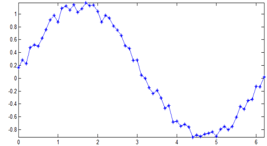
Lecture 3: Linear Filters

Signal Denoising

- Suppose we have a noisy 1D signal f(x).
- For example, it could represent a company's stock price over time.
- In order to see the overall trend of the data, economists "smooth" out the noise using a moving average.
- A 3-point moving average will look at each data point and one point to either side.
- We then average the 3 original points and write this into a new signal g(x).
- We then move to the next data point and repeat.



3-Point Moving Average

$\frac{1}{3}$ $\frac{1}{3}$ $\frac{1}{3}$

f =

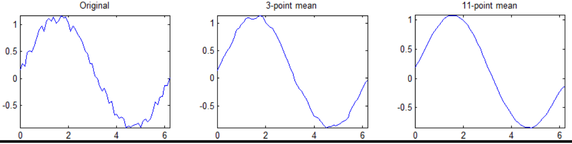
100	201	99	183	27	48	156	255

g =

		161	103				

1D Mean Filter

- In signal processing, this moving average process is called a 1D mean filter.
- We could do a 3-point mean filter where the weights are $\mathrm{w} = \left[\frac{1}{3} \ \ \frac{1}{3} \ \ \frac{1}{3}\right]$
- Or we could do a 5-point mean filter: $\mathrm{w} = \left[\frac{1}{5} \ \frac{1}{5} \ \frac{1}{5} \ \frac{1}{5} \ \frac{1}{5}\right]$
- In general, we can compute a N-point filter where N is an odd integer: $\mathrm{w} = \left[\frac{1}{N} \ \ \frac{1}{N} \ \ \frac{1}{N} \ \ \ldots \frac{1}{N}\right]$
- As N gets larger, the data gets more "smooth."



1D Convolution

- We can think of doing a moving average with any set of weights in the vector *w*.
- We express this moving average procedure as a convolution denoted by *.

$$g(x) = (f * w)(x) = \sum_{n} f(n)w(x-n)$$

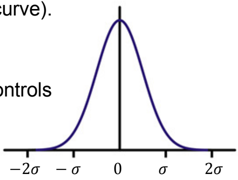
- A procedure that can be written as a convolution is called a linear filter.
- The weights w are called the filter or kernel.

- A 1D convolution can be computed using the Matlab command conv.

1D Gaussian Filter

- We could compute a weighted average by choosing any weights w that sum to 1.
- For example, we might want to give more emphasis to the center point and less weight to far-away points.
- The weights could be sampled from a zero-mean Gaussian distribution (bell curve).

$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$



- The standard deviation σ controls the width of the Gaussian.
- The area under the Gaussian is always one.

1D Gaussian Filter

- For Gaussian weights, the number of points N does not matter too much. The σ value controls the distribution.

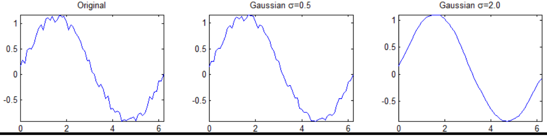
N=9, $\sigma = 0.5$

w=	0	0	0.0003	0.1065	0.7866	0.1065	0.0003	0	0

N=9, $\sigma = 2.0$

					center				
w=	0.0276	0.0663	0.1238	0.1802	0.2042	0.1802	0.1238	0.0663	0.0276

- As $\sigma \to 0$, the Gaussian filter makes little change to the data. As σ gets larger, it resembles a mean filter.



2D Convolution

- A 2D convolution is a weighted average of a image neighborhood.
- Generally the neighborhood is a NxN square, where N is an odd integer.

$$g(x,y) = (f * w)(x,y) = \sum_m \sum_n f(m,n)w(x-m, y-n)$$

g = filtered image   f = original image   w = filter (weights)

2D Convolution

- Digitally, we slide the filter *w* around the image *f* and compute the weighted average of that neighborhood.
- To illustrate convolution with a 3x3 neighborhood, let's compute $g = f * w$ at pixel 13 below.

W=

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

f =

f_1	f_2	f_3	f_4	f_5
f_6	f_7	f_8	f_9	f_{10}
f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
f_{21}	f_{22}	f_{23}	f_{24}	f_{25}

$$g_{13} = w_1f_7 + w_2f_8 + w_3f_9 + w_4f_{12} + w_5f_{13} + w_6f_{14} + w_7f_{17} + w_8f_{18} + w_9f_{19}$$

- We then slide the 3x3 box to the next pixel and compute the weighted average of that neighborhood.

Boundary Conditions

- What happens when our neighborhood window partly off the side of the image?
- There are several choices for the boundary conditions (BCs).

<u>Dirichlet BCs</u>

0	0	0		
0	12	13	21	39
0	41	33	88	62
	32	44	71	55
	16	64	10	28

<u>Periodic BCs</u>

28	16	64		
39	12	13	21	39
62	41	33	88	62
	32	44	71	55
	16	64	10	28

<u>Neumann (replicate) BCs</u>

12	12	13		
12	12	13	21	39
41	41	33	88	62
	32	44	71	55
	16	64	10	28

Convolution in Matlab

- We compute a 2D convolution with the Matlab command imfilter.

B = imfilter (A, w);

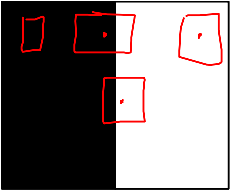
Filtered image   Original image   Filter

- The resulting image B will have the same size as the image A.
- The imfilter command works on both grayscale and color images.

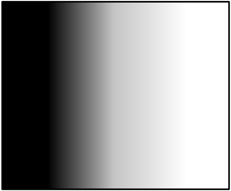
- The Matlab command fspecial can generate a variety of useful image filters.

Image Blur

- Applying a mean filter will "smooth" the edges of an image. We call this blur.



Mean Filter



Mean Filter

- In a $N \times N$ mean filter, all the weights are the same.
- As the window size N gets larger, the mean filter blurs the image more.

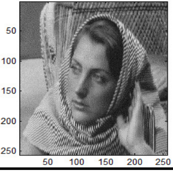
```
load woman;
imagesc(X);
```

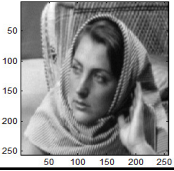
3x3 Mean Filter

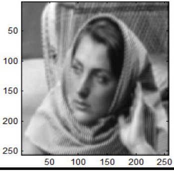
```
w=1/9*ones(3,3);
Y = imfilter(X,w);
imagesc(Y);
```

5x5 Mean Filter

```
w=1/25*ones(5,5);
Y = imfilter(X,w);
imagesc(Y);
```







Gaussian Filter

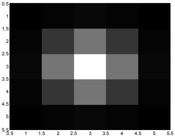
- To build a Gaussian filter, we need to tell fspecial what window size to use and the standard deviation σ.
- To create a 5x5 Gaussian window with $\sigma = 0.8$:

```
G = fspecial('gaussian', [5,5], 0.8)

G =
    0.0005    0.0050    0.0109    0.0050    0.0005
    0.0050    0.0522    0.1141    0.0522    0.0050
    0.0109    0.1141    0.2491    0.1141    0.0109
    0.0050    0.0522    0.1141    0.0522    0.0050
    0.0005    0.0050    0.0109    0.0050    0.0005
```

- It is sometimes helpful to display the filter weights.

- A Gaussian filter should be high in the middle (white) and low on the sides (black).



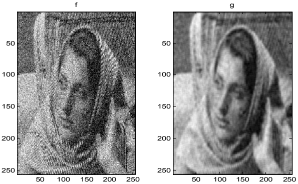
Gaussian Filter

- Although a Gaussian filter is good at removing Gaussian noise, it also blurs the image.

```
load woman;
f = imnoise(uint8(X),'gaussian', 0, 0.02);    %Add noise
G = fspecial('gaussian', [5,5], 0.5);
g = imfilter(f,G);
```

window size σ

- We'll talk more about denoising next week.
- Increasing σ will increase the amount of blur.



Motion Blur

- We can simulate a moving object or camera by adding motion blur.
- We specify the angle indicating direction of motion (in degrees) and the number of pixels moved.

length of motion (in pixels)

angle of motion direction (in degrees)

```
w = fspecial('motion', 6, 45)
```

0	0	0	0.0438	0.1004
0	0	0.0438	0.1496	0.0438
0	0.0438	0.1496	0.0438	0
0.0438	0.1496	0.0438	0	0
0.1004	0.0438	0	0	0

Motion Blur

Laplacian Filter

- The Laplacian Filter has a large negative value in the center.
- The Laplacian approximates the second derivatives.

```
w=fspecial('laplacian',0)
```

0	1	0
1	-4	1
0	1	0

- Note these weights do not sum to 1.
- For any filter that contains negative values, we should change the image to double format.

```
A = double(A);
B = imfilter(A, w);
```

Laplacian Filter

- The Laplacian Filter detects edges, taking on negative value near edges.
- Subtracting the Laplacian filtered image from the original image can help sharpen edges and remove blur.

```
w = fspecial('laplacian', 0);
A = double(A);     B = A - imfilter(A, w);
```

Original

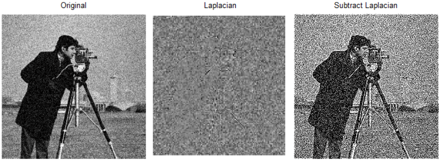
Laplacian

Subtract Laplacian

Laplacian Filter

- If the image is noisy, the Laplacian filter will pick up on the noise.
- Subtracting the Laplacian will just make the noise worse.

```
A = imnoise(A);   % Add noise to image.
w = fspecial('laplacian', 0);
A = double(A);    B = A - imfilter(A, w);
```



Prewitt Filter

- The Prewitt filter is designed to detect horizontal edges.
- Its transpose detects vertical edges.

w = fspecial('prewitt')

1	1	1
0	0	0
-1	-1	-1

```
A =double(A);
H = imfilter(A,w);
V = imfilter(A,w');
```

A

H

V

Difference of Gaussians (DoG)

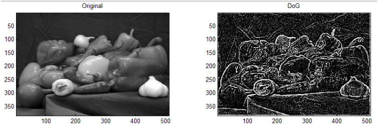
- We can extract features of an image f by subtracting images blurred at two different levels.

$$G_{\sigma_1} * f - G_{\sigma_2} * f \quad \text{where} \quad \sigma_1 < \sigma_2$$

Slightly blurry image

Very blurry image

```
G1 = fspecial('gaussian',[7,7],0.2);
G2 = fspecial('gaussian',[7,7],1.5);
DoG = imfilter(A,G1) - imfilter(A,G2);
```

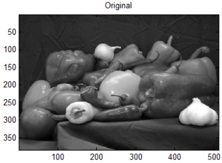


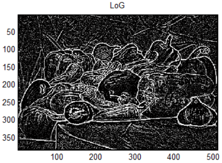
Laplacian of Gaussian (LoG)

- A LoG filter is another popular way to locate image features.

```
w = fspecial('log',  [5,5], 0.8)
```

0.0094	0.0470	0.0742	0.0470	0.0094
0.0470	0.0933	-0.0765	0.0933	0.0470
0.0742	-0.0765	-0.7770	-0.0765	0.0742
0.0470	0.0933	-0.0765	0.0933	0.0470
0.0094	0.0470	0.0742	0.0470	0.0094





Unsharp Masking

- The curiously named Unsharp Filter enhances the edges in an image by subtracting off a Gaussian blurred image.

$$U * f = f - \alpha\,(G_\sigma * \mathrm{f})$$

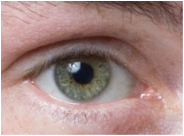
- The unsharp filter is large in the middle and subtracts off the surrounding pixels.

```
w = fspecial('unsharp', 0.5)
```

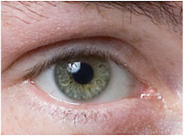
-0.3333	-0.3333	-0.3333
-0.3333	3.6667	-0.3333
-0.3333	-0.3333	-0.3333

Unsharp Masking

Original



Small Alpha



Large Alpha

- Unsharp masking is a ***deblurring*** operation, but it is **not** really a ***deconvolution*** operation since it does not take into account the blur process.

The Convolution Theorem

- The Fourier Transform $\mathcal{F}$ of a sequence f_n is defined as

$$\mathcal{F}(f)_k = \sum_{n=0}^{N-1} f_n \; e^{-\frac{2\pi i k n}{N}} \qquad k = 0, \ldots, N-1$$

- A convolution is equivalent to pointwise multiplication of the Fourier transforms.

$$\mathcal{F}[f * w] = \mathcal{F}(f)\, \mathcal{F}(w)$$

- Then taking the Inverse Fourier Transform $\mathcal{F}^{-1}$ gives

$$f * w = \mathcal{F}^{-1}[\mathcal{F}(f)\, \mathcal{F}(w)]$$

- A convolution on an image with N pixels can be computed in O(N logN) time.
- This means linear filters can be computed very quickly. This trick is built into the code of the imfilter command.

Point Spread Function (PSF)

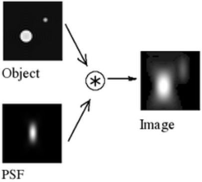
- Every camera will have add some amount of blur.
- The <u>Point Spread Function</u> (PSF) is how the camera will respond to a point source.
- We think of the PSF as the blur kernel or filter weights inherent to that physical sensor.

$$g = f * w$$

Observed Image
(camera output)

Ideal Image
(no blur)

Point Spread Function
(camera blur)



Deconvolution

$$g = f * w$$

- Given *g* (observed image) and *w* (camera PSF), the goal of deconvolution is to recover the ideal image *f*.
- If we apply Fourier Transforms to both sides, the Convolution Theorem says:

$$\mathcal{F}(g) = \mathcal{F}[f * w] = \mathcal{F}(f)\,\mathcal{F}(w)$$

- Solve for *f* using the Inverse Fourier Transform $\mathcal{F}^{-1}$

$$\boxed{\mathcal{F}(f) = \frac{\mathcal{F}(g)}{\mathcal{F}(w)} \quad \Longrightarrow \quad f = \mathcal{F}^{-1}\left[\frac{\mathcal{F}(g)}{\mathcal{F}(w)}\right]}$$

- This process is called Wiener Deconvolution.

Wiener Deconvolution

- The Matlab command deconvwnr performs deconvolution with a given PSF.

```
PSF = fspecial('gaussian', [5,5], 0.8);
blurred = imfilter(A, PSF);
deblurred = deconvwnr(blurred, PSF);
```

Create a Gaussian PSF.
Blur the image A with this PSF.
Then try to remove the blur.

Blurred Image

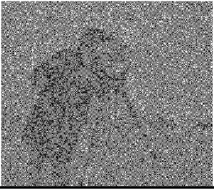
Wienered

Wiener Deconvolution

- Wiener Deconvolution is usually not practical because it is extremely sensitive to noise.

```
PSF = fspecial('gaussian', [5,5], 0.8);
blurred = imnoise(imfilter(A, PSF));
deblurred = deconvwnr(blurred, PSF);
```

Add small amount of noise to the blurred image.



Summary

- A linear filter is a weighted average around each pixel's neighborhood. We can write the filter as a convolution:

$$g = f * w.$$

- Linear filters have many uses:
  - Mean, Gaussian: Remove noise, but also blur the image.
  - Laplacian, Prewitt, DoG, LoG: Detect edges and other features.
  - Unsharp: Sharpens images. Can get similar result by subtracting a Laplacian filtered image.
- Wiener Deconvolution can remove blur, but only if the image was noise-free or if we know the signal-to-noise ratio (SNR).