ECE 484 Digital Image Processing

Lec 05 - Linear Filtering

Zhu Li

Dept of CSEE, UMKC

Office: FH560E, Email: lizhu@umkc.edu, Ph: x 2346.

http://l.web.umkc.edu/lizhu

Outline

- Recap of Lec 04
- Linear Filters
- Smoothing and Edge Detection
- Accelerating Linear Filters
- Summary

Gamma Correction - Adjust dynamic ranges

- Matching Display characteristics

$$v = c \cdot u^{\gamma}$$

  - power-law response functions in practice

    CRT Intensity-to-voltage function has $\gamma \approx 1.8\sim2.5$

    Camera capturing distortion with $\gamma_c = 1.0\text{-}1.7$

    Similar device curves in scanners, printers, …

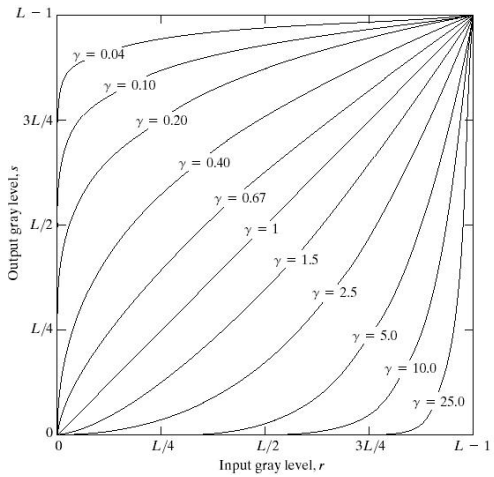


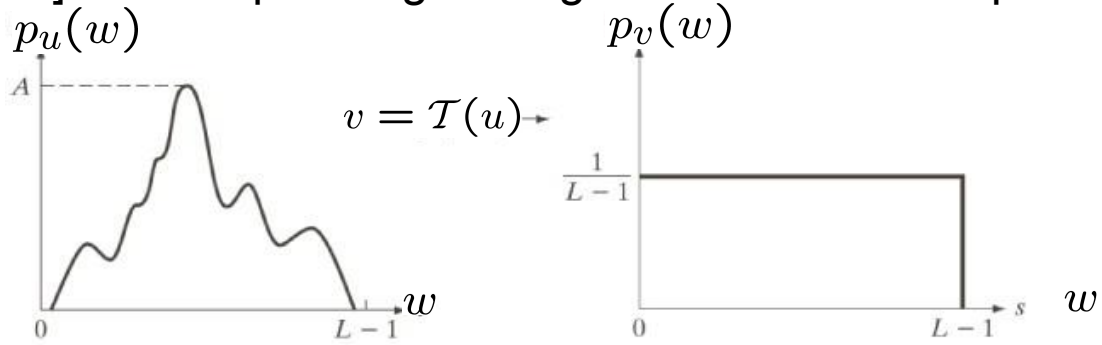
FIGURE 3.6 Plots of the equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases).

- power-law transformations are also useful for general purpose contrast manipulation

Histogram Equalization

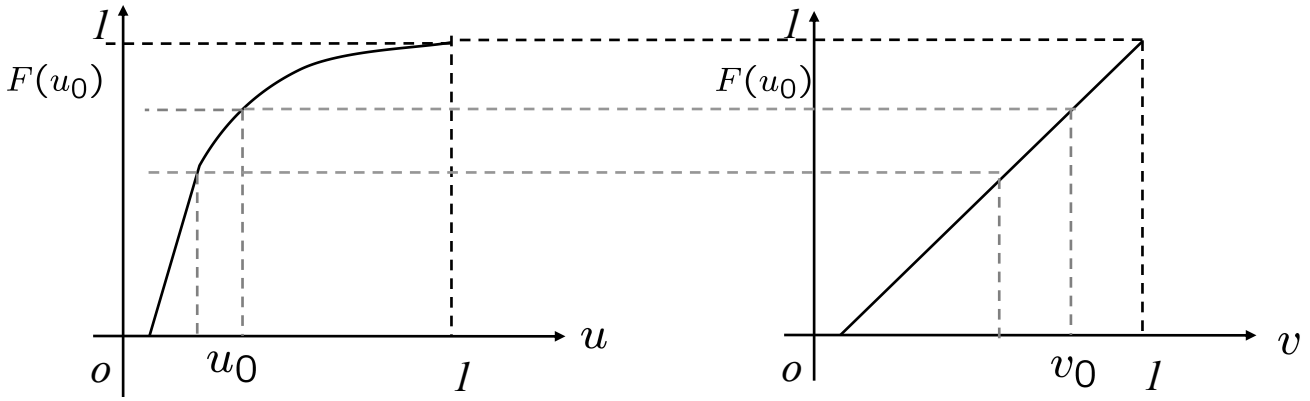
- Objectives
  - goal: map the each luminance level to a new value such that the output image has approximately uniform distribution of gray levels
  - two desired properties
    - monotonic (non-decreasing) function: no value reversals
    - [0,1]→[0.1] : the output range being the same as the input range

pdf



$$F_u(U) = P(U < u) = \int_0^u p_u(w)dw \qquad F_v(V) = P(V < v) = \int_0^v p_v(w)dw$$

cdf



Histogram Equalization

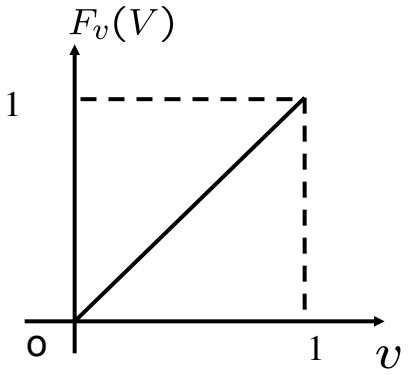
Algorithm

- make

$$v = F_u(u) = P(U < u) = \int_0^u p_u(w)dw$$

- show

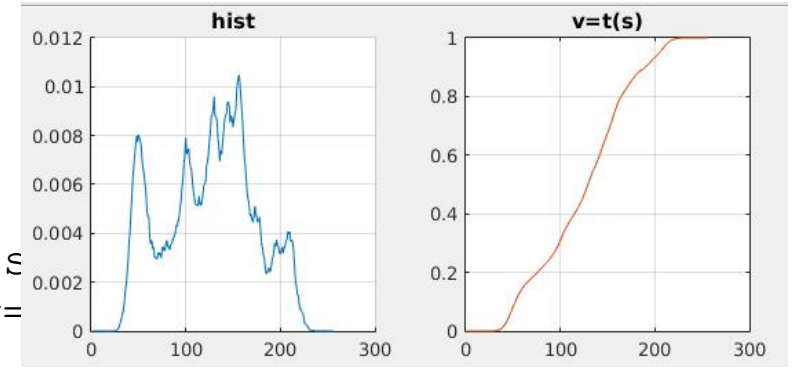
$$F_v(V) = v, \quad v \in [0, 1]$$



$$F_v(V) = P(V < v) = P(F_u(U) < v)$$
$$= P(U < F_u^{-1}(v)) = F_u(F_u^{-1}(v)) = v$$

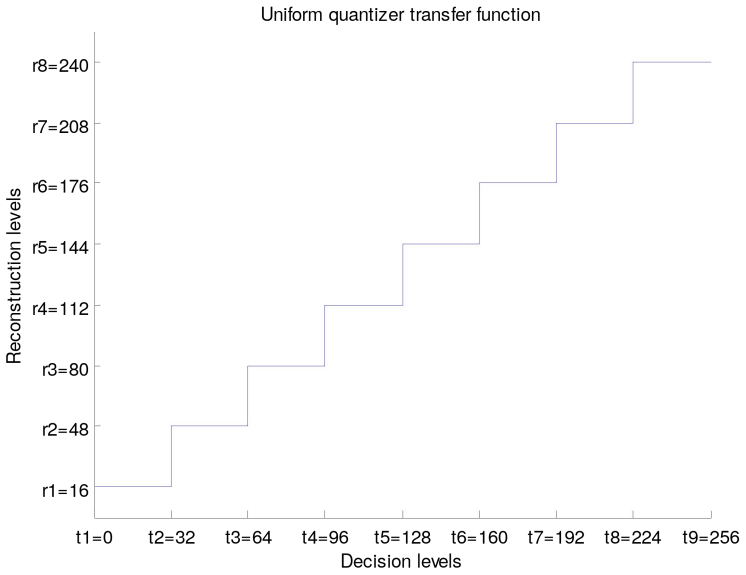
Matlab

```
im = imread('lena.png');
h1 = imhist(rgb2gray(im));
h1=h1/sum(h1);
cf1 = cumsum(h1);
subplot(1,2,1); plot(h1); title('his
subplot(1,2,2); plot(cf1); title('v=
```



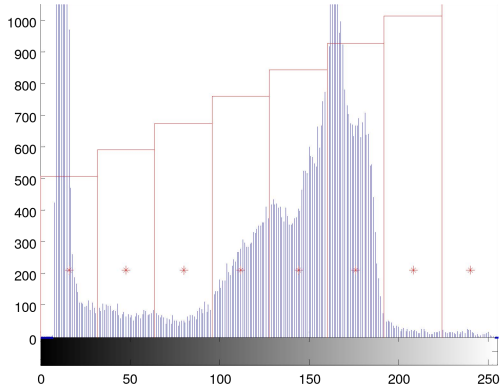
Uniform Quantization: B=3, L=8

- Finer quantization: B=3 => L=8; false contours present



Non-quantized image

Quantized image

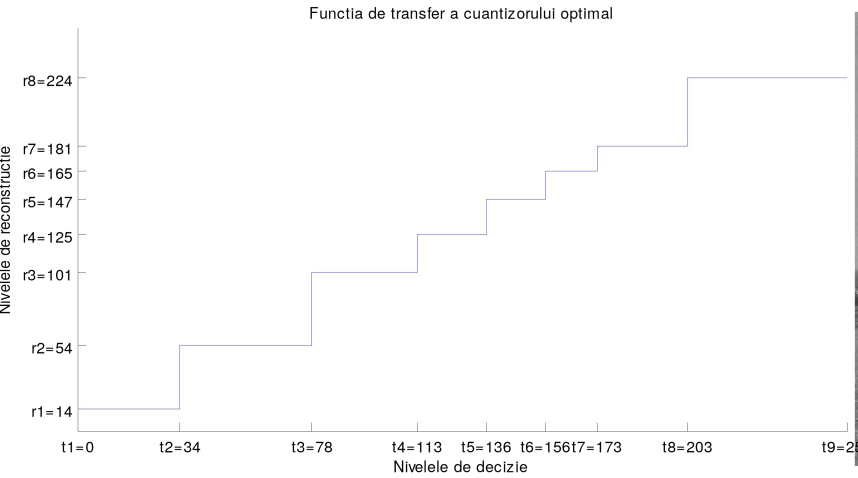


histogram

Quantization error; MSE=7.33

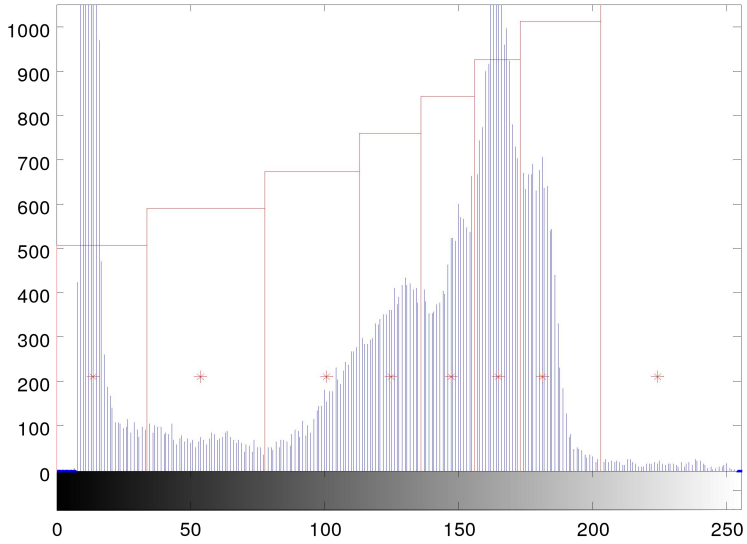
LMQ

- LMQ: B=3, L=8



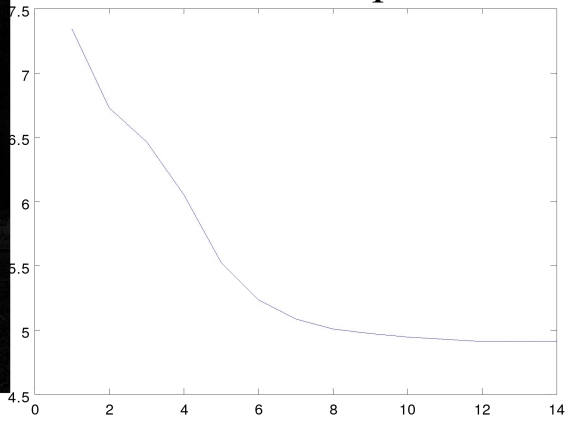
Non-quantized image

Quantized image



The quantization error; MSE=5

The evolution of MSE in the optimization, starting from the uniform quantizer

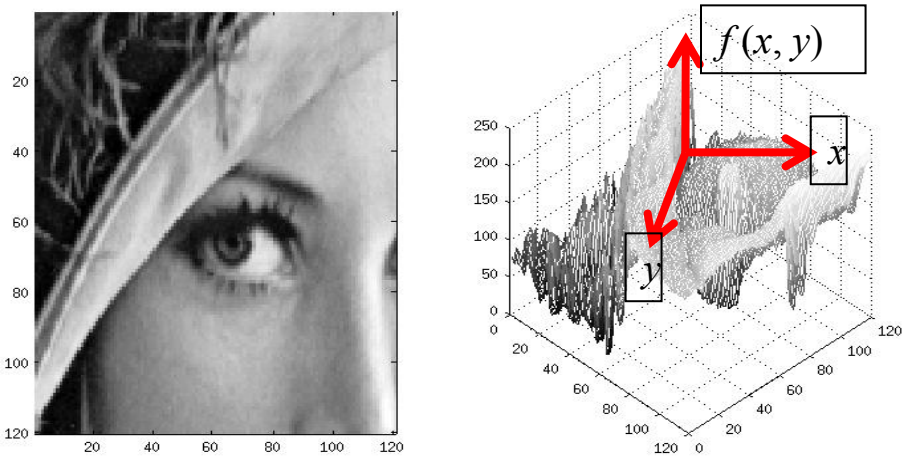


Outline

- Recap of Lec 04
- Linear Filters
- Smoothing and Edge Detection
- Accelerating Linear Filters
- Summary

What is an image?

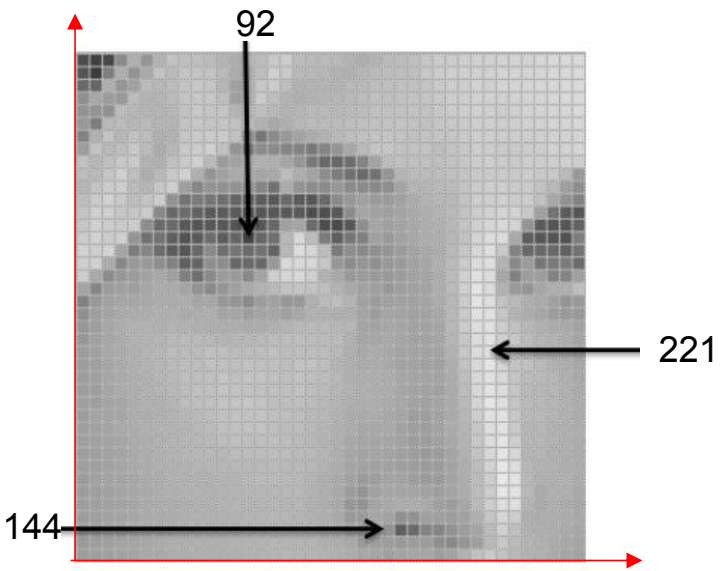
- We can think of a (grayscale) image as a **function**, f, from R^2 to R (or a 2D *signal*):
  - $f(x,y)$ gives the **intensity** at position (x,y)

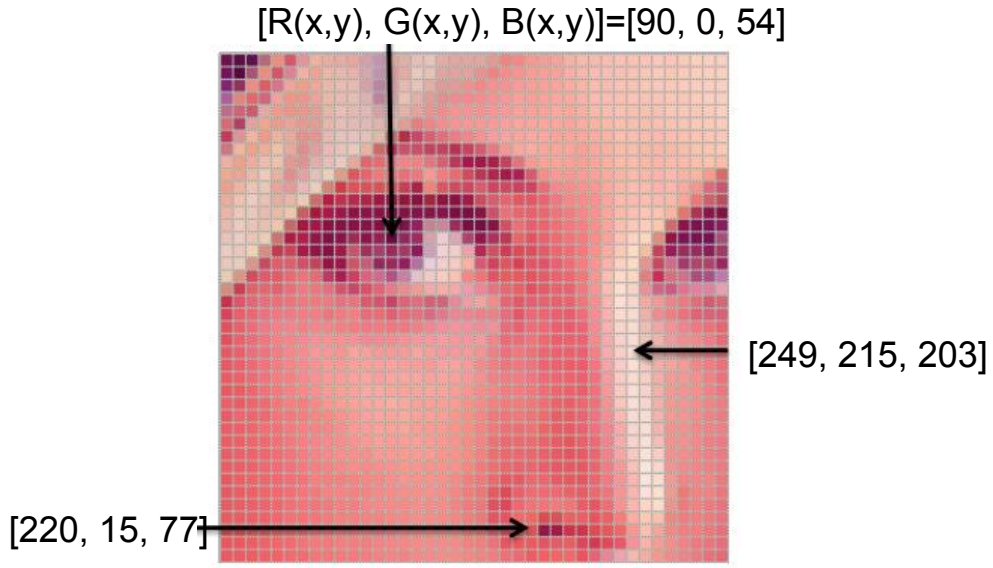


  - A **digital** image is a discrete (**sampled**, **quantized**) version of this function

Image as a function: I=f(x,y);

- A grid (matrix) of intensity values





(common to use one byte per value: 0 = black, 255 = white)

Motivation: Why Filtering ?

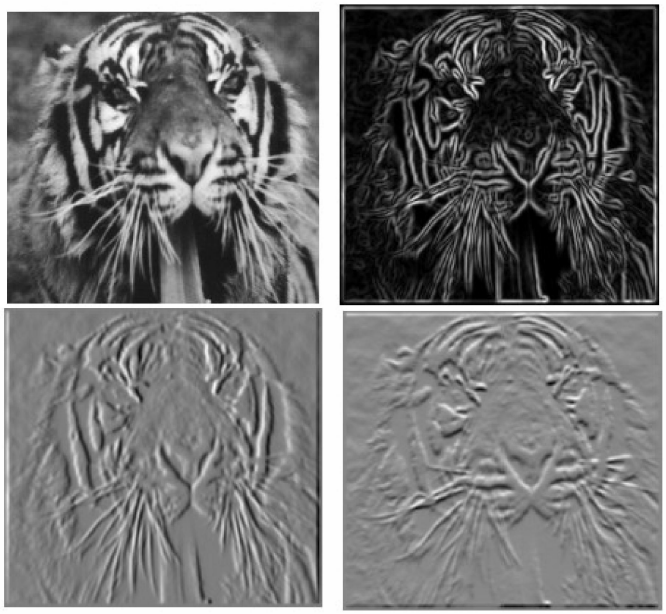
- Some use cases…

De-noising

Salt and pepper noise

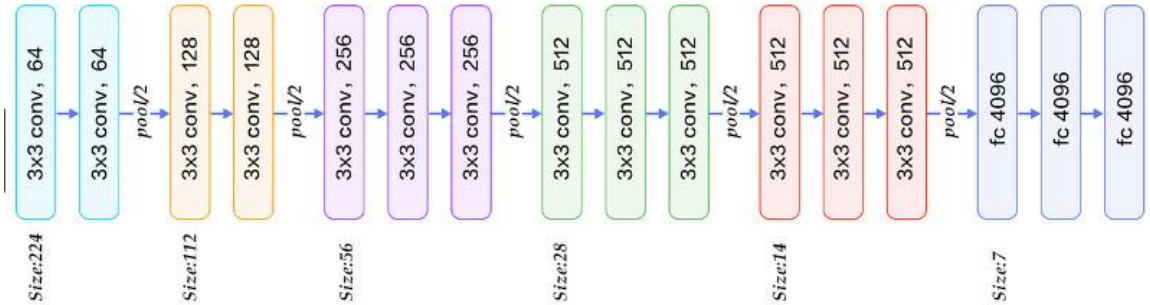
Super-resolution

Edge detection



In-painting

Bertamio et al



Deep Learning with Conv Nets: VGG16

Image filtering – Looking at pixel neighbors

- Modify the pixels in an image based on some function of a local neighborhood of each pixel
  - Discrete form:

$$f[n,m] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow g[n,m]$$

$$g = \mathcal{S}[f], \quad g[n,m] = \mathcal{S}\{f[n,m]\}$$

$$f[n,m] \xrightarrow{\mathcal{S}} g[n,m]$$

10	5	3
4	5	1
1	1	7

Local image data

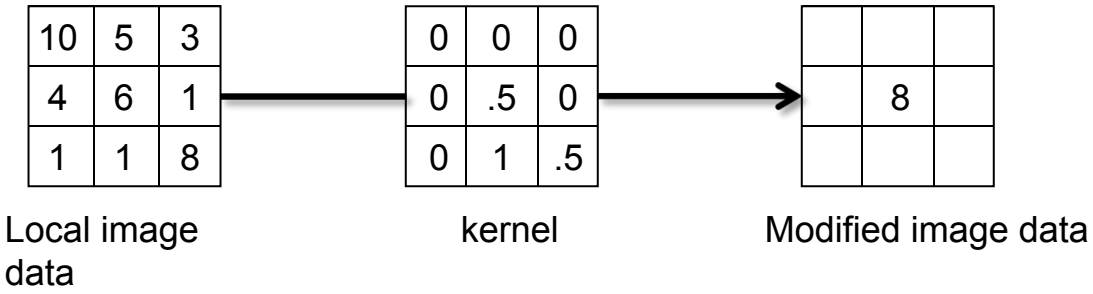
Some function

	7	

Modified image data

Linear filtering

- One simple version: linear filtering (cross-correlation, convolution)
  - Replace each pixel by a linear combination of its neighbors
- The prescription for the linear combination is called the "*kernel*" (or "mask", "filter")



Source: L. Zhang

Cross-correlation

- Cross-Correlation:
  - No flipping of kernel

Let ***F*** be the image, ***H*** be the kernel (of size 2k+1 x 2k+1), and ***G*** be the output image

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

This is called a **cross-correlation** operation:

$$G = H \otimes F$$

Convolution

- Same as cross-correlation, except that the kernel is "flipped" (horizontally and vertically), called convolution

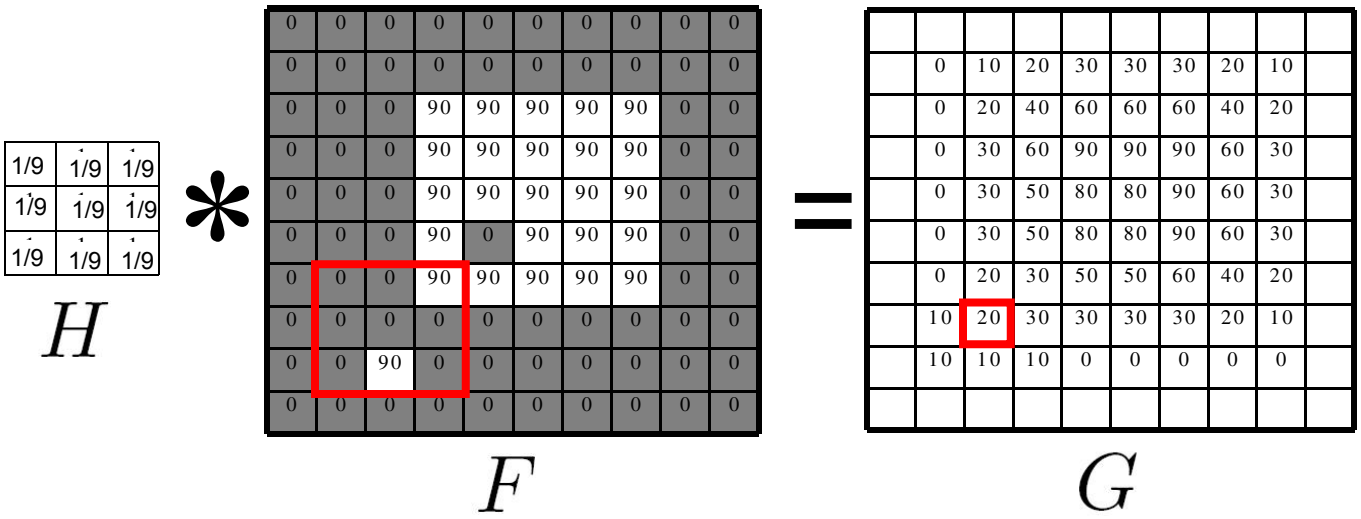
$$G = H * F$$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

- Convolution is **commutative** and **associative**
  - ***F=G*h + G*f = G*(h+f)***
  - ***F=G*h*f = G*(h*f)***

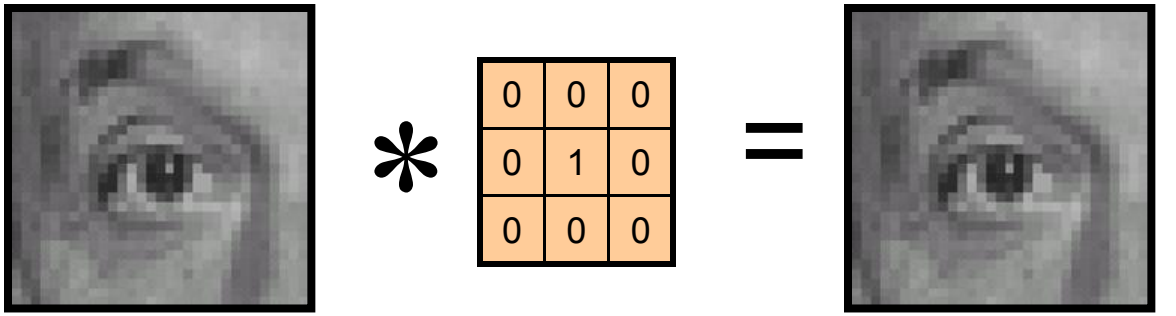
Mean filtering

- Find the average



Linear filters: examples

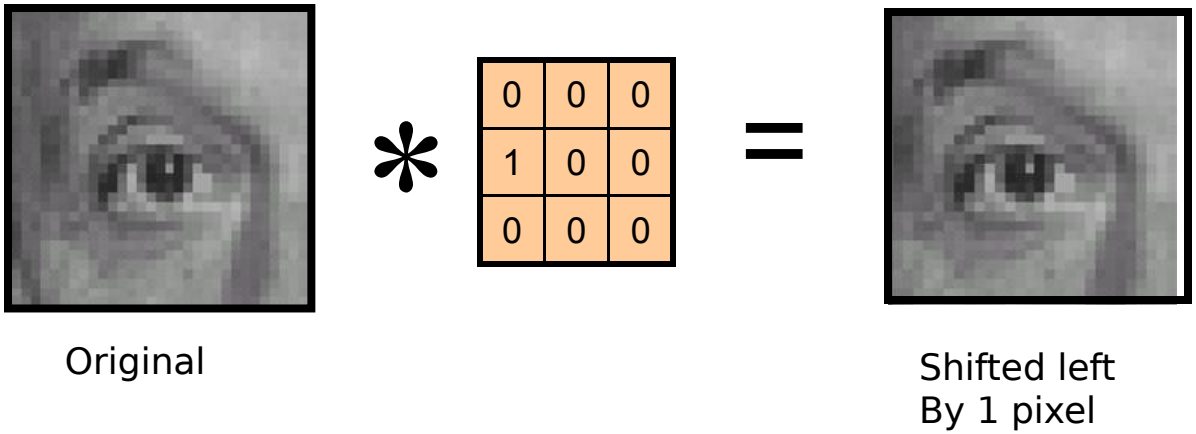
- Identity filter:



Source: D. Lowe

Linear filters: examples

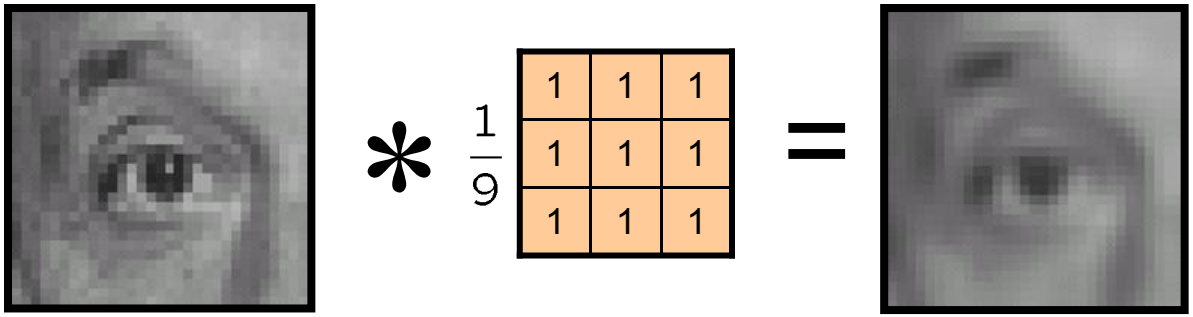
- Shift Filter



Source: D. Lowe

Linear filters: examples

- Average/Blurring



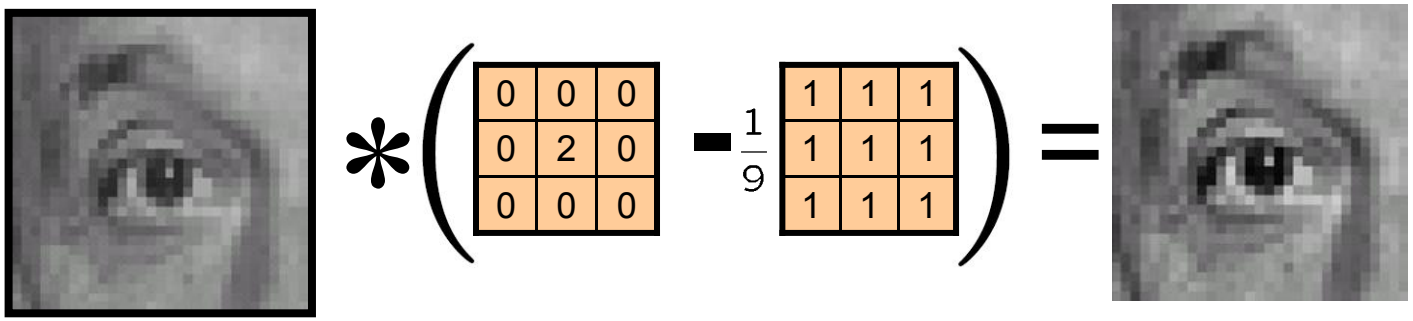
Original

Blur (with a mean filter)

Source: D. Lowe

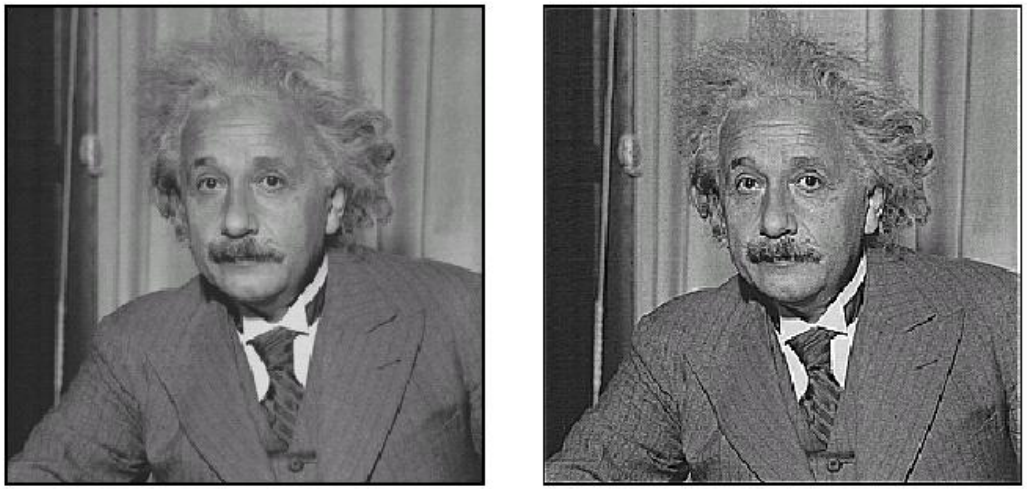
Linear filters: examples

- Sharpen: remove the average.



Original

Sharpening filter



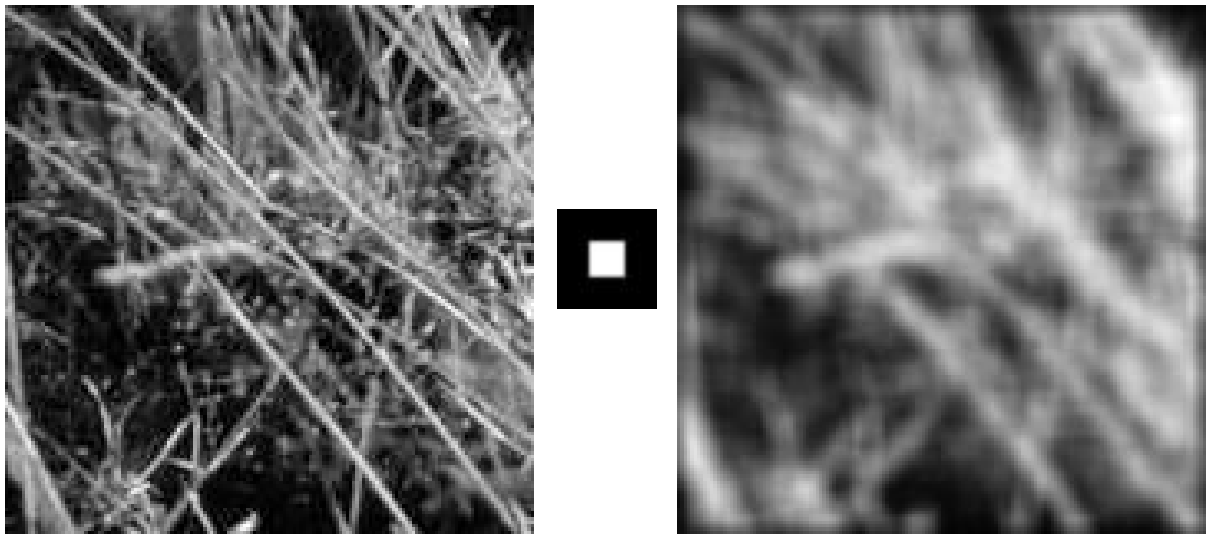
before

after

ve

Smoothing with box filter

- Box Filter:



- Box filter in integral image domain:
  - much faster, just 4 ADD operations.

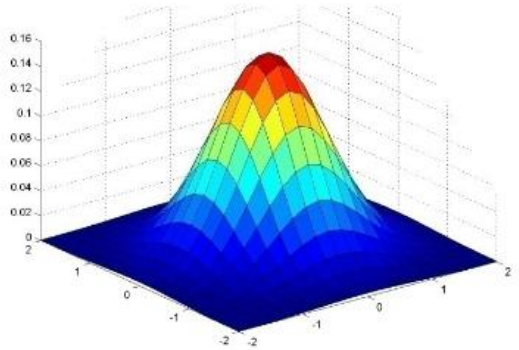
Source: D. Forsyth

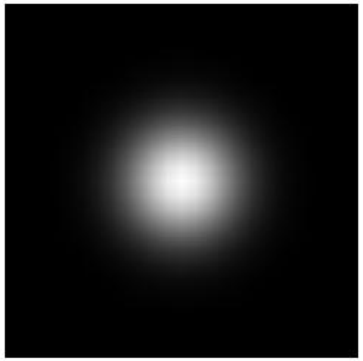
Gaussian Kernel

- Gaussian Kernel with scale σ

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

Matlab: h=**fspecial**('gaussian', 5, 1.0);



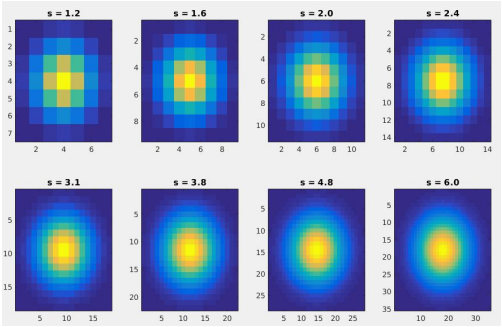


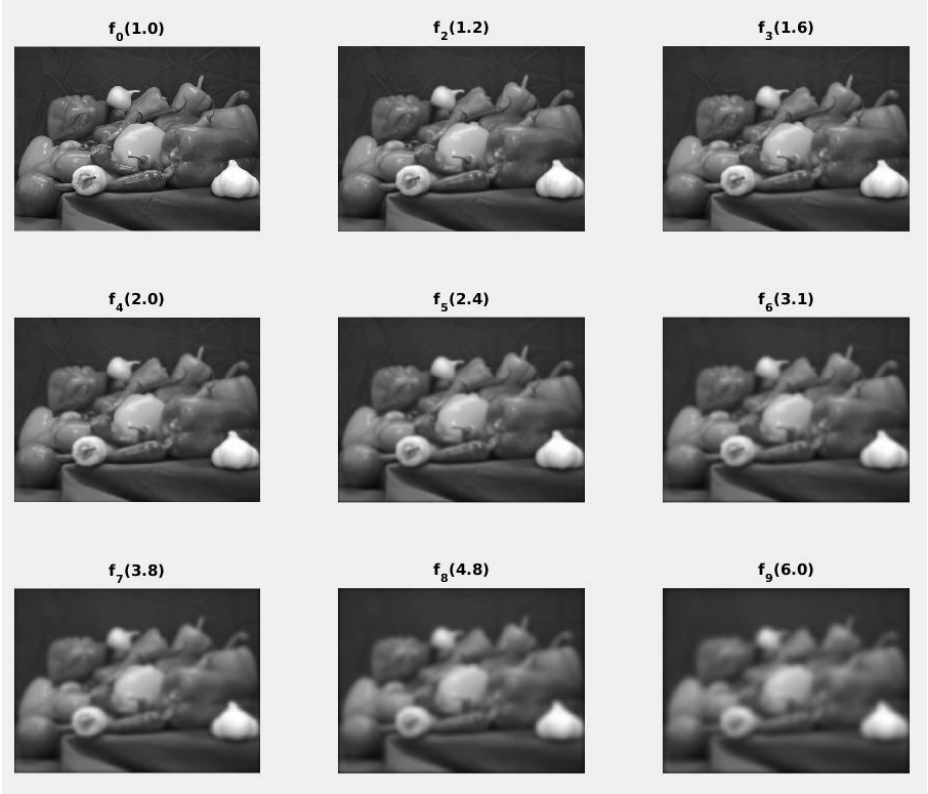
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

Gaussian Filters – A Scale Space Approximation

- Gaussian Blur



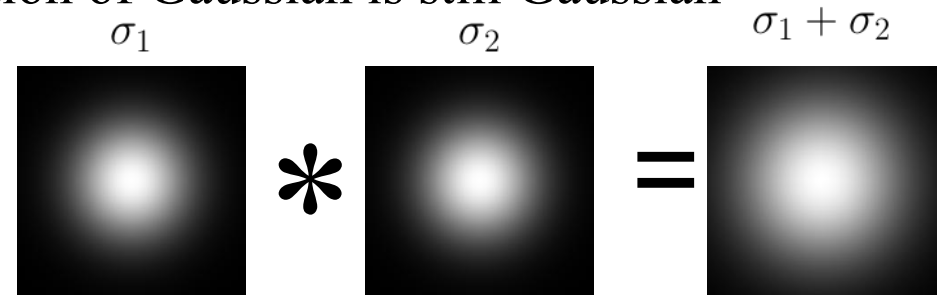


Matlab:

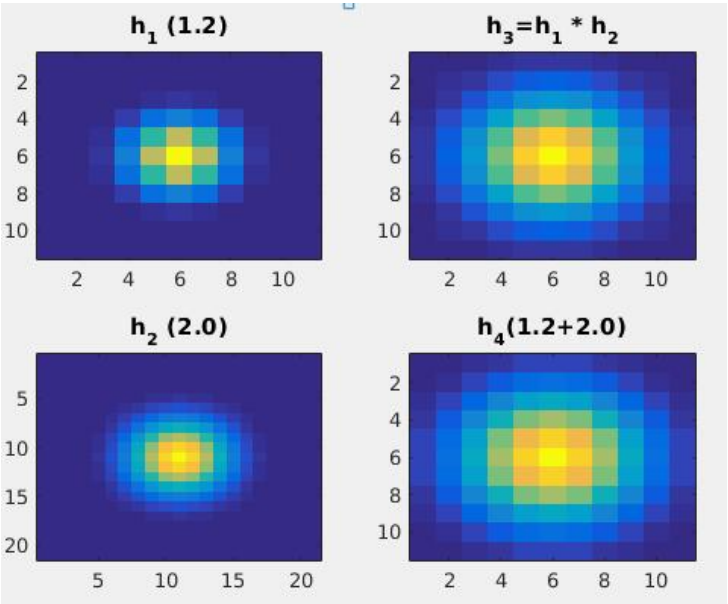
```
n=8;s = 1.25.^[1:n]; m=fix(6.*s);
figure(30);
for k=1:8
    subplot(2,4,k); h = fspecial('gaussian',
m(k), s(k));
    imagesc(h); title(sprintf('s = %1.1f',
s(k)));
end

figure(31);
subplot(3,3,1); imshow(im);
title('f_0(1.0)');
for k=1:n
    subplot(3,3,k+1);
    h = fspecial('gaussian', m(k), s(k));
    f = imfilter(im, h);
    imshow(f);
    title(sprintf('f_%d(%1.1f)', k+1, s(k)));

end
```

Gaussian filter properties

- Successive Gaussian filtering
  - convolution of Gaussian is still Gaussian



new kernel sigma: h1+ h2



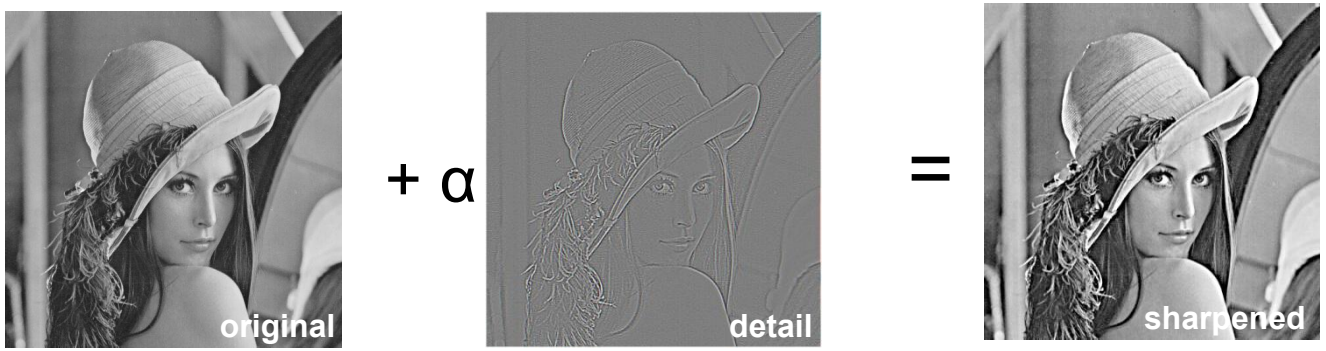
Matlab:

```
h1=fspecial('gaussian', 11, 1.2);
h2=fspecial('gaussian', 11, 2.0);
h3 = conv2(h1, h2);
h4=fspecial('gaussian', 11, 2.0+1.2);
```

Sharpening revisited

- What does blurring take away?

Let's add it back:



Source: S. Lazebnik

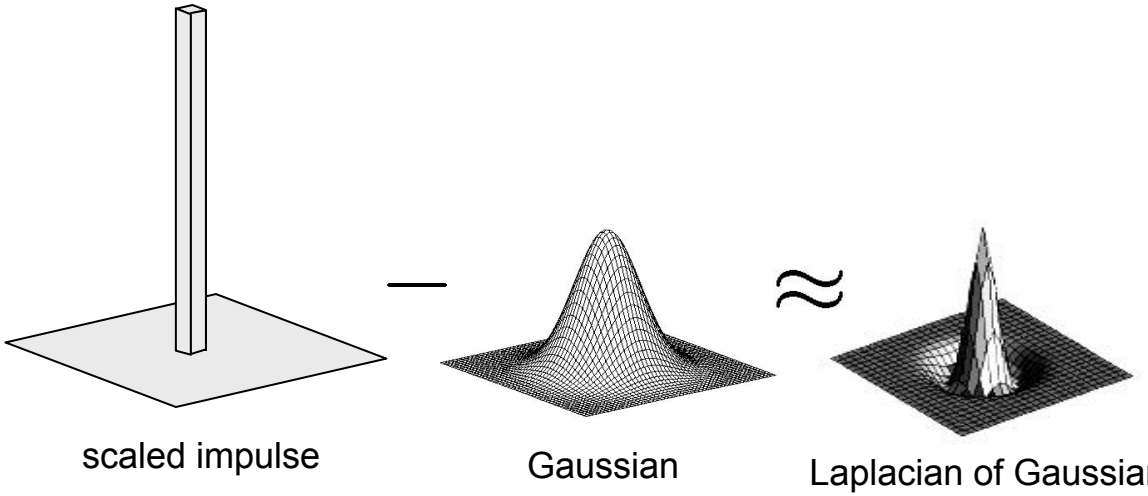
Sharpen filter - LoG

- Laplacian of Gaussian

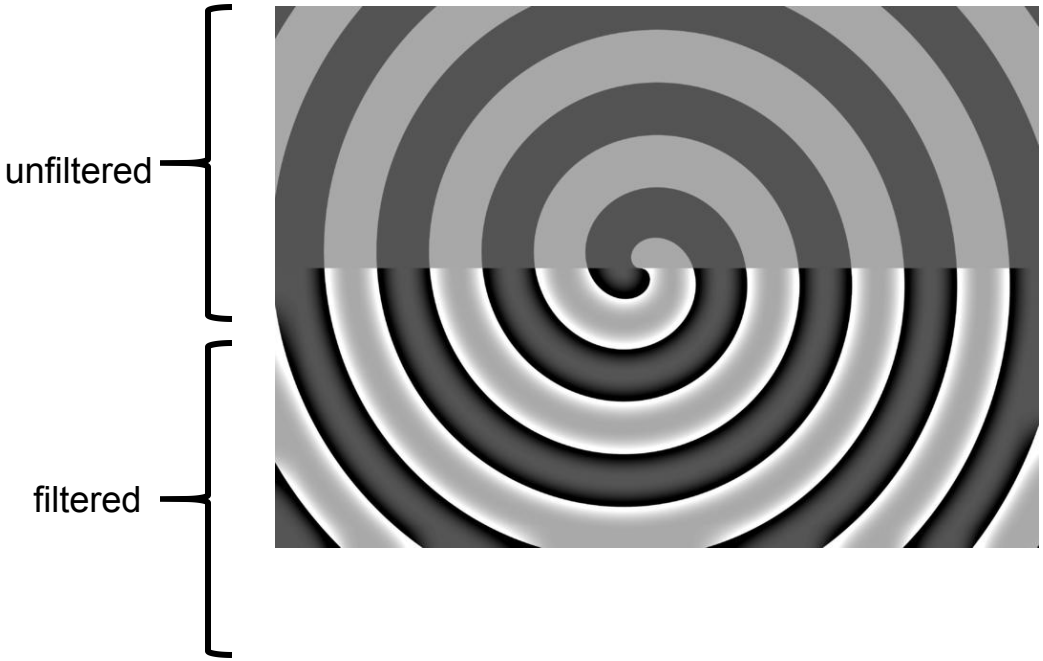
$$F + \alpha\,(F - \underbrace{F * H}_{\text{blurred image}})$$

image

unit impulse (identity)

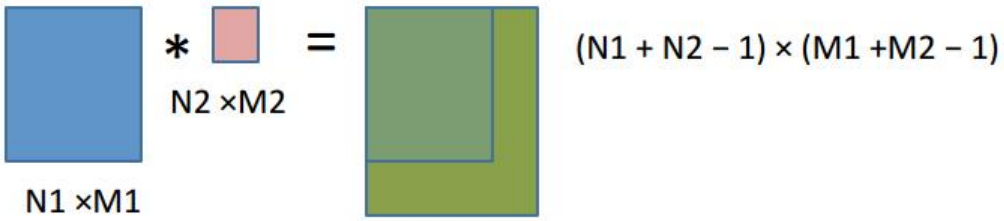


Sharpen filter



Filtering in Matlab

- Area of support for the operations



- To give the same n1 x m1 output, need to padding the edge
  - Default is zero padding
  - Also replicate the last edge pixel
  - Or, mirroring (used in MPEG codec)

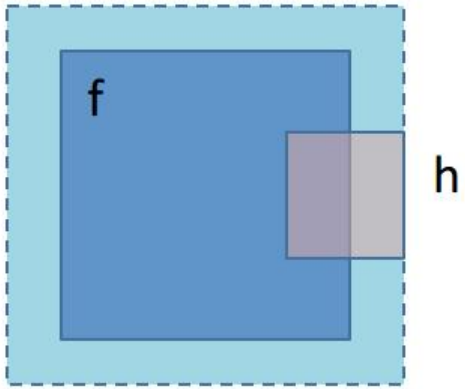
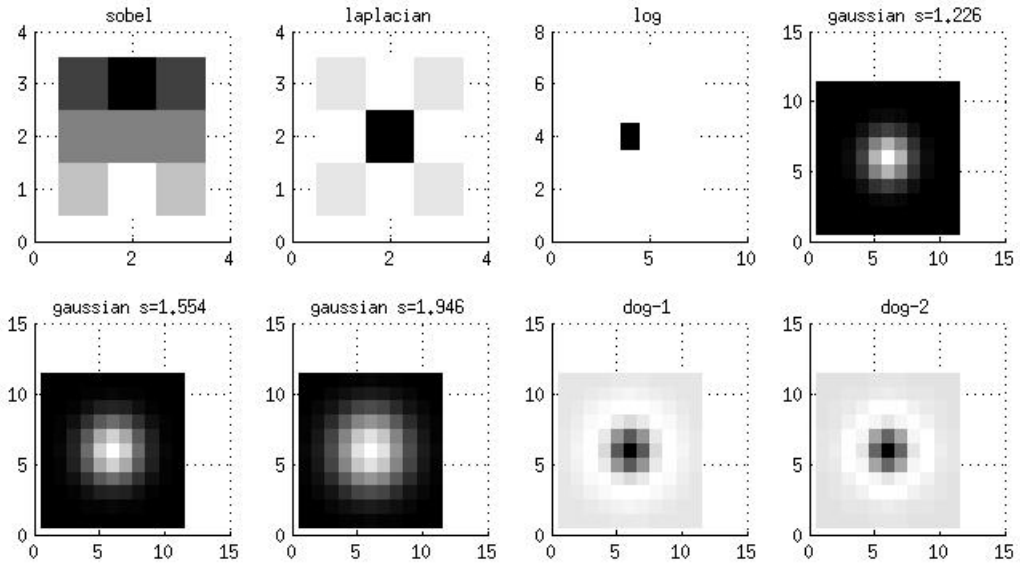


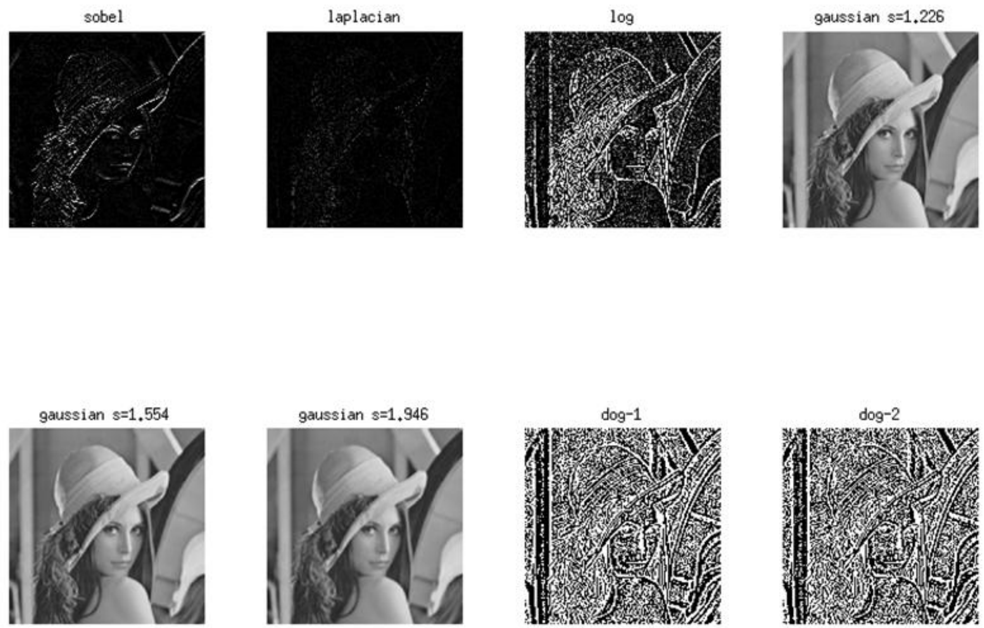
Image Filtering, Sweet Deal with Matlab

- It is such a nice tool
  - Main filter operation: im2 = imfilter(im, h, 'replicate')
  - Design your filter: h= fspecial('filter_type', kernel_size, options)
- Filter Design Examples:
  - Sobel
  - Laplacian, Laplacian of Gaussian
  - Gaussian, Difference of Gaussian (SIFT)



Matlab Image Filtering Example

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% image filters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
im = imread('../pics/Lenna.png');
im = rgb2gray(im);
im1 = double(im(201:320, 201:320));

% edge filters
h{1} = fspecial('sobel');
h{2} = fspecial('laplacian', 0.25);
h{3} = fspecial('log', 7, 0.25);
% gaussians
sigmas = [1.226, 1.554, 1.946];
for k=1:length(sigmas)
   h{3+k} = fspecial('gaussian',11, sigmas(k));
end
% diff of gaussian
h{7} = (h{6} - h{4}); h{7} = h{7}/sum(sum(h{7}));
h{8} = h{5} - h{4}; h{8} = h{8}/sum(sum(h{8}));

for k=1:8
   fprintf('\n k=%d', k);
   figure(26);  subplot(2,4,k); grid on; hold on; colormap('gray'); imagesc(h{k});
   figure(27);  subplot(2,4,k); imshow(imfilter(im, h{k}, 'replicate'));
end
```



Outline

- Recap of Lec 04
- Linear Filters
- Smoothing and Edge Detection
- Accelerating Linear Filters
- Summary

Filtering Properties

- Linear operation that is
  - Shift-Invariant:

    f(m-k, n-j)*h = g(m-k, n-j), if f*h=g

  - Associative:

    $f*h_1*h_2 = f*(h_1*h_2)$

    this can save a lot of complexity

  - Distributive:

    f*h1 + f*h2 = f*(h1+h2)

    useful in SIFT's DoG filtering.

Separability of the Gaussian filter

- Gaussian smoothing is separable

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{y^2}{2\sigma^2}}\right)$$

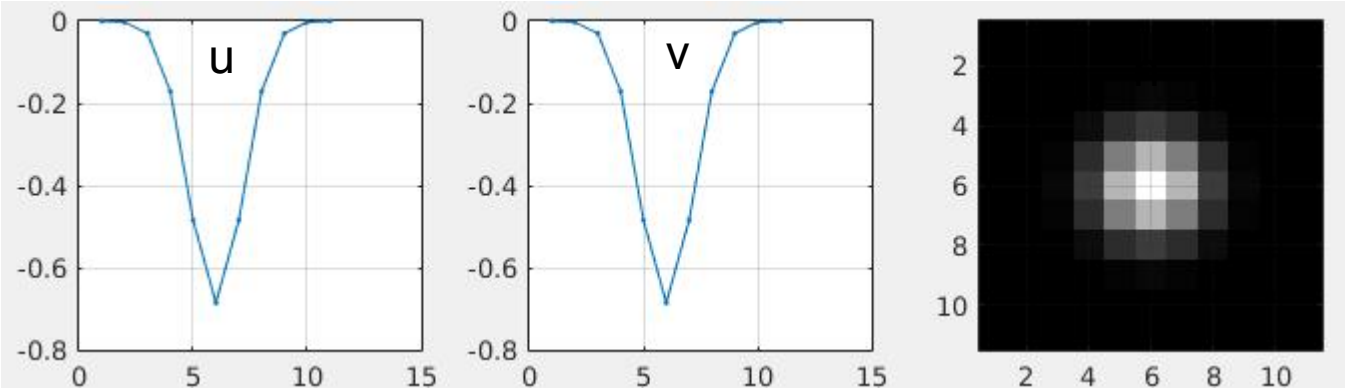
The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Separatable Fitler

- Gaussian filter is separatable:

$$h = u \otimes v$$



- verify via SVD:
  - h = fspecial('gaussian', 11, 1.2);
  - [u,s,v]=svd(h); plot(diag(s));
- Benefits of separable filters: much faster
  - Implement as 1D filter by row with u, then transpose filter again by v.
  - instead of O(m*n*k*k), O(m*n*k)

Separability example

- Separable filtering complexity

2D convolution (center location only)

$$\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} * \begin{bmatrix}2&3&3\\3&5&5\\4&4&6\end{bmatrix}$$

The filter factors into a product of 1D filters:

$$\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} = \begin{bmatrix}1\\2\\1\end{bmatrix} \times \begin{bmatrix}1&2&1\end{bmatrix}$$

Perform convolution along rows:

$$\begin{bmatrix}1&2&1\end{bmatrix} * \begin{bmatrix}2&3&3\\3&5&5\\4&4&6\end{bmatrix} = \begin{bmatrix}&11&\\&18&\\&18&\end{bmatrix}$$

Followed by convolution along the remaining column:

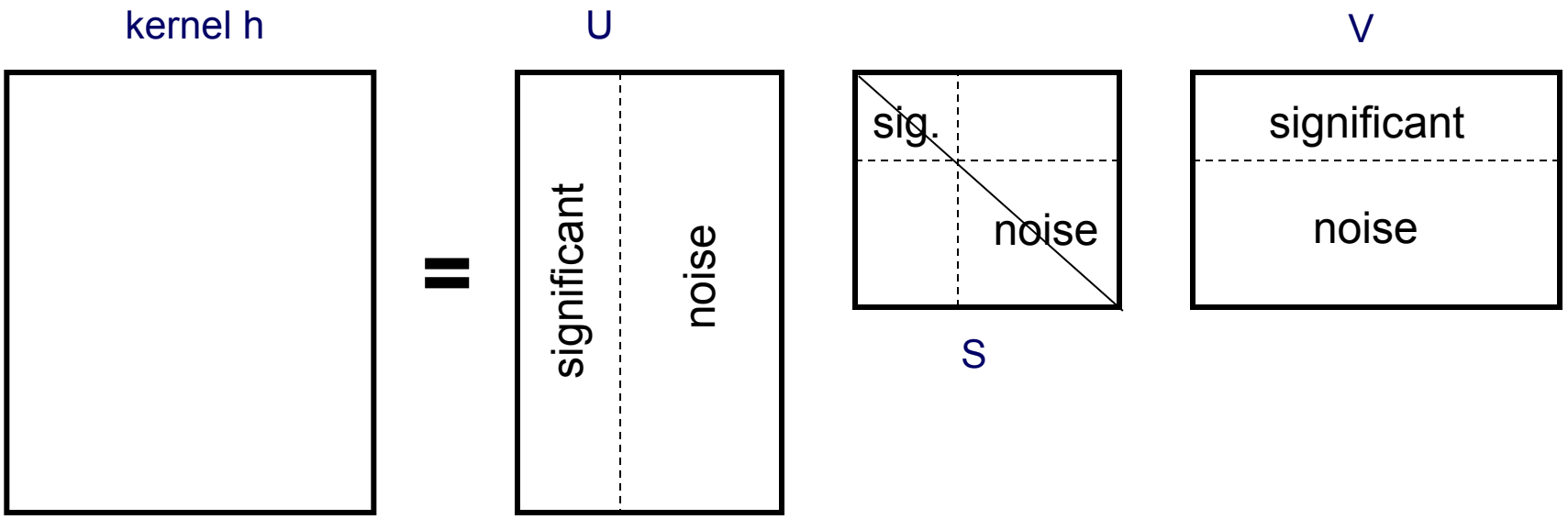
$$\begin{bmatrix}1\\2\\1\end{bmatrix} * \begin{bmatrix}&11&\\&18&\\&18&\end{bmatrix} = \begin{bmatrix}&&\\&65&\\&&\end{bmatrix}$$

Approx. Non-Separable Filters

- k-SVD separable filters
  - fitler kernel is approximated by first SVD basis

$$[U, S, V] = svd(h)$$

$$h = \sum_k s_{k,k} u_k v_k T$$

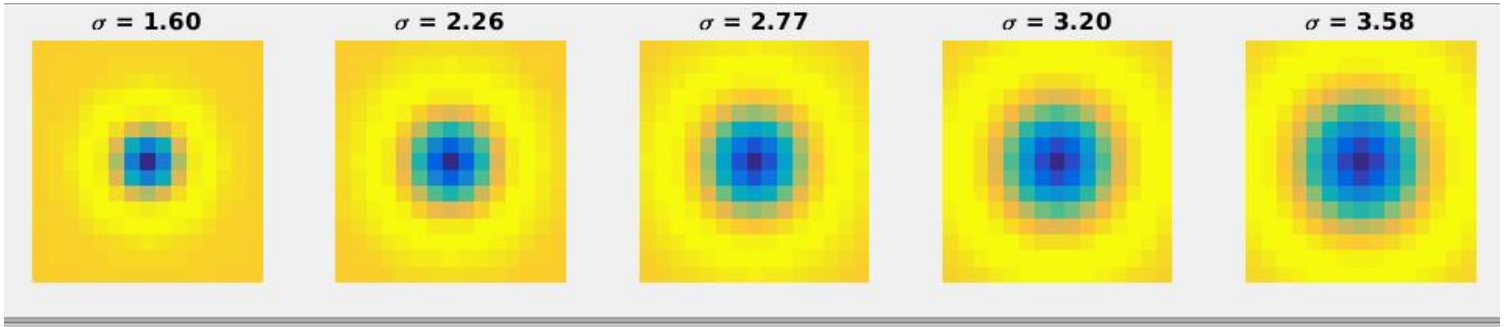


Matlab Implementation

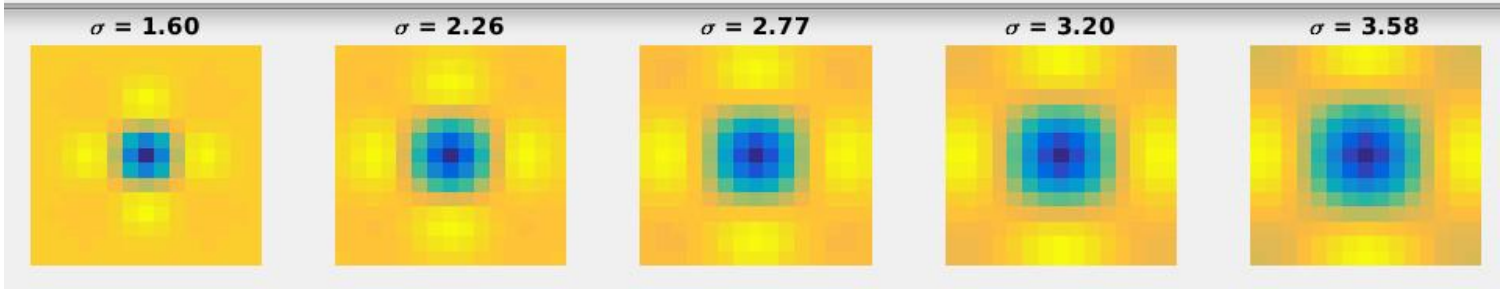
- DoG approximation with k-SVD separable fitlering

```
% k-SVD DoG filter
for k=1:length(sigma) -1
    [u,s,v]=svd(h_dog{k});
    % 1-SVD approx
    dog_svd{k} = s(1,1)*u(:,1)*v(:,1)';
    ss(k,:) = diag(s);
    figure(24); subplot(1,5,k); imagesc(h_dog{k});
    figure(25); subplot(1,5,k); imagesc(dog_svd{k});
end
```

dog



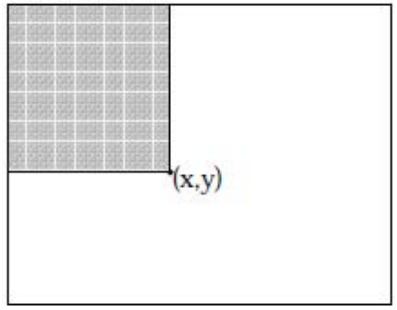
1-svd dog

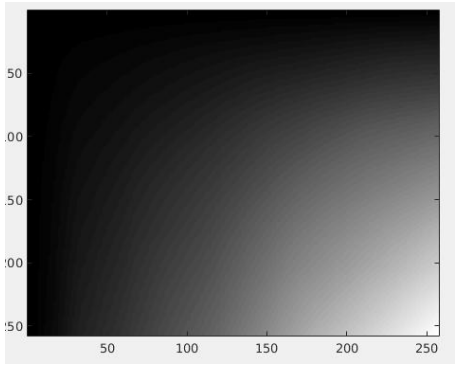


2D Box Filtering

- Integral Images:

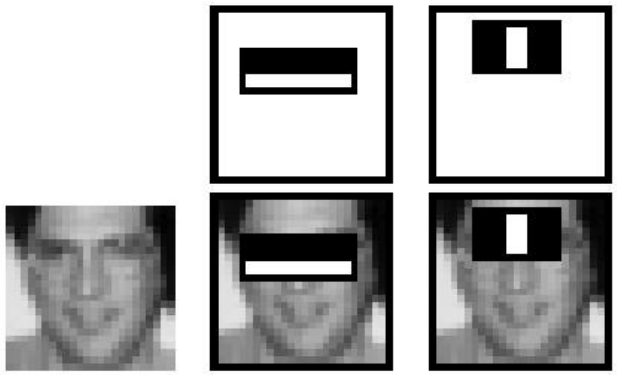
$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y')$$





lena integral image

- Fast 2D box filtering
  - applying a rectangle/box fitler can be computed as sum of 4 corners
  - much faster than convolution
- One of the key applications:
  - AdaBoost face detection



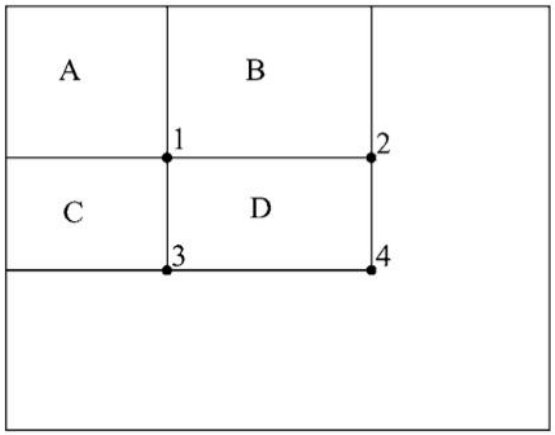


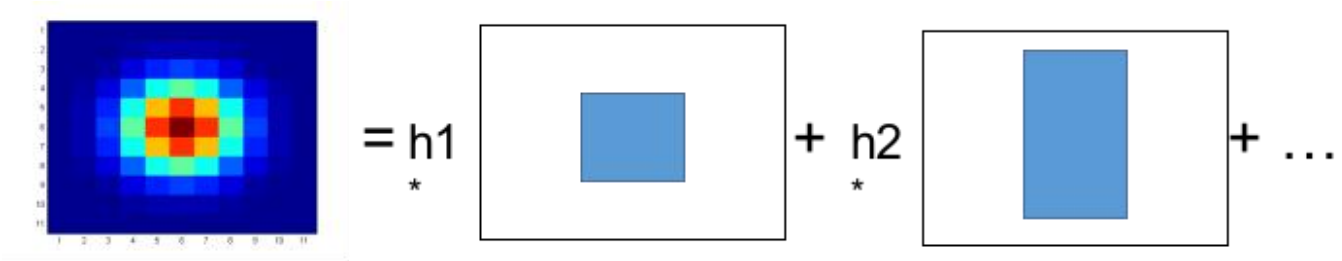
Figure 3. The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

Complexity Reduction

- The overhead of a 2D convolution can be significant, especially when the kernel is big.
  - Cost: $O(MNPQ)$ multiplications and $O(MNPQ)$ additions for an image of size MxN and a kernel of size PxQ.
- The complexity can be alleviated if the kernel is separable.
  - Cost: $O(MN(P+Q))$ multiplications and $O(MN(P+Q))$ additions.
- For large images and kernels the convolution can still be computationally expensive, e.g., detecting key points using Difference of Gaussians (DoG.)
- What if we could use Integral images and box filters to approximate the original filter?
  - The cost of computing an Integral image is $O(MN)$ additions
  - The cost of convolving with a k-box linear combination filters approximation is $O(MNK)$ multiplications and $O(4MNK)$ additions, where k is the number of boxes used for the approximation.

Approixmating Gaussian Filter

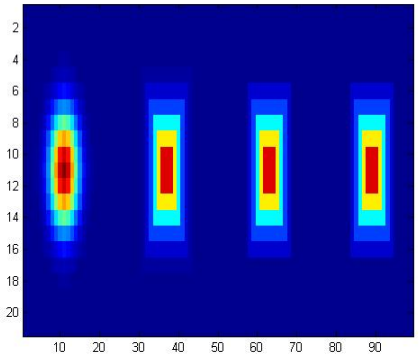
- Idea: A filter can be approximated as a linear combination of several box filters.

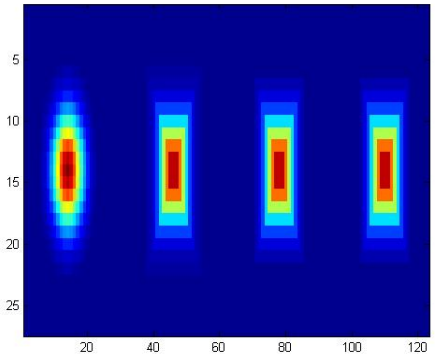


We need to find the set of weights {h_i}, such that:

- the number of boxes to use is as minimal as possible
- keep a low approximation error

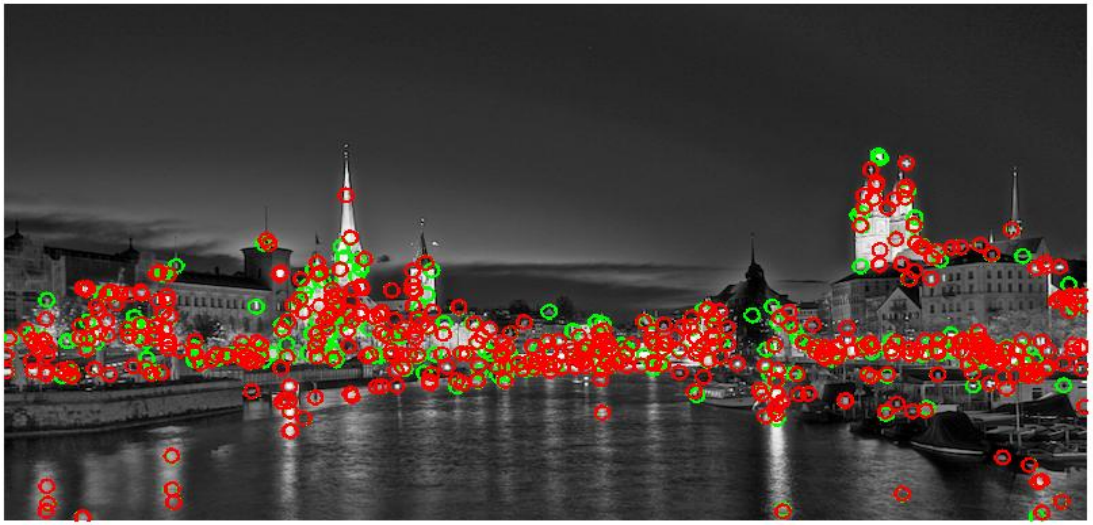
Sigma	Residuals			Num. of Boxes		
	LS	LS-LASSO	LASSO	LS	LS-LASSO	LASSO
1.249	0.05535	0.055372	0.055372	5	3	3
1.226273	0.057776	0.057791	0.057791	5	3	3
1.545008	0.035789	0.035802	0.035802	7	4	4
1.946588	0.024847	0.024848	0.024848	8	6	6
2.452547	0.018447	0.019152	0.019152	10	5	5
3.090016	0.014137	0.014175	0.014175	13	8	8





Application: CABOX detection

- We use the dictionary generation approach (3) and LASSO.



- The box filter (red circles) approach was able to detect **88%** of the features detected by the Gaussian approach (green circles).
- Another experiment with 190 images revealed that the box filter approach was able to detect **65%** of the features detected by Vlfeat.
- The criterion for declaring overlapping features is if their distance is less than 5 pixels (typically used in SfM)

Summary

- Linear Filters
  - involves a neighbourhood in output pixel values computing
  - characterized by a kernel: y = conv2(x, h)
  - Shift-Invariant:

    f(m-k, n-j)*h = g(m-k, n-j), if f*h=g
  - Associative:

    $f*h_1*h_2 = f*(h_1*h_2)$

    this can save a lot of complexity
  - Distributive:

    f*h1 + f*h2 = f*(h1+h2) useful in SIFT's DoG filtering.
- Smoothing/Edge Detection
  - Gaussian blurring
  - DoG edge detection
- Filtering acceleration
  - Separable filter
  - Integral image domain box filtering