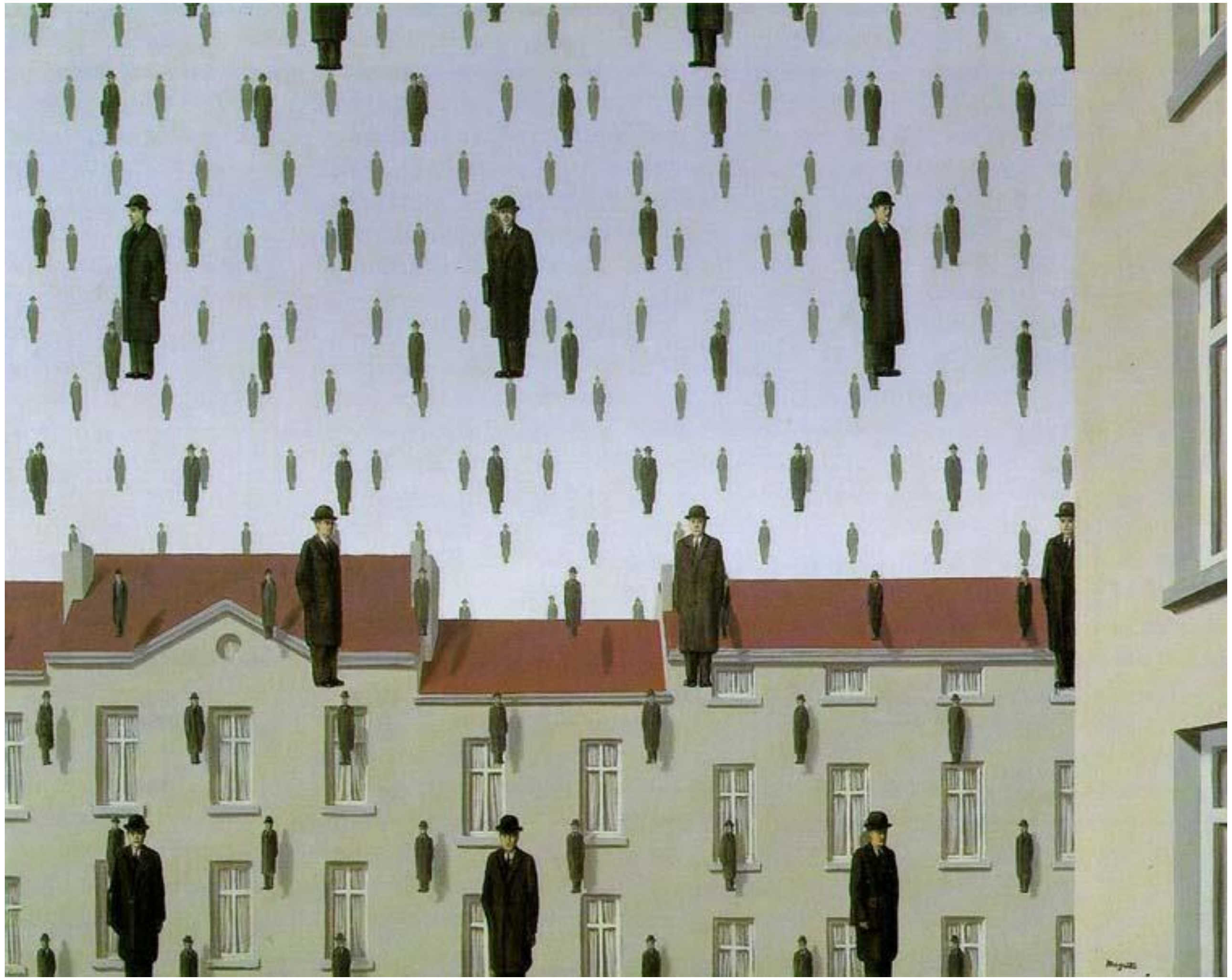
Templates and Image Pyramids



Computational Photography

Derek Hoiem, University of Illinois

Why does a lower resolution image still make sense to us? What do we lose?

Image: http://www.flickr.com/photos/igorms/136916757/

Why does a lower resolution image still make sense to us? What do we lose?

FFT linear scale

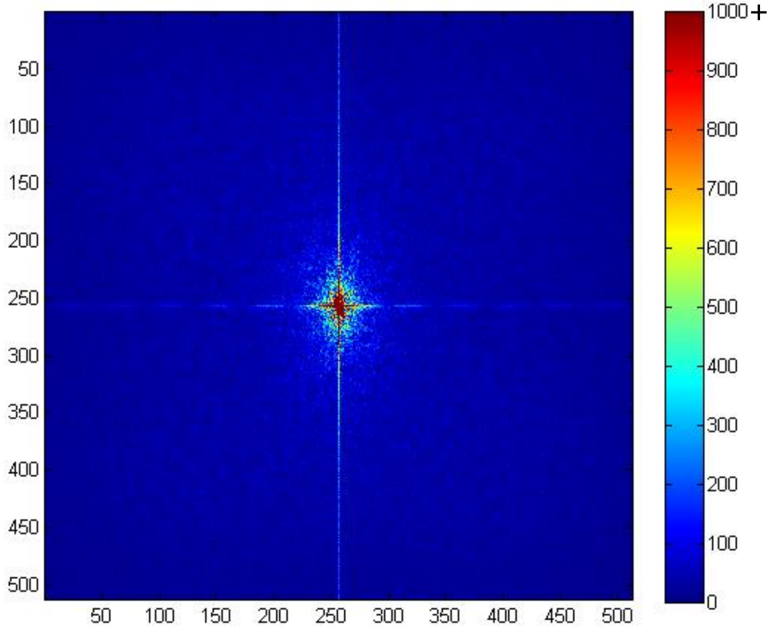
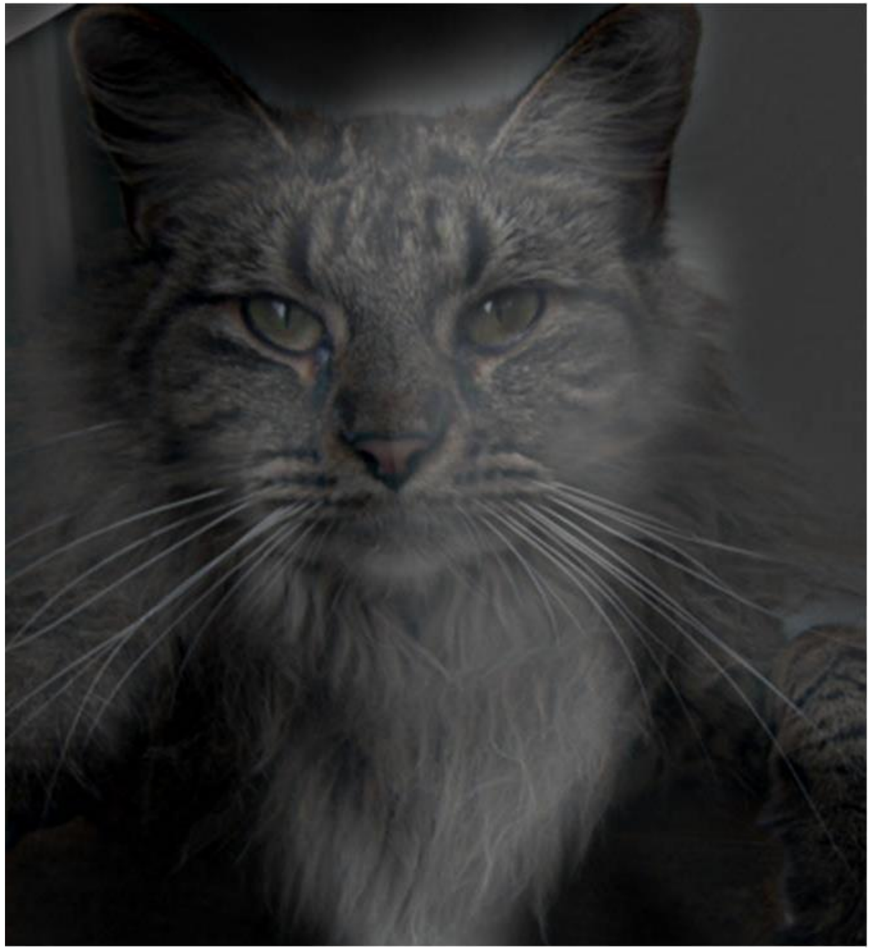


Image: http://www.flickr.com/photos/igorms/136916757/

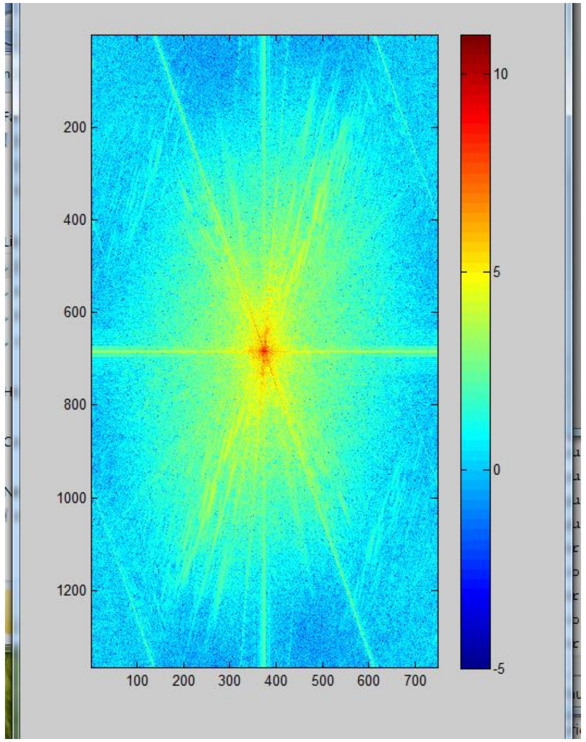
Why do we get different, distance-dependent interpretations of hybrid images?



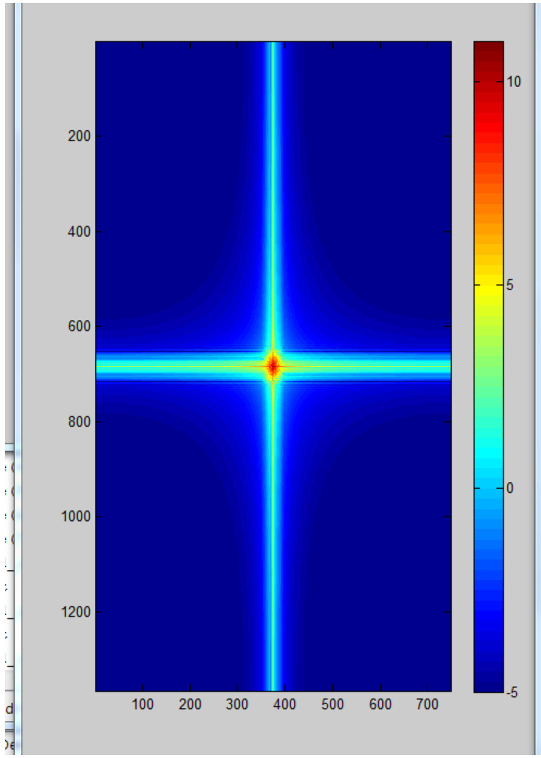
?

Hybrid Image in FFT

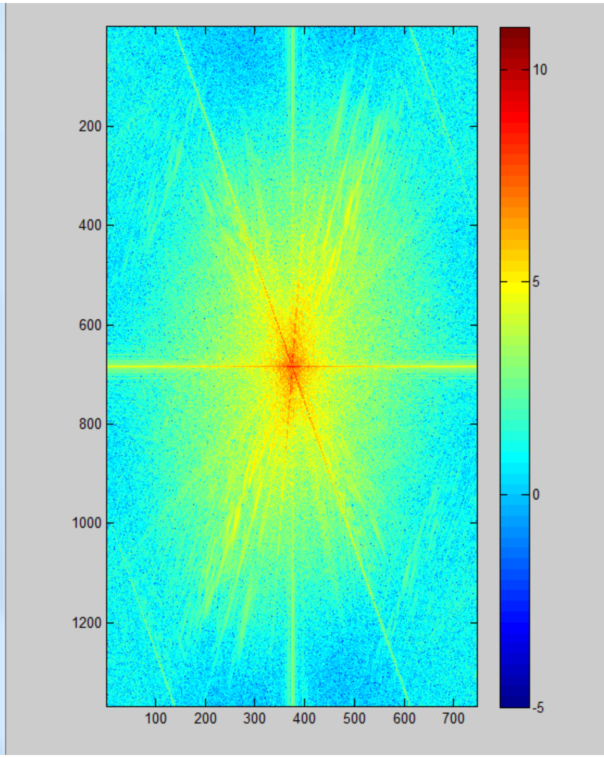
Hybrid Image



Low-passed Image



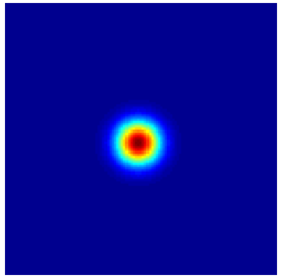
High-passed Image



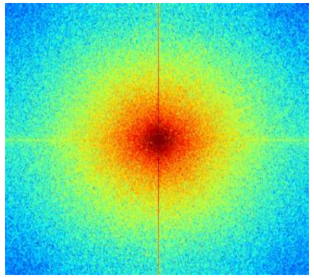
Review

1. Match the spatial domain image to the Fourier magnitude image

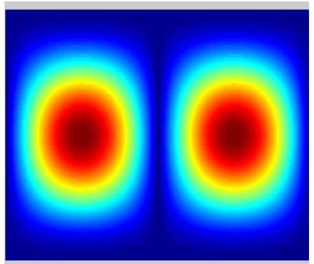
1



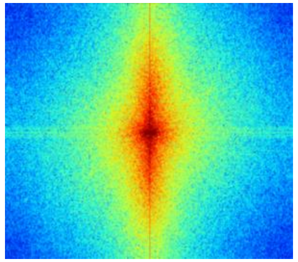
2



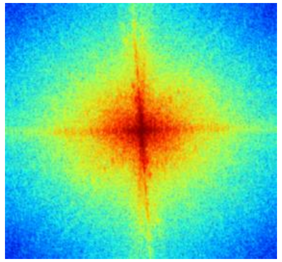
3



4



5

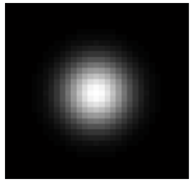


A

B

C

D



E

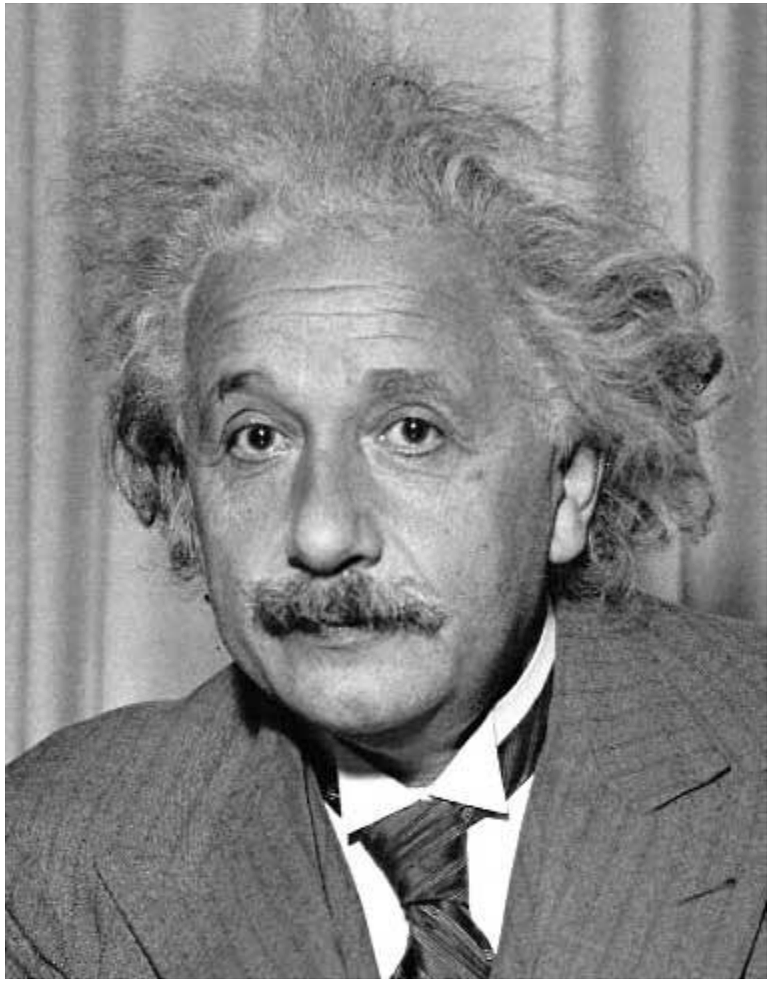
Today’s class: applications of filtering

- Template matching
- Coarse-to-fine alignment
- Denoising, Compression

Template matching

- Goal: find  in image

- Main challenge: What is a good similarity or distance measure between two patches?
  - Correlation
  - Zero-mean correlation
  - Sum Square Difference
  - Normalized Cross Correlation

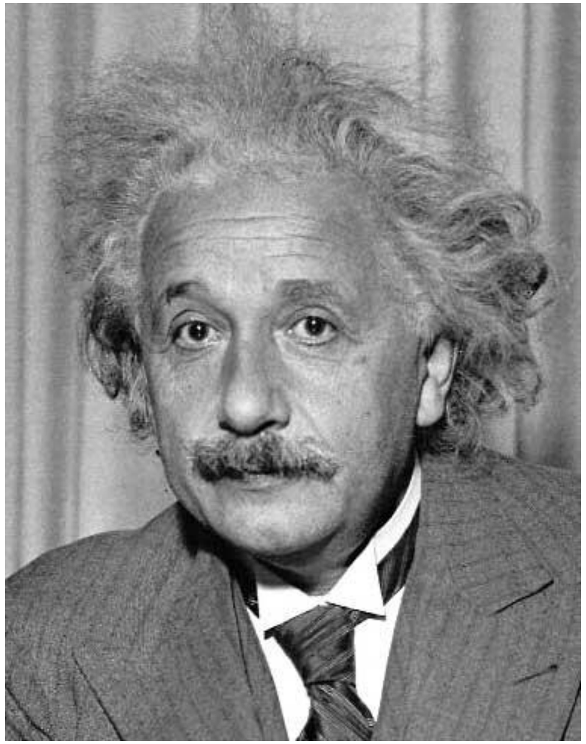


Matching with filters

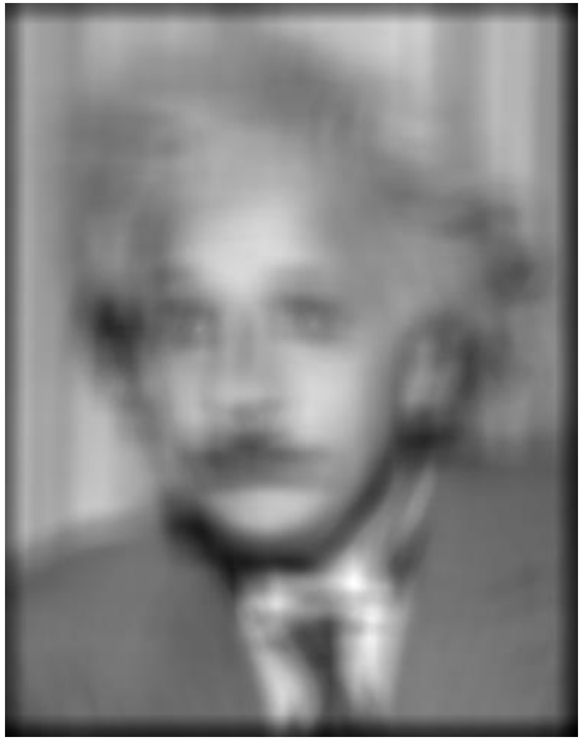
- Goal: find  in image
- Method 0: filter the image with eye patch

$$h[m,n] = \sum_{k,l} f[k,l]\ \ im[m+k, n+l]$$

im = image
f = filter



Input



Filtered Image

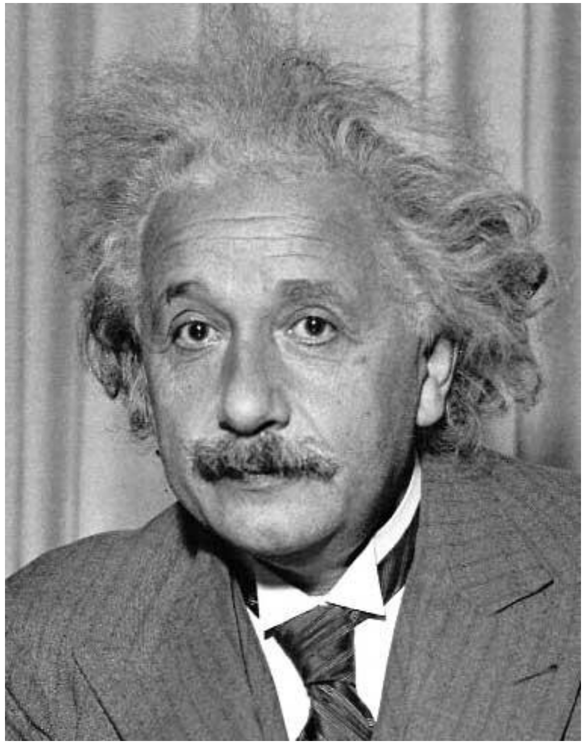
What went wrong?

Matching with filters

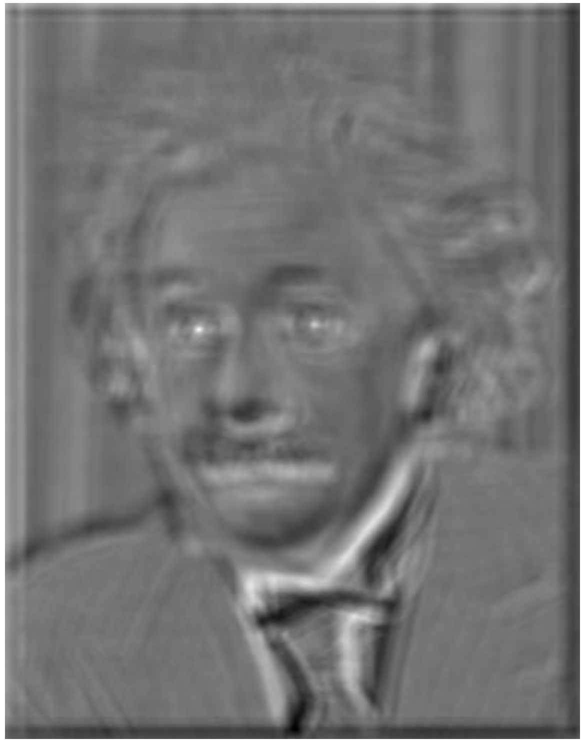
- Goal: find  in image
- Method 1: filter the image with zero-mean eye

$$h[m,n] = \sum_{k,l} (f[k,l] - \bar{f}) \; im[m+k, n+l]$$

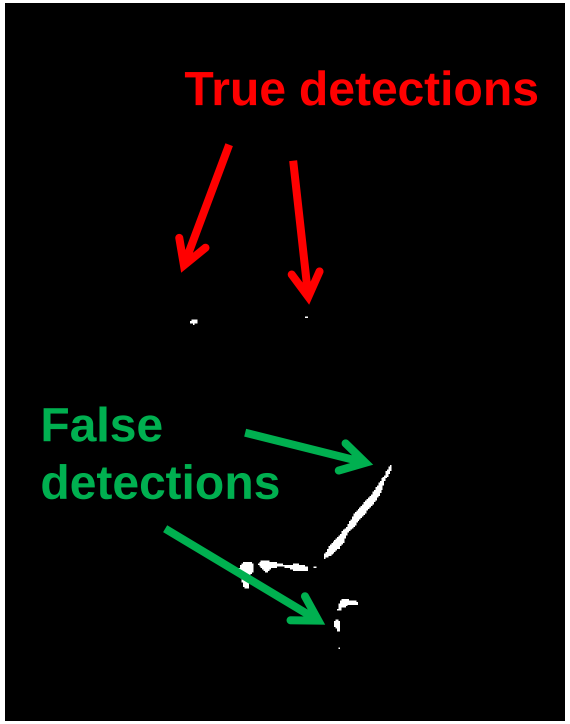
mean of filter f



Input



Filtered Image (scaled)

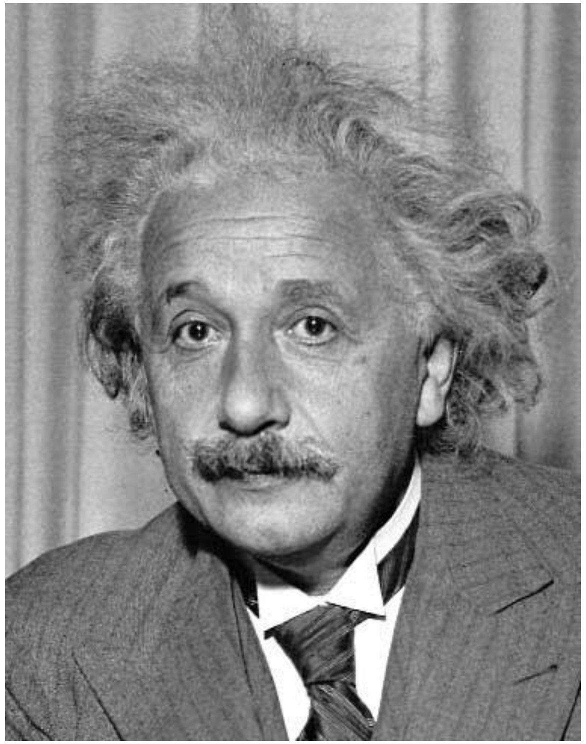


Thresholded Image

Matching with filters

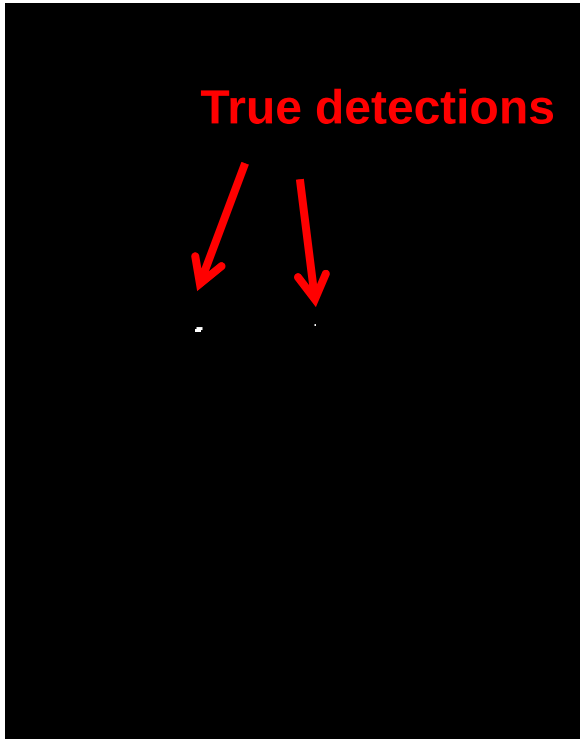
- Goal: find  in image
- Method 2: SSD

$$h[m,n] = \sum_{k,l} (f[k,l] - im[m+k, n+l])^2$$



Input

1- sqrt(SSD)



Thresholded Image

Matching with filters

Can SSD be implemented with linear filters?

$$h[m,n] = \sum_{k,l} (f[k,l] - \ im[m+k, n+l])^2$$

$$h[m,n] = \sum_{k,l} \left(f[k,l]^2 - 2 \cdot im[m+k, n+l] \cdot f[k,l] + im[m+k, n+l]^2\right)$$

$$h[m,n] = \sum_{k,l} f[k,l]^2 - \ 2 \sum_{k,l} im[m+k, n+l] \cdot f[k,l] + \sum_{k,l} im[m+k, n+l]^2$$

$$h = \underbrace{\sum_{k,l} f[k,l]^2}_{\text{constant}} - \ 2\, \underbrace{\mathrm{filter}(\,im, f\,)}_{\text{linear filter}} + \underbrace{\mathrm{filter}(\,im.\,\hat{}\,2, \mathrm{ones}(\,f.\mathrm{shape}))}_{\text{Element-wise square f, then sum with ones kernel of size f}}$$

constant

linear filter

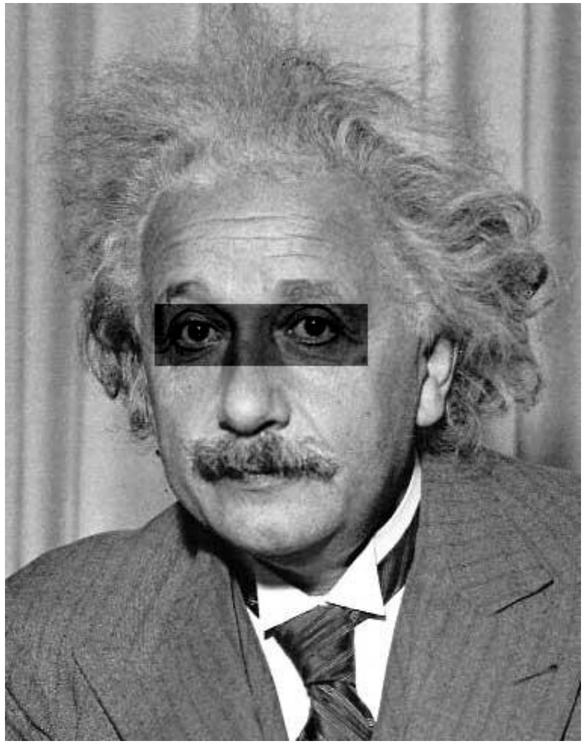
Element-wise square f, then
sum with ones kernel of size f

Matching with filters

- Goal: find  in image
- Method 2: SSD

$$h[m,n] = \sum_{k,l} (f[k,l] - im[m+k, n+l])^2$$

What's the potential downside of SSD?



Input

1- sqrt(SSD)

Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation

mean template

mean image patch

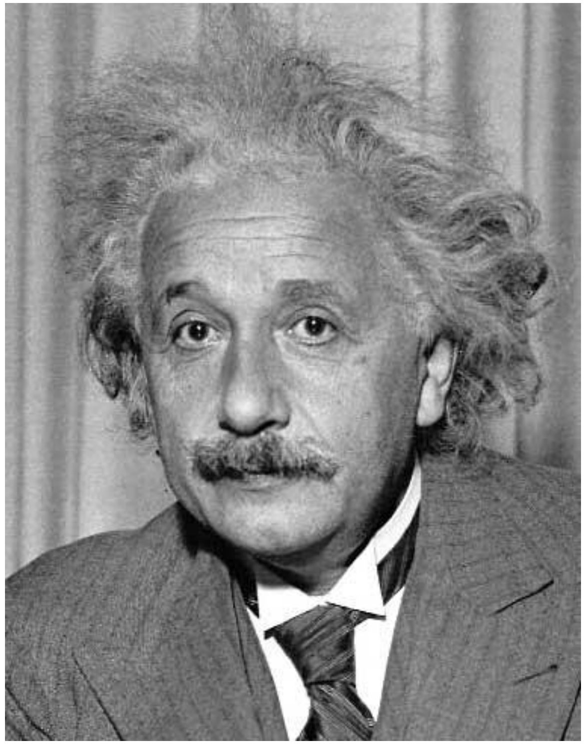
$$h[m,n] = \frac{\sum_{k,l}(f[k,l] - \bar{f})(\,im[m+k,n+l] - \overline{im_{m,n}})}{\left(\sum_{k,l}(f[k,l] - \bar{f})^2 \sum_{k,l}(\,im[m+k,n+l] - \overline{im_{m,n}})^2\right)^{0.5}}$$

(divide by product of standard deviations of template and image patch)

Python: `cv2.matchTemplate(im,template,cv2.TM_CCOEFF_NORMED)`

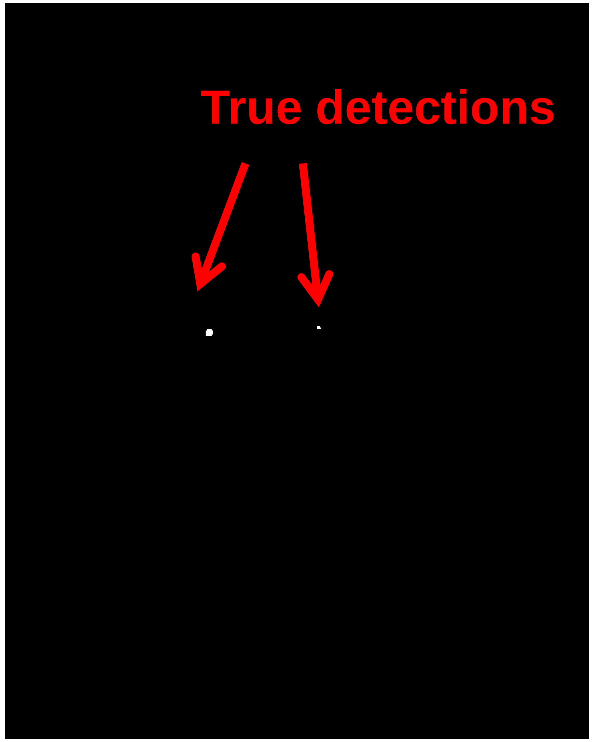
Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

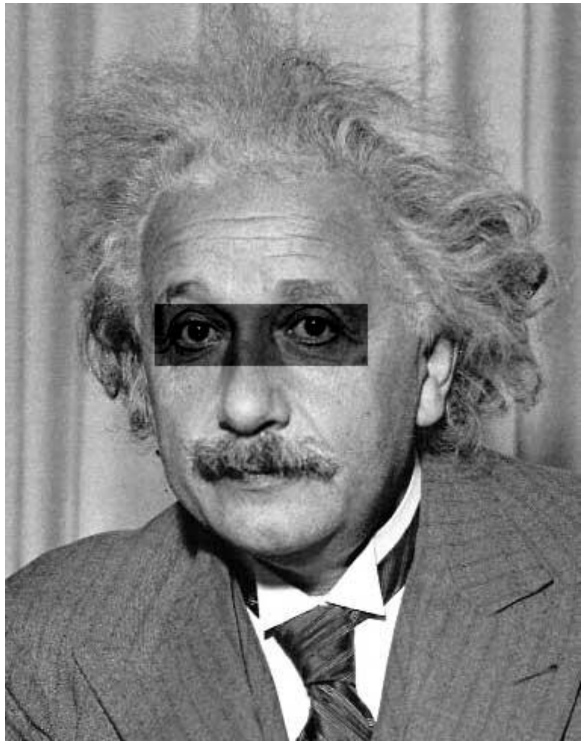
Normalized X-Correlation



Thresholded Image

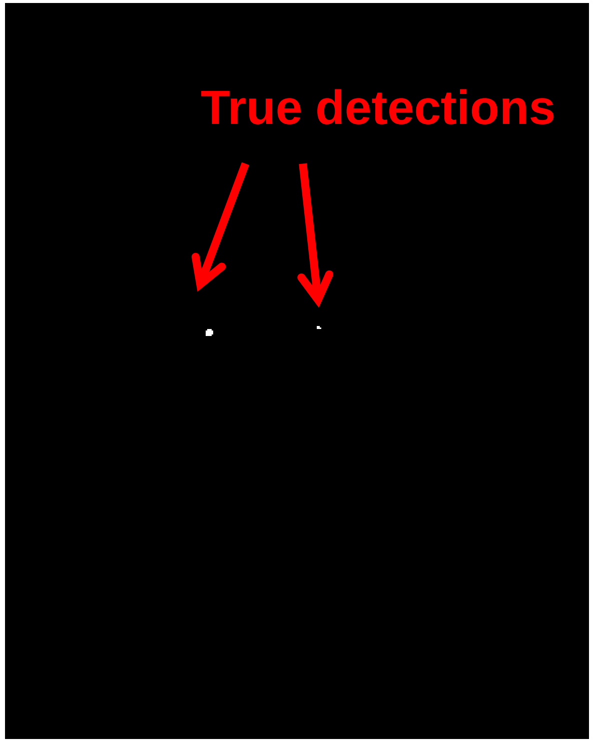
Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

Normalized X-Correlation



Thresholded Image

Q: What is the best method to use?

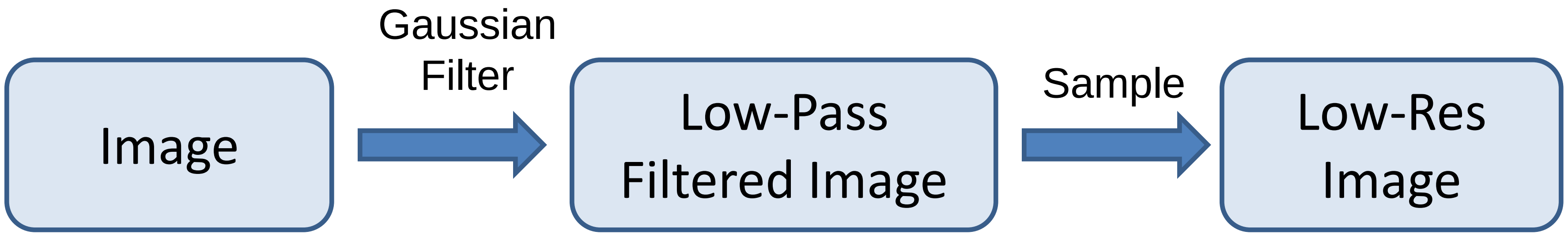
A: Depends

- Zero-mean filter: fastest but not a great matcher
- SSD: next fastest, sensitive to overall intensity
- Normalized cross-correlation: slowest, invariant to local average intensity and contrast

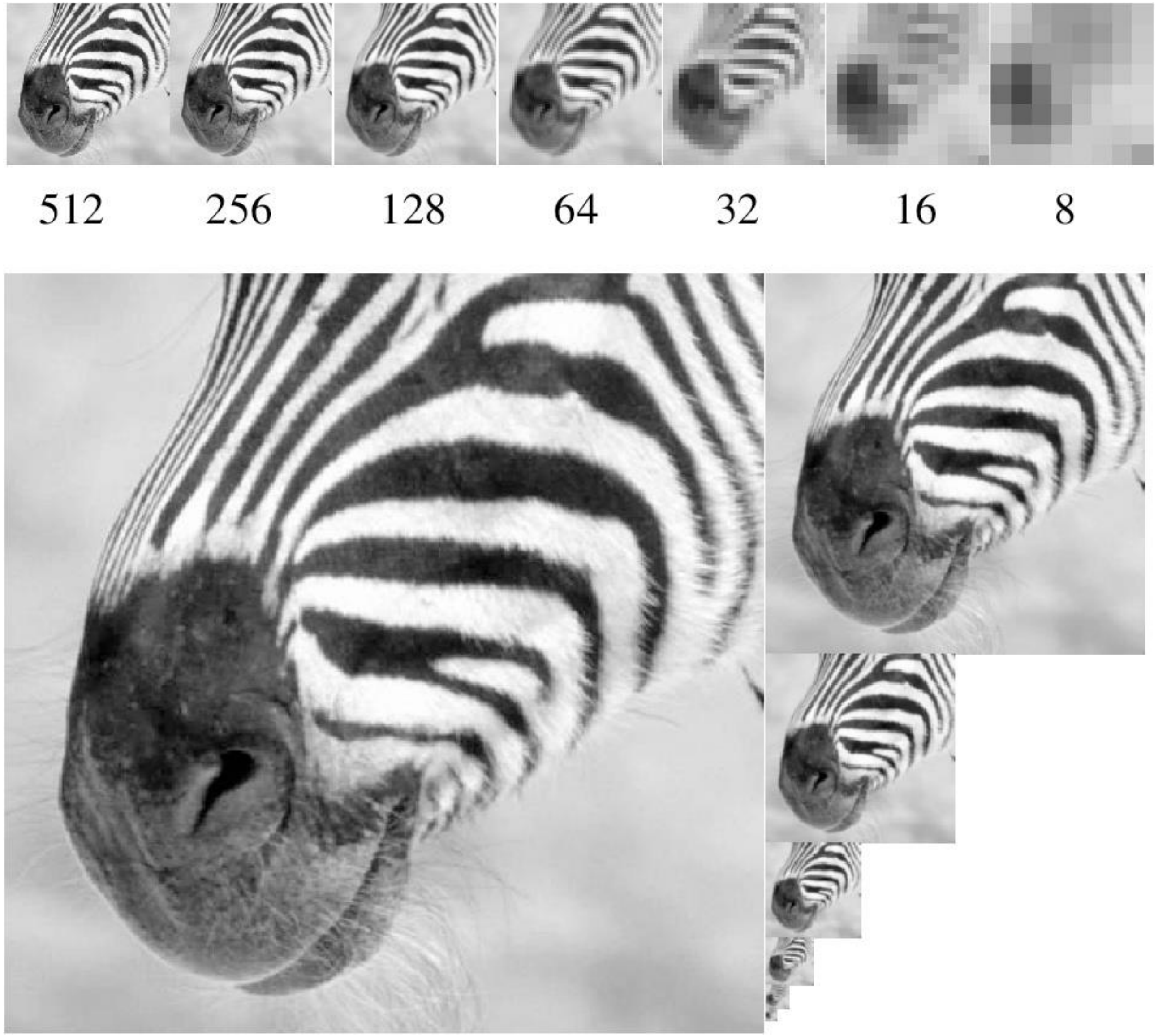
Q: What if we want to find larger or smaller eyes?

A: Image Pyramid

Review of Sampling

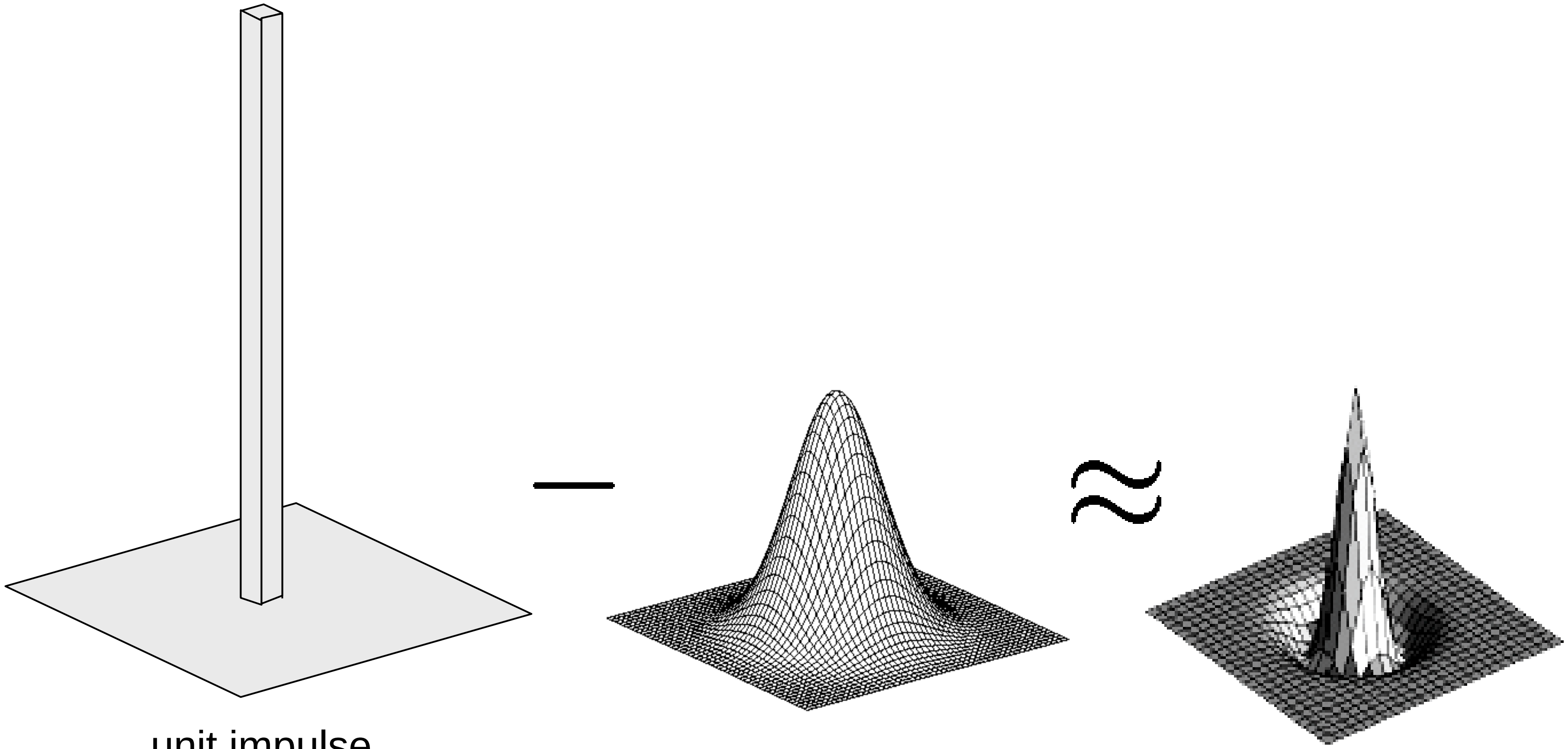


Gaussian pyramid



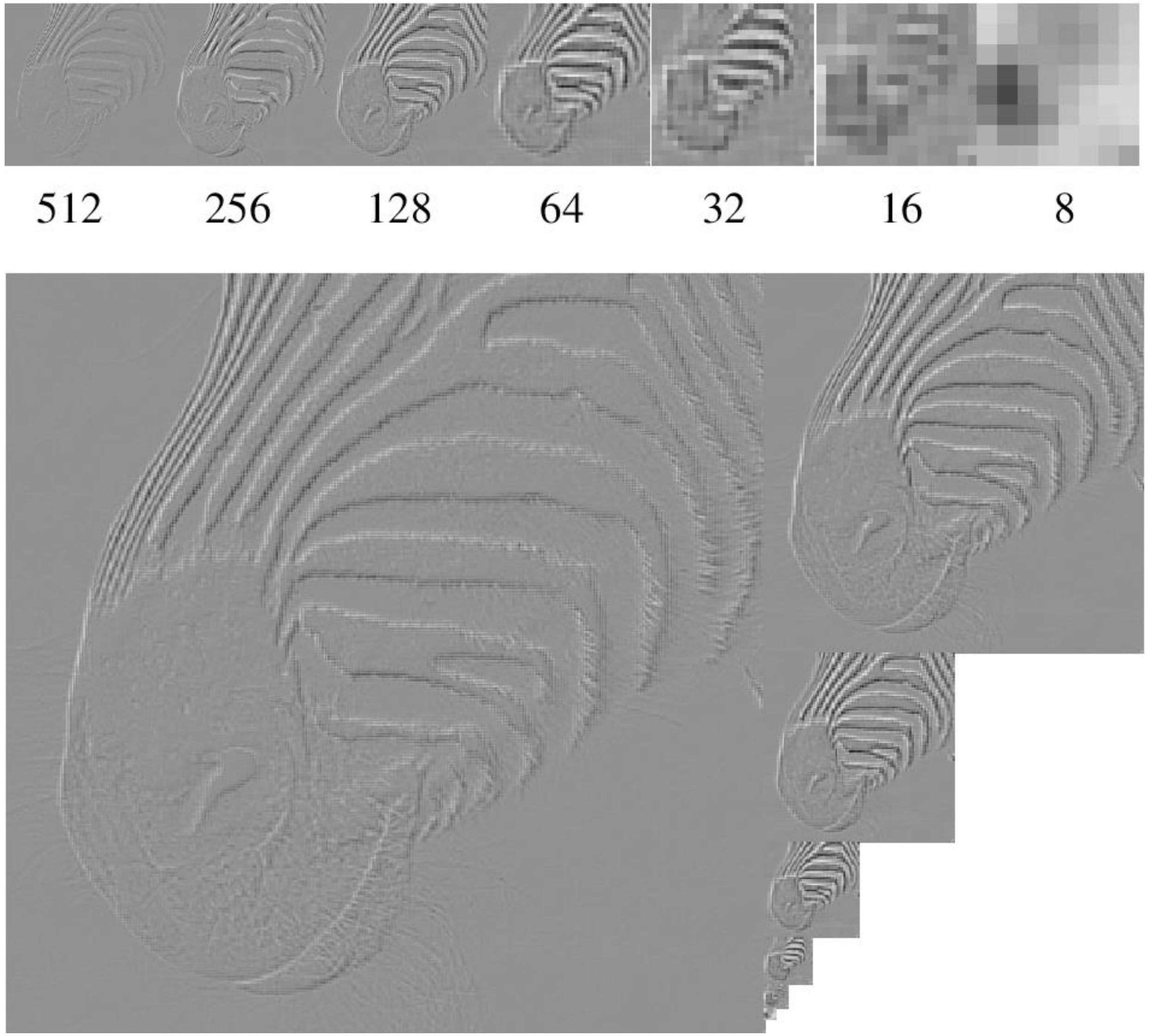
Source: Forsyth

Laplacian filter



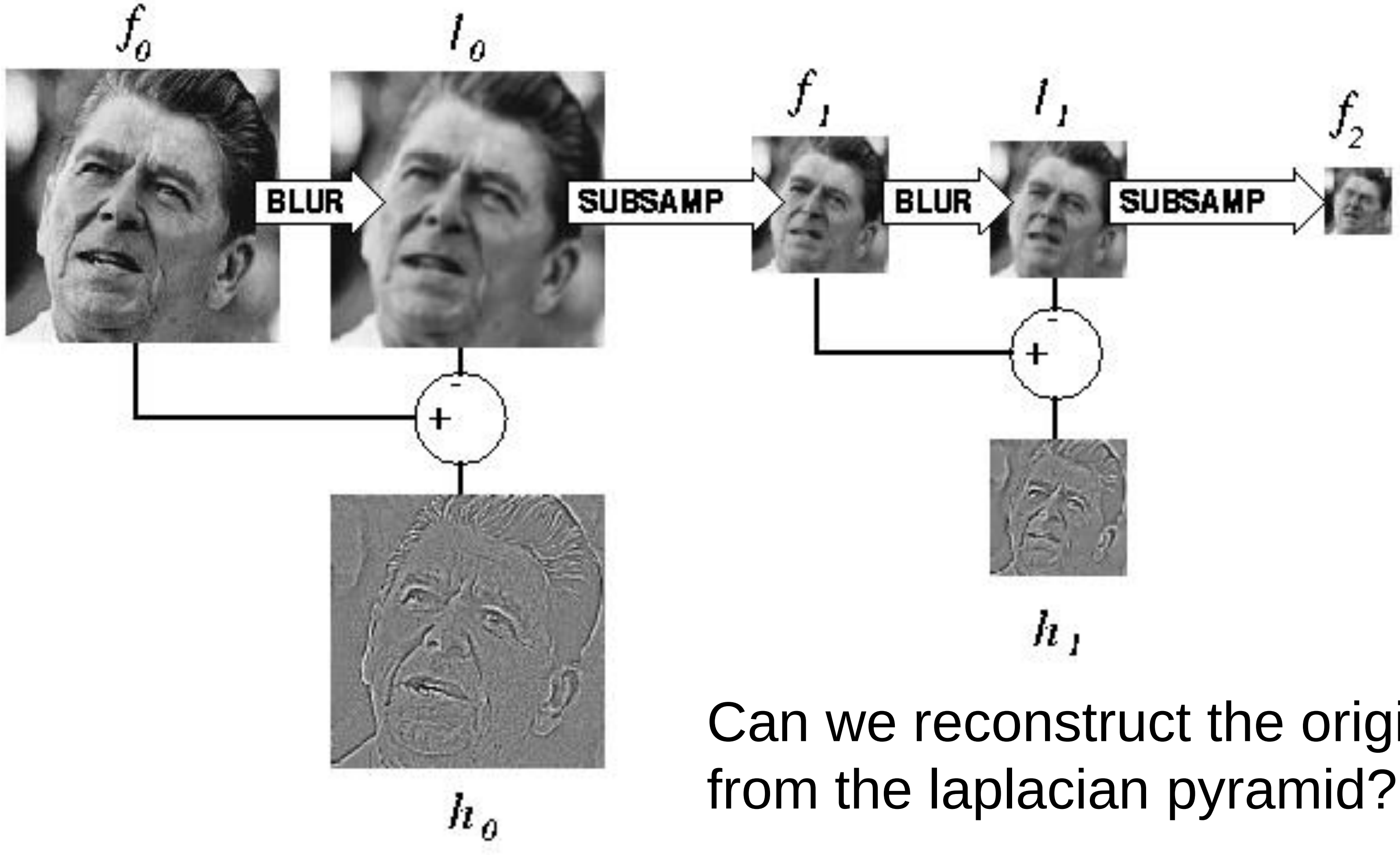
Source: Lazebnik

Laplacian pyramid



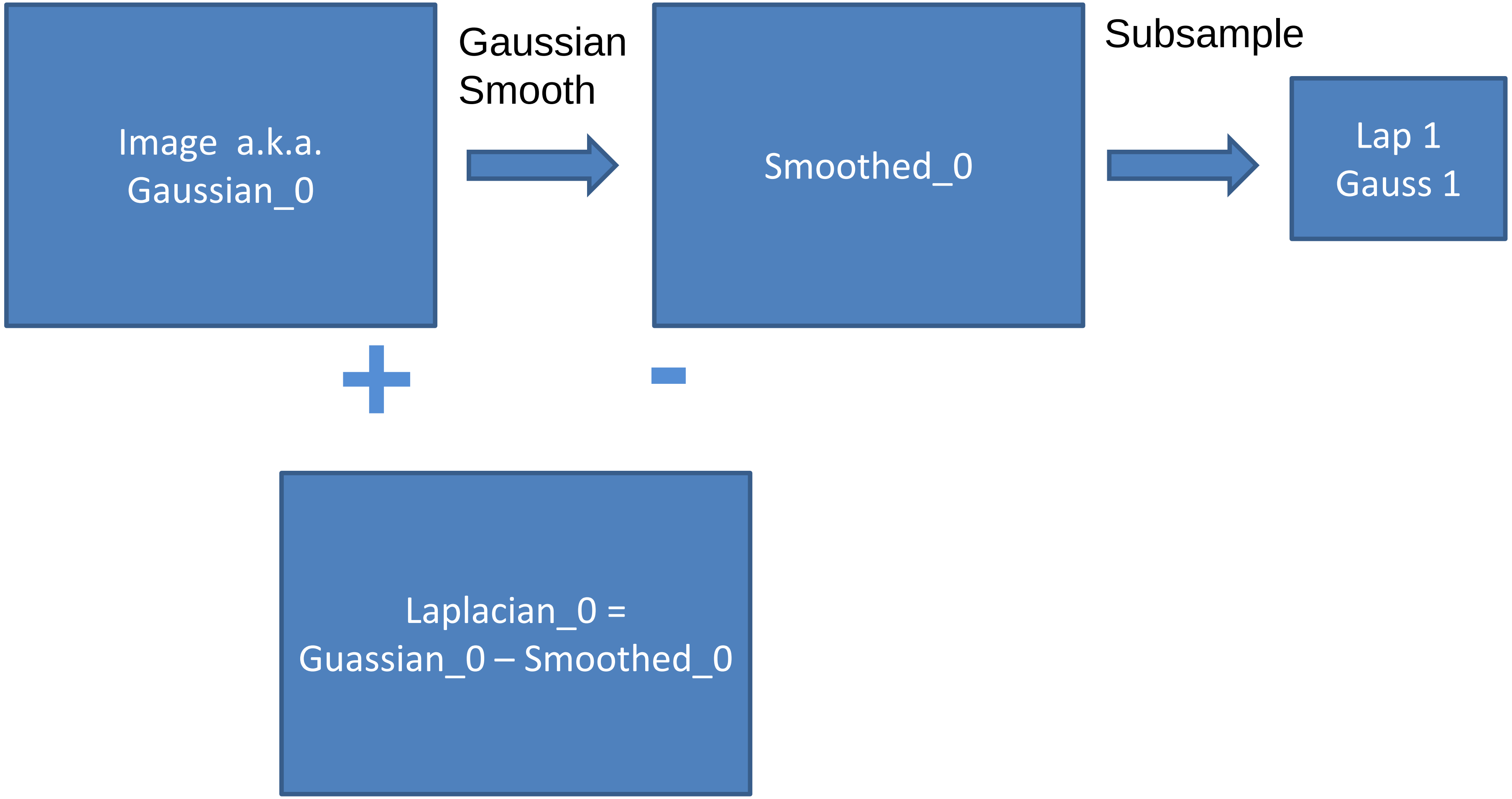
Source: Forsyth

Computing Gaussian/Laplacian Pyramid

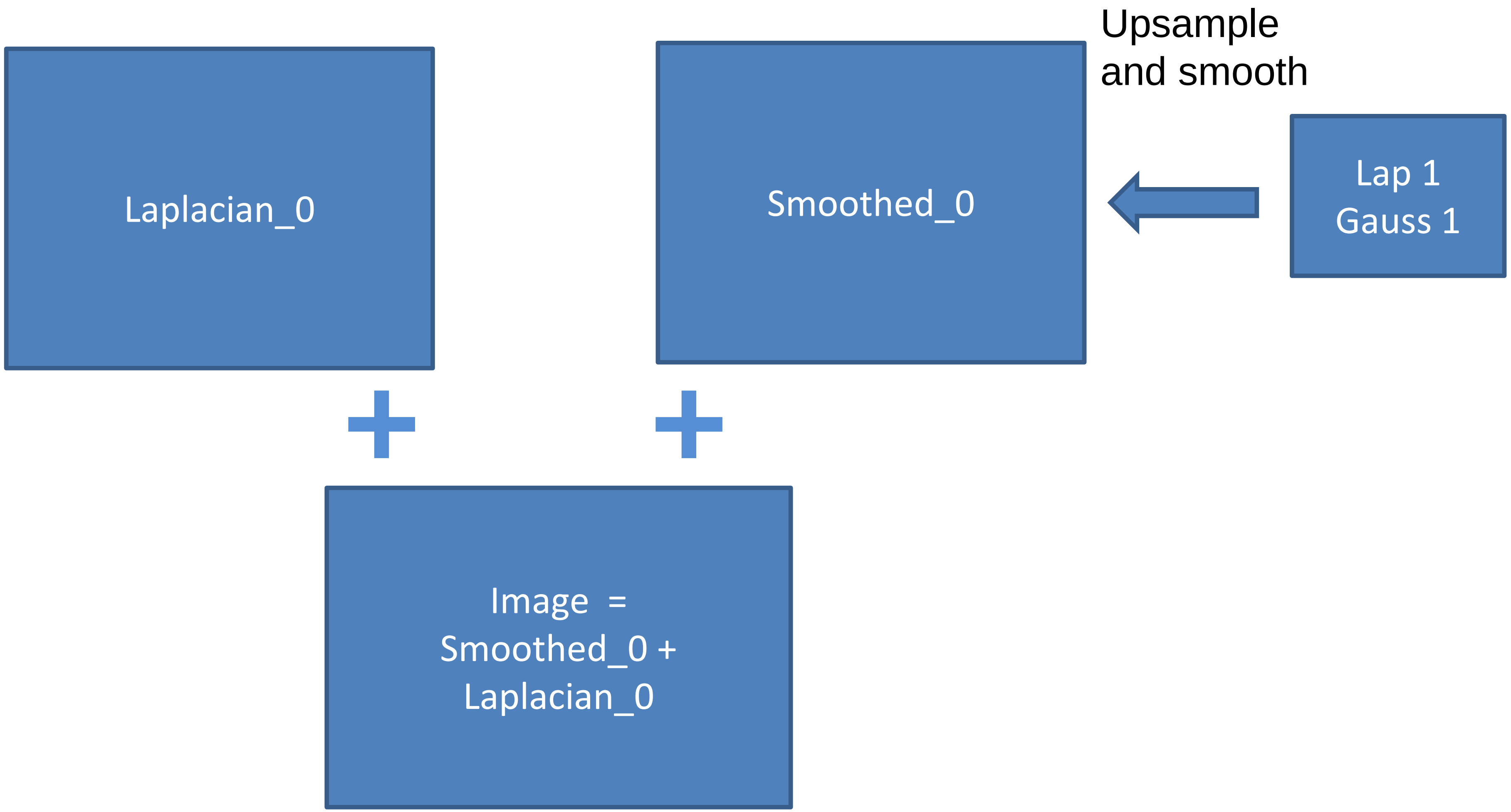


Can we reconstruct the original from the laplacian pyramid?

Creating a 2-level Laplacian pyramid



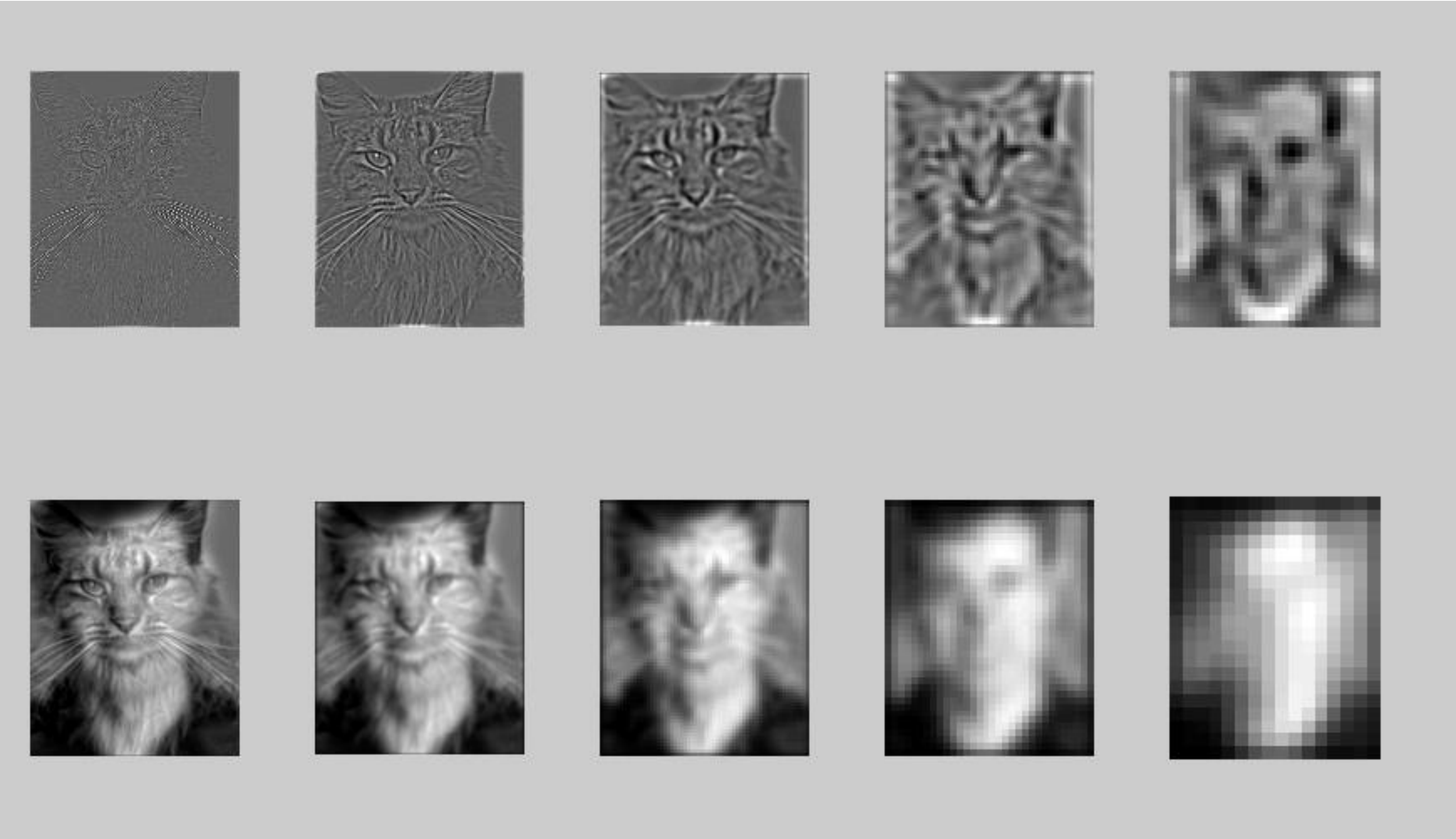
Reconstructing the image from Laplacian pyramid



Hybrid Image in Laplacian Pyramid

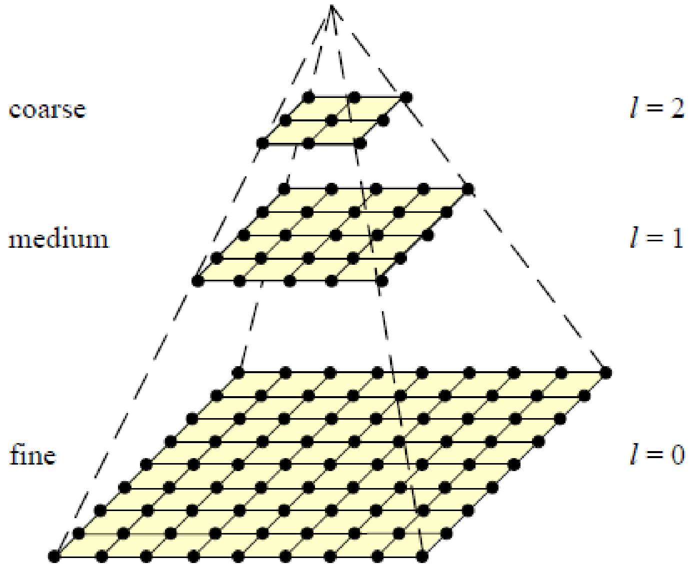
Extra points for project 1

High frequency → Low frequency

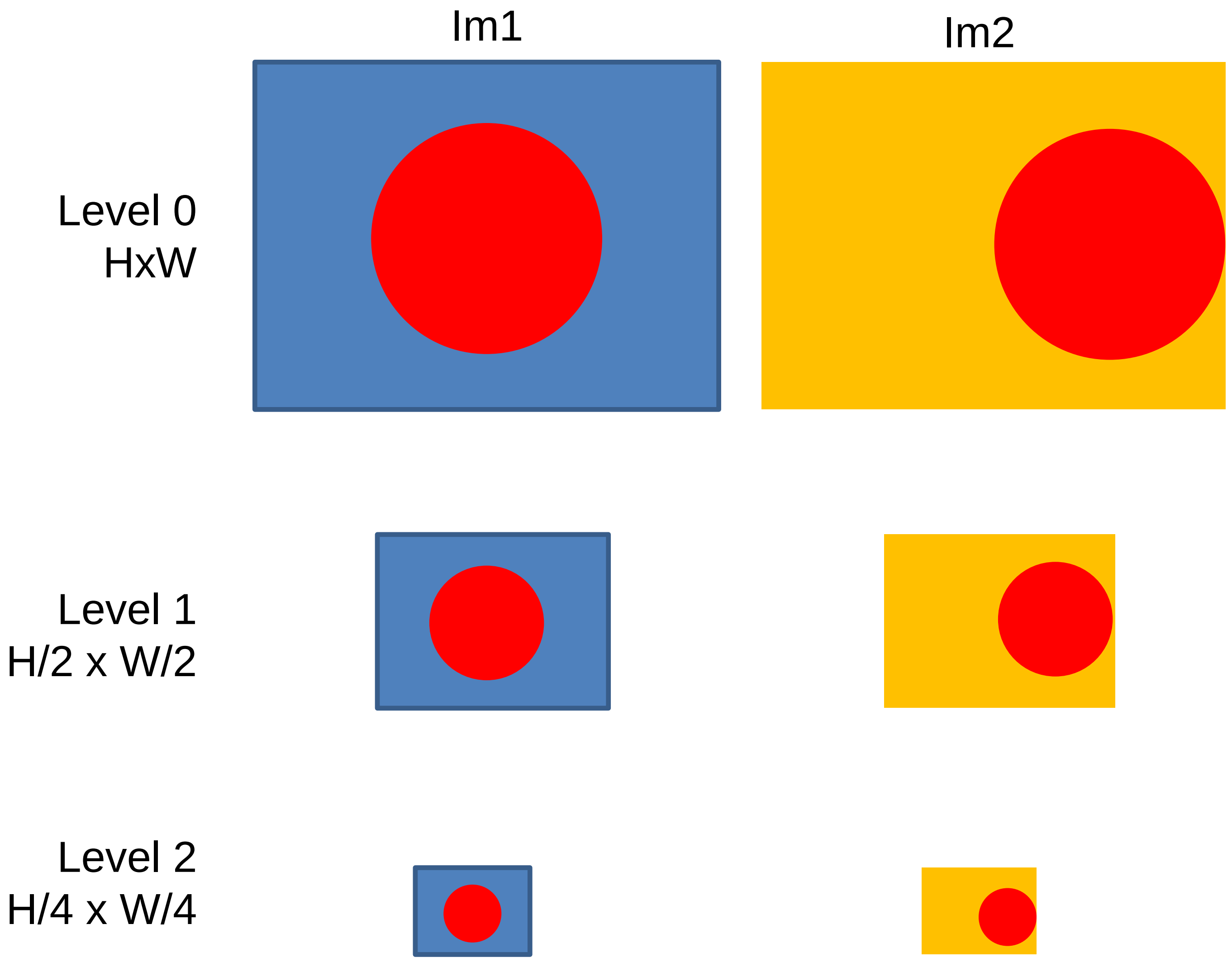


Coarse-to-fine Image Registration

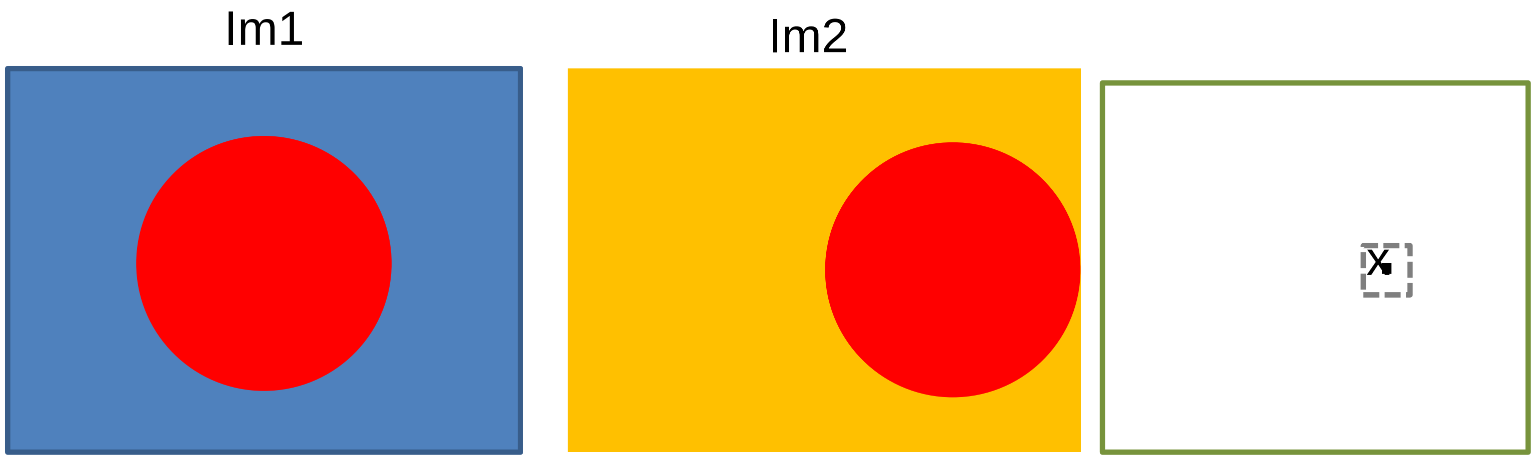
1. Compute Gaussian pyramid
2. Align at coarse level
   – Find minimum SSD position
3. Successively align at finer levels
   – Search small range (e.g., 5x5) centered around position determined at coarser scale



Coarse-to-fine Image Registration



Coarse-to-fine Image Registration



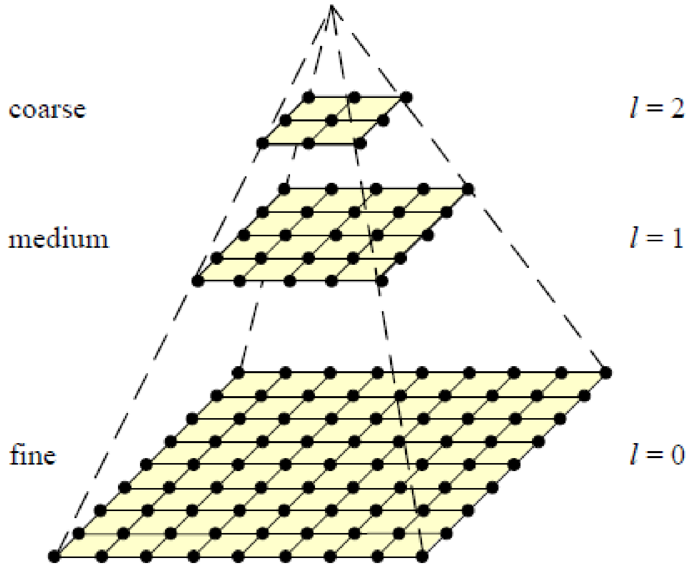
$$tx_0, ty_0 = \operatorname*{argmin}_{tx \in 2 \cdot tx_1 + \{-s..s\}, ty = 2 \cdot ty_1 + \{-s..s\}} \mathrm{SSD}(im1_0, \mathrm{translate}(im2_0, tx, ty))$$

$$tx_1, ty_1 = \operatorname*{argmin}_{tx \in 2 \cdot tx_2 + \{-s..s\}, ty = 2 \cdot ty_2 + \{-s..s\}} \mathrm{SSD}(im1_1, \mathrm{translate}(im2_1, tx, ty))$$

$$tx_2, ty_2 = \operatorname*{argmin}_{tx \in \left\{-\frac{W}{8} .. \frac{W}{8}\right\}, ty \in \{-\frac{H}{8} .. \frac{H}{8}\}} \mathrm{SSD}(im1_2, \mathrm{translate}(im2_2, tx, ty))$$

Coarse-to-fine Image Registration

1. Compute Gaussian pyramid
2. Align at coarse level
   – Find minimum SSD position
3. Successively align at finer levels
   – Search small range (e.g., 5x5) centered around position determined at coarser scale



Why is this faster?

Are we guaranteed to get the same result?

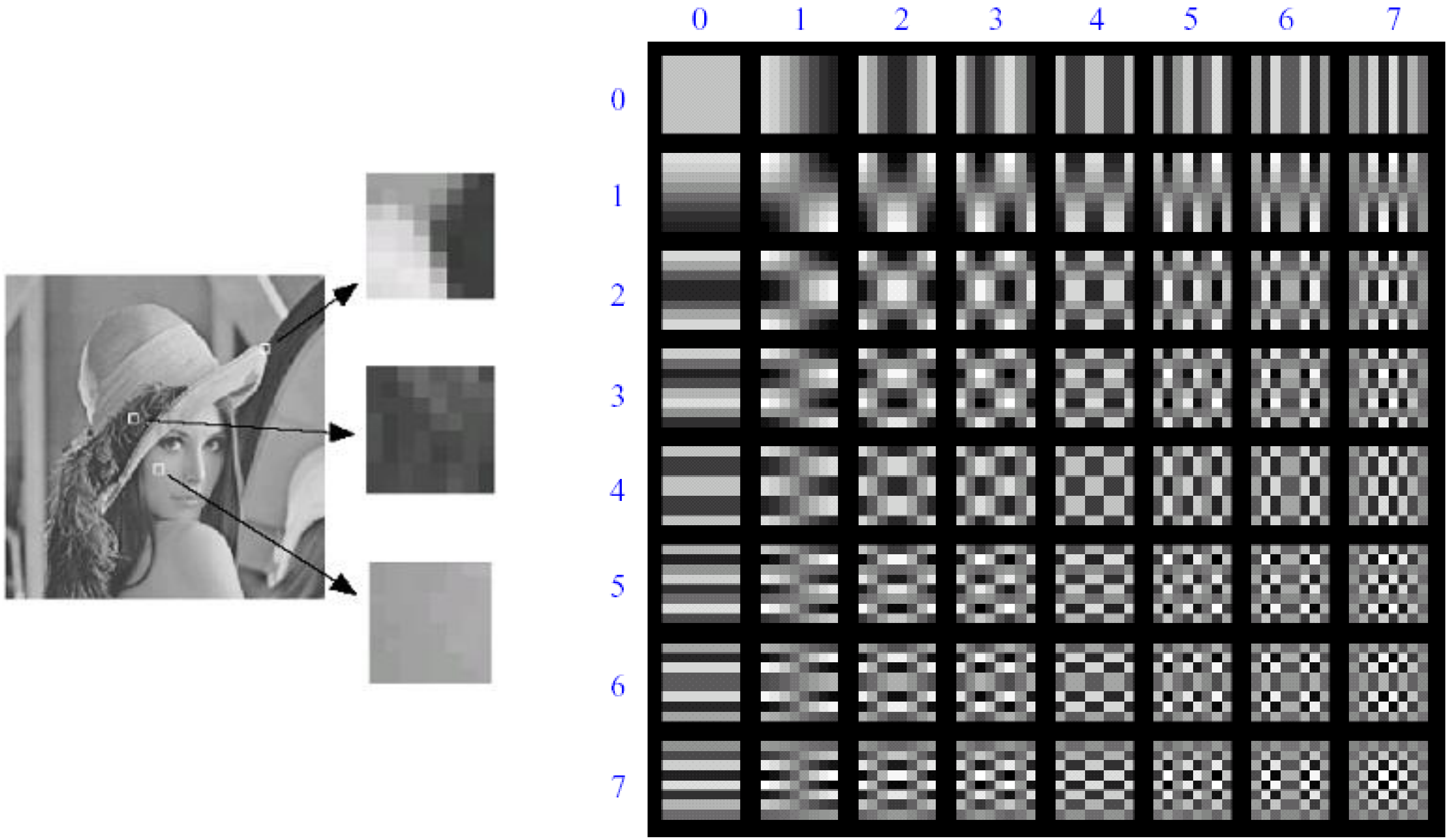
Question

Can you align the images using the FFT?

Compression

How is it that a 4 megapixel image (12MB) can be compressed to a few hundred KB without a noticeable change?

Lossy Image Compression (JPEG)

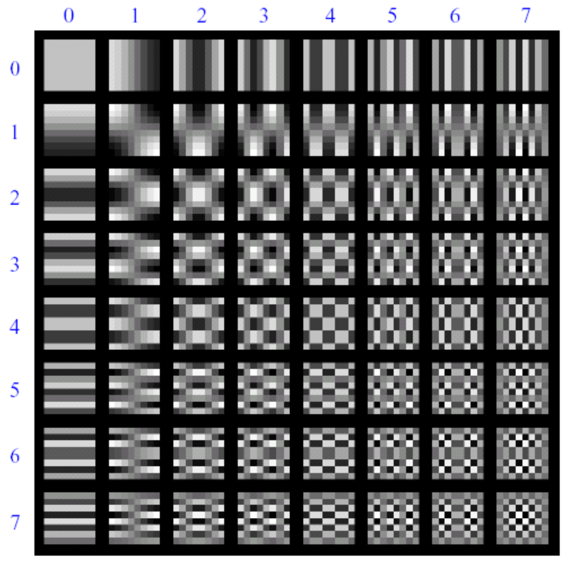


Block-based Discrete Cosine Transform (DCT)

Slides: Efros

Using DCT in JPEG

- The first coefficient $B(0,0)$ is the DC component, the average intensity
- The top-left coeffs represent low frequencies, the bottom right – high frequencies



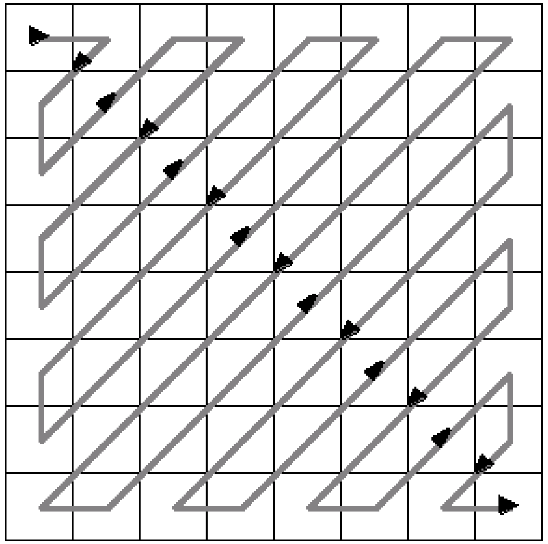


Image compression using DCT

- Quantize
  - More coarsely for high frequencies (which also tend to have smaller values)
  - Many quantized high frequency values will be zero
- Encode
  - Can decode with inverse dct

Filter responses

$$G = \begin{array}{c} \xrightarrow{u} \\ \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \end{array} \Big\downarrow v$$

Quantized values

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Quantization table

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

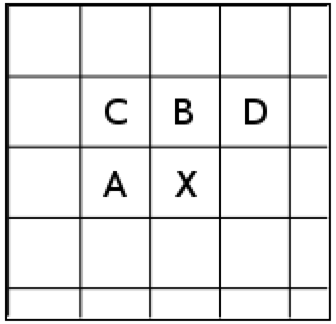
JPEG Compression Summary

1. Convert image to YCrCb
2. Subsample color by factor of 2
   - People have bad resolution for color
3. Split into blocks (8x8, typically), subtract 128
4. For each block
   a. Compute DCT coefficients
   b. Coarsely quantize
      - Many high frequency components will become zero
   c. Encode (e.g., with Huffman coding)

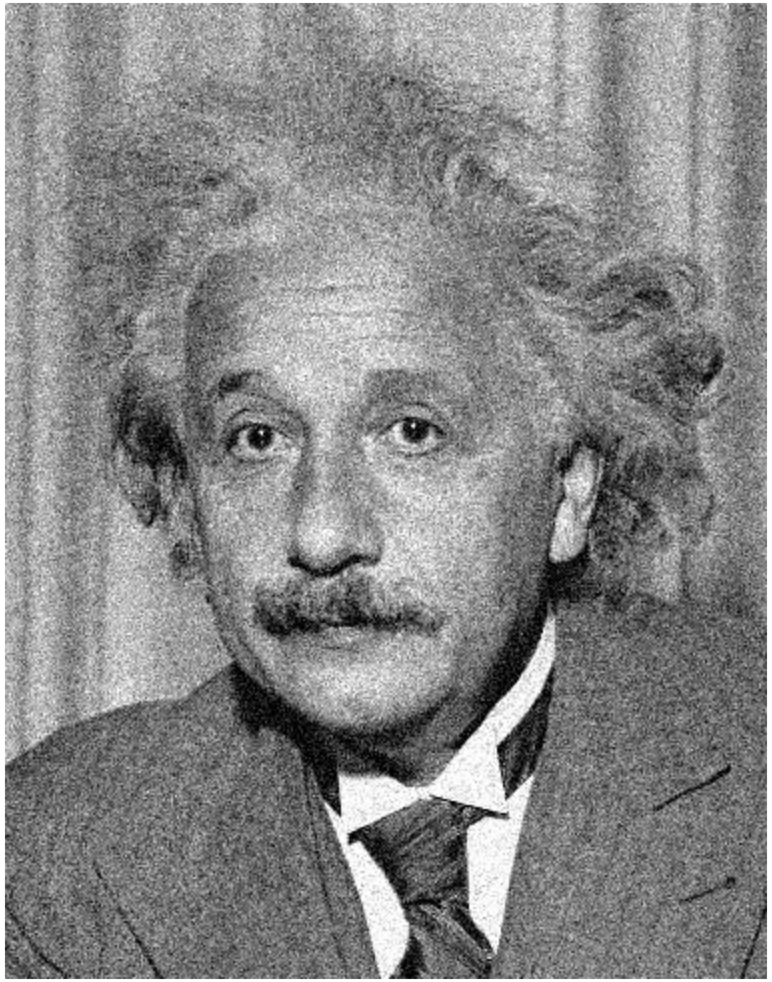
http://en.wikipedia.org/wiki/YCbCr
http://en.wikipedia.org/wiki/JPEG

Lossless compression (PNG)

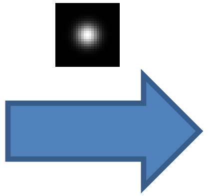
1. Predict that a pixel's value based on its upper-left neighborhood
   – Choose one rule per row, e.g. x~=A or x~=floor((A+B)/2)
2. Store difference of predicted and actual value
3. Pkzip it (DEFLATE algorithm)



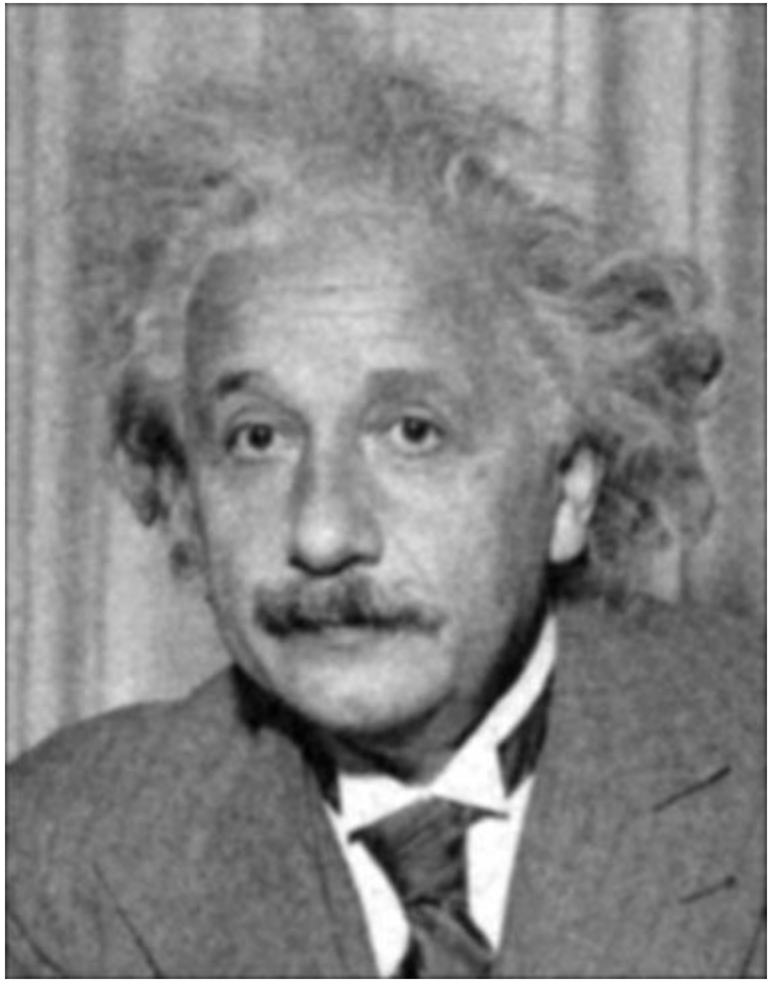
Denoising



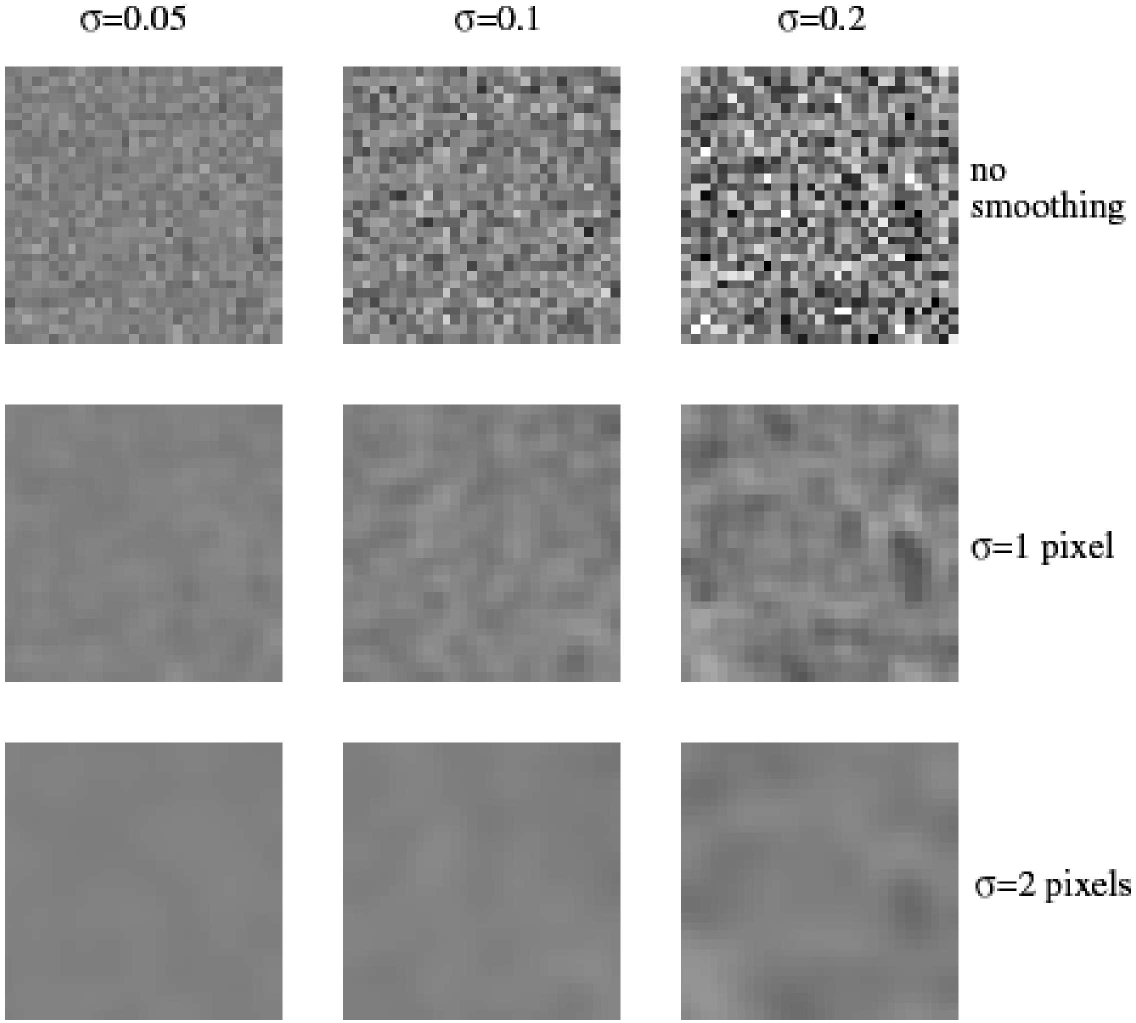
Additive Gaussian Noise



Gaussian
Filter



Reducing Gaussian noise



Smoothing with larger standard deviations suppresses noise, but also blurs the image

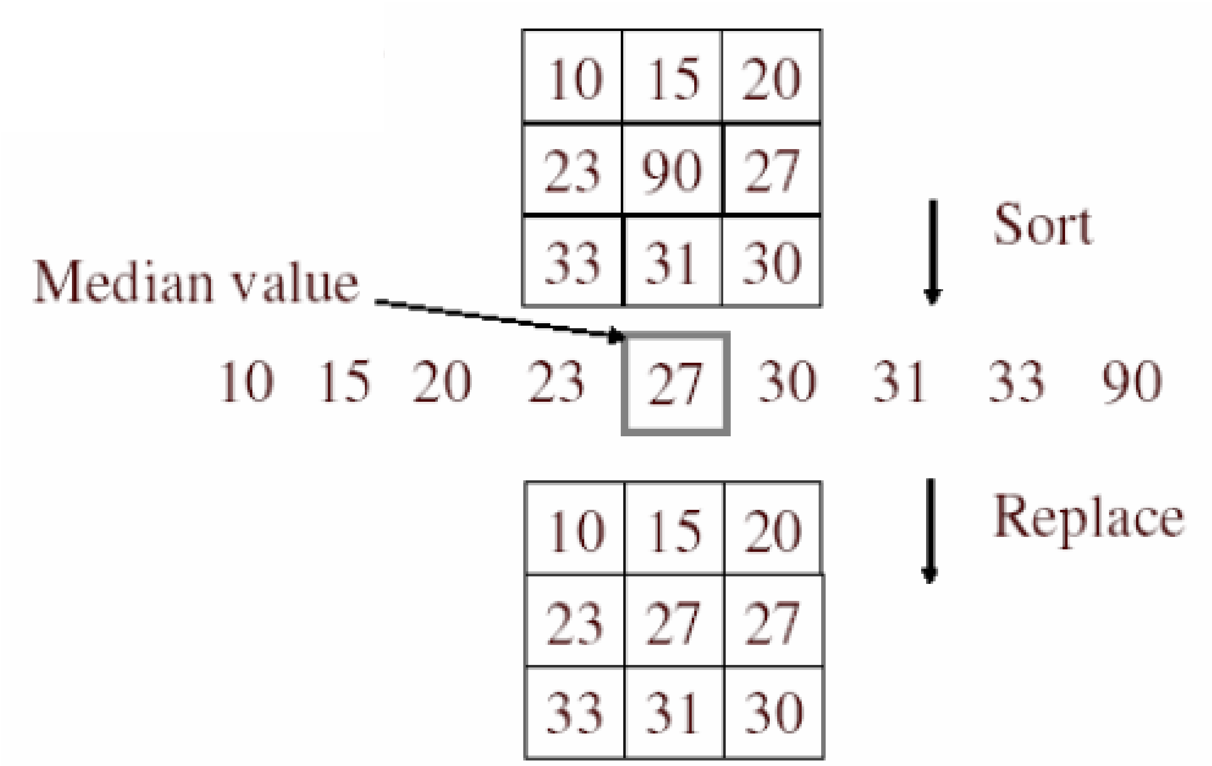
Source: S. Lazebnik

Reducing salt-and-pepper noise by Gaussian smoothing

3x3 5x5 7x7

Alternative idea: Median filtering

- A **median filter** operates over a window by selecting the median intensity in the window

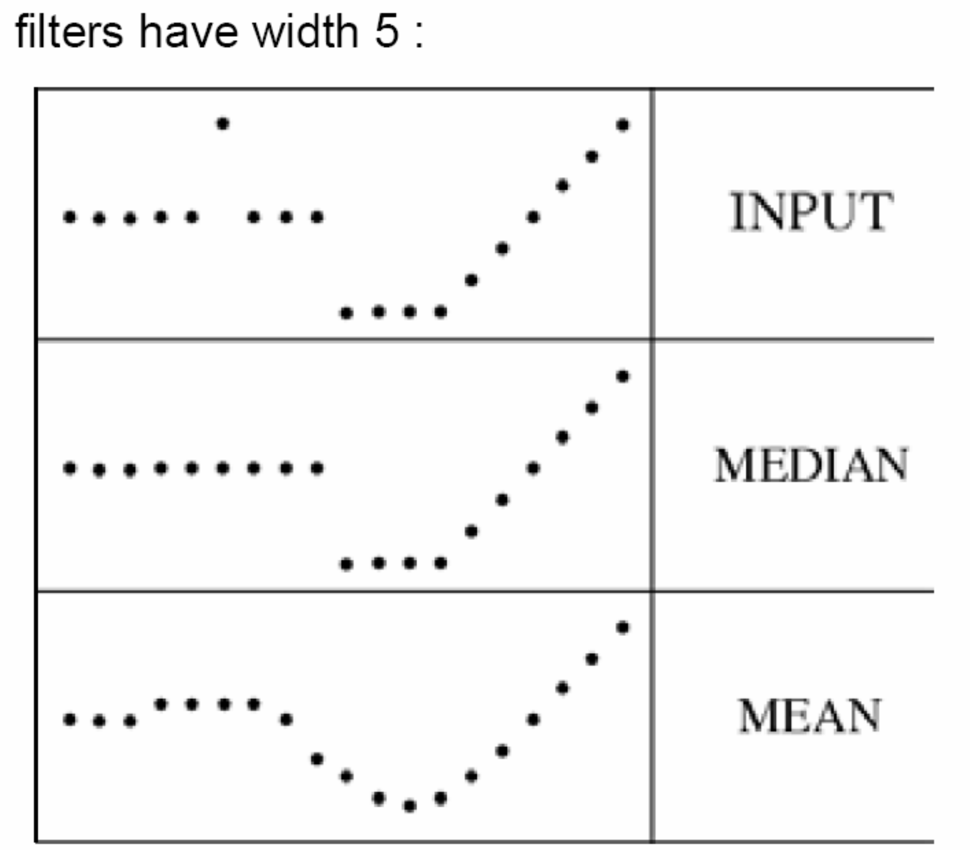


- Is median filtering linear?

Source: K. Grauman

Median filter

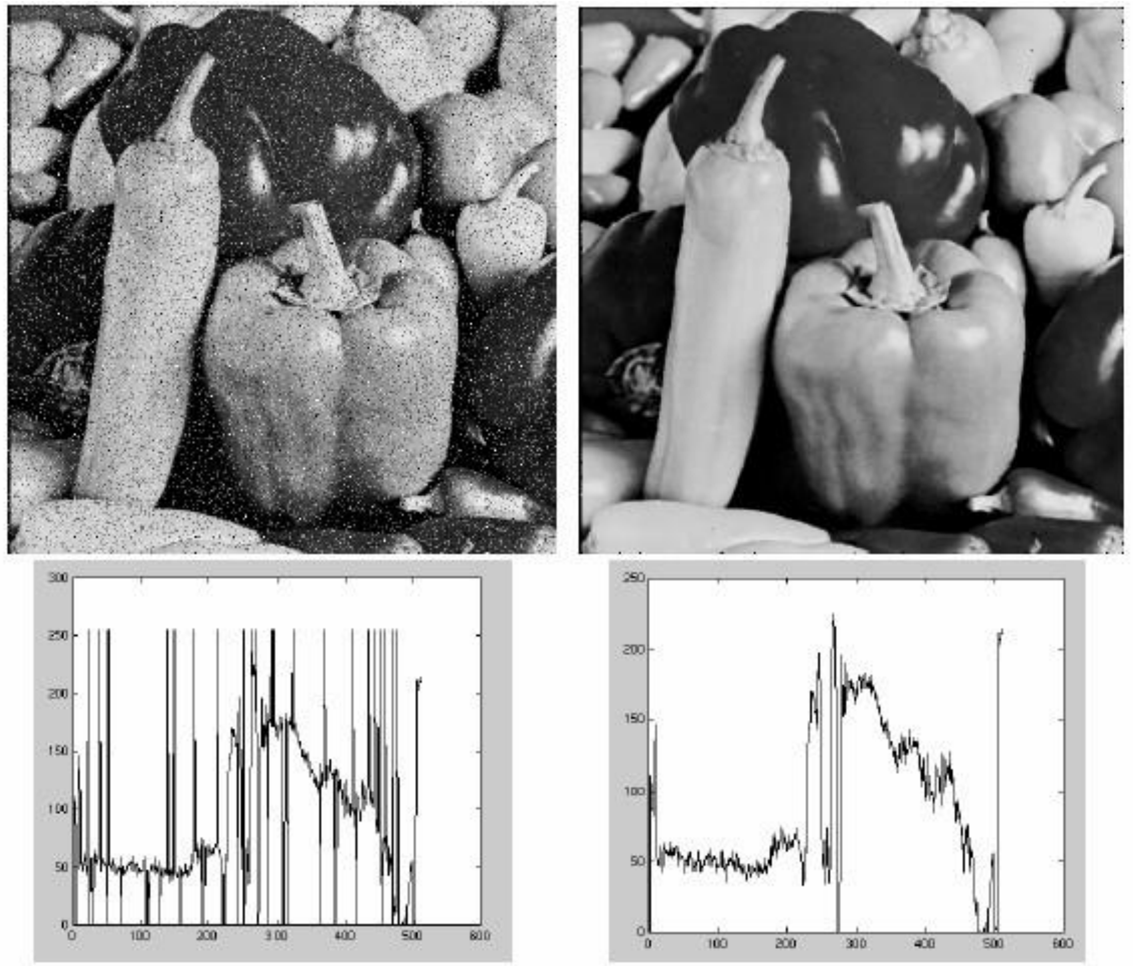
- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers



Source: K. Grauman

Median filter

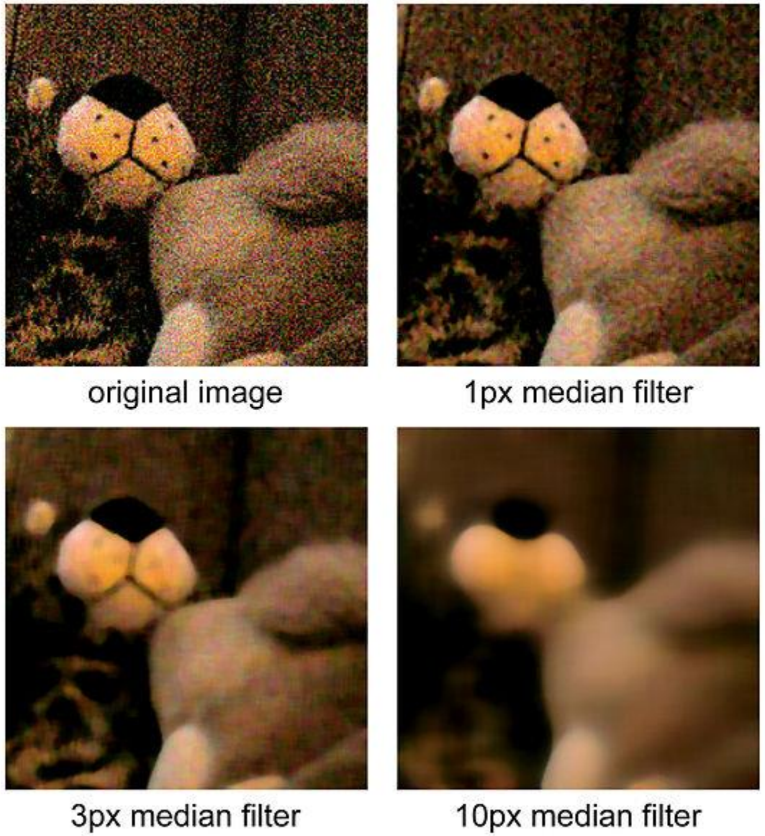
Salt-and-pepper noise Median filtered



Python: scipy.ndimage.median_filter (image, size)

Source: M. Hebert

Median Filtered Examples



http://en.wikipedia.org/wiki/File:Medianfilterp.png
http://en.wikipedia.org/wiki/File:Median_filter_example.jpg

Median vs. Gaussian filtering

3x3 5x5 7x7

Gaussian

Median

Other filter choices

- Weighted median (pixels further from center count less)
- Clipped mean (average, ignoring few brightest and darkest pixels)
- Bilateral filtering (weight by spatial distance *and* intensity difference)

```
cv2.bilateralFilter(size, sigma_color, signal_spatial)
```

Bilateral filtering

Image: http://vision.ai.uiuc.edu/?p=1455

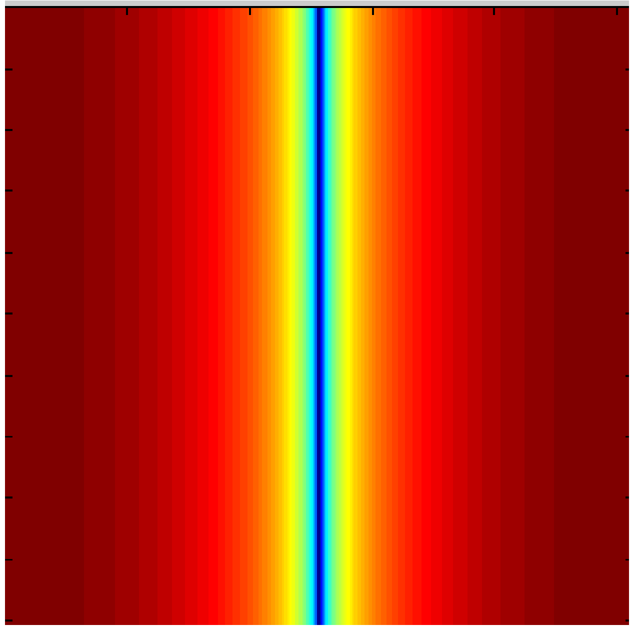
Review of Last 3 Days

- Filtering in spatial domain
  - Slide filter over image and take dot product at each position
  - Remember properties of linear filters

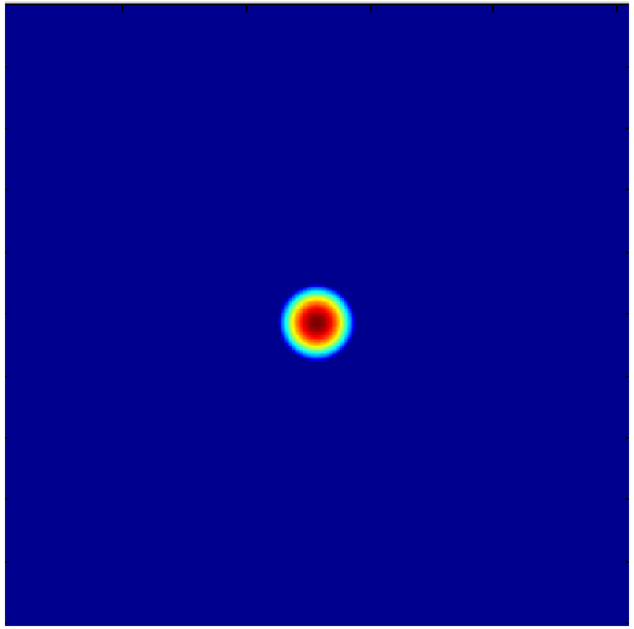
Review of Last 3 Days

- Linear filters for basic processing
  - Edge filter (high-pass)
  - Gaussian filter (low-pass)

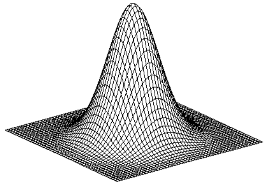
[-1 1]



FFT of Gradient Filter



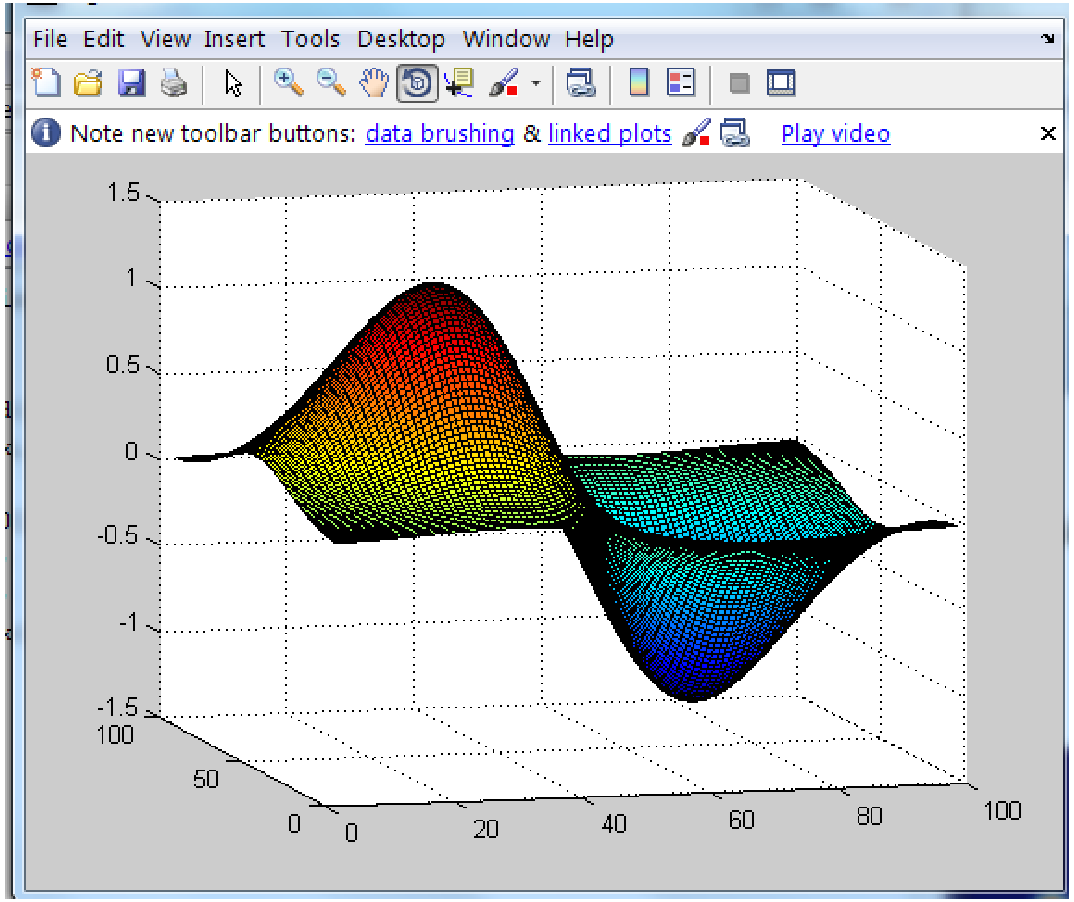
FFT of Gaussian



Gaussian

Review of Last 3 Days

- Derivative of Gaussian



Review of Last 3 Days

- Filtering in frequency domain
  - Can be faster than filtering in spatial domain (for large filters)
  - Can help understand effect of filter
  - Algorithm:
    1. Convert image and filter to FFT
    2. Pointwise-multiply FFTs
    3. Convert result to spatial domain with inverse FFT

Review of Last 3 Days

- Applications of filters
  - Template matching (SSD or normalized x-corr)
    - SSD can be done with linear filters, is sensitive to overall intensity
  - Gaussian pyramid
    - Coarse-to-fine search, multi-scale detection
  - Laplacian pyramid
    - Can be used for blending (later)
    - More compact image representation

Review of Last 3 Days

- Applications of filters
  - Downsampling
    - Need to sufficiently low-pass before downsampling
  - Compression
    - In JPEG, coarsely quantize high frequencies
  - Reducing noise (important for aesthetics and for later processing such as edge detection)
    - Gaussian filter, median filter, bilateral filter

Next lecture

- Light and color