LECTURE 05
IMAGE RESTORATION

Punnarai Siricharoen, Ph.D.

OBJECTIVES

- To understand different types of image degradations
- To understand and be able to apply image restoration techniques for diffferent types of degradations properly

CONTENT

- Image degradations
- Image restoration

IMAGE RESTORATION

Concerns the removal or reduction of degradations that have occurred during the acquisition of the image.

Degradations: noise (errors in pixel values), optical effects (out-of-focus) blurring or blurring due to camera motion.

Neighborhood operations or **frequency domain**

IMAGE RESTORATION

- Image restoration attempts to reconstruct or recover an image that has been degraded by using prior knowledge of the degration phenomenon.
- Image degradation / restoration process is modeled as degradation function h(x,y) that, together with an additive noise term $\eta(x, y)$, operates on an input image f(x,y). The degraded image g(x,y) in spatial domain is given by

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y)$$

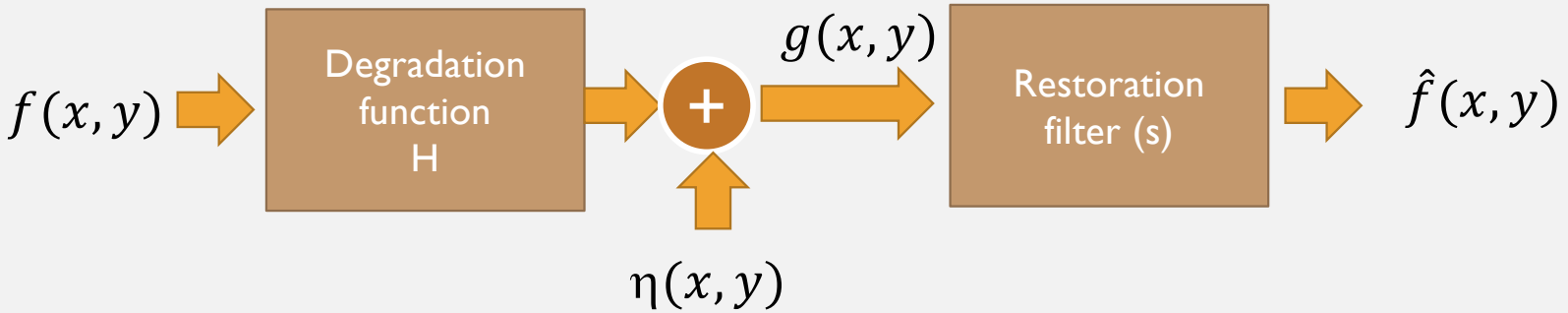
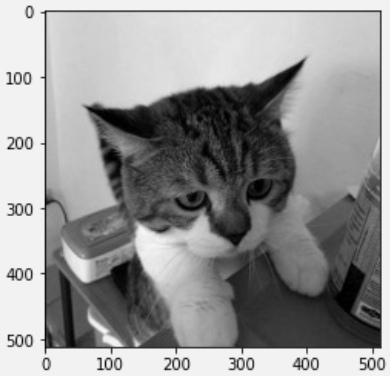


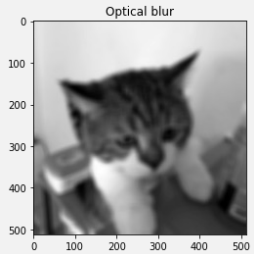
IMAGE DEGRADATIONS

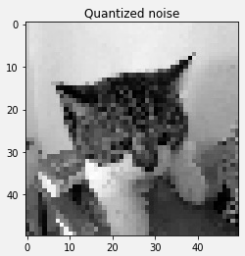
- The * convolution in spatial domain is equaivalent to multiplication in frequency domain representation below,

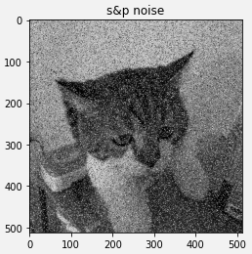
$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

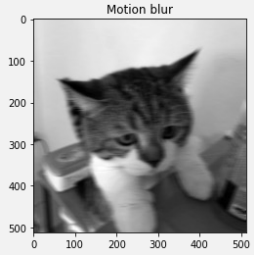
- Degradations:

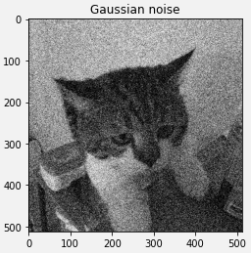


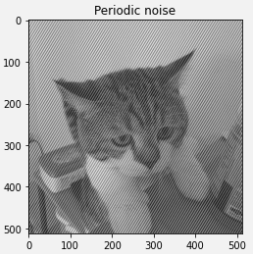












NOISE

- Considering a noisy image g(x,y) formed by the addition of noise η(x,y) to an original image:

$$g(x,y) = f(x,y) + \eta(x,y)$$

- Noise: any degradation caused by external disturbance. E.g., image sent from one place to another via Satellite or wireless transmission, network cables
- Cleaning an image for image restoration.

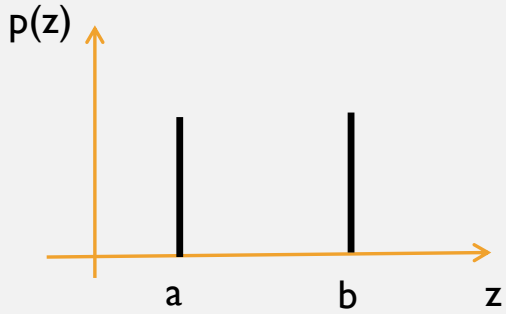
NOISE

- Various types of noise and how they appear on an image:
  - Salt and Pepper noise
  - Gaussian noise
  - Speckle noise
  - Periodic noise

1 NOISE: SALT & PEPPER

- Impulse noise, shot noise or binary noise.
- Resemble salt-and-pepper granules randomly distributed over the image.
- Probability density function (PDF) of the impulse noise is given by

$$p(z) = \begin{cases} P_a & for\ z = a \\ P_b & for\ z = b \\ 0 & otherwise \end{cases}$$



where z – gray level, if b>a b will appear as a light dot in the image, a is a dark dot.
a and b are saturated values.

1 NOISE: SALT & PEPPER

- During image digitizing process, impulse corruption is usually large compared with the strength of the image signal. So it is digitized to black or white values in an image.
- **Impulse noise** is found in situations where quick transients such as faulty switching take place during imaging.
- What do you think what is a and b could be for 8 bit image?

1 NOISE: SALT & PEPPER

- Python: 1) generate a grayscale image from a color image.
- 2) To add noise use `skimage.util import random_noise` as follows:

```
import cv2
from skimage.util import random_noise

img = cv2.imread("twins.jpg",0)

sp = random_noise(img, mode='s&p')
sp2 = random_noise(img, mode='s&p', amount = 0.2)
```

where amount is the noise density (default:0.05)

amount : float, optional

Proportion of image pixels to replace with noise on range [0, 1]. Used in 'salt', 'pepper', and 'salt & pepper'. Default : 0.05

https://scikit-image.org/docs/dev/api/skimage.util.html#skimage.util.random_noise

random_noise

`skimage.util.` `random_noise`(*image*, *mode='gaussian'*, *seed=None*, *clip=True*, ***kwargs*) [source]

Function to add random noise of various types to a floating-point image.

Parameters

image : ndarray

Input image data. Will be converted to float.

mode : str, optional

One of the following strings, selecting the type of noise to add:

- 'gaussian' Gaussian-distributed additive noise.
- **'localvar' Gaussian-distributed additive noise, with specified** local variance at each point of *image*.
- 'poisson' Poisson-distributed noise generated from the data.
- 'salt' Replaces random pixels with 1.
- **'pepper' Replaces random pixels with 0 (for unsigned images) or** -1 (for signed images).
- **'s&p' Replaces random pixels with either 1 or** *low_val*, **where** *low_val* is 0 for unsigned images or -1 for signed images.
- **'speckle' Multiplicative noise using out = image + n*image, where** n is uniform noise with specified mean & variance.

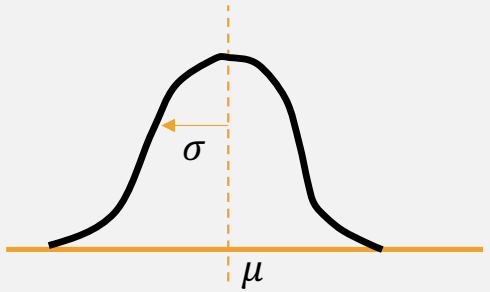
2 NOISE: SALT & PEPPER

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (16,8))
fig.add_subplot(1,3,1), plt.imshow(img, cmap="gray"), plt.title('Original')
fig.add_subplot(1,3,2), plt.imshow(sp, cmap="gray"), plt.title('Salt & Pepper')
fig.add_subplot(1,3,3), plt.imshow(sp2, cmap="gray"), plt.title('Salt & Pepper (noise
density = 0.2)')
plt.show()
```

2 NOISE: GAUSSIAN NOISE

- Gaussian also called, 'Normal'.
- Idealized form of white noise which is caused by random fluctuations of the signal.
- Noise is normally distributed.

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\mu)^2/2\sigma^2}$$



- z represents gray level, μ is the mean of z and σ is its standard deviation.

2 NOISE: GAUSSIAN NOISE

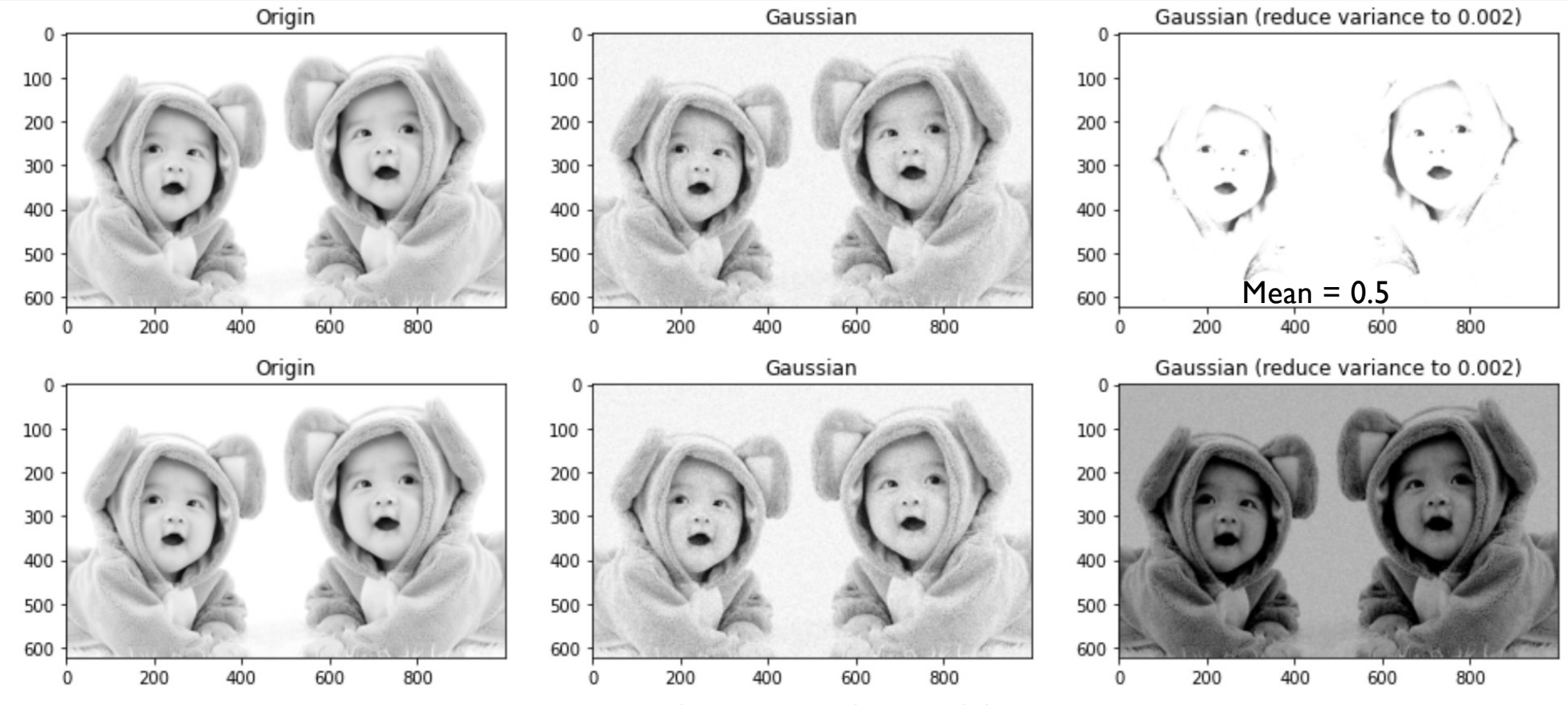
- Adds Gaussian white noise of mean m and variance v to the image img. The default is zero mean noise with 0.01 variance

```
gauss2 = random_noise(img, mode='gaussian',var = VAR , mean=MEAN)
```

```
gauss = random_noise(img, mode='gaussian')
gauss2 = random_noise(img, mode='gaussian',var = 0.002, mean=0.5)

import matplotlib.pyplot as plt
fig = plt.figure(figsize = (16,8))
fig.add_subplot(1,3,1), plt.imshow(img, cmap="gray"), plt.title('Origin')
fig.add_subplot(1,3,2), plt.imshow(gauss, cmap="gray"), plt.title('Gaussian')
fig.add_subplot(1,3,3), plt.imshow(gauss2, cmap="gray"),
plt.title('Gaussian (reduce variance to 0.002)')
plt.show()
```

2

NOISE: GAUSSIAN NOISE



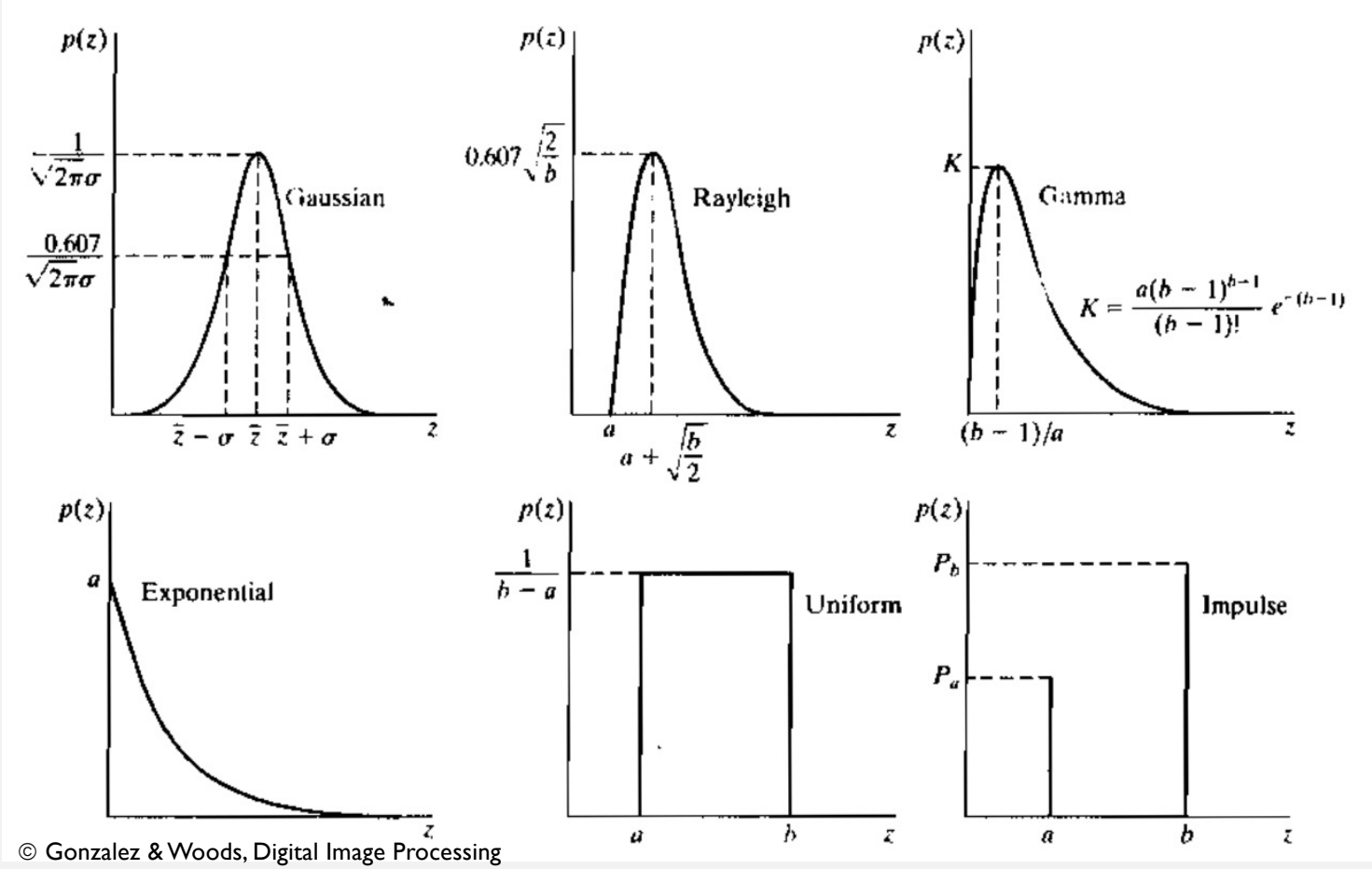
Making to mean of the noise to the dark values

Default mean = 0, var = 0.01

Mean = -0.5

NOISE: PROBABILITY DENSITY FUNCTIONS FOR NOISE

- Gaussian noise
  - Most frequently in practice
  - Electronic circuit noise
  - Sensor noise from poor illumination
- Rayleigh noise
  - Medical imaging, such as X-rays & MRI
- Exponential & Gamma
  - Laser imaging
- Exponential – PET/SPECT
- Uniform – dust in a camera lens



© Gonzalez & Woods, Digital Image Processing

3 NOISE: SPECKLE NOISE

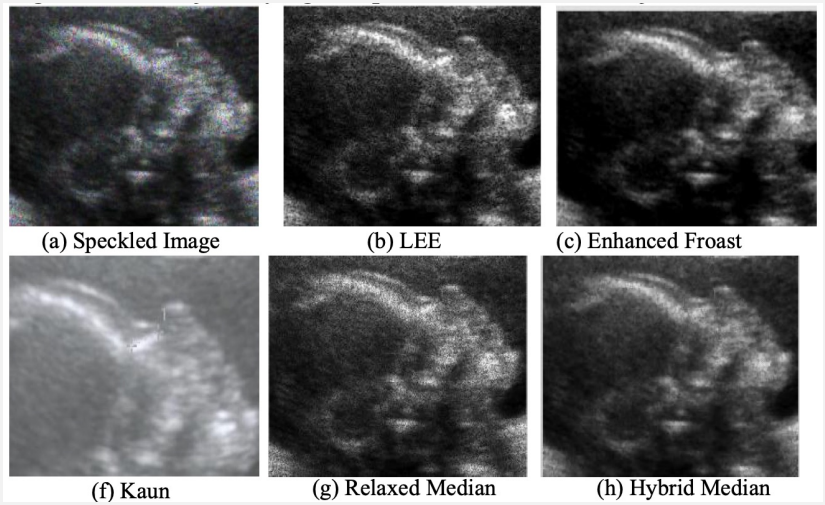
- Can be modeled by random values multiplied by pixel values -> **multiplicative noise.**

$$g(x, y) = f(x, y) * n(x, y)$$

- $g(x, y)$ – distorted image, $f(x, y)$ – original image, $n(x, y)$ – speckle noise
- The major problem in some laser, radar applications and ultrasound imaging
  - Ultrasound imaging – granular patterns in an image due to wave interference by adding up or canceling
  - Synthetic Aperture Radar (SAR) - radar waves hit the surface, they scatter in different directions and results in a granular patterns

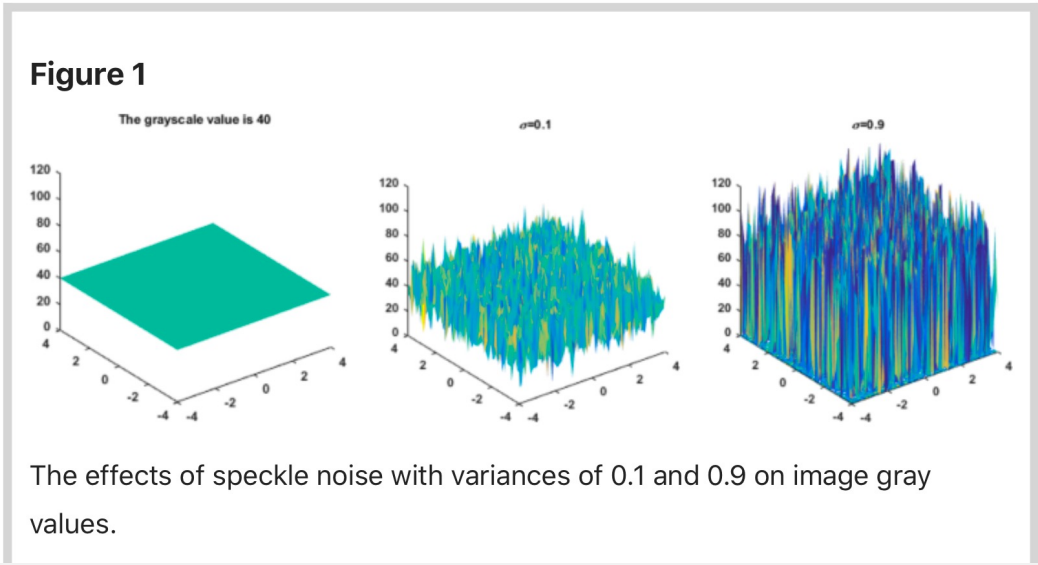
3 NOISE: SPECKLE NOISE

- The influence of the speckle noise, the effects of speckle noise with variances of 0.1 and 0.9 on an image is shown below. The larger the variance of the speckle noise is, the more difficult it is to recover [Ren et al., 2019]



Denoising a speckled ultrasound image

B.Priestly Shan and M.Madheswaran, A Generalized Despeckling Filter for Enhancing Fetal Structures to Aid Automated Obstetric Pathologies

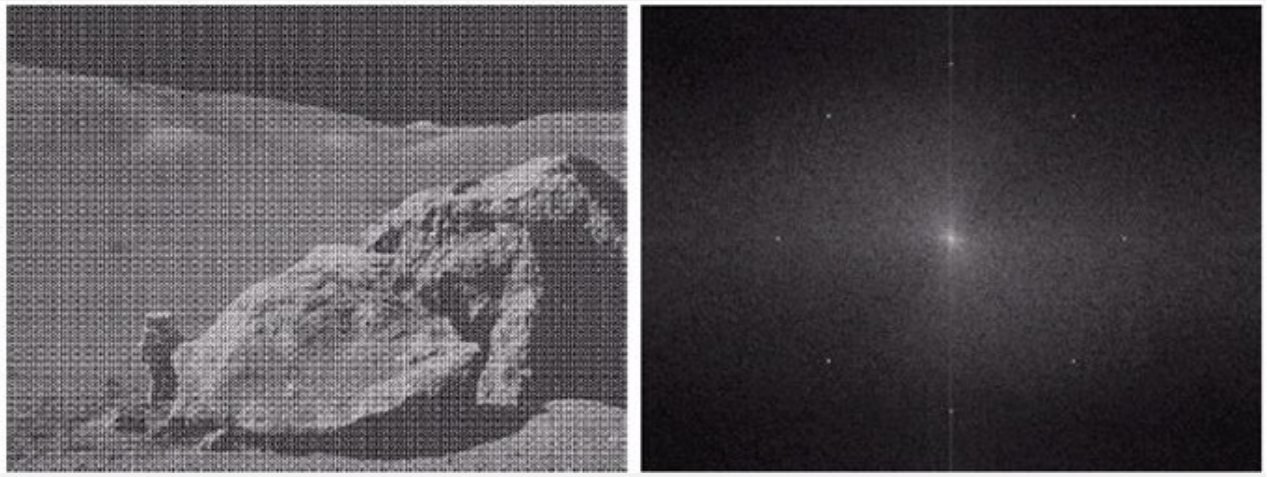


The effects of speckle noise with variances of 0.1 and 0.9 on image gray values.

https://www.poroussilicon.com/uploads/2/9/7/0/29708759/9541123_orig.png

4 NOISE: PERIODIC NOISE

- **Periodic Noise –** <u>electrical</u> or <u>electromagnetical</u> interference during image acquisition.
- Can be reduced significantly via frequency domain filtering.



a b

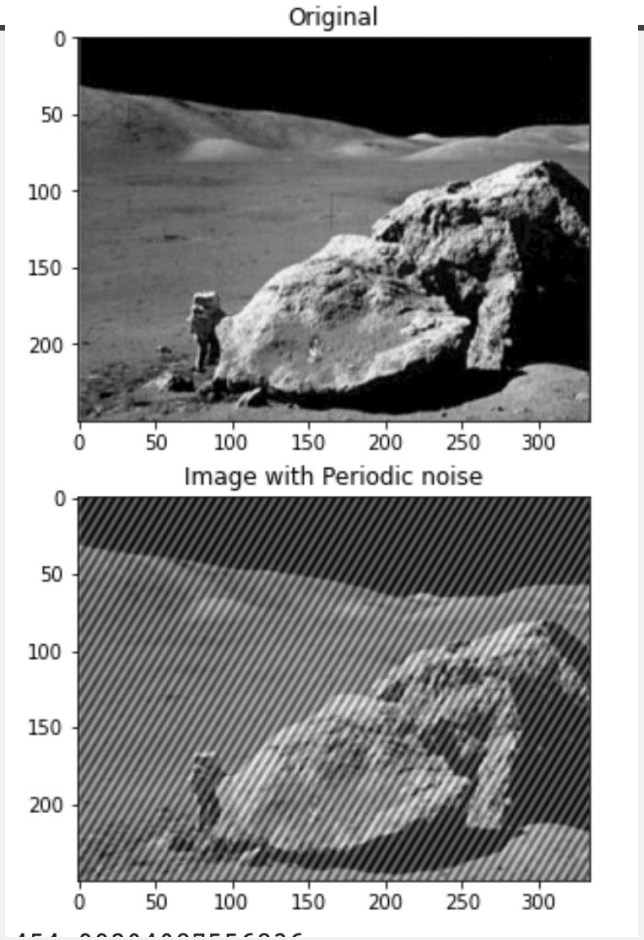
FIGURE 5.5 (a) Image corrupted by sinusoidal noise. (b) Spectrum (each pair of conjugate impulses corresponds to one sine wave). (Original image courtesy of NASA.)

- If amplitude strong enough, will see spectrum of each sine wave.

© Gonzalez & Woods, Digital Image Processing

4 NOISE: PERIODIC NOISE

```
img = cv2.imread("nasa_image.png",0)


# strength
s = 100

nx, ny = img.shape
y = np.linspace(-nx/2,nx/2,nx)
x = np.linspace(-ny/2,ny/2,ny)
xv, yv = np.meshgrid(x,y)
p = s*(np.sin(np.pi*xv/3+np.pi*yv/5)+1.0)
print(img.shape)


img_pn = img.astype(float)+p
```



4 EXERCISE #1 NOISE: PERIODIC NOISE

- **Exercise #1**: use Fourier transform to investigate the "nasa_image.png" in frequency domain.

METHODS FOR RESTORATIONS

IMAGE RESTORATION

- Considering a noisy image g(x,y) formed by the addition of noise η(x,y) to an original image:

$$g(x,y) = f(x,y) + \eta(x,y)$$

- Frequency domain:

$$G(u,v) = F(u,v) + N(u,v)$$

METHODS FOR RESTORATION

- Low-pass filtering/Averaging filtering
  - Cleaning Salt & Pepper and Gaussian filtering.
- Median filtering
  - Cleaning Salt & pepper noise.
- Removal of periodic noise.

1 LOW-PASS FILTERING/ AVERAGING FILTERING

Averaging filters

ω_1	ω_2	ω_3
ω_4	ω_5	ω_6
ω_7	ω_8	ω_9

$\frac{1}{9}$ x

1	1	1
1	1	1
1	1	1

$\frac{1}{16}$ x

1	2	1
2	4	2
1	2	1

Mask coefficients

	z_1	z_2	z_3
	z_4	z_5	z_6
	z_7	z_8	z_9

Image pixels

$$R = \sum_{i=1}^{mn} \omega_i z_i \qquad R = \frac{1}{9} \sum_{i=1}^{9} z_i$$

which is the average of the gray levels of the pixels in the 3 x 3 neighborhood

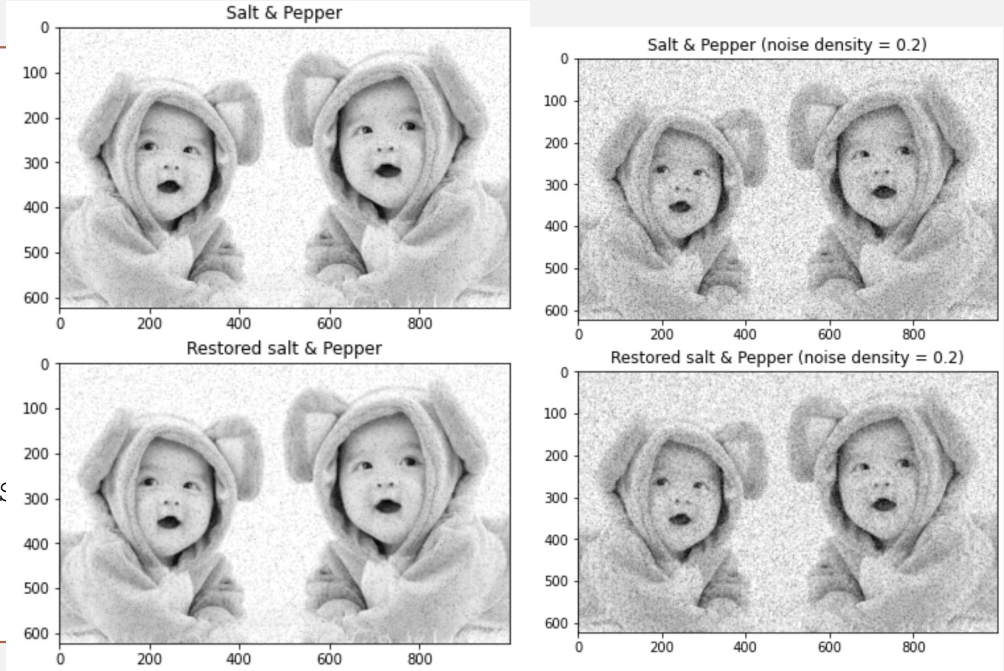
-A spatial averaging filter -> **box filter**

-the left mask (1/16) called **weighted average**

1 LOW-PASS FILTERING/ AVERAGING FILTERING

- Salt & Pepper – high-frequency components of an image. Low-pass filter (or averaging filter) should reduce S&P noise:

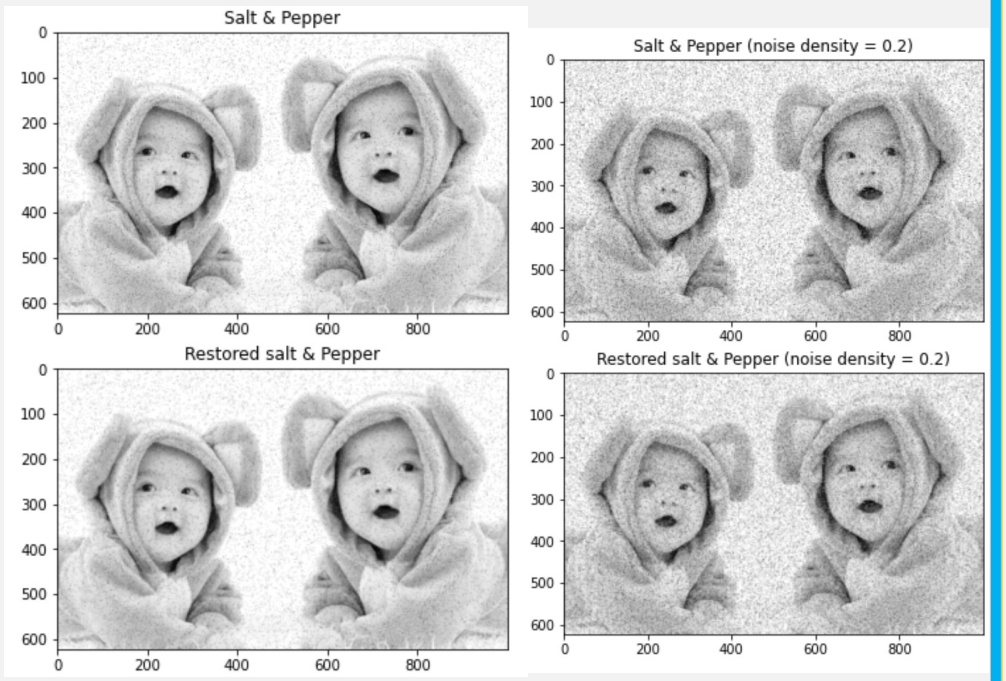
```
import cv2
from skimage.util import random_noise
import numpy as np

img = cv2.imread("twins.jpg",0)

image_sp = random_noise(img, mode='s&p')
image_sp2 = random_noise(img, mode='s&p', amount = 0.2)

K_SIZE = 3

avg_filter = np.ones((K_SIZE,K_SIZE),np.float32)/(K_SIZE*K_S

image_sp_a3 = cv2.filter2D(image_sp, -1, avg_filter)
image_sp2_a3 = cv2.filter2D(image_sp2, -1, avg_filter)
```



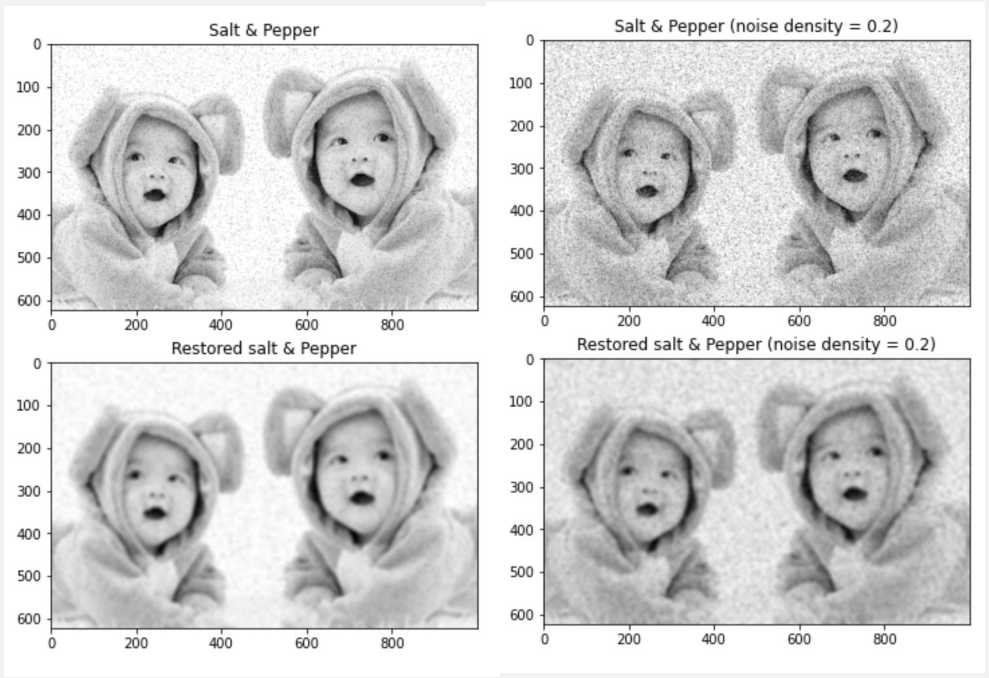
3x3 is so small to see differences!

1 LOW-PASS FILTERING/ AVERAGING FILTERING

- With larger filter size:



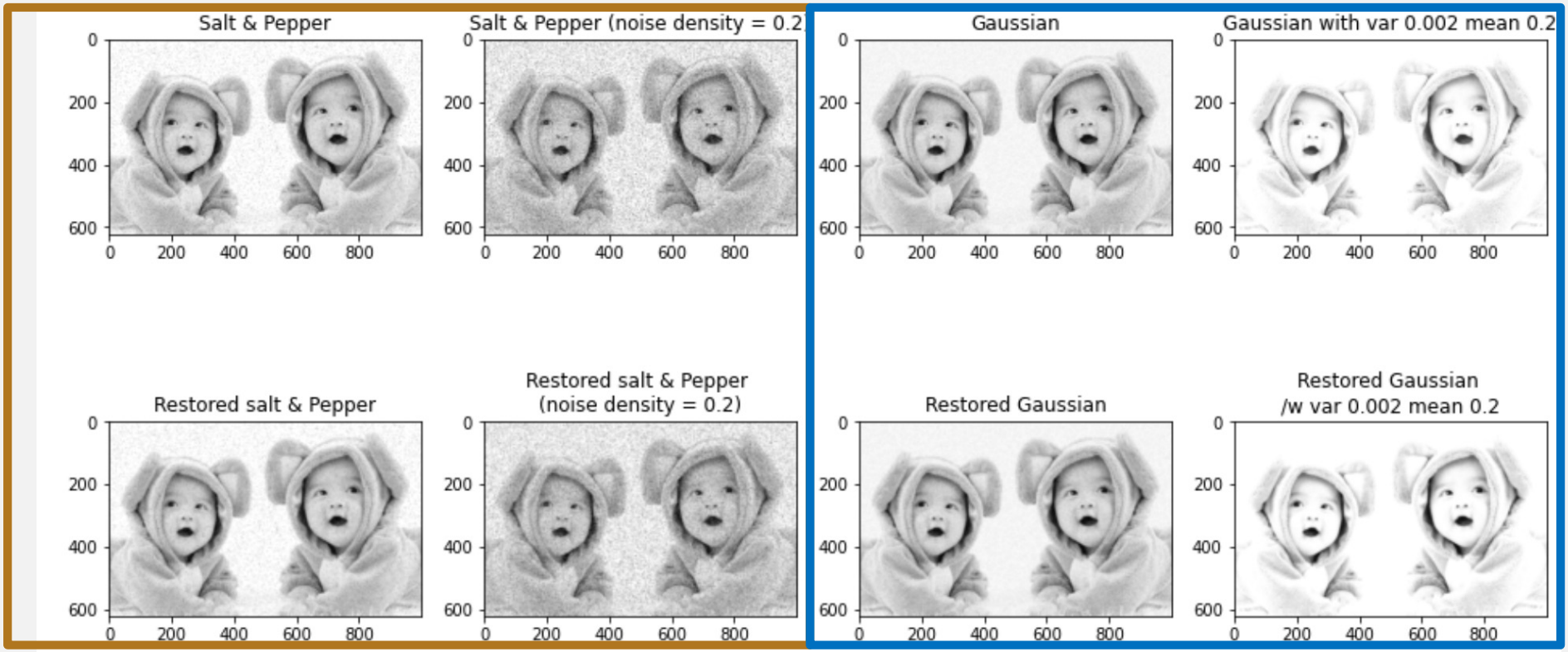
3x3 Average filter



11x11 Average filter

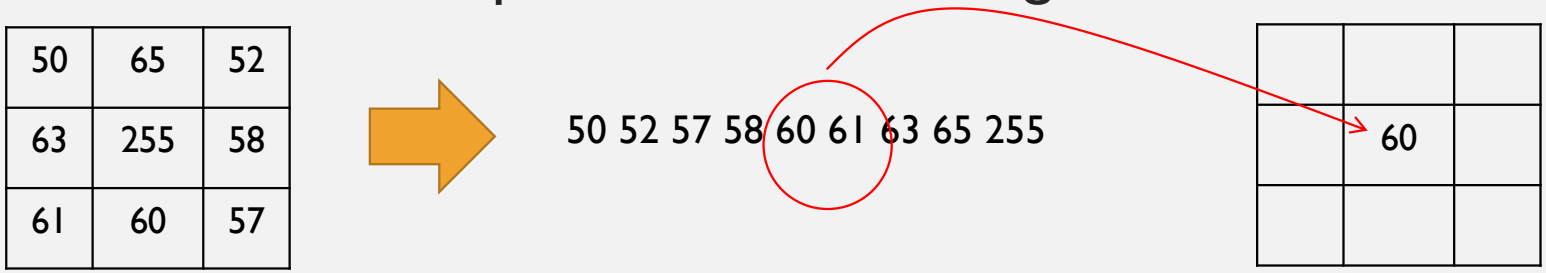
1 LOW-PASS FILTERING/ AVERAGING FILTERING

- Gaussian vs. Salt and Pepper noise



2 MEDIAN FILTERING

- Median values: 1 2 3 4 5 → ***3***
- Perform well when impulse noise is not large.



- What do you see from this example?
- Find the median values for the rest of the pixels.

2 MEDIAN FILTERING

- Python – **`medianBlur()`**

medianBlur()

```
void cv::medianBlur ( InputArray  src,
                      OutputArray dst,
                      int         ksize
                    )
```

Python:

```
dst = cv.medianBlur( src, ksize[, dst] )
```

```
#include <opencv2/imgproc.hpp>
```

Blurs an image using the median filter.

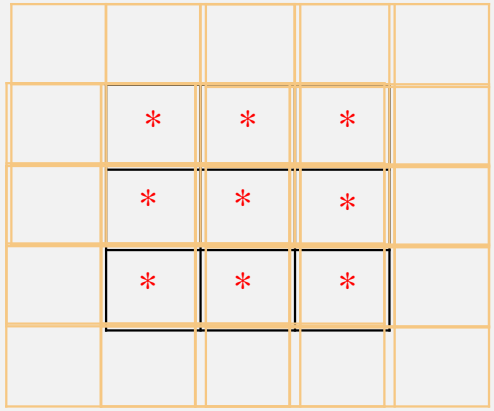
The function smoothes an image using the median filter with the `ksize` × `ksize` aperture. Each channel of a multi-channel image is processed independently. In-place operation is supported.

Note

The median filter uses **BORDER_REPLICATE** internally to cope with border pixels, see **BorderTypes**

Parameters

src	input 1-, 3-, or 4-channel image; when ksize is 3 or 5, the image depth should be CV_8U, CV_16U, or CV_32F, for larger aperture sizes, it can only be CV_8U.
dst	destination array of the same size and type as src.
ksize	aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...



- **`B = medianBlur(A,3)`** performs median filtering of the matrix A in two dimensions. Each output pixel contains the median value in a 3-by-3 neighborhood around the corresponding pixel in the input image using border replicate.

https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9

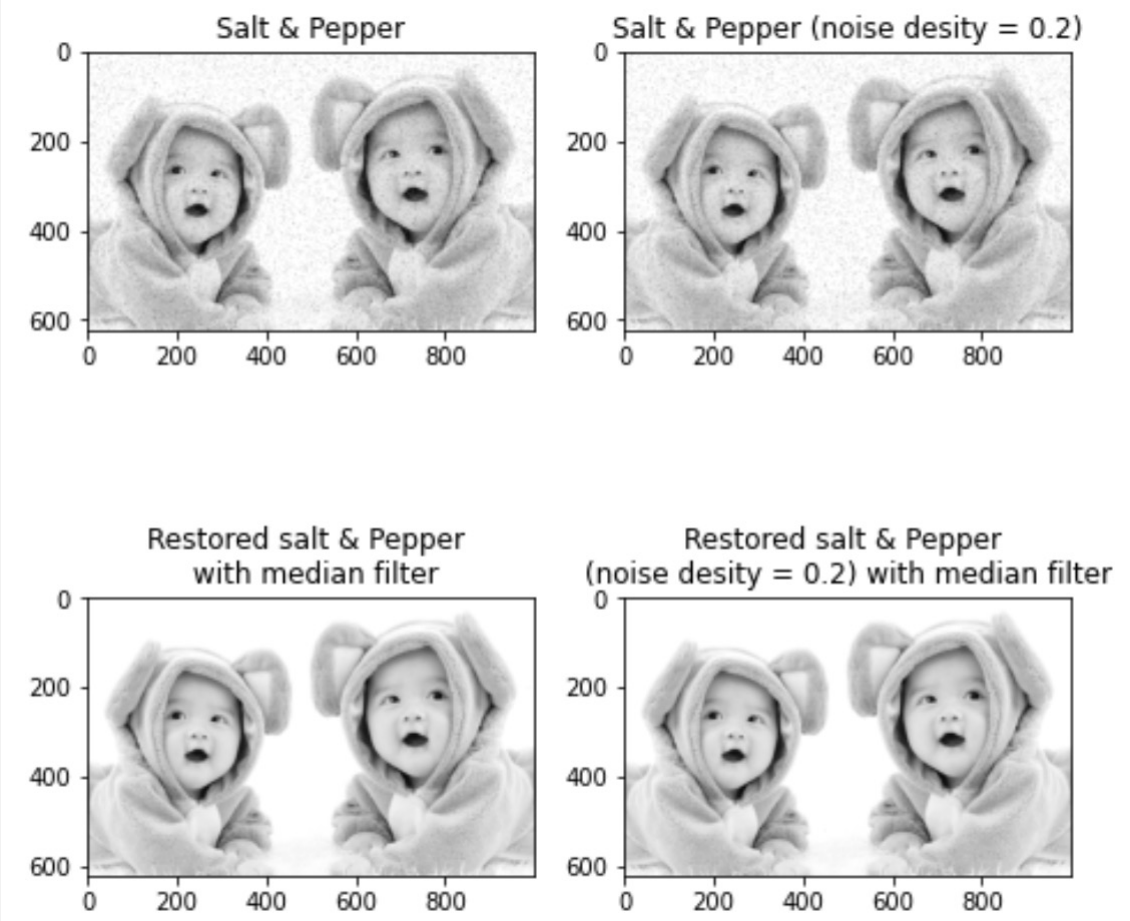
2 MEDIAN FILTERING

```
from skimage import io, color
from skimage.util import random_noise
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("twins.jpg",0)

image_sp = random_noise(img, mode='s&p')
image_sp2 = random_noise(img, mode='s&p')
image_sp = np.array(255*image_sp, dtype = 'uint8')
image_sp2 = np.array(255*image_sp2, dtype = 'uint8')

fig = plt.figure(figsize = (16,8))
fig.add_subplot(2,4,1), plt.imshow(image_sp, cmap="gray"), plt.title('Salt & Pepper')
fig.add_subplot(2,4,2), plt.imshow(image_sp2, cmap="gray"), plt.title('Salt & Pepper (noise desity = 0.2)')


image_md = cv2.medianBlur(image_sp,3)
image_md2 = cv2.medianBlur(image_sp2,3)
fig.add_subplot(2,4,5), plt.imshow(image_md, cmap="gray"), plt.title('Restored salt & Pepper \n with median
filter')
fig.add_subplot(2,4,6), plt.imshow(image_md2, cmap="gray"), plt.title('Restored salt & Pepper \n(noise
desity = 0.2) with median filter')
plt.show()
```

2

MEDIAN FILTERING

- Salt & Pepper noise



2

MEDIAN FILTERING

- Median filters vs. Gaussian filters

s&p noise

s&p noise with d = 0.2

restored s&p noise with med filter

restored s&p noise with d = 0.2 with med filte

Median filters

- The default for d is 0.05.

s&p noise

s&p noise with d = 0.2

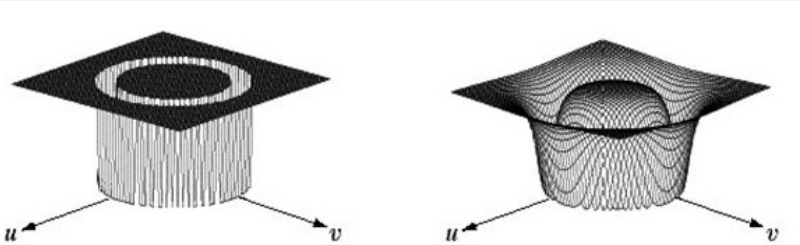
restored s&p noise with 7x7 avg filter

restored s&p noise with d = 0.2 with 7x7 avg filter

Gaussian filters

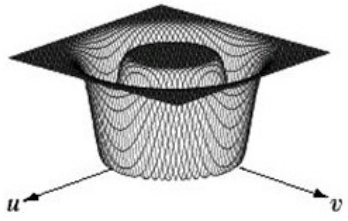
3 REMOVING PERIODIC NOISE

- Band-reject filters

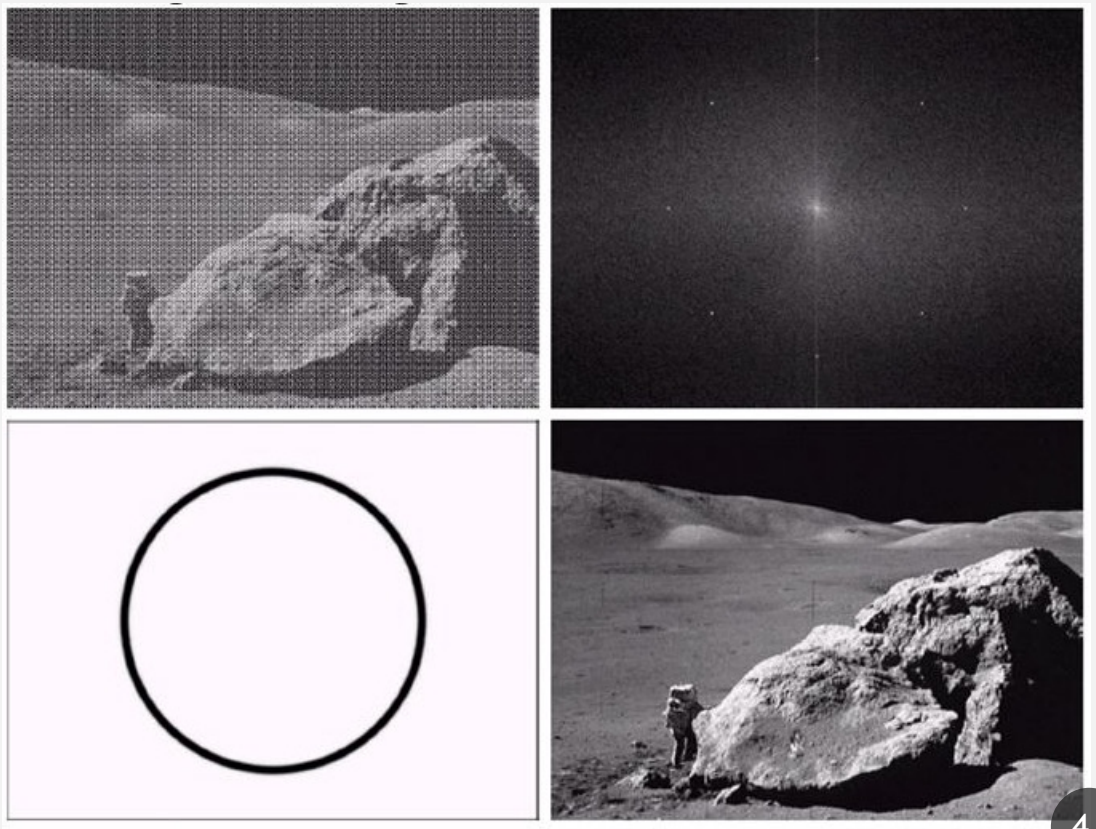


a b c

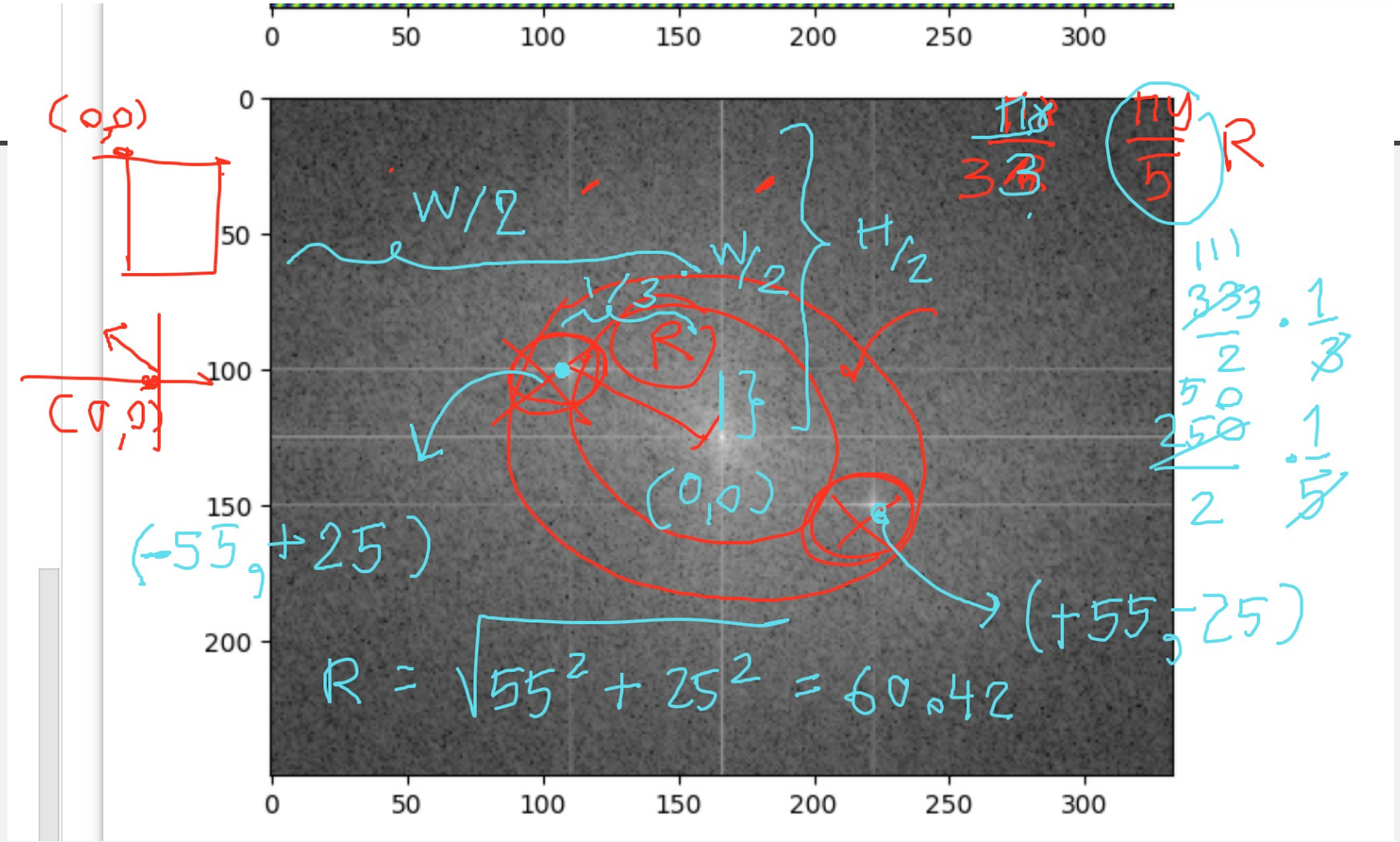
FIGURE 5.15 From left to right, perspective plots of ideal, Butterworth filters.



(of order 1), and Gaussian bandreject

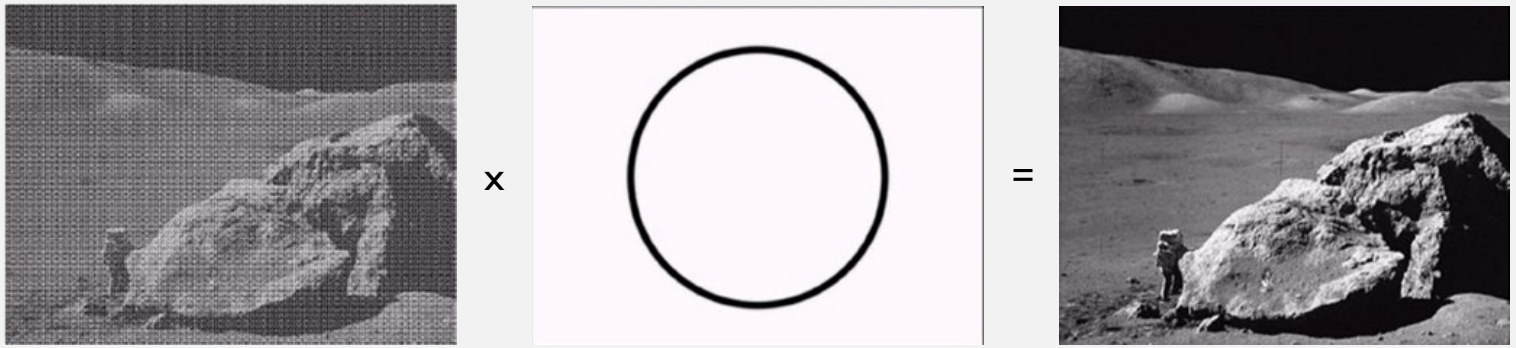


© Gonzalez & Woods, Digital Image Processing

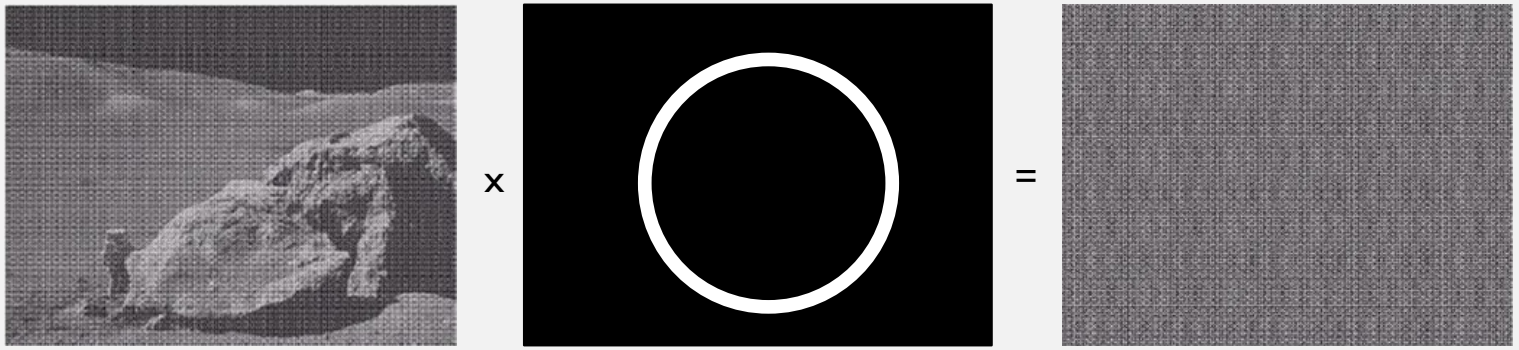

0
50
100
150
200
250
300
(0,0)
W/2
H/2
W/3
R
(0,0)
(-55, +25)
(+55, -25)
R = √(55² + 25²) = 60.42

3 REMOVING PERIODIC NOISE

- Bandpass filters



bandreject



1-bandreject

=bandpass

© Gonzalez & Woods, Digital Image Processing

3 REMOVING PERIODIC NOISE

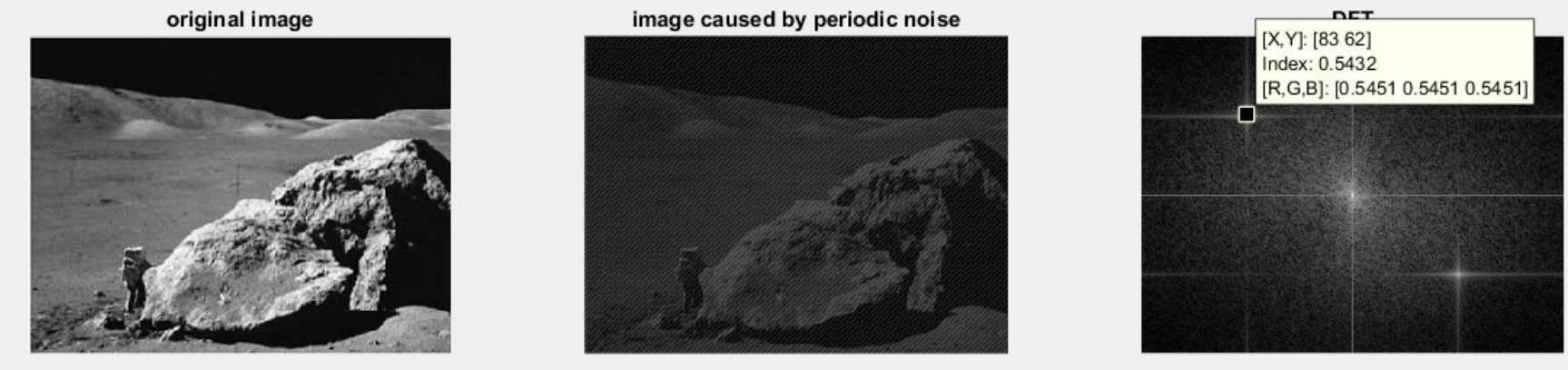
- A ***notch filter*** rejects (or passes) frequencies in predefined neighborhoods about a center frequency.
- Due to the symmetry of the Fourier transform, notch filters must appear in symmetric pairs about the origin in order to obtain meaningful results.
- So transfer function of an ideal notch filter (reject) of radius D_0 with a center at (u_0, v_0) and by symmetry at $(-u_0, -v_0)$:

FIGURE 5.18 Perspective plots of (a) ideal, (b) Butterworth (of order 2), and (c) Gaussian notch (reject) filters.

© Gonzalez & Woods, Digital Image Processing

3 REMOVING PERIODIC NOISE

- **Band Reject Filtering**



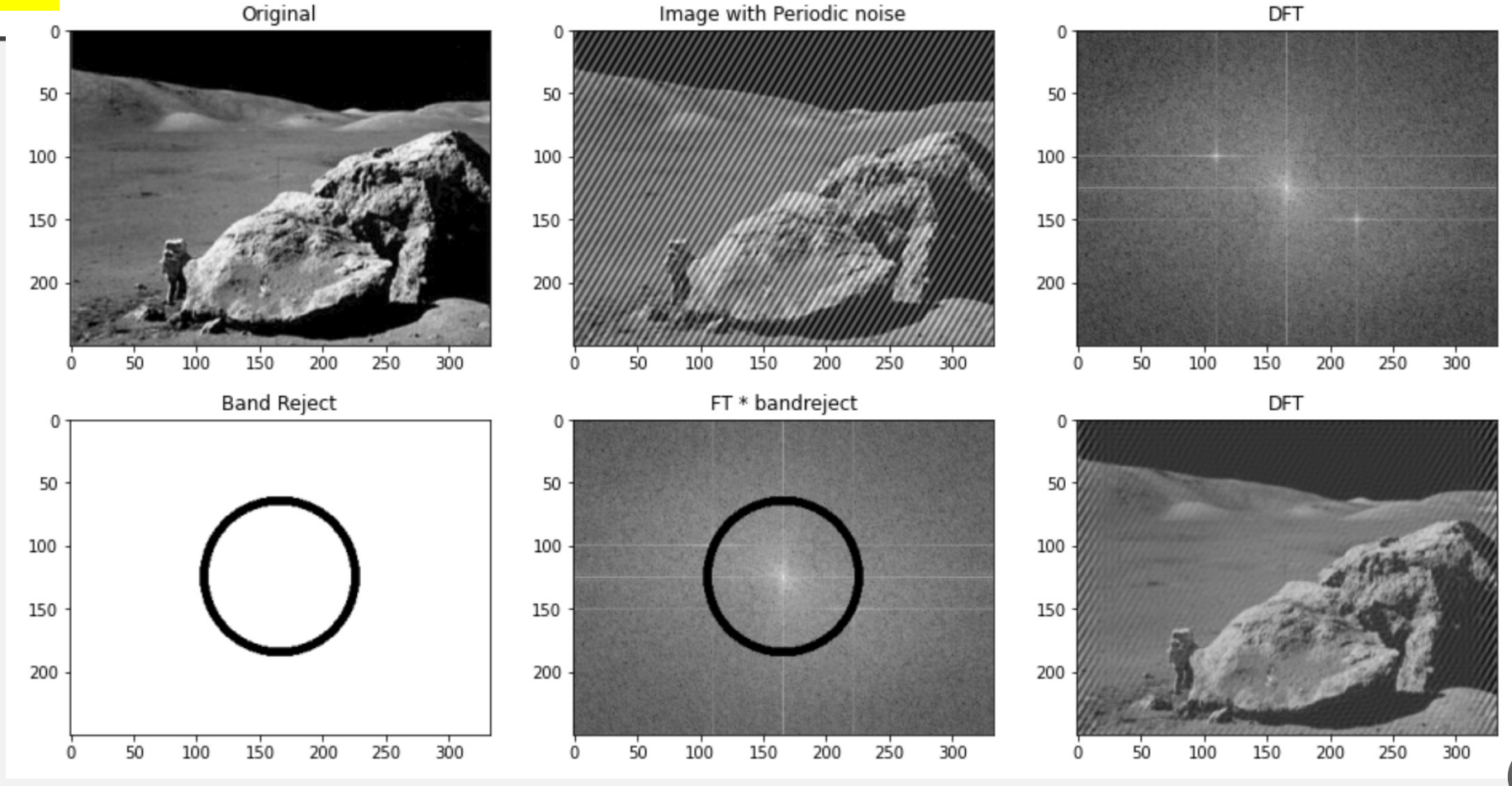
- Using check where the noise frequencies are.
- For band reject filtering -> create radius-angle coordinates to define the rejected band

3 EXERCISE #2: BAND REJECT FILTERING

```
nx, ny = img.shape
y = np.linspace(-nx/2,nx/2,nx)
x = np.linspace(-ny/2,ny/2,ny)
xv, yv = np.meshgrid(x,y)
radiusCoor = np.sqrt(xv**2 + yv**2)

r1 = radiusCoor < _____________________
r2 = radiusCoor >= _____________________
mask1 = np.ones([nx,ny])
bandReject =
cv2.bitwise_or(r1.astype(np.uint8),r2.astype(np.uint8),mask=mask1.astype(np.uint8))
plt.imshow(bandReject,cmap='gray')
plt.show()

img_bandReject = bandReject * fshift
img_ft_filter_hp_spectrum = np.log(1+np.abs(img_bandReject))
plt.imshow(img_ft_filter_hp_spectrum)
plt.show()

f_ishift_br = np.fft.ifftshift(img_bandReject)
img_restored = np.fft.ifft2(f_ishift_br)
img_restored = np.abs(img_restored)
```

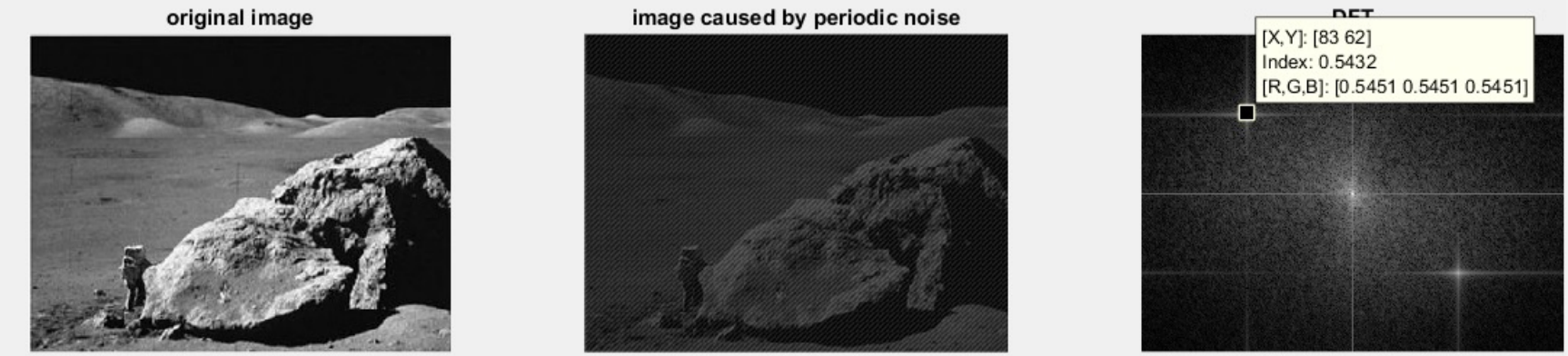
3

EXERCISE #2: BAND REJECT FILTERING



3 REMOVING PERIODIC NOISE

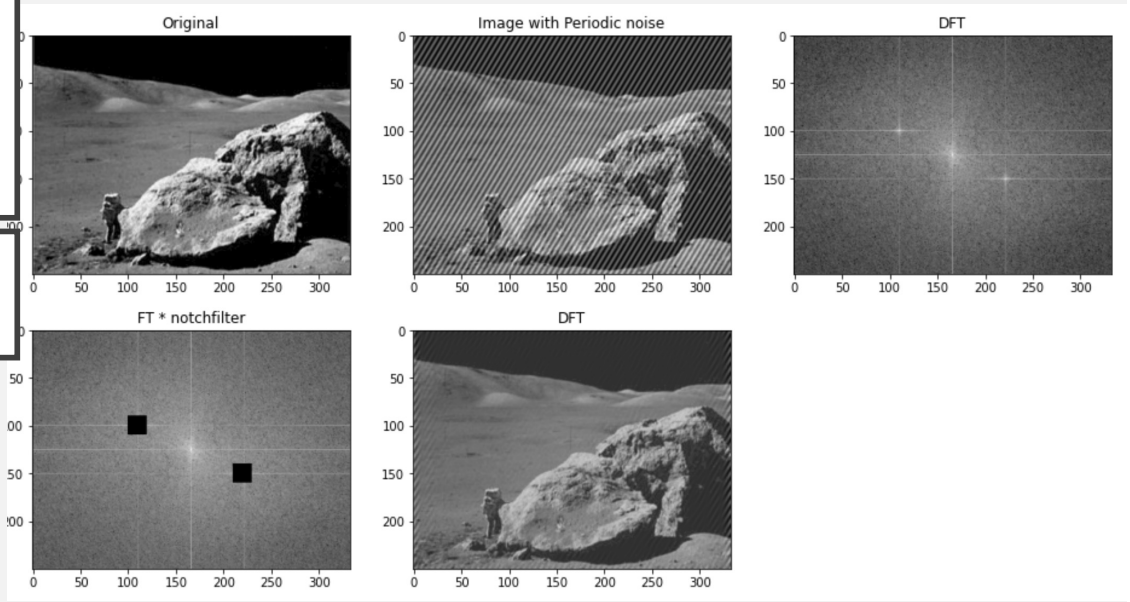
- **Notch filter**



- Check where the noise frequencies are.
- For notch filter: we simply make rows and columns of the spikes 0.

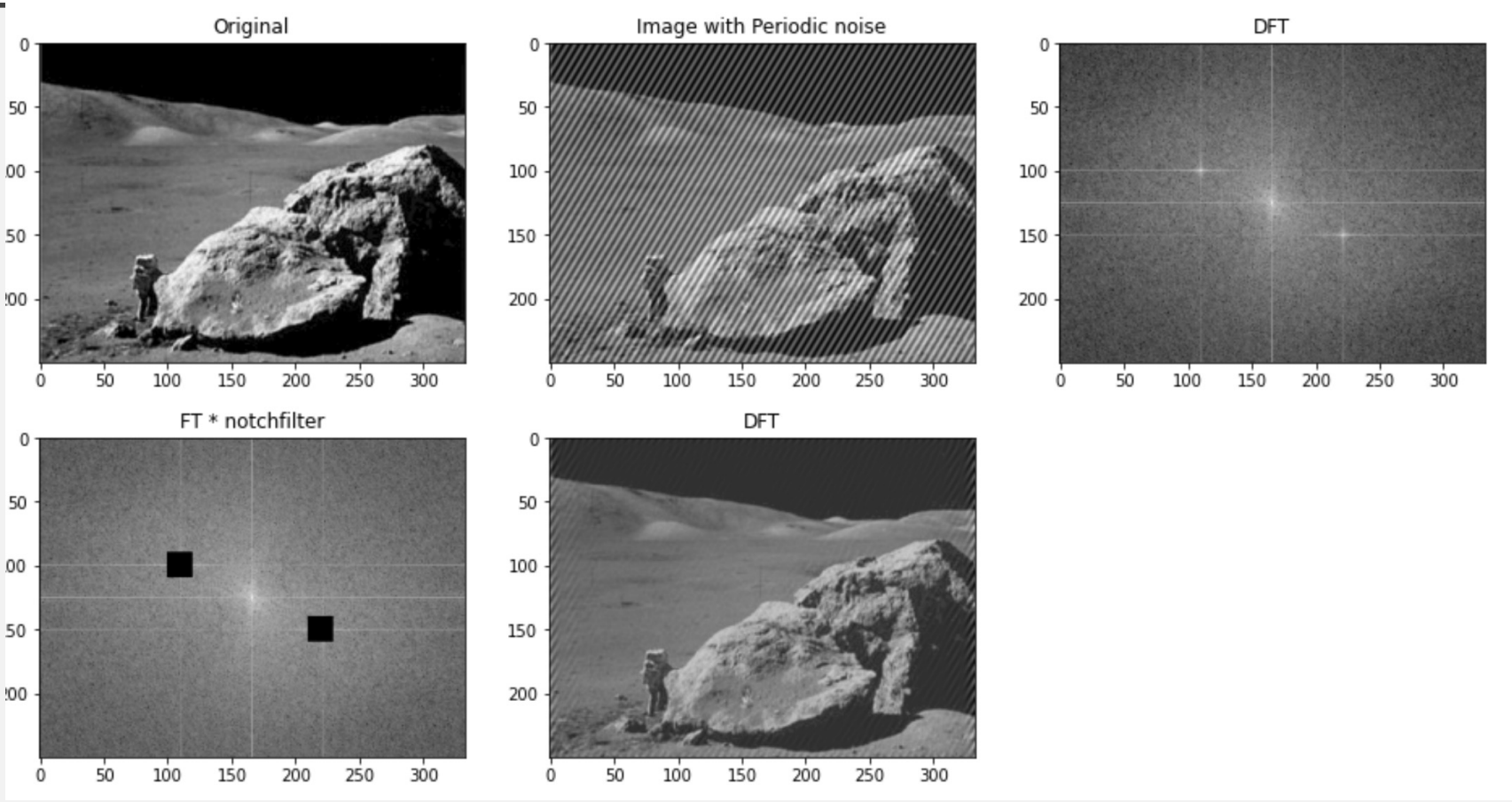
3 PYTHON: NOTCH FILTERING

EXERCISE #3: NOTCH FILTERING



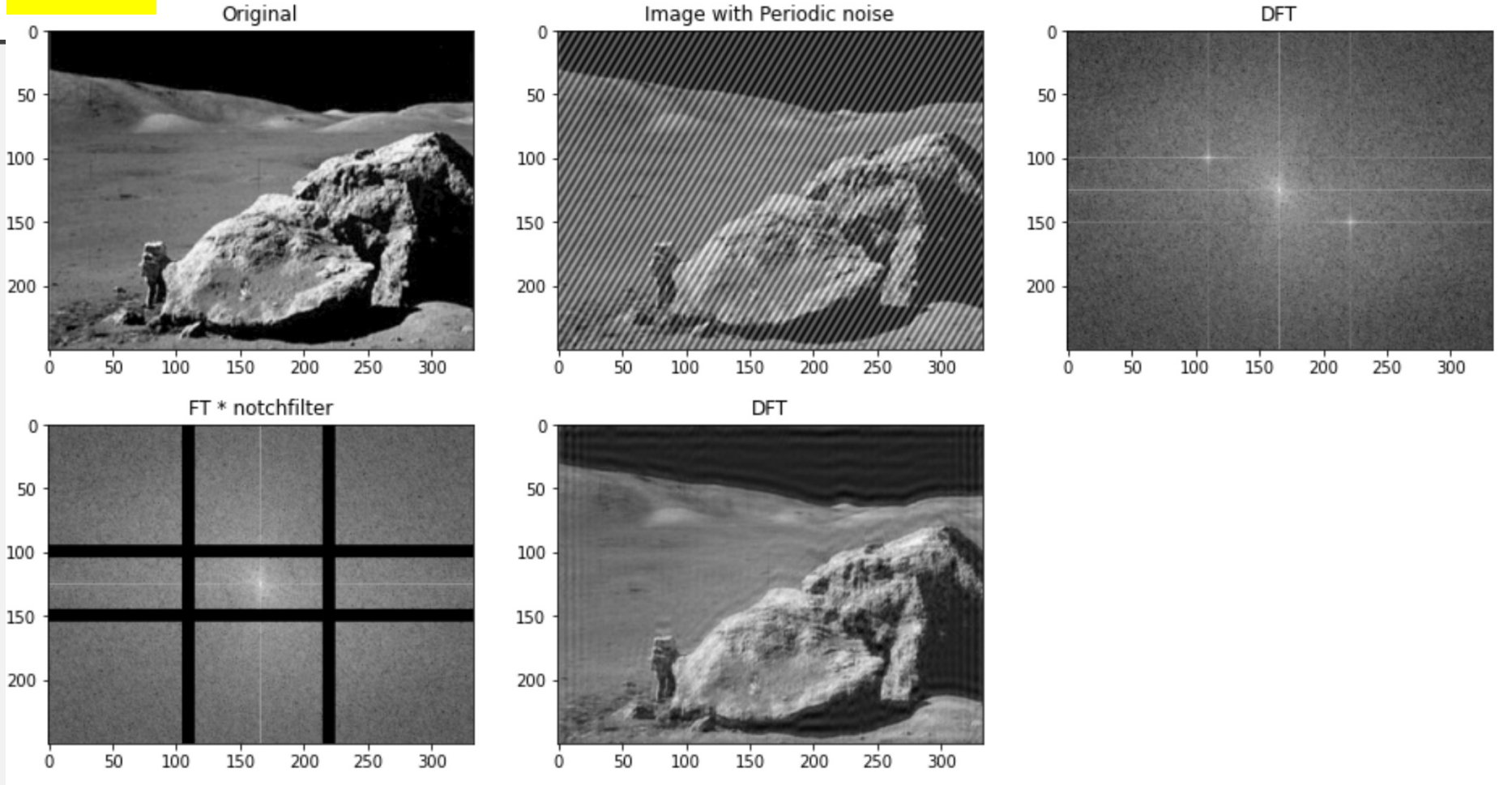

```
im_notchFilter = fshift.copy()
im_notchFilter[__________________]= 0 #top left
im_notchFilter[__________________]= 0 #bottom right

img_ft_filter_nf_spectrum = np.log(1+np.abs(im_notchFilter))

f_ishift_nf = np.fft.ifftshift(im_notchFilter)
img_restored = np.fft.ifft2(f_ishift_nf)
img_restored = np.abs(img_restored)
```

EXERCISE #3: NOTCH FILTERING

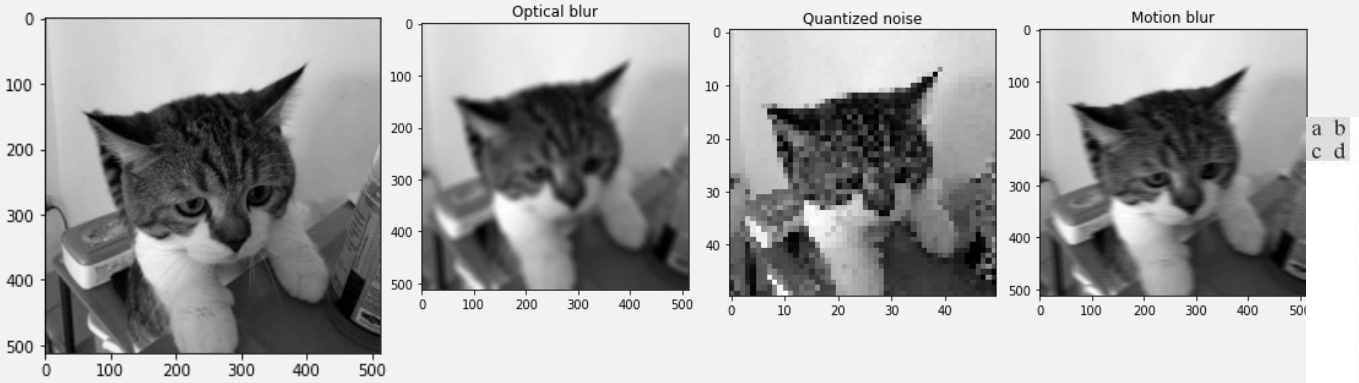


3

PYTHON: NOTCH FILTERING

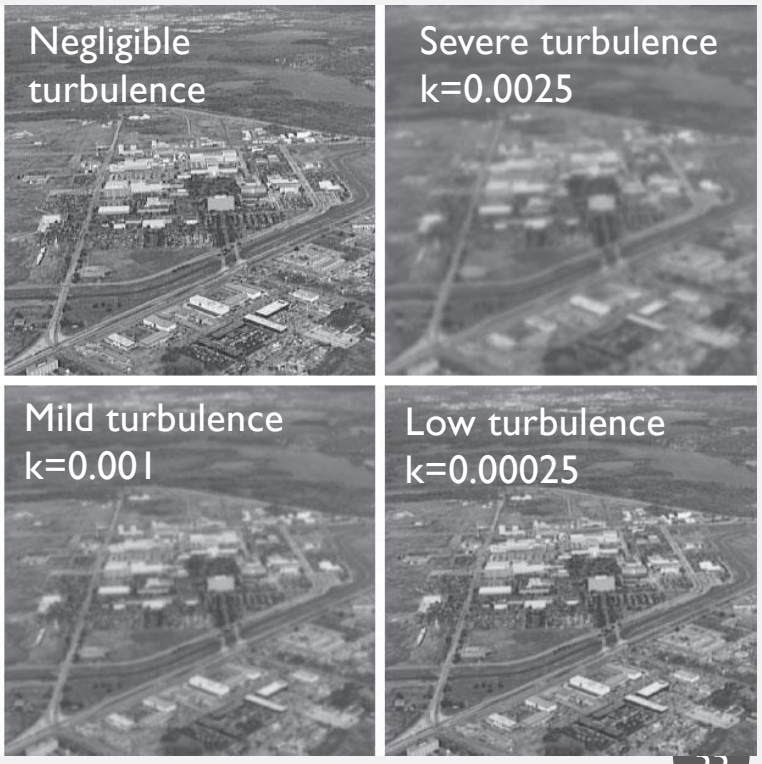


OTHER DEGRADATIONS



- Estimating the degradation function
  - Estimation by Image observation
  - Estimation by Experimentation
  - Estimation by Modelling
    - Atmosphere tubulence: Hufnagel and Stanley (1964)

$$H(u,v) = e^{-k(u^2+v^2)^{5/6}}$$



© Gonzalez & Woods, Digital Image Processing

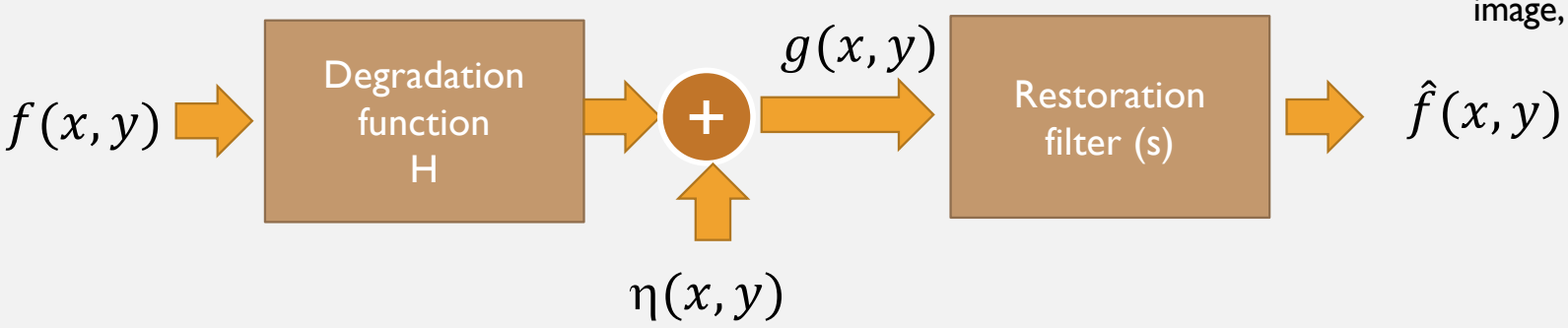
INVERSE FILTERING

- An image degraded by a degradation function H and added by noise N can be represented in spatial domain and frequency domain below,

$$g(x,y) = h(x,y) * f(x,y) + \eta(x,y)$$

$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

Spatial Domain
$\eta(x,y)$ – noise term
$f(x,y)$ – original image
$g(x,y)$ – degraded image
$\hat{f}(x,y)$ - reconstructed image
N, F, G, $\hat{F}$ are the frequency domain of noise term, original image, degraded image and reconstructed image, respectively



INVERSE FILTERING

- An image degraded by a degradation function H. The simplest approach to restoration is direct inverse filtering, where we compute an estimate, $\hat{F}(u,v)$ by

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)}$$

- If there is noise term, we get

$$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}$$

Introduction to Inverse Problems in Imaging, M. Bertero, P. Boccacci, Christine De Mol (2021)

N, F, G, $\hat{F}$ are the frequency domain of noise term, original image, degraded image and reconstructed image, respectively.

4 MINIMUM MEAN SQUARE ERROR (WIENER) FILTERING

- Wiener filter is to find the reconstructed image $\hat{f}(x,y)$ that minimizing the mean squre error of $\hat{f}(x,y)$ and $f(x,y)$ by

$$e^2 = E\{(f-\hat{f})^2\}$$

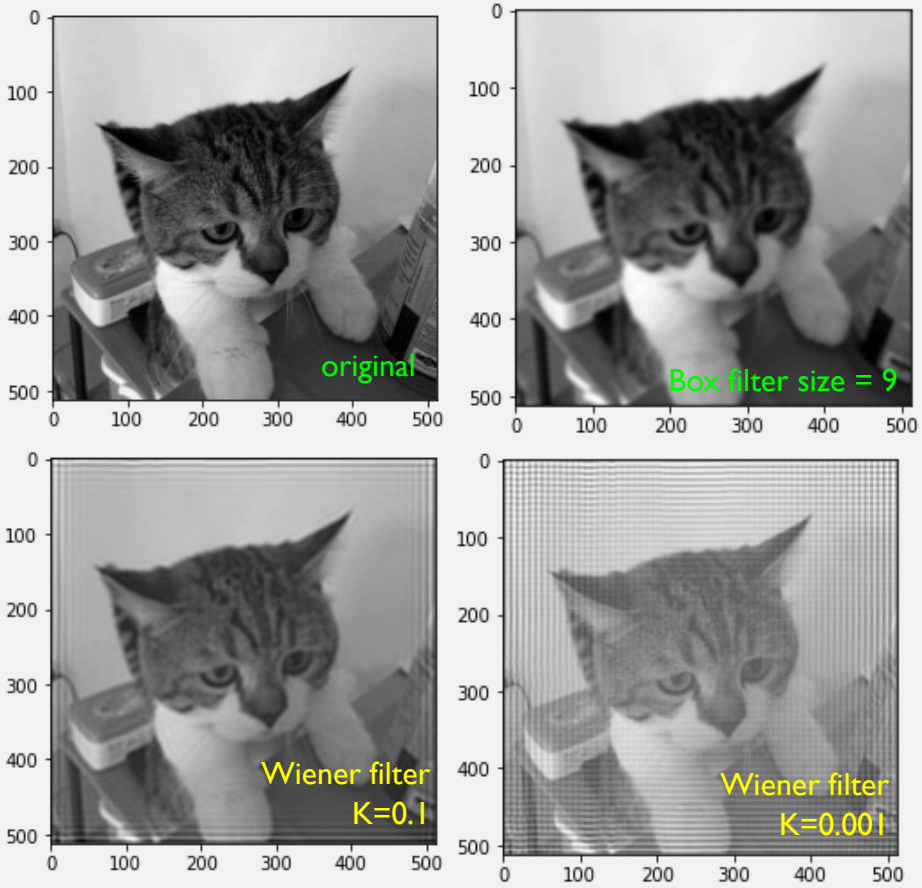
- where E{·} is the expected value of the argument. Based on this conditions, the minimum error is given in the frequency domain by the expression

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}\right] G(u,v)$$

Constant K

$H(u,v)$-degradation function
$H^*(u,v)$-complex conjugate of $H(u,v)$
$|H(u,v)|^2 = H^*(u,v)\,H(u,v)$
$S_\eta(u,v) - |N(u,v)|^2$ - power spectrum of the noise
$S_f(u,v) - |F(u,v)|^2$ - power sepectrum of the undegraded image

4 MINIMUM MEAN SQUARE ERROR (WIENER) FILTERING



wiener

`skimage.restoration.`**wiener**(*image*, *psf*, *balance*, *reg=None*, *is_real=True*, *clip=True*) [source]

Wiener-Hunt deconvolution

Return the deconvolution with a Wiener-Hunt approach (i.e. with Fourier diagonalisation).

Parameters

image : (M, N) ndarray

Input degraded image

psf : ndarray

Point Spread Function. This is assumed to be the impulse response (input image space) if the data-type is real, or the transfer function (Fourier space) if the data-type is complex. There is no constraints on the shape of the impulse response. The transfer function must be of shape *(M, N)* if *is_real is True*, *(M, N // 2 + 1)* otherwise (see *np.fft.rfftn*).

balance : float

The regularisation parameter value that tunes the balance between the data adequacy that improve frequency restoration and the prior adequacy that reduce frequency restoration (to avoid noise artifacts).

Returns

im_deconv : (M, N) ndarray

The deconvolved image.

François Orieux, Jean-François Giovannelli, and Thomas Rodet, "Bayesian estimation of regularization and point spread function parameters for Wiener-Hunt deconvolution", J. Opt. Soc. Am. A 27, 1593-1607 (2010) https://www.osapublishing.org/josaa/abstract.cfm?URI=josaa-27-7-1593
http://research.orieux.fr/files/papers/OGR-JOSA10.pdf
B. R. Hunt "A matrix theory proof of the discrete convolution theorem", IEEE Trans. on Audio and Electroacoustics, vol. au-19, no. 4, pp. 285-288, dec. 1971

RESTORATION: EVALUATION

Peak signal-to-noise ratio (PSNR) is the ratio between the maximum possible power of an image and the power of corrupting noise that affects the quality of its representation. To estimate the PSNR of an image, it is necessary to compare that image to an ideal clean image with the maximum possible power.

PSNR is defined as follows:

$$PSNR = 10log_{10}(\frac{(L-1)^2}{MSE}) = 20log_{10}(\frac{L-1}{RMSE})$$

Here, **L** is the number of maximum possible intensity levels (minimum intensity level suppose to be 0) in an image.

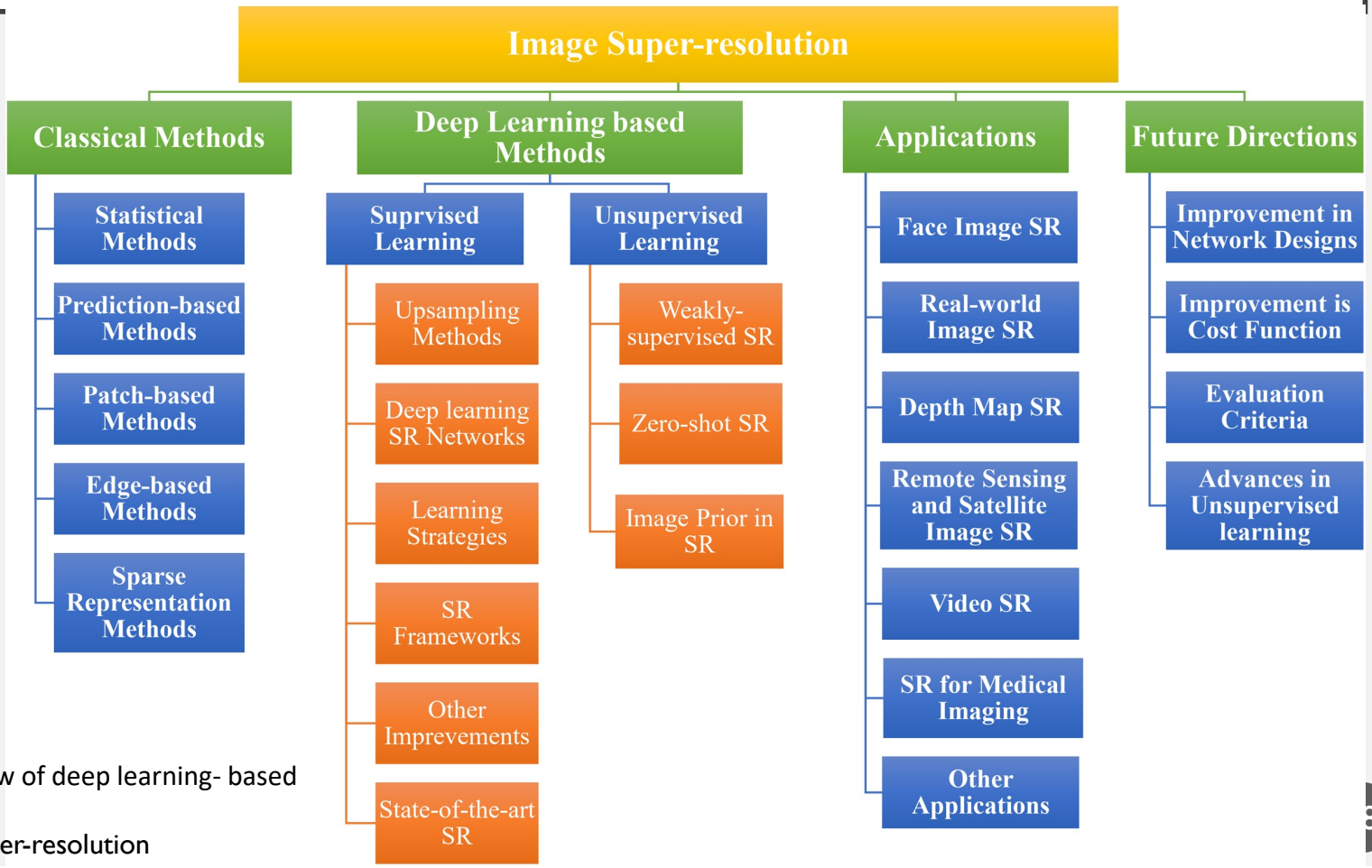
MSE is the mean squared error & it is defined as:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(O(i,j) - D(i,j))^2$$

- Other metrics
  - **Structural Similarity Index** (SSIM)

IMAGE SUPERRESOLUTION

- Image super-resolution (SR) is one of the vital image processing methods that **improve the resolution** of an image in the field of computer vision



S.M.A. Bashir, et al. A comprehensive review of deep learning- based single image super-resolution (2021)
https://paperswithcode.com/task/image-super-resolution

REFERENCES

- Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing, Addison- Wesley
  - Chapter 5, Section 5.1-5.8
- Image restoration slides
  - https://www.robots.ox.ac.uk/~az/lectures/ia/lect3.pdf