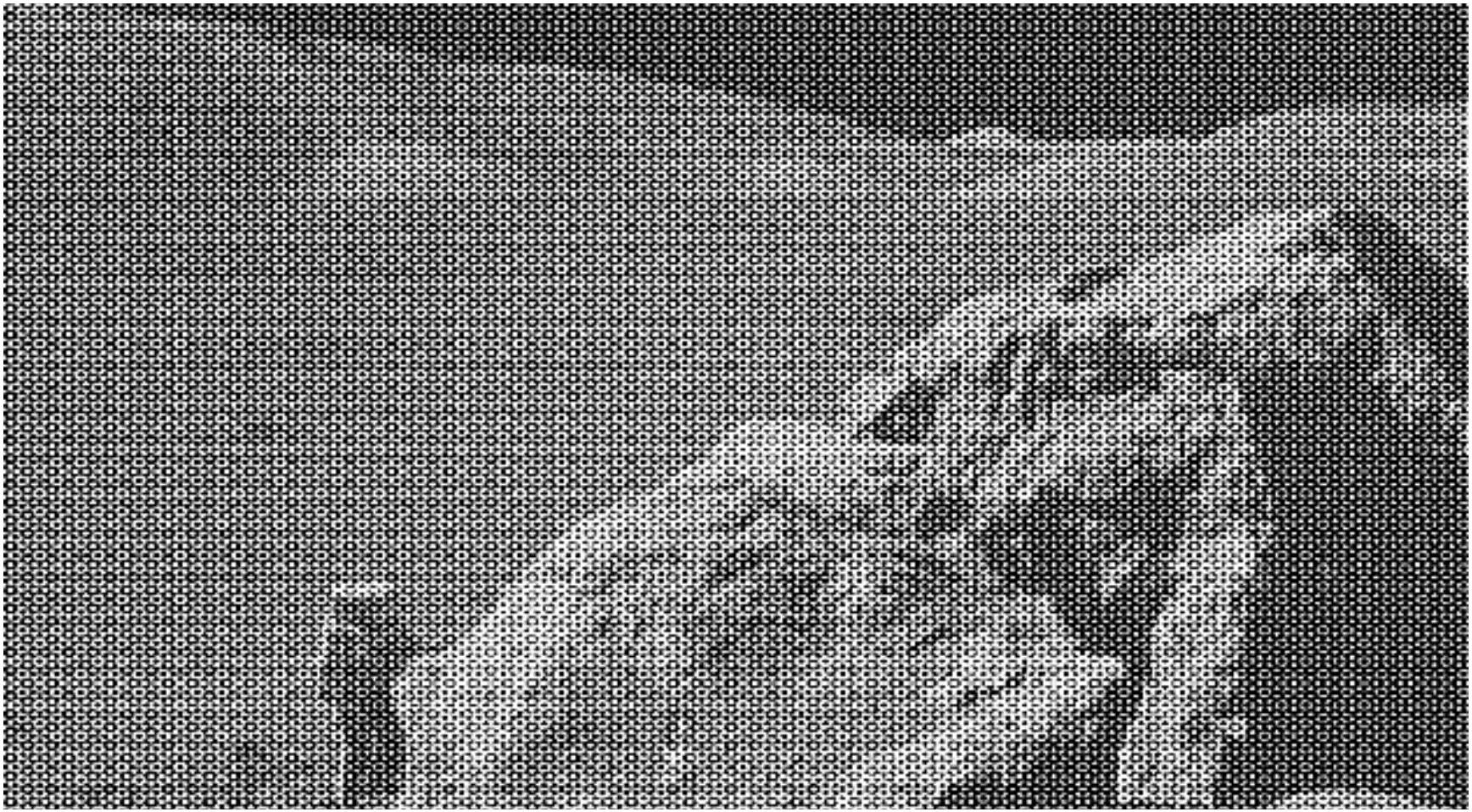
- Download the Cheetha and Zebra images

- DFT both images in Fourier domain and compute magnitude and phase. I have already did this in the helper code PhaseandMagnitude.py

- You can compute phase and magnitude as this:
- magnitude_zebra = 30*np.log(np.abs(fshift))
- phase_zebra = np.angle(fshift)

- Reconstruct the image with Cheetha phase and Zebra magnitude and vice versa. You have to do this yourself!

Exercise 3: Simple Image DE nosing



Application: Simple Image DE nosing

Examine the provided image moonlanding.png, which is heavily contaminated with periodic noise. In this exercise, we aim to clean up the noise using the Fast Fourier Transform.

- 1. load in the image using plt.imread() or scipy.misc()
- 2. Use the 2DFFT in numpy.fft and plot the spectrum of the image.
- 3. The spectrum consists f high and low frequency components. The noise is contained in the high frequency part of the spectrum, so set some of those components to zero.
- 4. Apply the inverse FT to see the resulting image

Python demo: FFT of gratings

- We want to create gratings in Python
- Create two sine gratings that has different special frequency and orientation
- Compute FFT of the gratings
- Sum them up.