

Denoising

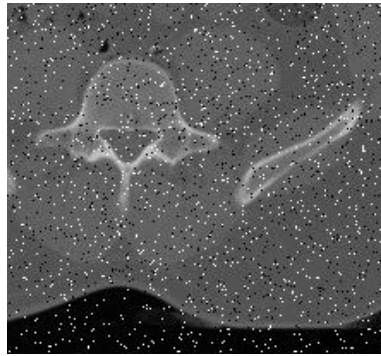
L20

Goal: To find out the types of compromises we use to try to remove noise from images.

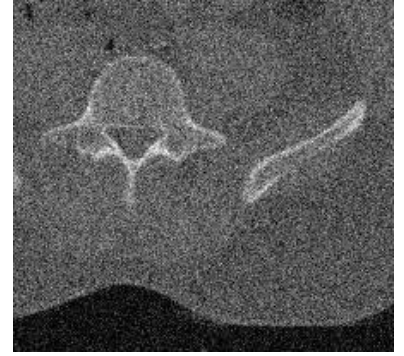
There are different types of noise in images.



Original
(8-bit image)



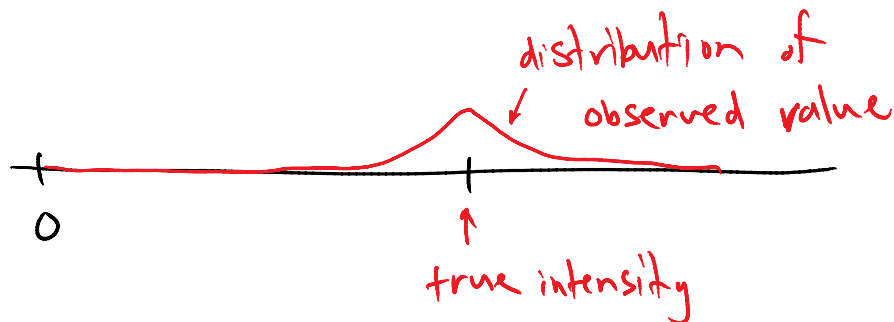
Speckle
("Salt and pepper")



Gaussian (Normal)
distributed
($\mu=0$, $\sigma=25$)

In an MRI, the acquired (raw) data is complex-valued. Both the real & imaginary parts have Gaussian noise. When you look at the noise in the magnitude of the reconstructed images, its distribution is called **Rician**.

But for this course we will usually talk about Gaussian noise. Gaussian noise is typically additive



These random fluctuations are introduced into the images

through a number of phenomena. Most notably

- electrical noise in analog circuits
- sample statistics (eg. # of photons that hit an x-ray detector)

Noise Removal

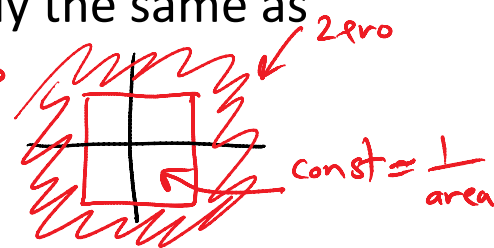
Let f be the true image (noise-free). What we actually observe is u , where

$$u = f + \varepsilon \quad (\text{additive noise model})$$

ε ← noise

The task of noise removal is to approximate f given only u (but not ε , though you may know some statistical characteristics of ε , eg. Zero-mean Gaussian distributed).

Many approaches:

- 1) Local median filter is good at removing speckle.
- 2) Windowed averaging: this is actually the same as convolving your noisy image with 
- 3) Gaussian smoothing/blurring
- 4) Wavelets (remove small-scale/low amplitude coefs)
- 5) Non-local Means

Median Filter

Look in a neighbourhood around a pixel and assign the median value.

eg.

10	15	21
11	18	30
9	11	12

Neighbourhood =

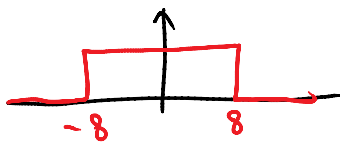
$\{9, 10, 11, 11, 12, 15, 18, 21, 30\}$

↑
middle = median

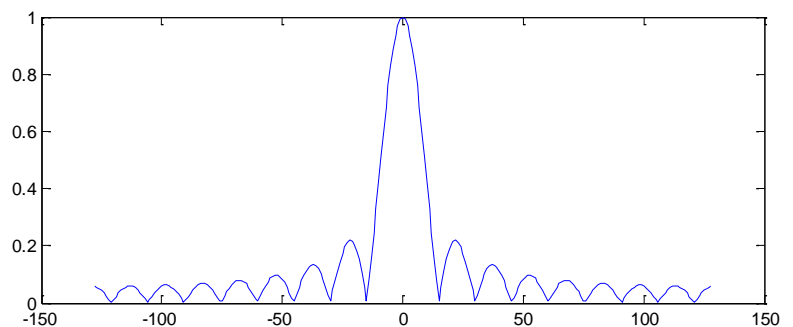
∴ assign an intensity of **12** where the 18 is.
(Peltis speckle demo)

Windowed Averaging

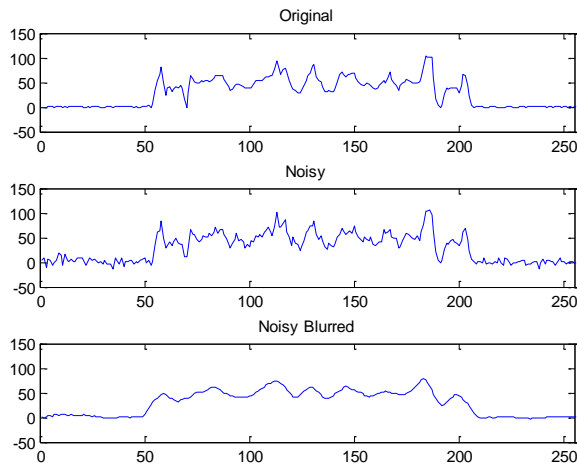
In 1D, it is like convolving with a **box filter**.



```
f = zeros(1,257);  
f((129-8):(129+8)) = 1/15;  
F = fftshift( fft( ifftshift(f) ) );  
x = (1:257) - 129;  
plot(x, abs(F));
```

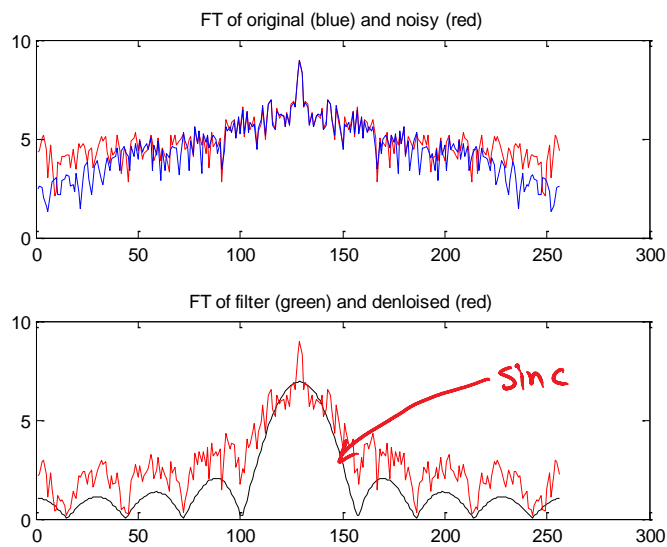


So windowed averaging is equivalent to multiplying the Fourier coefficients by the **sinc** function. Thus, the **low** frequencies are maintained, but the **high** frequencies are largely dampened.



Additive Gaussian with $\sigma=5$

Filtered with 



Note: FT is linear.

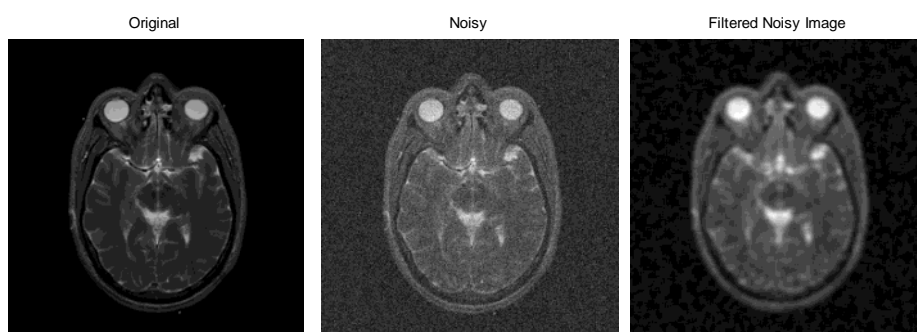
$$F = \mathcal{F}\{f\}$$

$$G = \mathcal{F}\{g\}$$

$$\alpha F = \mathcal{F}\{\alpha f\}$$

$$F + G = \mathcal{F}\{f + g\}$$

How about on an image?



8-bit

$\sigma=15$

Filtered \rightarrow Box filter
with radius = 2



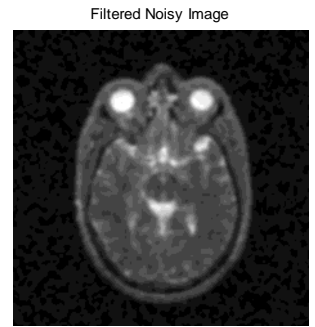
Gaussian Smoothing

Filtered Noisy Image



Gaussian Smoothing

Similar to windowed averaging above, but using a Gaussian kernel. The FT of a Gaussian is a Gaussian, so the high frequencies are all but gone.

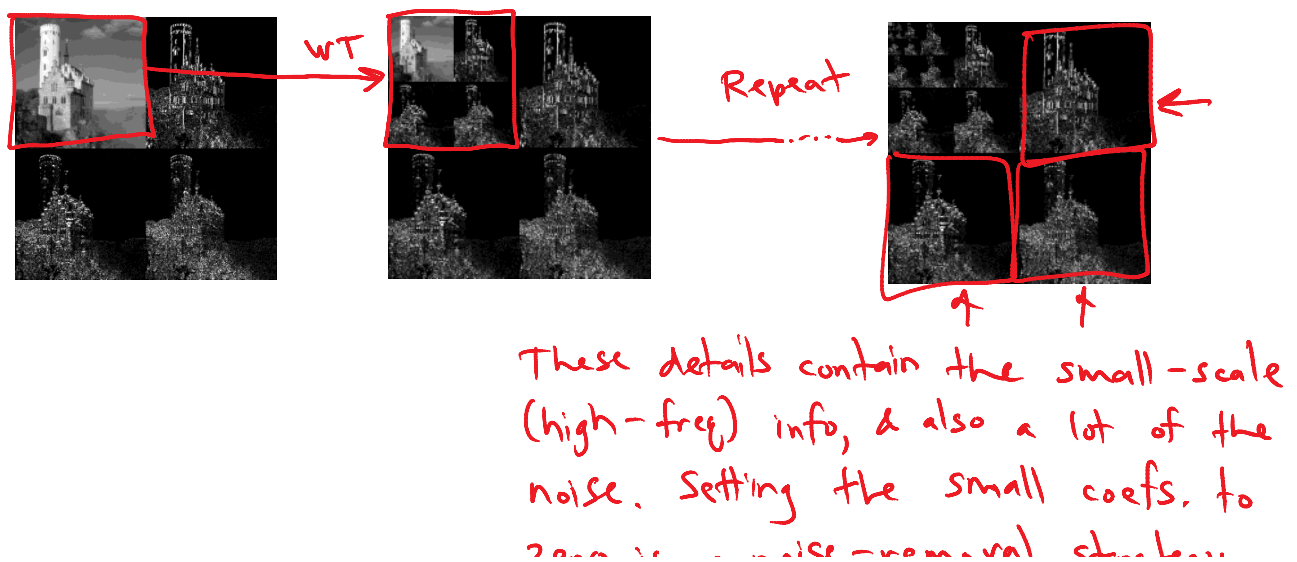


Wavelets

Unfortunately, wavelets are not part of this course. Briefly, it is a transform that applies smoothing along rows and columns, and records both the smoothed values, as well as the details that the smoothing removed.



Repeat the process on the smoothed version.



noise. Setting the small coeffs. to zero is a noise-removal strategy.

Non-Local Means

Replace each pixel in the image with a weighted average of other pixels with similar neighbourhoods.

$$\begin{aligned} \{f\}_n &= \sum_{j=0}^{N-1} \left(\sum_{k=0}^{N-1} f_k g_{j-k} \right) e^{-2\pi i \frac{jn}{N}} \\ &= \sum_{j=0}^{N-1} f_k e^{-2\pi i \frac{kn}{N}} \sum_{k=0}^{N-1} g_{j-k} e^{-2\pi i \frac{n}{N}(j-k)} \\ &= \hat{f}_n \hat{g}_n . \end{aligned}$$

the above derivation assumes that f and g are

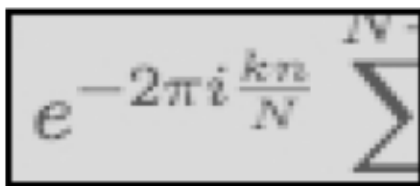
(a) Test image

$$\begin{aligned} \{f\}_n &= \sum_{j=0}^{N-1} \left(\sum_{k=0}^{N-1} f_k g_{j-k} \right) e^{-2\pi i \frac{jn}{N}} \\ &= \sum_{j=0}^{N-1} f_k e^{-2\pi i \frac{kn}{N}} \sum_{k=0}^{N-1} g_{j-k} e^{-2\pi i \frac{n}{N}(j-k)} \\ &= \hat{f}_n \hat{g}_n . \end{aligned}$$

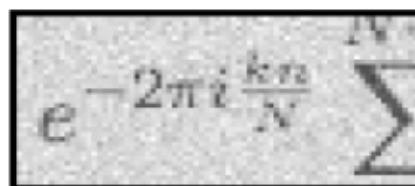
the above derivation assumes that f and g are

(b) Noisy, $\sigma=11$, PSNR=27.0

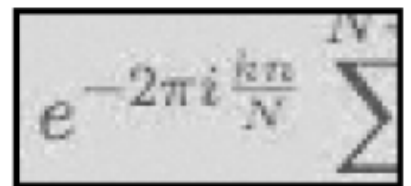
Bigger = better



Original

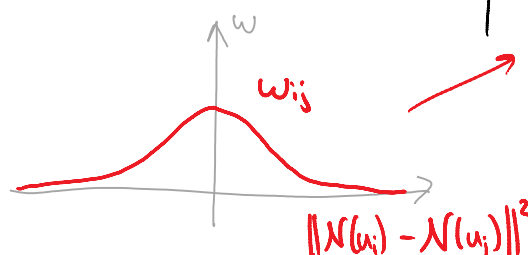


Noisy
PSNR = 27.0 dB



Denoised
PSNR = 33.7 dB

Recall: $u = f + \varepsilon$
 \uparrow \uparrow \uparrow
 noisy image ideal image noise



We replace u_i with \tilde{u}_i

$$\tilde{u}_i = \sum_j w_{ij} u_j$$

weight of pixel j on new pixel i

$$w_{ij} = \frac{1}{\sum_j w_{ij}} \exp \left(-\frac{1}{h^2} \|N(u_i) - N(u_j)\|^2 \right)$$

\uparrow \uparrow \uparrow
 $\sum_j w_{ij}$ Parameter neighbourhood around pixel j

To denoise a single pixel i , compare $N(u_i)$ to all other neighbourhoods $N(u_j)$.

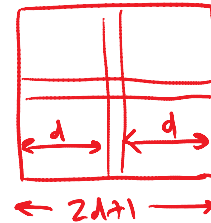
To denoise a single pixel i , compare x_i to all other neighbourhoods $\mathcal{N}(u_j)$.

For an $N \times N$ image, this requires $\mathcal{O}(N^2)$ neighbourhood comparisons!

But there are N^2 pixels to denoise $\Rightarrow \mathcal{O}(N^4)$

Suppose each neighbourhood has a radius of d , so contains $(2d+1) \times (2d+1)$ pixels.

\Rightarrow NL-Means is $\mathcal{O}(N^4 d^2)$!



This is why much of the research in NL-Means is devoted to speed improvements and shortcuts.